Are distributions really boring and a solved problem?

Lucas Nussbaum

lucas@debian.org

Debian Project Leader

(Slides will be available)

Alternate title, briefly considered, but rejected (thankfully):

Alternate title, briefly considered, but rejected (thankfully):

Debian Jessie is released, now what?

Lucas Nussbaum

lucas@debian.org

Debian Project Leader

(Slides will be available)

The truth about this talk:

The truth about this talk:

Hard problems I would like someone to solve :-)

Lucas Nussbaum

lucas@debian.org

Debian Project Leader

(Slides will be available)

The problems that distros were trying to solve a decade ago largely are seen as not only solved, but kind of boring.

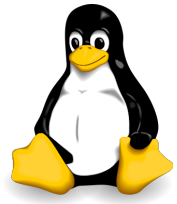
Matthew Miller (Fedora Project Leader)

The problems that distros were trying to solve a decade ago largely are seen as not only solved, but kind of boring.

Matthew Miller (Fedora Project Leader)

OK, which problems should we try to solve today?

Distributions in the Free Software world



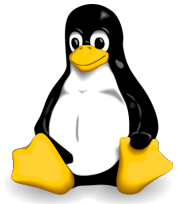
Alice

Bob

Carol

Dave

Distributions in the Free Software world



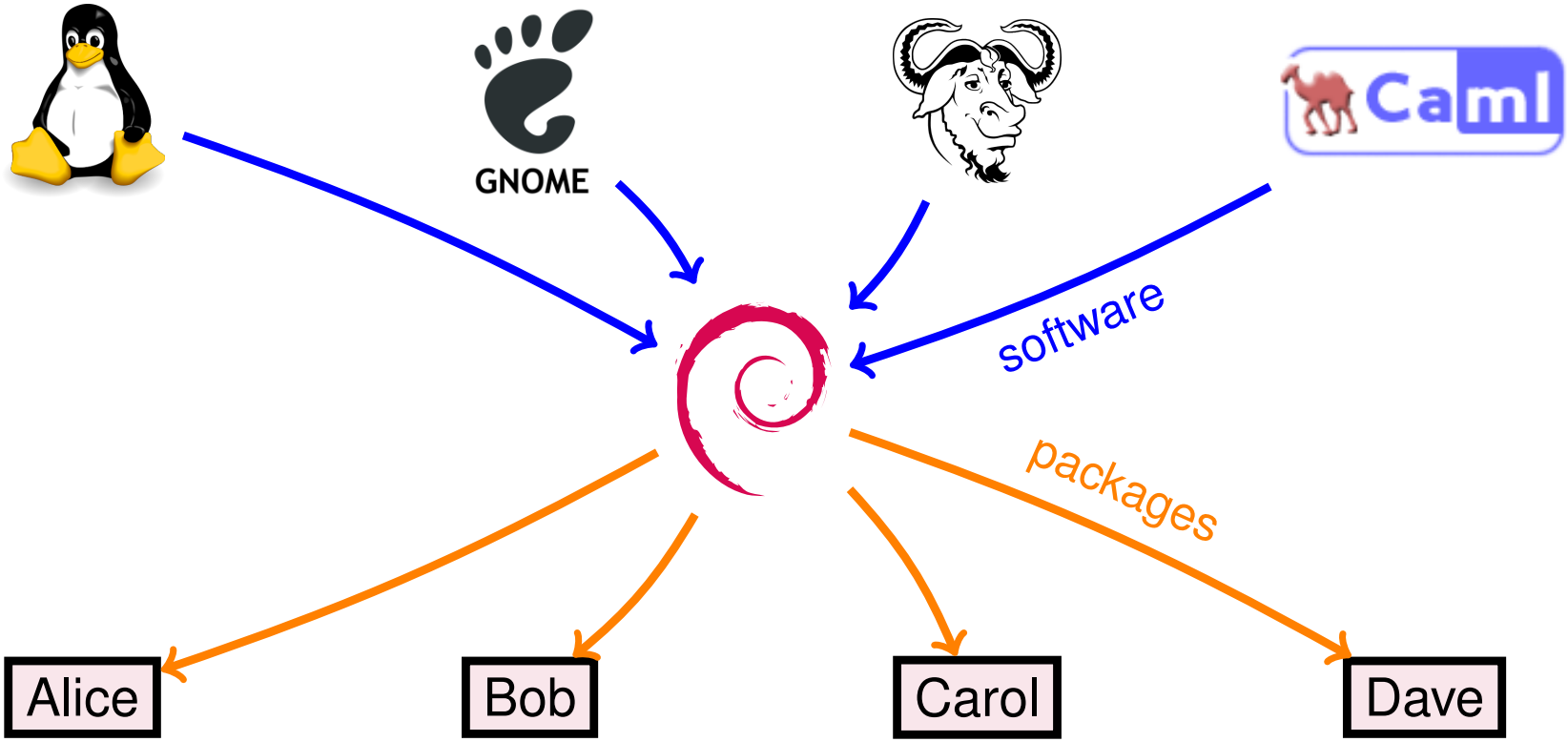
Alice

Bob

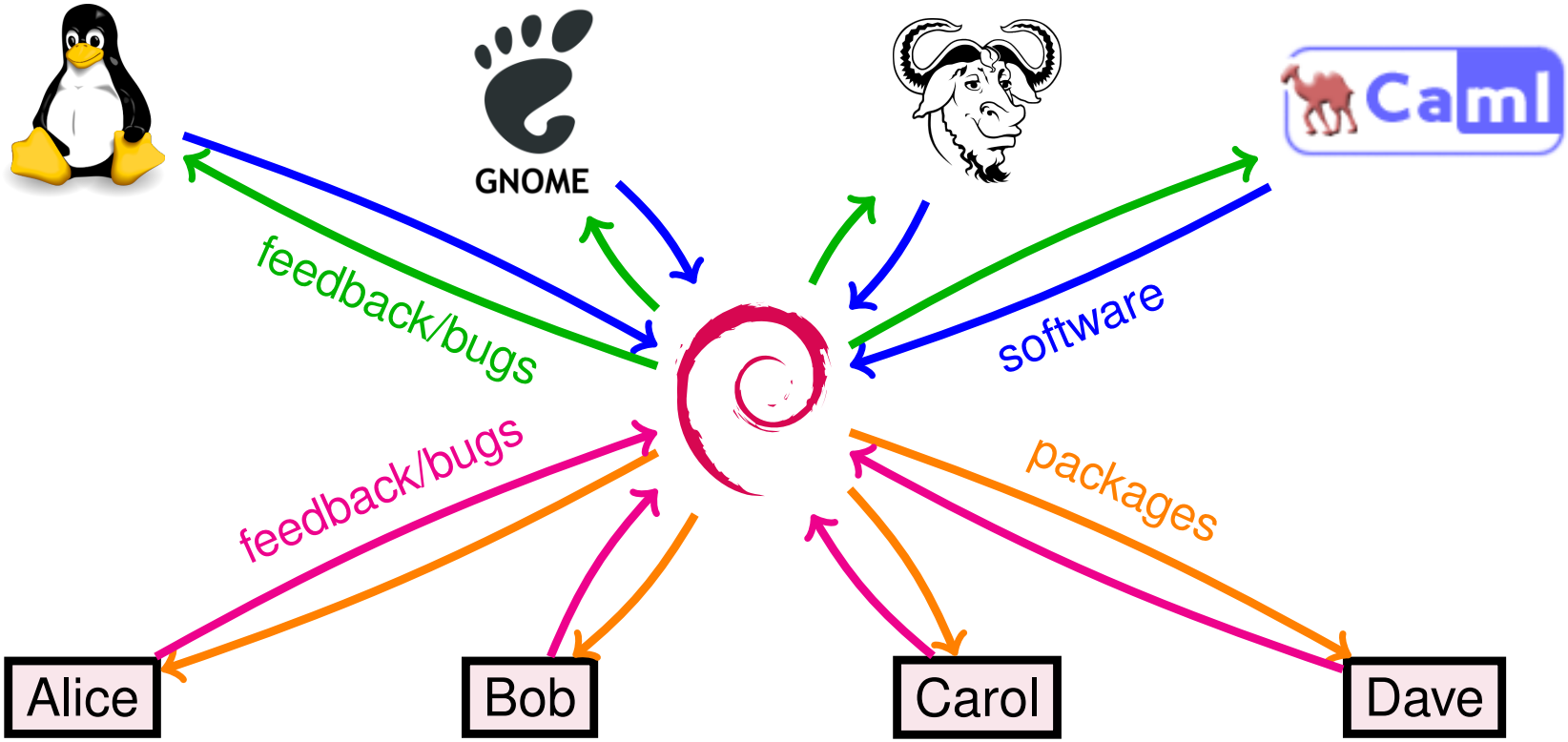
Carol

Dave

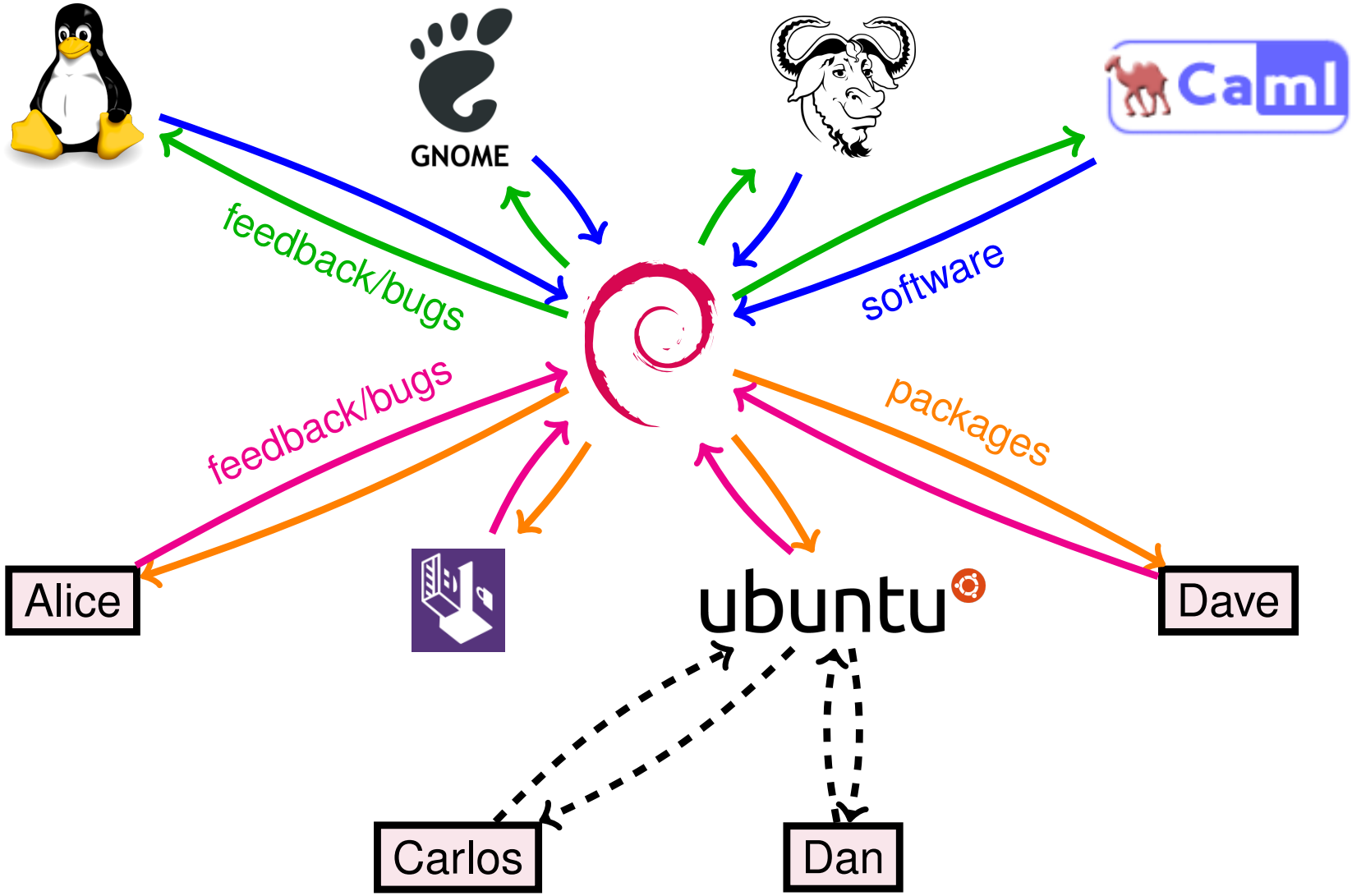
Distributions in the Free Software world



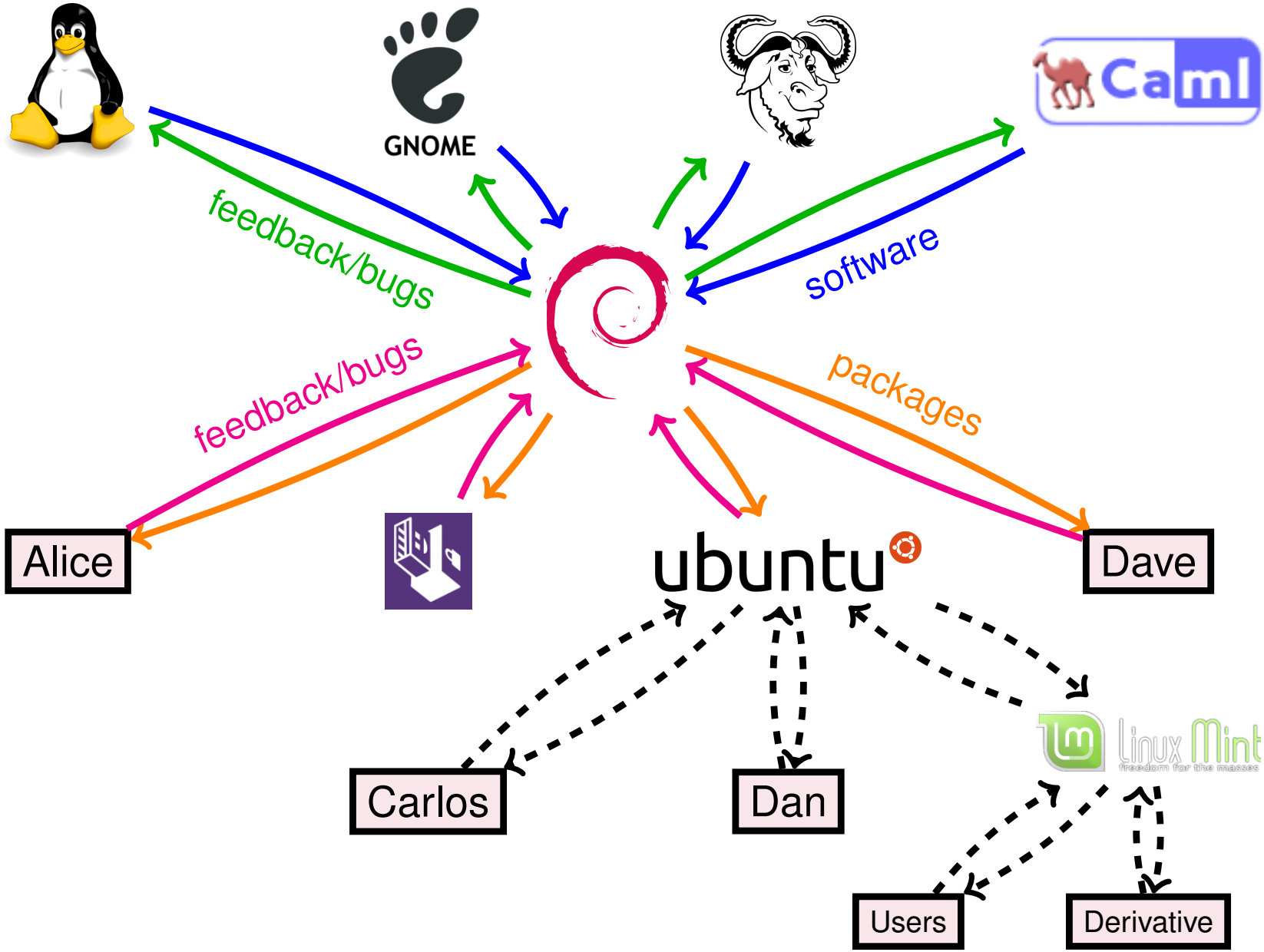
Distributions in the Free Software world



Distributions in the Free Software world



Distributions in the Free Software world



Distributions' role in Free Software

What we do well:

- Provide a unified interface for users to upstream projects: package managers, mirrors network
  - Hiding all the subtle, annoying differences
  - Supplementing upstreams, sometimes
- Integrate upstream projects, resolving incompatibilities
  - Cleaning Free Software from problems that upstreams often ignore

Distributions' role in Free Software

What we do well:

- Provide a unified interface for users to upstream projects: package managers, mirrors network
  - Hiding all the subtle, annoying differences
  - Supplementing upstreams, sometimes
- Integrate upstream projects, resolving incompatibilities
  - Cleaning Free Software from problems that upstreams often ignore

What we don't do that well:

- Provide an intermediate support layer
- Act as mediators between upstreams, derivatives, users
- Meet all our users' needs

Distributions' role in Free Software

What we do well:

- Provide a unified interface for users to upstream projects: package managers, mirrors network
  - Hiding all the subtle, annoying differences
  - Supplementing upstreams, sometimes
- Integrate upstream projects, resolving incompatibilities
  - Cleaning Free Software from problems that upstreams often ignore

What we don't do that well:

- Provide an intermediate support layer
- Act as mediators between upstreams, derivatives, users
- Meet all our users' needs

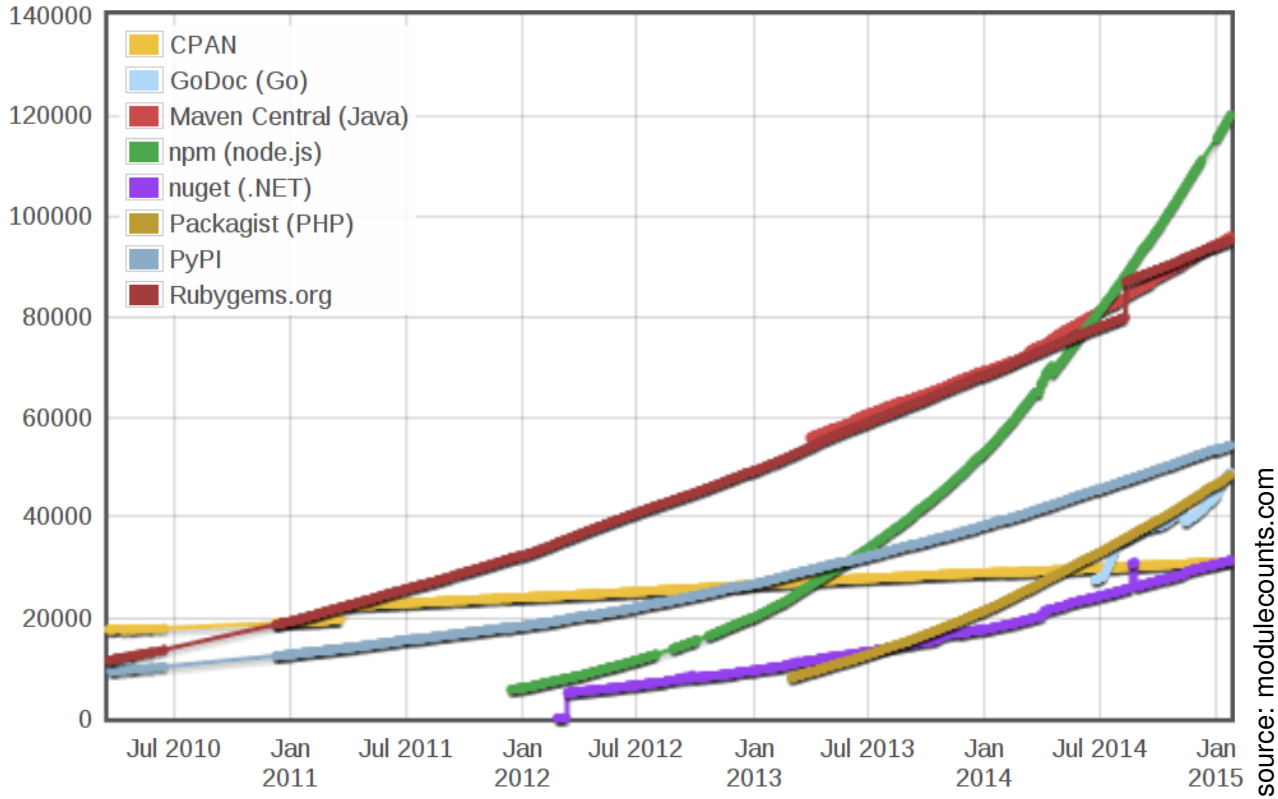
Can we do better?

Meeting all our users' needs

- Quick poll:
  - During the last year, who has had to install something from sources, or using unofficial packages, or gems?

Meeting all our users' needs

- Quick poll:
  - During the last year, who has had to install something from sources, or using unofficial packages, or gems?
- A lot of software not packaged, and we are losing the race



Meeting all our users' needs

- Quick poll:
  - During the last year, who has had to install something from sources, or using unofficial packages, or gems?
- A lot of software not packaged, and we are losing the race
- Not the version you need in the release you are using (or in backports)
  974 packages in wheezy-backports, vs 21151 packages in jessie

Meeting all our users' needs

- Quick poll:
  - During the last year, who has had to install something from sources, or using unofficial packages, or gems?
- A lot of software not packaged, and we are losing the race
- Not the version you need in the release you are using (or in backports)
  974 packages in wheezy-backports, vs 21151 packages in jessie

How can aim for 100% coverage of our users' needs?

Meeting all our users' needs

- Quick poll:
  - During the last year, who has had to install something from sources, or using unofficial packages, or gems?
- A lot of software not packaged, and we are losing the race
- Not the version you need in the release you are using (or in backports)

  974 packages in wheezy-backports, vs 21151 packages in jessie

How can aim for 100% coverage of our users' needs?

1. Get more efficient at packaging
2. Provide additional levels of support

Distributions contributors

- More devops than pure developers
  - Started by scratching an itch: ease installation of software
    - Frequent need for sysadmins, not so much for developers
  - Community that is excellent at:
    - Dealing with obscure dirty Unix stuff in various languages
    - Forcing various things into working together

Distributions contributors

- More devops than pure developers
  - Started by scratching an itch: ease installation of software
    - Frequent need for sysadmins, not so much for developers
  - Community that is excellent at:
    - Dealing with obscure dirty Unix stuff in various languages
    - Forcing various things into working together
  - But often, not so great at designing and writing complex frameworks
  - Tendency to add layers of glue, and avoid deep refactoring

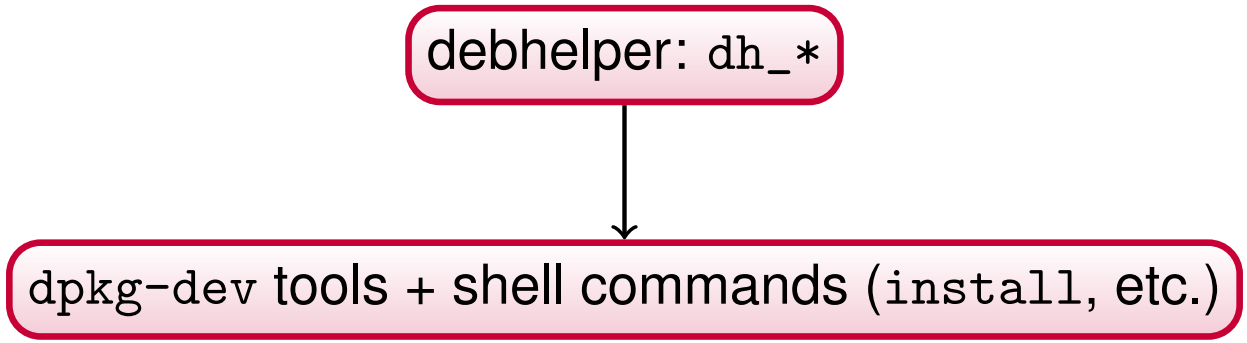
Distributions contributors

- More devops than pure developers
  - Started by scratching an itch: ease installation of software
    - Frequent need for sysadmins, not so much for developers
  - Community that is excellent at:
    - Dealing with obscure dirty Unix stuff in various languages
    - Forcing various things into working together
  - But often, not so great at designing and writing complex frameworks
  - Tendency to add layers of glue, and avoid deep refactoring
- That's also why it's harder to recruit compared to other projects
  - Mismatch with typical university curriculums

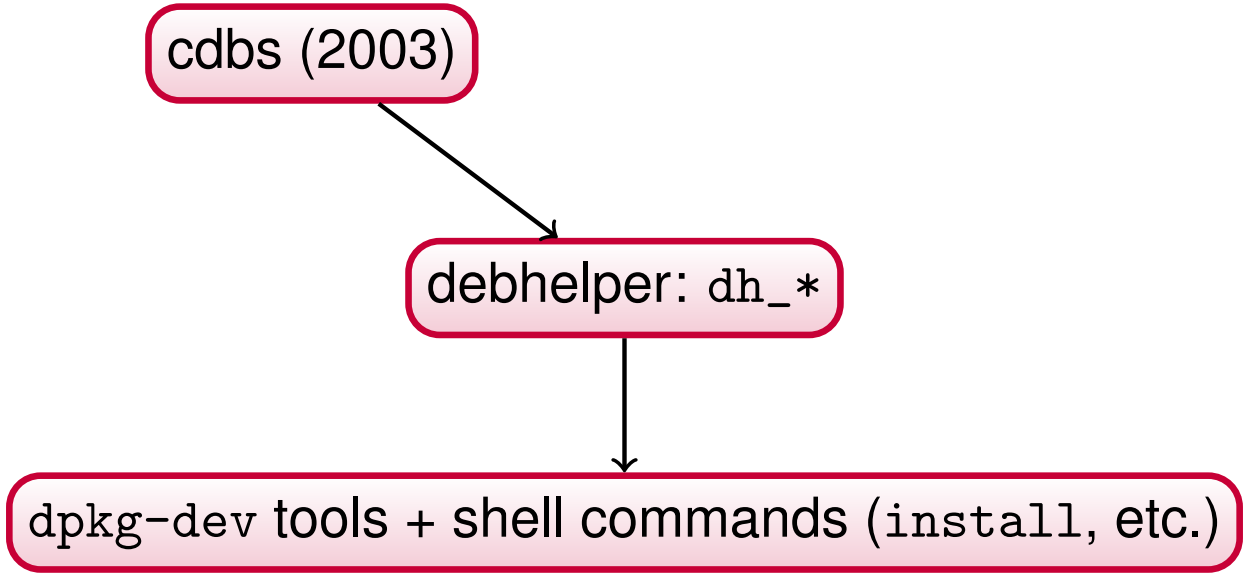
Debian packaging stack

`dpkg-dev` tools + shell commands (`install`, etc.)

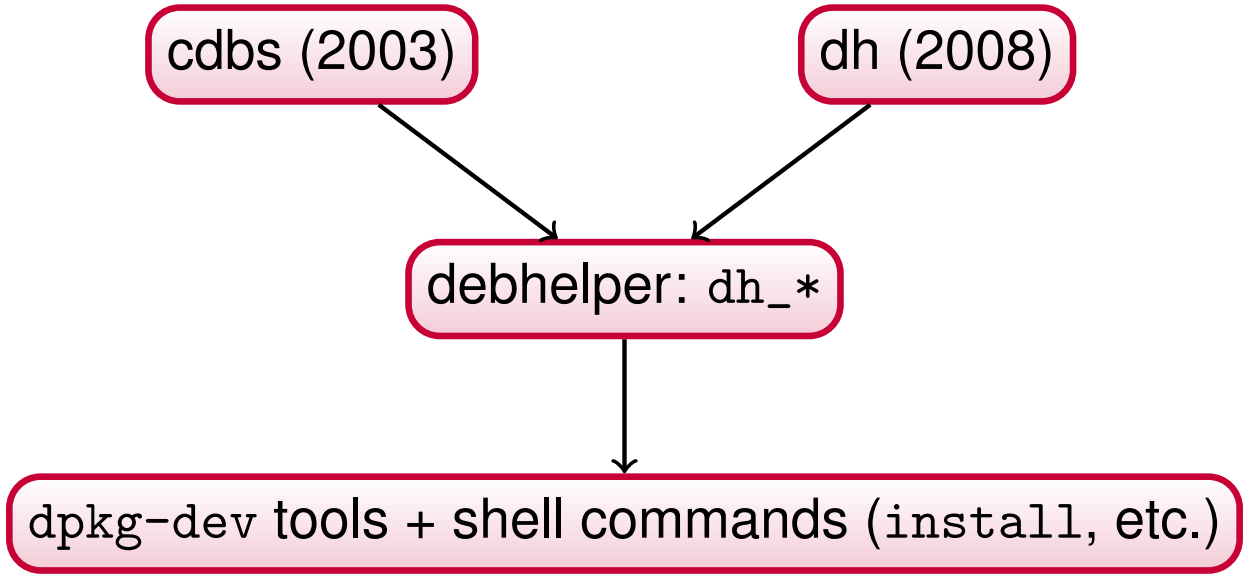
Debian packaging stack



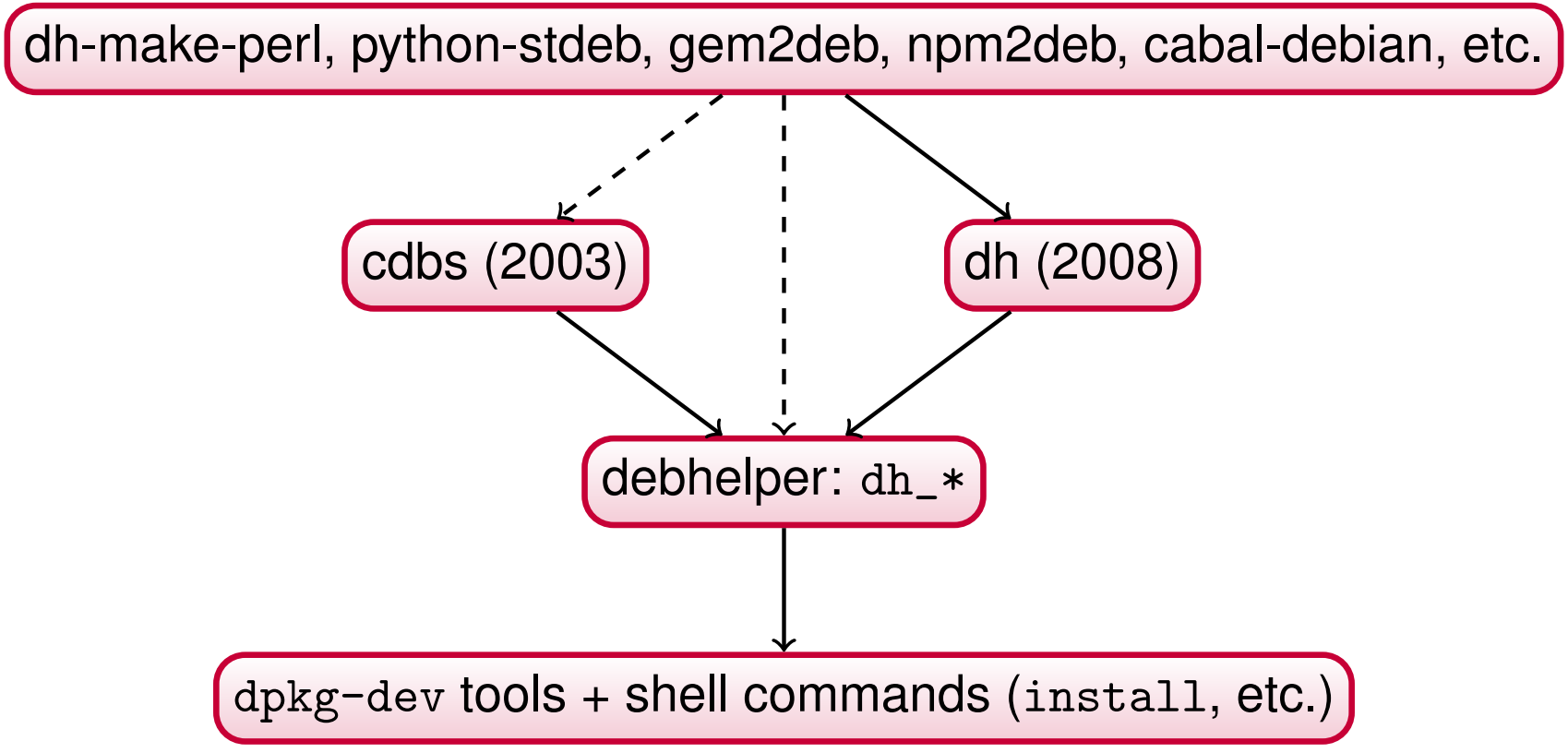
Debian packaging stack



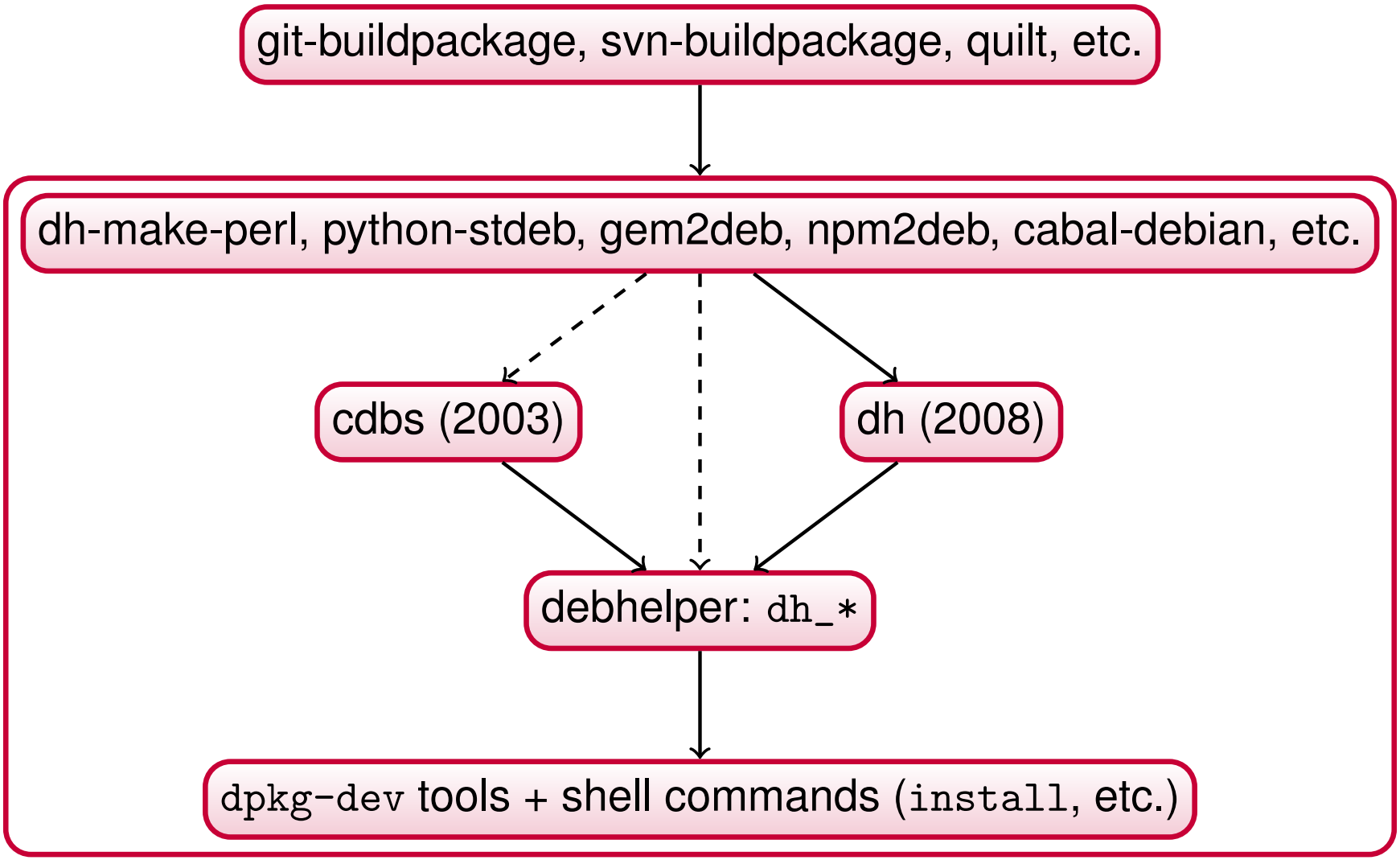
Debian packaging stack



Debian packaging stack

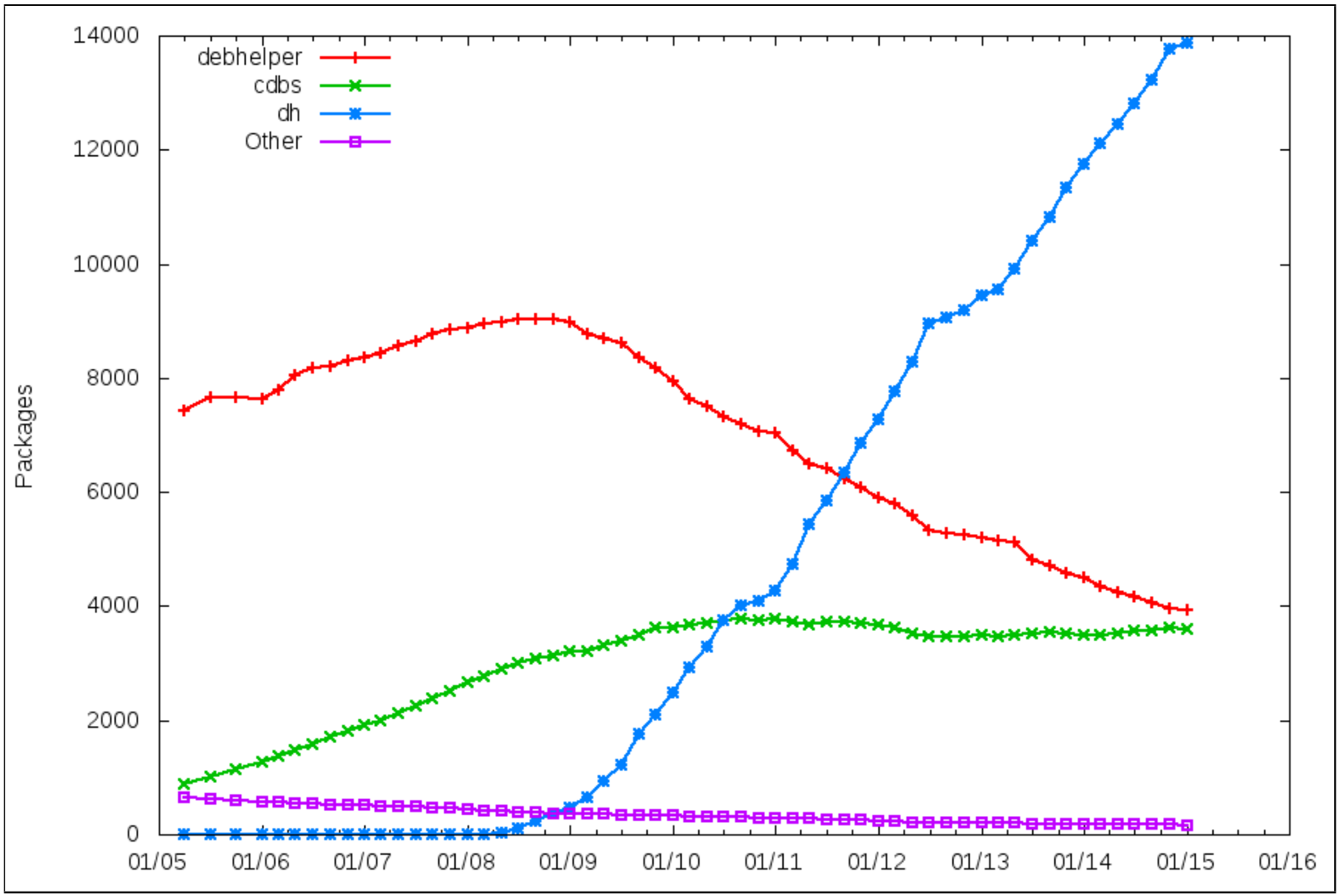


Debian packaging stack



Debian packaging stack: problems

1. **We are not really moving away from deprecated tools**



Debian packaging stack: problems

2. **We are not hiding lower-level tools**

visible surface area: debhelper

-- --add-udeb --autodest --dbg-package --destdir --dirs-only --dpkg-gencontrol-params --dpkg-shlibdeps-params
--error-handler --fail-missing --filename --flavor --ignore --init-script --keep-debug --language --list-missing
--mainpackage --name --no-act --no-restart-on-upgrade --no-start --priority --remove-d --restart-after-upgrade
--sourcedir --version-info

-A -L -N -P -V -X -a -d -i -k -l -m -n -o -p -s -u -v -x

DH_ALWAYS_EXCLUDE DH_COMPAT DH_NO_ACT DH_OPTIONS DH_VERBOSE

138 items

debian/<package>.bug-control debian/<package>.bug-presubj debian/<package>.bug-script
debian/<package>.compress debian/<package>.cron.<type> debian/<package>.default debian/<package>.dirs
debian/<package>.docs debian/<package>.emacsen-install debian/<package>.emacsen-remove
debian/<package>.emacsen-startup debian/<package>.examples debian/<package>.files
debian/<package>.gconf-defaults debian/<package>.gconf-mandatory debian/<package>.if-<script>
debian/<package>.info debian/<package>.init debian/<package>.init.d debian/<package>.links
debian/<package>.lintian-overrides debian/<package>.logcheck.<type> debian/<package>.logrotate
debian/<package>.manpages debian/<package>.menu debian/<package>.modprobe debian/<package>.modules
debian/<package>.pam debian/<package>.ppp.ip-<script> debian/<package>.sharedmimeinfo
debian/<package>.suid debian/<package>.symbols debian/<package>.udev debian/<package>.wm
debian/compat

dh_bugfiles dh_builddeb dh_clean dh_compress dh_desktop dh_fixperms dh_gconf dh_gencontrol dh_icons
dh_install dh_installcatalogs dh_installchangelogs dh_installcron dh_installdeb dh_installdebconf dh_installdirs
dh_installdocs dh_installemacsen dh_installexamples dh_installifupdown dh_installinfo dh_installinit
dh_installlogcheck dh_installlogrotate dh_installman dh_installmanpages dh_installmenu dh_installmime
dh_installmodules dh_installpam dh_installppp dh_installudev dh_installwm dh_installxfonts dh_link dh_lintian
dh_listpackages dh_makeshlibs dh_md5sums dh_movefiles dh_perl dh_prep dh_python dh_scrollkeeper
dh_shlibdeps dh_strip dh_suidregister dh_testdir dh_testroot dh_testversion dh_undocumented dh_usrlocal

(Joey Hess at DebConf 9: *Not Your Grandpa's Debhelper*)

Debian packaging stack: problems

2. **We are not hiding lower-level tools**

visible surface area: CDBS

/usr/share/cdbs/1/class/ant.mk /usr/share/cdbs/1/class/autotools.mk /usr/share/cdbs/1/class/cmake.mk /usr/share/cdbs/1/class/docbookxml.mk
/usr/share/cdbs/1/class/gnome.mk /usr/share/cdbs/1/class/hbuild.mk /usr/share/cdbs/1/class/hlibrary.mk /usr/share/cdbs/1/class/kde.mk
/usr/share/cdbs/1/class/langcore.mk /usr/share/cdbs/1/class/makefile.mk /usr/share/cdbs/1/class/perlmodule.mk /usr/share/cdbs/1/class/python-distutils.mk
/usr/share/cdbs/1/class/qmake.mk /usr/share/cdbs/1/rules/buildcore.mk /usr/share/cdbs/1/rules/debhelper.mk /usr/share/cdbs/1/rules/dpatch.mk
/usr/share/cdbs/1/rules/patchsys-quilt.mk /usr/share/cdbs/1/rules/simple-patchsys.mk /usr/share/cdbs/1/rules/tarball.mk

ANT_ARGS ANT_ARGS_<package> ANT_HOME ANT_OPTS BUILD_GHC6 CABAL_PACKAGE CFLAGS CMAKE CXXFLAGS DEB_AC_AUX_DIR
DEB_ALL_PACKAGES DEB_ANT_BUILD_TARGET DEB_ANT_CHECK_TARGET DEB_ANT_CLEAN_TARGET DEB_ANT_INSTALL_TARGET DEB_ARCH
DEB_ARCH_PACKAGES DEB_AUTO_UPDATE_ACLOCAL DEB_AUTO_UPDATE_AUTOCONF DEB_AUTO_UPDATE_AUTOHEADER
DEB_AUTO_UPDATE_AUTOMAKE DEB_AUTO_UPDATE_DEBIAN_CONTROL DEB_AUTO_UPDATE_LIBTOOL DEB_BUILDDIR
DEB_BUILD_DEPENDENCIES DEB_BUILD_MAKE_TARGET DEB_CLASSPATH DEB_CLEAN_ARGS DEB_CLEAN_MAKE_TARGET
DEB_CMAKE_INSTALL_PREFIX DEB_CMAKE_NORMAL_ARGS DEB_COMPLETE_EXCLUDE DEB_CONFIGURE_EXTRA_FLAGS
DEB_CONFIGURE_INCLUDEDIR DEB_CONFIGURE_INFODIR DEB_CONFIGURE_INVOKE DEB_CONFIGURE_LIBEXECDIR
DEB_CONFIGURE_LOCALSTATEDIR DEB_CONFIGURE_MANDIR DEB_CONFIGURE_NORMAL_ARGS DEB_CONFIGURE_PREFIX
DEB_CONFIGURE_SCRIPT DEB_CONFIGURE_SCRIPT_ENV DEB_CONFIGURE_SYSCONFDIR DEB_DESTDIR DEB_DH_ALWAYS_EXCLUDE
DEB_DH_DESKTOP_ARGS DEB_DH_GCONF_ARGS DEB_DH_GENCONTROL_ARGS DEB_DH_GENCONTROL_ARGS_<package>
DEB_DH_GENCONTROL_ARGS_ALL DEB_DH_ICONS_ARGS DEB_DH_MAKESHLIBS_ARGS DEB_DH_MAKESHLIBS_ARGS_<package>
DEB_DH_MAKESHLIBS_ARGS_ALL DEB_DH_PERL_ARGS DEB_DH_PREP DEB_DH_SCROLLKEEPER_ARGS DEB_DH_SHLIBDEPS_ARGS
DEB_DH_SHLIBDEPS_ARGS_<package> DEB_DH_SHLIBDEPS_ARGS_ALL DEB_FIXPERMS_EXCLUDE DEB_HADDOCK_DIR
DEB_HADDOCK_HTML_DIR DEB_HBUILD_INVOKE DEB_HOST_ARCH_CPU DEB_HOST_ARCH_OS DEB_INDEP_PACKAGES
DEB_INSTALL_CHANGELOGS_ALL DEB_INSTALL_DIRS_<package> DEB_INSTALL_DIRS_ALL DEB_INSTALL_DOCS_<package>
DEB_INSTALL_DOCS_ALL DEB_ISNATIVE DEB_JARS DEB_KDE_APIDOX DEB_KDE_ENABLE_FINAL DEB_MAKEMAKER_INVOKE
DEB_MAKEMAKER_PACKAGE DEB_MAKEMAKER_USER_FLAGS DEB_MAKE_BUILD_TARGET DEB_MAKE_CHECK_TARGET
DEB_MAKE_CLEAN_TARGET DEB_MAKE_ENVVARS DEB_MAKE_INSTALL_TARGET DEB_MAKE_INVOKE DEB_MAKE_MAKEFILE
DEB_MAKE_TEST_TARGET DEB_NOEPOCH_VERSION DEB_NOREVISION_VERSION DEB_NO_IMPLICIT_HADDOCK_HYPERLINK DEB_PACKAGES
DEB_PATCHDIRS DEB_PATCHDIRS_READONLY DEB_PATCH_SUFFIX DEB_PERL_INCLUDE DEB_PERL_INCLUDE_package DEB_PHONY_RULES
DEB_PYTHON_BUILD_ARGS DEB_PYTHON_CLEAN_ARGS DEB_PYTHON_DESTDIR DEB_PYTHON_INSTALL_ARGS_ALL
DEB_PYTHON_MODULE_PACKAGE DEB_PYTHON_MODULE_PACKAGES DEB_PYTHON_PRIVATE_MODULES_DIRS DEB_PYTHON_SETUP_CMD
DEB_PYTHON_SYSTEM DEB_QMAKE_CONFIG_VAL DEB_QUILT_TOPDIR DEB_SETUP_BIN_NAME DEB_SHLIBDEPS_INCLUDE
DEB_SHLIBDEPS_INCLUDE_<package> DEB_SHLIBDEPS_LIBRARY_<package> DEB_SHLIBDEPS_LIBRARY_package DEB_SOURCE_PACKAGE
DEB_SRCDIR DEB_TARBALL DEB_TAR_SRCDIR DEB_UDEB_PACKAGES DEB_UPDATE_RCD_PARAMS DEB_VERBOSE_ALL DEB_VERSION
GHC6_VERSION JAVACMD JAVA_HOME MAKEFILE MAKEFLAGS NUMJOBS OPTIMIZE QMAKE

+153 items

binary-install/<package>:: clean:: debian/ant.properties install/<package>::

(Joey Hess at DebConf 9: *Not Your Grandpa's Debhelper*)

Debian packaging stack: problems

2. **We are not hiding lower-level tools**

visible surface area: dh

+12 items

dh override_dh_<command>:

dh_auto_clean dh_auto_build dh_auto_install dh_auto_test

--with --sourcedirectory --buildsystem --builddirectory --list

(Joey Hess at DebConf 9: *Not Your Grandpa's Debhelper*)

Debian packaging stack: problems

2. **We are not hiding lower-level tools**

visible surface area: dh

+12 items

dh override_dh_<command>:

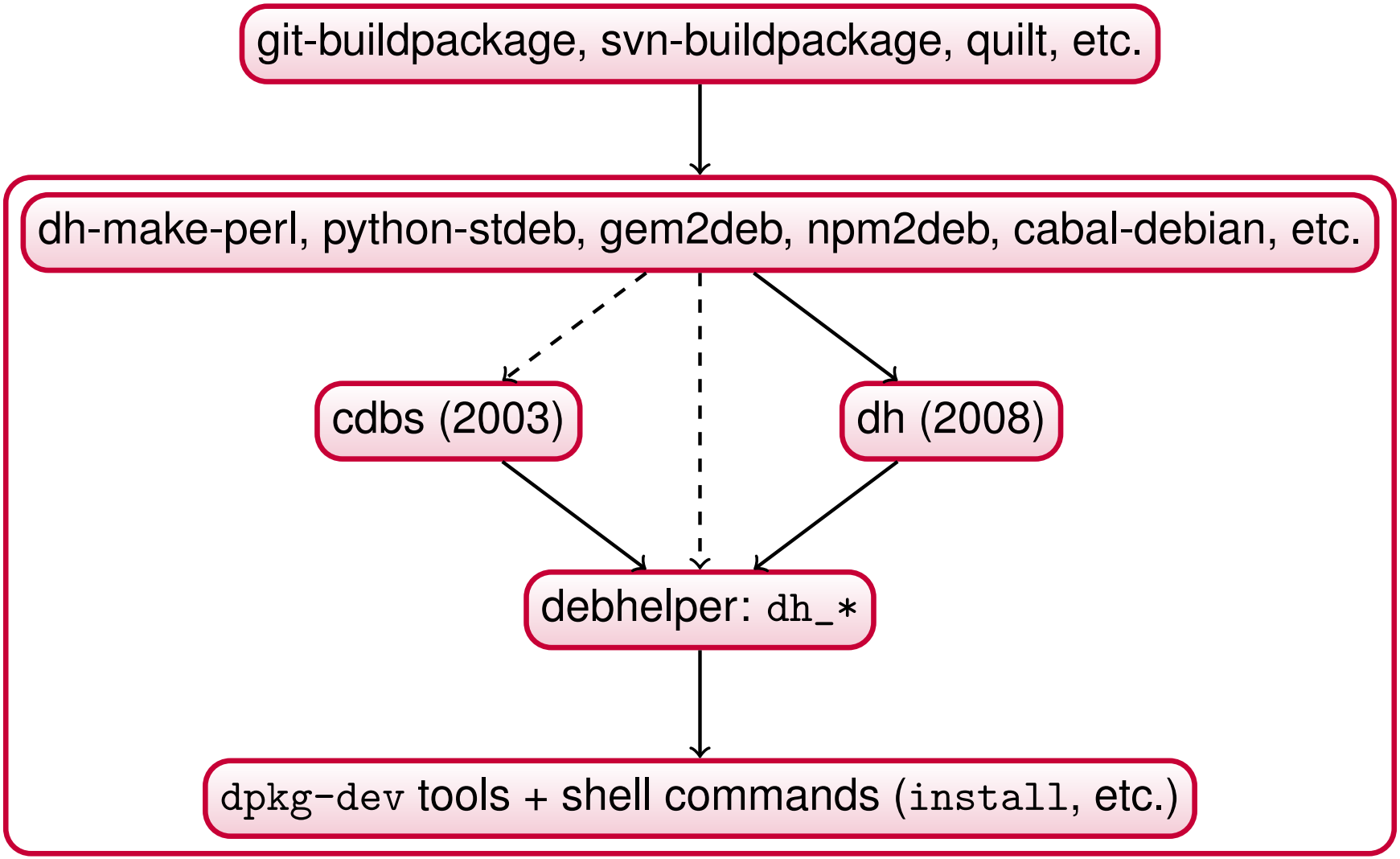
dh_auto_clean dh_auto_build dh_auto_install dh_auto_test

--with --sourcedirectory --buildsystem --builddirectory --list

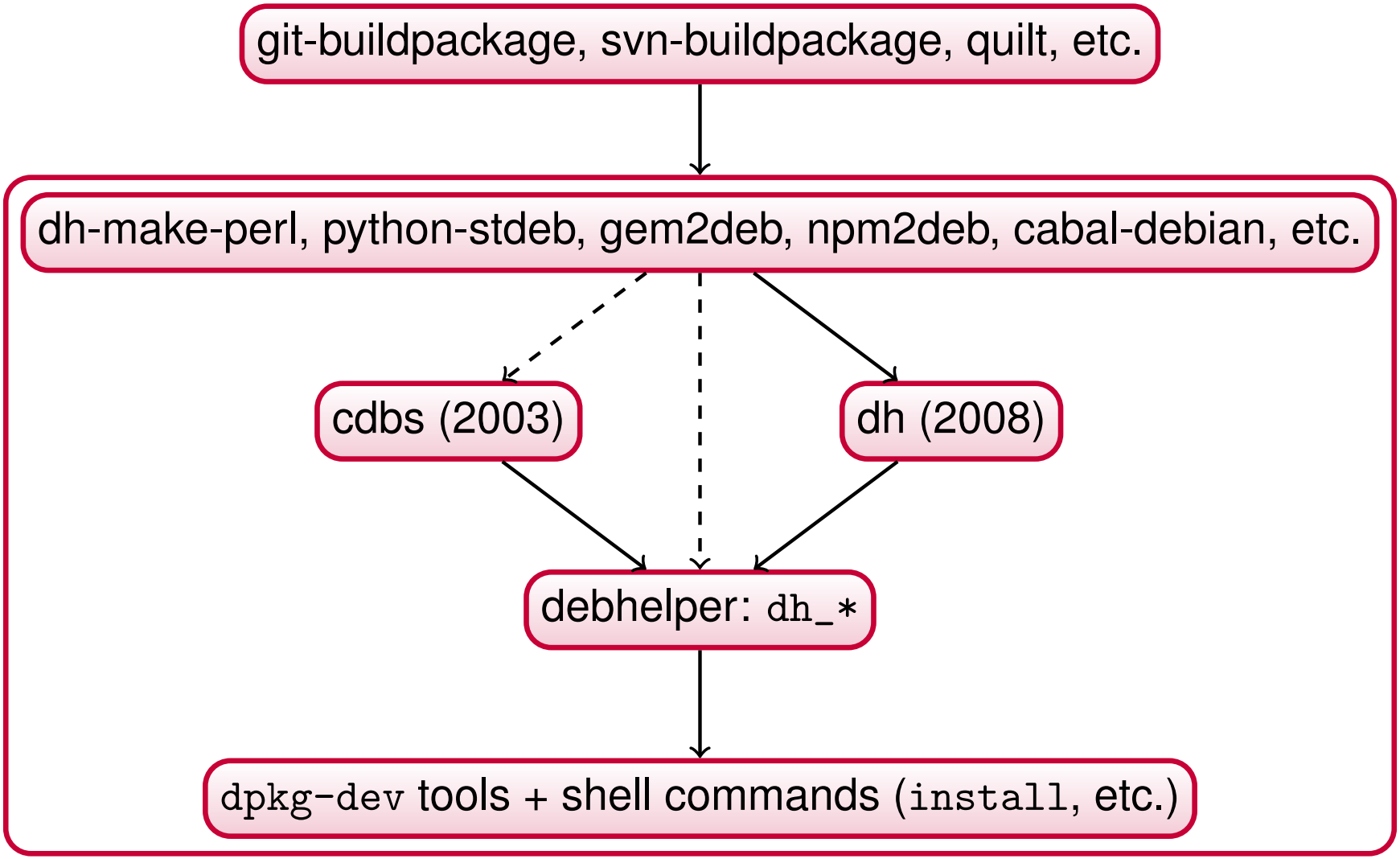
Still, 138 + 12 = 150 visible items!

(Joey Hess at DebConf 9: *Not Your Grandpa's Debhelper*)

Debian packaging stack

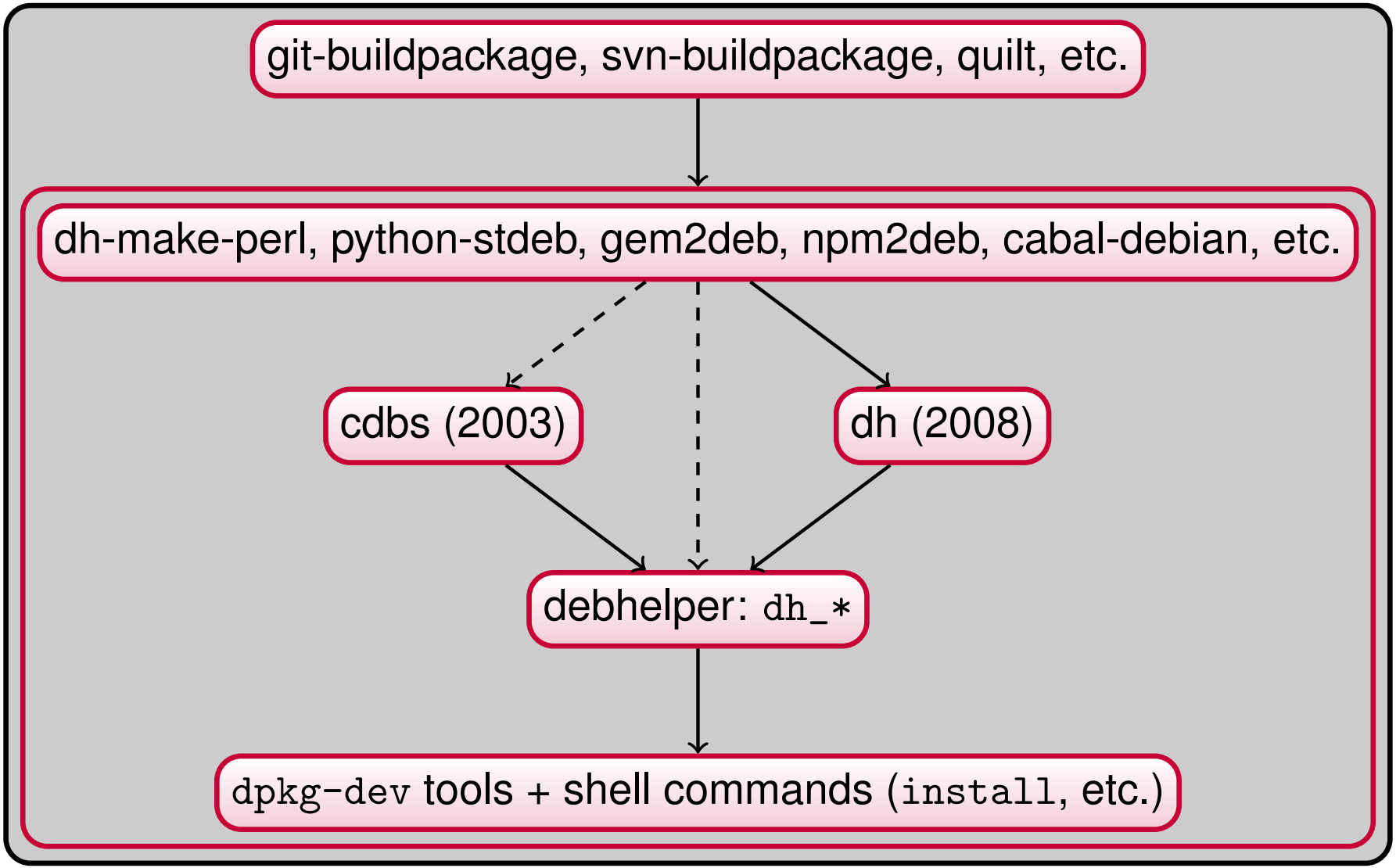


Debian packaging stack



What you need to master to do Debian packaging

Debian packaging stack



What you need to master to do Debian packaging

Debian packaging stack: problems

3. **We are not good at maintaining our packaging code**

- Influenced by:
  - Changes in the upstream code
  - Changes in the Debian Policy
  - Changes in the packaging team's policy & practices

Debian packaging stack: problems

3. **We are not good at maintaining our packaging code**

- Influenced by:
  - Changes in the upstream code
  - Changes in the Debian Policy
  - Changes in the packaging team's policy & practices
- Packaging $\approx$ manual merging from those three different branches

Debian packaging stack: problems

3. **We are not good at maintaining our packaging code**

- Influenced by:
  - Changes in the upstream code $\rightsquigarrow$ `uupdate`
  - Changes in the Debian Policy $\rightsquigarrow$?
  - Changes in the packaging team's policy & practices $\rightsquigarrow$?
- Packaging $\approx$ manual merging from those three different branches

Debian packaging stack: problems

3. **We are not good at maintaining our packaging code**

- Influenced by:
  - Changes in the upstream code $\rightsquigarrow$ `uupdate`
  - Changes in the Debian Policy $\rightsquigarrow$?
  - Changes in the packaging team's policy & practices $\rightsquigarrow$?
- Packaging $\approx$ manual merging from those three different branches
- A lot of duplication in our 3000 Perl modules, or 700 Python libs

Debian packaging stack: problems

3. **We are not good at maintaining our packaging code**

- Influenced by:
  - Changes in the upstream code $\rightsquigarrow$ `uupdate`
  - Changes in the Debian Policy $\rightsquigarrow$?
  - Changes in the packaging team's policy & practices $\rightsquigarrow$?
- Packaging $\approx$ manual merging from those three different branches
- A lot of duplication in our 3000 Perl modules, or 700 Python libs
- A lot of outdated packaging code

What should we do?

What should we do?

- Design a higher-level packaging framework

What should we do?

debdry

- When `debdry` is run, it:
  1. Moves the contents of your `debian/` directory aside;
  2. Chooses and runs an automatic debianisation tool;
  3. Applies your manual changes on top of the autogenerated `debian/`, to produce the final source package;
  4. Stores the original `debian/` directory in `debian/debdry` so that the process can be reversed.
- So your packaging code becomes
  (output of `dh_make_*`) + (diff of `debian/` for manual changes)
- Clearly a step in the right direction
- But the maintainer is still editing files in `debian/`
  - Low-level
  - Diffing+patching/merging will likely fail for some cases
    (e.g. SOVERSION change)
  - Yet another tool $\rightsquigarrow$ higher entry barrier

What should we do?

- Design a higher-level packaging framework that:
  - Relies on our known-working tools and formats
  - But hides them in 99% of cases (Think of C & assembly language)

What should we do?

- Design a higher-level packaging framework that:
  - Relies on our known-working tools and formats
  - But hides them in 99% of cases (Think of C & assembly language)
- A compiler or generator for the `debian/` directory?

What should we do?

- Design a higher-level packaging framework that:
  - Relies on our known-working tools and formats
  - But hides them in 99% of cases (Think of C & assembly language)
- A compiler or generator for the `debian/` directory?
- Expected outcomes:
  - Less time on boring pure-packaging stuff
  - More time on hard & interesting problems
  - Lower the entry barrier for newcomers
  - Automate the packaging work: all common tasks in the framework
  - Does not prevent NMUs: one can still work from the generated files

What should we do?

- Design a higher-level packaging framework that:
  - Relies on our known-working tools and formats
  - But hides them in 99% of cases (Think of C & assembly language)
- A compiler or generator for the `debian/` directory?
- Expected outcomes:
  - Less time on boring pure-packaging stuff
  - More time on hard & interesting problems
  - Lower the entry barrier for newcomers
  - Automate the packaging work: all common tasks in the framework
  - Does not prevent NMUs: one can still work from the generated files

Automated packaging

- Automated backports
  - To stable, for most of testing, unstable, experimental
- Automated packages
  - From CPAN, PyPI, RubyGems, Maven Central, npm

Automated packaging

- Automated backports
  - To stable, for most of testing, unstable, experimental
- Automated packages
  - From CPAN, PyPI, RubyGems, Maven Central, npm
- Different compromises: no security support, no manual testing
  - Acceptable for users who would install from sources anyway

Automated packaging

- Automated backports
  - To stable, for most of testing, unstable, experimental
- Automated packages
  - From CPAN, PyPI, RubyGems, Maven Central, npm
- Different compromises: no security support, no manual testing
  - Acceptable for users who would install from sources anyway

Make distribution packages the universal way to manage software again

Automated packaging

- Automated backports
  - To stable, for most of testing, unstable, experimental
- Automated packages
  - From CPAN, PyPI, RubyGems, Maven Central, npm
- Different compromises: no security support, no manual testing
  - Acceptable for users who would install from sources anyway

Make distribution packages the universal way to manage software again

Package the Free Software world (including every version of it)
= Really meet our users' needs

Automated packaging

- Automated backports
  - To stable, for most of testing, unstable, experimental
- Automated packages
  - From CPAN, PyPI, RubyGems, Maven Central, npm
- Different compromises: no security support, no manual testing
  - Acceptable for users who would install from sources anyway

Make distribution packages the universal way to manage software again

Package the Free Software world (including every version of it)
= Really meet our users' needs

Note: The legal side of this needs to be carefully thought. But CPAN & RubyGems are doing it.
Mailing list thread: `https://lists.debian.org/debian-project/2015/01/msg00046.html`

Other challenges

Applications and services $\neq$ packages

- What users want: (working) applications and services
- What we provide: (working) packages

Applications and services $\neq$ packages

- What users want: (working) applications and services
- What we provide: (working) packages
- Not a problem for simple applications & services

Applications and services $\neq$ packages

- What users want: (working) applications and services
- What we provide: (working) packages
- Not a problem for simple applications & services
- But complex services requiring interaction between packages?
  (Mail server, Cloud infrastructure, web application using a complex stack, etc.)

Applications and services $\neq$ packages

- What users want: (working) applications and services
- What we provide: (working) packages
- Not a problem for simple applications & services
- But complex services requiring interaction between packages?
  (Mail server, Cloud infrastructure, web application using a complex stack, etc.)
- Important to enable users to keep controlling their computing

Debian in the Dark Ages of Free Software
(Stefano Zacchiroli @ DebConf'14)

Applications and services ≠ packages (2)

- Technical issue: packages that configure other packages is hard
  (Policy §10.7.4)
- What can we learn from configuration management and containers?
  - Packages are *ingredients*, not really the *cooking recipe*
  - Should we package *cooking recipes*?
    Puppet/Chef recipes to automate the configuration of sets of packages?
  - Should we ship fully-prepared meals? containers?
    - What about *preferred form for making modifications*?
    - What about all-you-can-eat buffets?
  - Should we help users install complex applications and services by inventing something merging packages, containers, *tasks*, blends?

Providing the Debian experience everywhere

A lot of computing is moving to new architectures:

- Smartphones and tablets
  - Current status: run Debian in chroots (or bind-mounts)
- Cloud infrastructures
  - Current status: semi-official images for several public clouds

Providing the Debian experience everywhere

A lot of computing is moving to new architectures:

- Smartphones and tablets
  - Current status: run Debian in chroots (or bind-mounts)
- Cloud infrastructures
  - Current status: semi-official images for several public clouds
- Similar situation: users giving up freedom, control and trust for comfort

Providing the Debian experience everywhere

A lot of computing is moving to new architectures:

- Smartphones and tablets
  - Current status: run Debian in chroots (or bind-mounts)
- Cloud infrastructures
  - Current status: semi-official images for several public clouds
- Similar situation: users giving up freedom, control and trust for comfort
- Can we help users re-gain those without losing comfort?
  - What could we bring with a Debian-powered smartphone/tablet?
  - Improve the quality of semi-official images in Clouds, and enforce it?
    - Certification kit for Cloud providers?

Increasing trust in distributions

Increasing trust in distributions

- Our users are putting a lot of trust in us
  - Blindly running our binaries and maintainer scripts
  - Are we really that trustworthy?

Increasing trust in distributions

- Our users are putting a lot of trust in us
  - Blindly running our binaries and maintainer scripts
  - Are we really that trustworthy?
- Trustable package manager and archive ✓
- Trustable development process: quite
  Only 25% not using VCS ; only 5% modifying upstream without patch system

Increasing trust in distributions

- Our users are putting a lot of trust in us
  - Blindly running our binaries and maintainer scripts
  - Are we really that trustworthy?
- Trustable package manager and archive ✓
- Trustable development process: quite
  Only 25% not using VCS ; only 5% modifying upstream without patch system
- Trustable packages?
  - Debian's dirtiest secret:
    Binary package built by developers are used in the archive

Increasing trust in distributions

- Our users are putting a lot of trust in us
  - Blindly running our binaries and maintainer scripts
  - Are we really that trustworthy?
- Trustable package manager and archive ✓
- Trustable development process: quite
  Only 25% not using VCS ; only 5% modifying upstream without patch system
- Trustable packages?
  - Debian's dirtiest secret:
    Binary package built by developers are used in the archive
  - Source-only uploads are now possible
    - Not mandatory yet ; not for architecture:all packages yet

Increasing trust in distributions

- Our users are putting a lot of trust in us
  - Blindly running our binaries and maintainer scripts
  - Are we really that trustworthy?
- Trustable package manager and archive ✓
- Trustable development process: quite
  Only 25% not using VCS ; only 5% modifying upstream without patch system
- Trustable packages?
  - Debian's dirtiest secret:
    Binary package built by developers are used in the archive
  - Source-only uploads are now possible
    - Not mandatory yet ; not for architecture:all packages yet
  - Reproducible builds: auditability via bit-for-bit comparison
    - Using `.buildinfo` to record version of dependencies
    - And various tricks to deal with timestamps, randomness, etc.
    - Talk today at 4pm, K1.105, by Holger Levsen

Increasing trust in distributions

- Our users are putting a lot of trust in us
  - Blindly running our binaries and maintainer scripts
  - Are we really that trustworthy?
- Trustable package manager and archive ✓
- Trustable development process: quite
  Only 25% not using VCS ; only 5% modifying upstream without patch system
- Trustable packages?
  - Debian's dirtiest secret:
    Binary package built by developers are used in the archive
  - Source-only uploads are now possible
    - Not mandatory yet ; not for architecture:all packages yet
  - Reproducible builds: auditability via bit-for-bit comparison
    - Using `.buildinfo` to record version of dependencies
    - And various tricks to deal with timestamps, randomness, etc.
    - Talk today at 4pm, K1.105, by Holger Levsen
- Trustable runtime environment?

More bandwidth to upstreams and derivatives

More bandwidth to upstreams and derivatives

Current status:

- Structured contact points for derivatives (Debian Derivatives Front Desk)
- Services to monitor new upstream versions (using `debian/watch`)
- Manual forwarding of bugs
- Some services to track bugs in other bug trackers
  (Launchpad's bugs watches, Debian's bts-link)
- Some attempts at facilitating the exchange of patches
  - Ubuntu's changelog entries that summarize divergence
  - Debian's Patch Tagging Guidelines (DEP3): standard headers
  - ~~`http://patch-tracker.d.o` to expose all patches~~ down, dead?
- Some dashboards with pointers to other distributions

More bandwidth to upstreams and derivatives

Current status:

- Structured contact points for derivatives (Debian Derivatives Front Desk)
- Services to monitor new upstream versions (using `debian/watch`)
- Manual forwarding of bugs
- Some services to track bugs in other bug trackers
  (Launchpad's bugs watches, Debian's bts-link)
- Some attempts at facilitating the exchange of patches
  - Ubuntu's changelog entries that summarize divergence
  - Debian's Patch Tagging Guidelines (DEP3): standard headers
  - ~~`http://patch-tracker.d.o` to expose all patches~~ down, dead?
- Some dashboards with pointers to other distributions

Not working smoothly:

- No communication with some upstreams (and they complain about it)
- Many bugs and patches not forwarded

More bandwidth to upstreams and derivatives

Current status:

- Structured contact points for derivatives (Debian Derivatives Front Desk)
- Services to monitor new upstream versions (using `debian/watch`)
- Manual forwarding of bugs
- Some services to track bugs in other bug trackers
  (Launchpad's bugs watches, Debian's bts-link)
- Some attempts at facilitating the exchange of patches
  - Ubuntu's changelog entries that summarize divergence
  - Debian's Patch Tagging Guidelines (DEP3): standard headers
  - ~~`http://patch-tracker.d.o` to expose all patches~~ down, dead?
- Some dashboards with pointers to other distributions

Not working smoothly:

- No communication with some upstreams (and they complain about it)
- Many bugs and patches not forwarded

Next step: a real cross-distro+upstreams dashboard or hub?

Conclusions

Several challenges ahead:

1. Scale and automate our packaging practices and tools
2. Bring complex services and applications to users
3. Improve our support of new computing environments: phones, clouds
4. Increase trust in our distributions and packages
5. Improve collaboration with upstreams and derivatives

lucas@debian.org

Conclusions

Several challenges ahead:

1. Scale and automate our packaging practices and tools
2. Bring complex services and applications to users
3. Improve our support of new computing environments: phones, clouds
4. Increase trust in our distributions and packages
5. Improve collaboration with upstreams and derivatives

Boring? no!

lucas@debian.org

Conclusions

Several challenges ahead:

1. Scale and automate our packaging practices and tools
2. Bring complex services and applications to users
3. Improve our support of new computing environments: phones, clouds
4. Increase trust in our distributions and packages
5. Improve collaboration with upstreams and derivatives

Boring? no!

(but let's release jessie first!)

lucas@debian.org