# Multiarch crossbuilding
# How to use it, and what still needs work

Wookey

The Cross-building victim

## MultiarchCross

- Historical Context
- Autobuilder
- Toolchains and $stuff
- Multiarch for cross-deps
- Examples of things that break
- Current Status & Outstanding issues
- Bootstrapping

## Outline

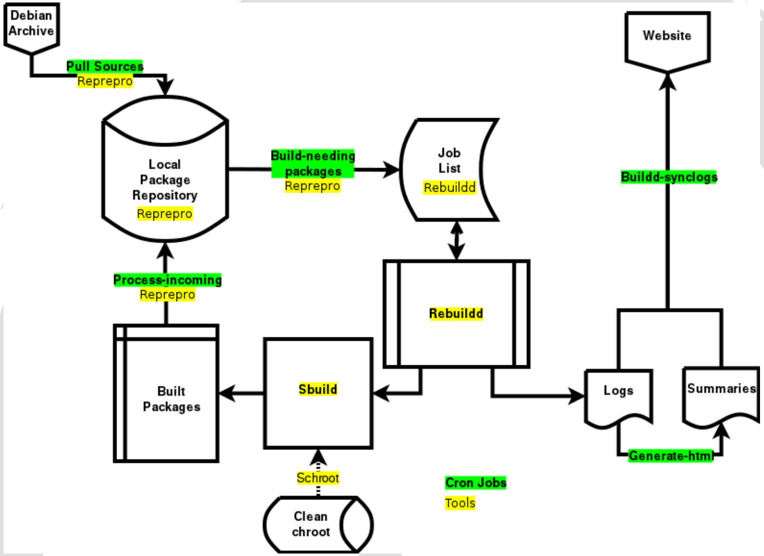
## Nomenclature

| | |
|---|---|
| Build | Machine/architecture you are building on |
| Host | Machine/architecture package is being built for |
| Target | Machine/architecture a compiler generates code for |

## Potted History

- 1997 - dpkg-cross (Roman hodek, Dave Schleef, Nikita Youschenko, Neil Williams)
- 2003 - emdebian cross-toolchains (Wookey, Hector Oron)
- 2004 - apt-cross
- 2007 - xapt, pdebuild-cross
- 2009 - chromiumos-build $\longrightarrow$ xdeb
- 2010 - linaro cross-toolchains
- 2011 - cross-build daemon
- 2012 - sbuild cross-support
- 2012 - multiarch-built cross-toolchains (Thibault Girka)

# Cross Build Daemon



xbuilder package in Linaro PPA

## Cross Build Daemon - Stats

- `http://people.linaro.org/~wookey/buildd/`

| Distro | live/dead | Total | Builds | Fails | Deps |
|---|---|---|---|---|---|
| Sid | live | 99 | 27 | 6 | 65 |
| Quantal | live | 93 | 37 | 24 | 32 |
| Precise | dead | 94 | 51 | 18 | 25 |

| | |
|---|---|
| Packages that build OK if deps present: | bzip2, dbus, fakeroot, fontconfig, gmp, gnupg, grep, libxcb, libtool, make-dfsg, ncurses, wget, xz-utils, zlib |

## Parts needed

- Toolchain
- Cross-build-deps
- Dpkg-cross autoconf caching
- Avoid running wrong-arch binaries

## Toolchain

There are 2 aspects to multiarching toolchains

- System search paths
  - path for libs and system headers (<> includes)
  - Previously previously /usr/include/ (native), /usr/<triplet>/include (cross)
  - Now always /usr/include/<triplet>:/usr/include/
  - Previously previously /usr/lib/ (native), /usr/<triplet>/lib (cross)
  - Now always /usr/lib/<triplet>:/usr/lib:/lib/<triplet>:/lib
- Build mechanism
  - Previously dpkg-cross libc6 for armel to make libc6-armel-cross arch all
  - Now Depend on libc6:armel (libgomp:armel, libmudflap:armel, etc)

## Autoconf caching

dpkg-cross provides `/etc/dpkg-cross/cross-config.cache` and `/etc/dpkg-cross/cross-config.<arch>`

- ac_cv_sizeof_float=4
- coreutils gl_cv_func_fstatat_zero_flag=yes
- dbus ac_cv_have_abstract_sockets=yes
- shadow ac_cv_func_setpgrp_void=yes
- bash bash_cv_job_control_missing=present
- sudo sudo_cv_func_unsetenv_void=no

## Bits and Bobs

Other things are needed for a smooth experience

- build-essential-<arch> packages
- cross-pkg-config
- toolchain defaults links (arm-linux-gnueabi-gcc → arm-linux-gnueabi-gcc-4.7)
- more cross-utilites. . .

## Outline

## Multiarch terminology

**Multi-arch-ready packages are given an extra field Multi-Arch**

- same: *(libraries)*
  can be co-installed and can only satisfy deps within the arch
- foreign: *(tools)*
  can not be co-installed can satisfy deps for any arch
- allowed: *(both)*
  can be either. Depending packages specify which is wanted

dpkg has support for reference-counting of (doc-)files from co-installable packages that overlap

## Dependency satisfaction

```
dpkg --add-architecture armhf
apt-get build-dep -a armhf <package>
```

- Described at `https://wiki.ubuntu.com/MultiarchCross`

| | Build-Depends: foo | Build-Depends: foo:any | Build-Depends: foo:native |
|---|---|---|---|
| no Multi-Arch field | DEB_HOST_ARCH | disallowed | DEB_BUILD_ARCH |
| Multi-Arch: same | DEB_HOST_ARCH | disallowed | DEB_BUILD_ARCH |
| Multi-Arch: foreign | any, pref DEB_BUILD_ARCH | disallowed | disallowed |
| Multi-Arch: allowed | DEB_HOST_ARCH | any, pref DEB_BUILD_ARCH | DEB_BUILD_ARCH |

- So tools all need to be marked Multi-Arch: foreign (over 1000)
- Or implement #666772 *apt cross-build-dep handling should be liberal with Arch: all packages*

## Transitive Build-deps

A package Build-depends: libdb-dev

```
Package: libdb-dev
Depends: libdb5.1-dev
```

libdb-dev used to be arch all. Now needs to be arch any to get libdb5.1-dev:DEB_HOST_ARCH

## Outline

## Crossbuilding Issues - Wrong arch tools

- libnih: /≪PKGBUILDDIR≫/nih-dbus-tool/.libs/lt-nih-dbus-tool: No such file or directory
- help2man Runs command –help to get manpage

## Crossbuilding Issues - config scripts

Arch-dependent config scripts

- tcl8.5 /usr/lib/tcl8.5/tclConfig.sh
- curl /usr/bin/curl-config
- freetype /usr/bin/freetype-config
- guile /usr/bin/guile-config
- icu /usr/bin/icu-config
- krb5 /usr/bin/krb5-config
- pcre /usr/bin/pcre-config –libs
  → `-L/usr/lib/x86_64-linux-gnu -lpcre`
- apr /usr/bin/apr-config –cc
  → `x86_64-linux-gnu-gcc`

## Crossbuilding Issues - cross-install failures

M-A: same packages which run foreign-arch binaries during install

- libgvc5: libgvc5-config-update
- libglib2.0-0: glib-compile-schemas, gio-querymodules (fixed)
- libgdk-pixbuf2.0-0: gdk-pixbuf-query-loaders
- libgtk2.0-0: gtk-query-immodules-2.0
- libgtk-3-0: gtk-query-immodules-3.0

## Crossbuilding Issues - Arch-dependent tools

- chrpath: Modifies rpath in binary
- gobject-introspection (atk, gstreamer, pango, udev, libsoup, gdk-pixbuf, gnome-everything)
  - `g-ir-scanner` dlopens binaries to scan for gobject interfaces and writes (arch-specific) xml descriptions

## Making your packages cross-friendly

- `include /usr/share/dpkg/architecture.mk`
- Use pkg-config and autotools or cmake
- Don't run just-built binaries when crossing
- read `http://wiki.debian.org/CrossBuildPackagingGuidelines` and `https://wiki.linaro.org/Platform/DevPlatform/CrossCompile/CrossPatching`

## Outline

## Setup

1. install dependencies
2. clone git
3. download Debian Sid Packages.bz2 and Sources.bz2
4. compile Ocaml code

```
\$ apt-get install --no-install-recommends libdose3-ocaml-dev \
> camlp4 make wget dctrl-tools bzip2 xdot
\$ git clone git://gitorious.org/debian-bootstrap/bootstrap.git
\$ cd bootstrap
\$ make setup
\$ make
```

- most programs take the -v and --progress options
- if errors occur, re-execute with -vv

## Utility Scripts

`./basebuildsystem.native` creates a list of source packages that are required to be cross compiled for a minimal native build system

`./reduced_dist.native` creates a minimal distribution to make it possible to start concentrating on *core* packages first

`./check_source_buildability.native` checks if all source packages in the distribution can potentially be built, given the amount of binary packages

`./crosseverything.native` find one of the possible sets of packages that, if cross compiled would make the whole archive buildable

`./check_binary_to_source_mapping.native` checks if there are binary packages that have no corresponding source package

## ./basebuildsystem.native

- why?
  - something must come out of nothing before one can start native building on a new architecture → crosscompiling
  - `./basenocycles.native` needs a minimal system to start dependency analysis
- what packages make a minimal build system?
  - `priority:essential` packages and dependencies
  - `build-essential` and dependencies
- how to execute?
  - `./basebuildsystem.native -v Pkg.bz2 Src.bz2`
  - output will be `min-cross-sources.list` containing a list of source packages that build above binary packages
  - `min-cross-sources.list` is needed by the main program, `./basenocycles.native`

## ./basebuildsystem.native - statistics

| | Debian Sid | Ubuntu Precise |
|---|---|---|
| priority:required | 37 | 70 |
| essential:true | 25 | 24 |
| buildessential:true | 11 | 44 |
| the above plus dependencies | 106 | 140 |
| number of source packages | 55 | 75 |

- packages to be crossed for ubuntu only: apt, busybox, cpio, dbus, elfutils, fakeroot, glib2.0, gnupg, ifupdown, initramfs-tools, iproute, klibc, libalgorithm-diff-xs-perl, libdrm, libffi, libnih, libpciaccess, libpng, libusb, module-init-tools, mountall, openssl, pcre3, plymouth, procps, python2.7, python-defaults, udev, upstart
- packages to be crossed for debian only: liblocale-gettext-perl, libsemanage, libsepol, libtext-charwidth-perl, libtext-iconv-perl, texinfo, ustr

## ./reduced_dist.native

- why?
  - analyzing all of Debian at once is slow due to its size
  - easier analysis if packages unrelated to a set of core packages are not considered
- what is a reduced distribution?
  - contains a set of source packages A and a set of binary packages B
  - all binary packages in B can be built from the source packages in A
  - all source packages in A are buildable with the binary packages in B
- how to execute?
  - `./reduced_dist.native -v Pkg.in Src.in Pkg.out Src.out`
  - program will ask if important packages should be included or not
  - program will ask for an additional packages (and its dependencies) to be included (eg: task-gnome-desktop)

## ./reduced_dist.native - statistics

| | Debian Sid | Ubuntu Precise |
|---|---|---|
| src/bin in original repositories | 18266/37781 | 3305/8076 |
| src/bin without important | 645/2324 | 522/1838 |
| src/bin with important | 679/2403 | 541/1871 |
| src/bin imp. + task-gnome-desktop | 855/2853 | - |
| src/bin imp. + ubuntu-desktop | - | 718/2467 |
| src/bin imp. + task-kde-desktop | 791/2769 | - |
| src/bin imp. + kubuntu-desktop | - | 618/2158 |

- Debian Sid reduced dists are incomplete because src:libvdpau is currently unbuildable (ia32-libs-dev)

## ./check_source_buildability.native

- `./check_source_buildability.native -v --progress Pkg.bz2 Src.bz2`
- checks which packages can potentially be compiled given all packages of the distribution

| | Debian Sid | Ubuntu Precise |
|---|---|---|
| number of source packages | 18266 | 3305 |
| compilable source packages | 18207 | 3305 |
| time needed to check | 3:50 h | 0:12 h |

## ./crosseverything.native

- there exist two methods to break dependency cycles: staged build dependencies and cross compilation
- through multi-arch and autoconf, many packages can be cross compiled without any modification to the package source
- if breaking of a cycle is possible by cross building a package that takes no extra effort to make it cross build, this solution should be taken to break the cycle
- to know which packages cross compile without modification, all packages in the archive must be tried to cross compile
- since the archive is too big to make this feasible, get a list of packages that, if cross compiled, would make the whole archive buildable
- this list is neither unique nor minimal but represents some crucial packages that it would make sense to check for cross compilability
- if time permits, the whole archive can be checked later

## ./crosseverything.native - statistics

- `./crosseverything.native -v --progress Pkg.bz2 Src.bz2`

| | task-gnome-desktop | Precise | ubuntu-desktop |
|---|---|---|---|
| to cross | 158 | 176 | 157 |
| time | 0:09 h | 2:33 h | 0:06 h |

## `./basenocycles.native`

- `./basenocycles.native -v --progress Pkg.bz2 Src.bz2`
- main analysis program
- needs `min-cross-sources.list` created by `./basebuildsystem.native` for the list of packages that are cross compiled for a basic build system
- reads from `add-cross-sources.list` if available for additional packages that were chosen to be cross compiled
- tries to build everything it can given the base system
- if not all packages could be built, assist in analyzing the situation

## ./basenocycles.native - main menu

1. investigate package
   1. find packages to cross compile
   2. calculate dependency graph
2. find a candidate package to investigate
   1. list binary packages that are most needed
   2. list source packages with the least dependencies missing
   3. TODO: list smallest cycles in the archive
   4. TODO: list source packages with only unimportant dependencies missing
   5. TODO: list binary packages with least vertices in their dependency graph

## ./basenocycles.native - graph menu

1. full graph
   1. show graph
   2. show statistics
   3. save DOT graph
2. scc with investigated package
   1. show graph
   2. show statistics
   3. save DOT graph
3. scc #2 [...]
4. scc #3 [...]
5. scc #N [...]

## ./basenocycles.native - example 1

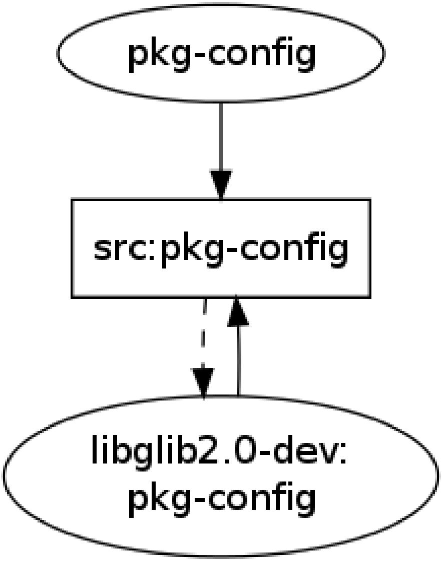
1. create a mini distribution for faster execution and to not consider packages unrelated to base packages; add important packages and task-gnome-desktop
   - `./reduced_dist.native -v Sid-Packages.bz2 Sid-Sources.bz2 Sid-Packages-reduced.bz2 Sid-Sources-reduced.bz2`
2. calculate set of base packages that have to be cross compiled for minimal native compilation
   - `./basebuildsystem.native -v Sid-Packages-reduced.bz2 Sid-Sources-reduced.bz2`

## ./basenocycles.native - example 2

1. `./basenocycles.native -v --progress Sid-Packages-reduced.bz2 Sid-Sources-reduced.bz2`
2. find out that just with the bare base system, only 8 of 905 packages can be built
3. select "find a candidate package to investigate" and find out that debhelper is a build dependency of 876 of the 905 source packages
4. therefor choosing debhelper as the first package to make available, choose "investigate package", type "debhelper"
5. find out that the dependency graph covers nearly the whole archive
6. decide that making all involved packages compile without debhelper is harder than making some packages cross compile
7. choosing "find packages to cross compile" will give the list of packages that, if cross compiled, would make debhelper available
8. exit the program and add the list to `add-cross-sources.list`

## ./basenocycles.native - example 3

1. start ./basenocycles.native again and discover that now 207 out of 905 packages can be built
2. find out that pkg-config is needed by 235 source packages
3. calculate the dependency graph and investigate the "full graph"
4. since pkg-config cannot be compiled without libglib2.0-dev, append it to the list of packages to additionally cross compile: add-cross-sources.list
5. exit the program
6. restarting it shows, that now with pkg-config, 237 out of 905 packages can be built



## Future

- use cycle enumeration capabilities to find dependency cycles and break cycles using staged build dependencies

## needed

- more input of what cross builders need or like to get to know for decision making
- list of likely optional and likely hard build dependencies
- another implementation of enumeration of elementary circuits of a directed graph
- papers on enumeration of elementary circuits as many are behind a paywall
- name for the software (the hardest bit)

## The End

Thanks to various people

- Linaro for funding this work
- Team GSOC
  - Johanes Schauer (bootsrtapping analysis)
  - Thibaut Girka (multiarch cross-toochains)
  - Patrick McDermott (utils and dep-cycle breaking)
- Various useful people: Steve Langasek, Colin Watson, Marcin Juśkiewicz, Hector Oron, Neil Williams, Pietro Abate, Jonathan Austin, Harry Liebel, Loic Minier

Further reading: `https://wiki.linaro.org/Platform/DevPlatform/CrossCompile/CrossBuilding`