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):
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
Distributions in the Free Software world
Distributions in the Free Software world

Alice

Bob

Carol

Dave

software

packages

GNOME
Distributions in the Free Software world

Alice

Bob

Carol

Dave

feedback/bugs

software

packages

feedback/bugs
Distributions in the Free Software world

Alice

Bob

Carol

Dave

software

feedback/bugs

feedback/bugs

packages

Lucas Nussbaum

Are distributions really boring and a solved problem?
Distributions in the Free Software world

Alice 

Bob 

Carol 

Dave 

feedback/bugs 

software 

feedback/bugs 

packages 

Ubuntu 

Carlos 

Dan 

Users 

Derivative 

Lucas Nussbaum  Are distributions really boring and a solved problem? 5 / 22
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

![Graph showing software package growth over time](source: modulecounts.com)
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

Lucas Nussbaum
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

Lucas Nussbaum  Are distributions really boring and a solved problem?
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

debhelper: dh_*

dpkg-dev tools + shell commands (install, etc.)
Debian packaging stack

cdbs (2003)

debhelper: dh_*

dpkg-dev tools + shell commands (install, etc.)
Debian packaging stack


debhelper: dh_*

dpkg-dev tools + shell commands (install, etc.)
Debian packaging stack

dh-make-perl, python-stdeb, gem2deb, npm2deb, cabal-debian, etc.


debhelper: dh_*

dpkg-dev tools + shell commands (install, etc.)
Debian packaging stack

- git-buildpackage, svn-buildpackage, quilt, etc.

- dh-make-perl, python-stddeb, gem2deb, npm2deb, cabal-debian, etc.

- cdb (2003)
- dh (2008)

- debhelper: dh_*

- dpkg-dev tools + shell commands (install, etc.)
Debian packaging stack: problems

- We are not really moving away from deprecated tools

![Graph showing package counts over time for different tools.](image)
Debian packaging stack: problems

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

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_installmacsen dh_installxamples dh_installupdown 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

138 items

(Joey Hess at DebConf 9: Not Your Grandpa's Debhelper)
Debian packaging stack: problems

We are not hiding lower-level tools

visible surface area: CDBS

(Joey Hess at DebConf 9: *Not Your Grandpa’s Debhelper*)
Debian packaging stack: problems

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

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 --builddirectory --list

Still, 138 + 12 = 150 visible items!

(Joey Hess at DebConf 9: Not Your Grandpa’s Debhelper)
Debian packaging stack

depth: dpkg-dev

tools + shell commands (install, etc.)

debhelper: dh_*


dh-make-perl, python-stddeb, gem2deb, npm2deb, cabal-debian, etc.

git-buildpackage, svn-buildpackage, quilt, etc.
Debian packaging stack

git-buildpackage, svn-buildpackage, quilt, etc.

dh-make-perl, python-stddeb, gem2deb, npm2deb, cabal-debian, etc.


debhelper: dh_*

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

What you need to master to do Debian packaging
Debian packaging stack

What you need to master to do Debian packaging

- dpkg-dev tools + shell commands (install, etc.)
- debhelper: dh_*
- dh-make-perl, python-stddeb, gem2deb, npm2deb, cabal-debian, etc.

Lucas Nussbaum: Are distributions really boring and a solved problem?
Debian packaging stack: problems

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 ≈ manual merging from those three different branches
Debian packaging stack: problems

We are not good at maintaining our packaging code

- Influenced by:
  - Changes in the upstream code $\leadsto$ update
  - Changes in the Debian Policy $\leadsto$ ?
  - Changes in the packaging team’s policy & practices $\leadsto$ ?

- Packaging $\approx$ manual merging from those three different branches
Debian packaging stack: problems

We are not good at maintaining our packaging code

- Influenced by:
  - Changes in the upstream code → update
  - Changes in the Debian Policy → ?
  - Changes in the packaging team’s policy & practices → ?

- Packaging ≈ manual merging from those three different branches

- A lot of duplication in our 3000 Perl modules, or 700 Python libs
Debian packaging stack: problems

We are not good at maintaining our packaging code

- Influenced by:
  - Changes in the upstream code \(\sim u\)update
  - Changes in the Debian Policy \(\sim ?\)
  - Changes in the packaging team’s policy & practices \(\sim ?\)
- 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?

- 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

- 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?

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 ~ 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 \texttt{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
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

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
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 ≠ packages

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

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

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?
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?

Lucas Nussbaum

Are distributions really boring and a solved problem?
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

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?

Lucas Nussbaum

Are distributions really boring and a solved problem?
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?
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