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
```

- implemented in OCaml using the Dose3 framework
- licensed under LGPL3+
- most programs take the `-v` and `--progress` options
- if errors occur, re-execute with `-vv`

Programs

`./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 are built from no source package

`./basenocycles.native` main program for dependency analysis

./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

- source 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
- source 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

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

./basenocycles.native - statistics

1. top X source packages with most/least unmet build dependencies
2. top X binary packages with most/least unbuilt sources
3. top X source packages with most/least binary packages needing it
4. top X binary packages with most/least source packages build-depending on it
5. top X source packages with highest/lowest ratio of binary packages that need it and binary packages it depends on
6. top X binary packages with highest/lowest ratio of source packages that build-depend on it and source packages it builds from
7. top X most/least connected source packages
8. top X most/least connected binary packages
9. TODO: top X source packages with only unimportant dependencies missing

./basenocycles.native - example 1

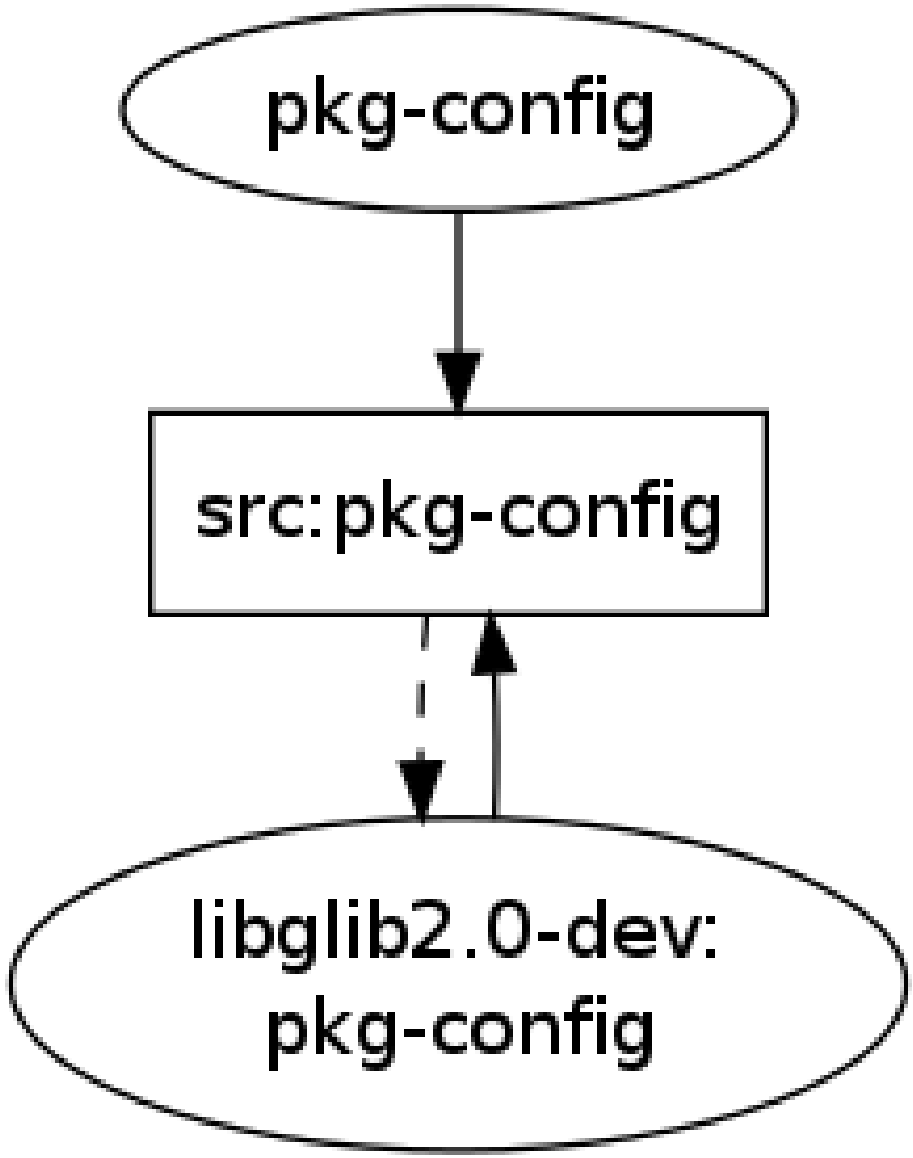
1. create a mini distribution for faster execution and to not consider packages unrelated to base packages; add important packages and task-gnome-desktop
   - `./reduced_dist.native -v Sid-Packages.bz2 Sid-Sources.bz2 Sid-Packages-reduced.bz2 Sid-Sources-reduced.bz2`
2. calculate set of base packages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



Improvements over last year

- native installability is checked, not cross installability
- the generated graph takes installability and availability of packages within the current state of the system into account
- if a source package build dependency is installable in the current state of the system, it is not added to the graph
- if a binary package runtime dependency is available in the current state of the system, it is not added to the graph
- allows to enumerate all cycles in the dependency graph

enumeration of elementary circuits in a directed graph

- needed to find dependency cycles as well as breaking cycles using staged build dependencies
- using Johnson's algorithm
- validation through comparing output with implementations of Johnsons's algorithm in Java, Tarjan's algorithm in Python and Hawick and James' algorithm in D
- there seems to be no well tested implementation of a circuit enumeration algorithm in well-known libraries
- code at `https://github.com/josch/cycle_test`

build dependency cycles in Precise

- no statistics for Debian yet as libvdpau is unbuildable
- limit maximum cycle size to 2 or 4, otherwise there are easily more cycles than will fit into RAM
- number of cycles in a reduced Precise distribution: 21 cycles of length 2 and 124 cycles of length 4
- most cycles of length four include one of the cycles of length 2
- smallest cycles should be broken first
- list of cycles up to length 4: `http://mister-muffin.de/p/G3kD.txt`

Future

- use cycle enumeration capabilities to find 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)