Grassroots Free Software
In-Depth Case Study: Debian

Stefano Zacchiroli

Debian Project Leader
Université Paris Diderot
IRILL

21 June 2012
Insubria International Open Source Summer School
Como, Italy

Outline

1. FOSS concepts
2. Debian overview
3. Philosophy
4. Organization
5. Processes
6. Derivatives
7. Appendix: packaging tutorial
8. Appendix: contribute

About the speaker

- Research
  - associate professor (*maître de conférences*) at Paris Diderot
  - research fellow at IRILL (*Initiative de Recherche et Innovation sur le Logiciel Libre*) — `http://www.irill.org`
- Research interests
  - component based software engineering (CBSE)
  - formal methods for component upgrades and QA
  - case in point: FOSS distribution packages
- Debian
  - Debian Developer since March 2001
  - packages: OCaml, Vim, Python, math-related sw
  - QA & infrastructure (PTS)
  - Debian Project Leader since April 2010, 3rd term

Outline

1. FOSS concepts
2. Debian overview
3. Philosophy
4. Organization
5. Processes
6. Derivatives
7. Appendix: packaging tutorial
8. Appendix: contribute

Free Software

1. an idea: software users should be in control of their software
   - AKA: software users should enjoy a set of fundamental freedoms while “using” it
2. a social and political movement to promote software freedoms world-wide
   - rooted in the hacker culture of the 70s and of the early UNIX-es
   - started in 1983 by Richard Stallman, launching the GNU Project
   - promoted since 1985 by the Free Software Foundation

Free Software — why it matters

Lester picked up a screwdriver. "You see this? It's a tool. You can pick it up and you can unscrew stuff or screw stuff in. You can use the handle for a hammer. You can use the blade to open paint cans. You can throw it away, loan it out, or paint it purple and frame it." He thumped the printer. "This [Disney in a Box] thing is a tool, too, but it's not your tool. It belongs to someone else — Disney. It isn't interested in listening to you or obeying you. It doesn't want to give you more control over your life." [...]

"If you don't control your life, you're miserable. Think of the people who don't get to run their own lives: prisoners, reform-school kids, mental patients. There's something inherently awful about living like that. Autonomy makes us happy."

— Cory Doctorow, Makers
http://craphound.com/makers/

Free Software, defined

Definition (Free Software)

A program is free software if the program's users have the four essential freedoms:

0. The freedom to run the program, for any purpose (freedom 0).
1. The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1).
2. The freedom to redistribute copies so you can help your neighbor (freedom 2).
3. The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes.

Access to the source is a precondition for freedoms 1 and 3.

Source: `http://www.gnu.org/philosophy/free-sw.html`

What About “Open Source”?

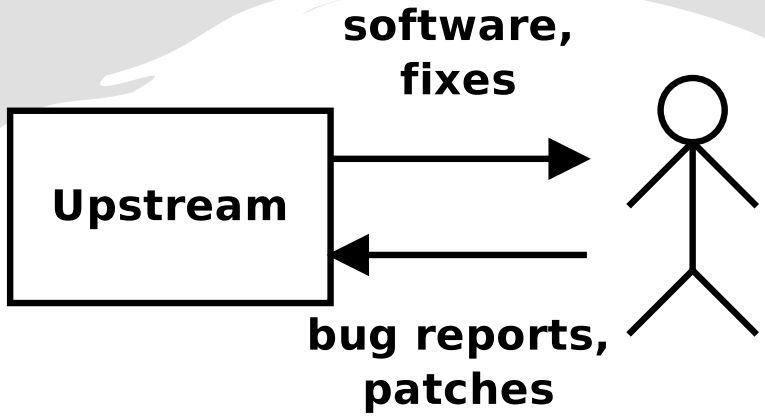
A different point of view on similar objectives

- Free Software best practices as a development methodology...
- ...that leads to better software
- major influence from “The Cathedral and the Bazaar”, by Eric S. Raymond
- that later influenced Netscape “open source”-ing
- synthesized by the Open Source Initiative (OSI) in the Open Source Definition (1981)
  - derived from the Debian Free Software Guidelines (more on this later...)

Origin of a heated debate since then

`http://www.gnu.org/philosophy/open-source-misses-the-point.html`

Distributing Free Software — the early days



Actors:

1. upstream software developer
2. final users

Notable flows: software, fixes, bug report, patches

Highlights:

- source distribution
  - compilation is done on user machines...
  - ...by every user

Distributing Free Software — the early days (cont.)

Practically, users need to:

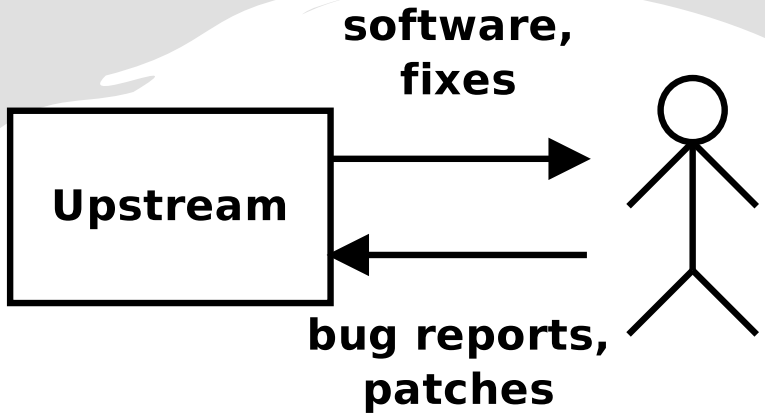
1. download
   - bonus point: verify checksums and GPG-sig
2. untar
3. ./configure
4. make
5. make install

(or language-specific variants)

Example

Let's have a look at:
`http://www.gnu.org/prep/standards/html_node/Managing-Releases.html`

Distributing Free Software — the early days (cont.)



Pros:

- tight relationships between upstream and users
- encourage becoming involved with development

Cons:

- confuses user and developer roles → developer knowledge needed to run the software
- scalability
  - update frequency
  - trust

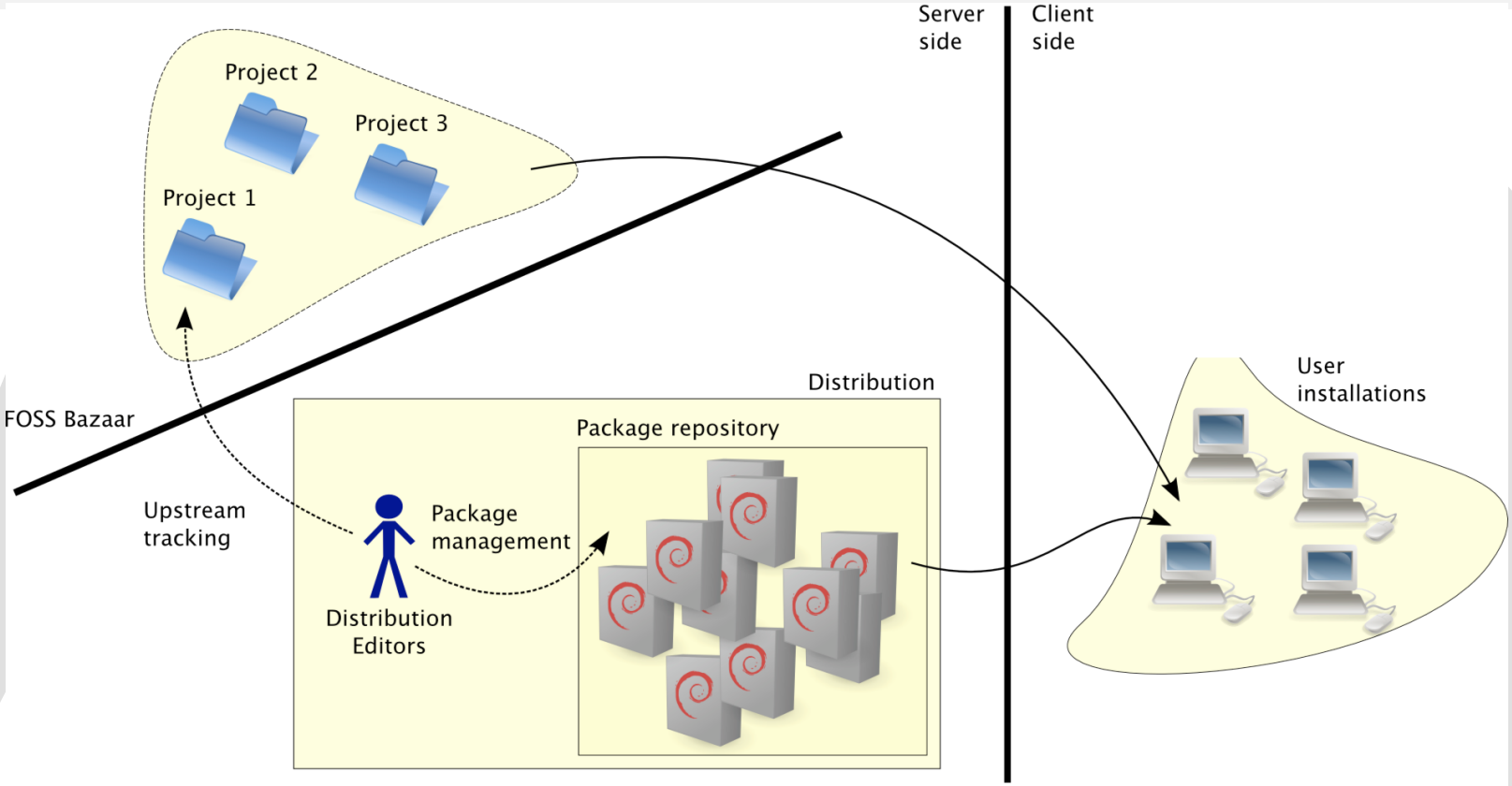
Example — installation issues in the early days

foo is cool, let's install it!

1. download foo-1.0.tar.gz
   - checksum mismatch, missing public key, etc.
2. ./configure
   - error: missing bar, baz, . . .
3. foreach (bar, baz, . . .) go to 1
   until (recursive) success
4. make
   - error: symbol not found
5. make install
   - error: cp: cannot create regular file /some/weird/path

now try scale that up to 17'000 sources releasing 3'000 new versions/month

The notion of “distribution”



- distributions are meant to ease software management
- key notion: the abstraction of package
- offer coherent collections of software
- killer application: package managers

Package manager

React to user requests to alter (upgrade / install / remove) software installation:

1. dependency solving
2. software download
3. software installation
   - as in: putting files in the right places
4. software configuration
   - as in: doing post-installation configuration

Package manager — example

Phase	Trace
User request	`# apt-get install aterm`

Constraint resolution	`Reading package lists... Done`
	`Building dependency tree... Done`
	`The following extra packages will be installed:`
	`  libafterimage0`
	`The following NEW packages will be installed`
	`  aterm libafterimage0`
	`0 upgraded, 2 newly installed, 0 to remove and 1786 not upgraded.`
	`Need to get 386kB of archives.`
	`After unpacking 807kB of additional disk space will be used.`
	`Do you want to continue [Y/n]? Y`

Package retrieval	`Get: 1 http://debian.ens-cachan.fr testing/main libafterimage0 2.2.8-2 [3`
	`Get: 2 http://debian.ens-cachan.fr testing/main aterm 1.0.1-4 [84.4kB]`
	`Fetched 386kB in 0s (410kB/s)`

Pre-configuration	`{`

Unpacking	`Selecting previously deselected package libafterimage0.`
	`(Reading database ... 294774 files and directories currently installed.)`
	`Unpacking libafterimage0 (from .../libafterimage0_2.2.8-2_i386.deb) ...`
	`Selecting previously deselected package aterm.`
	`Unpacking aterm (from .../aterm_1.0.1-4_i386.deb) ...`

Configuration	`Setting up libafterimage0 (2.2.8-2) ...` `Setting up aterm (1.0.1-4) ...`

Distribution concerns

Distributions act as intermediaries between upstream software authors and final users. Distributions are meant to ease Free Software life cycle management.

Within distributions scope:

- package management
- trusted sw delivery
- sw integration
- initial installation
- sw packaging
- upstream release tracking
- bug triage and forwarding
- (porting)

Outside distribution scope:

- upstream sw development (but beware of overlaps)
- "shielding" users from upstream and vice-versa

Source vs binary distribution

Source distribution

- packages contain source code
- which get compiled (automatically) on user machines
- distribution takes care of compilability

Binary distribution

- packages contain compiled (or “binary”) code
- which get installed on user machines
- distribution takes care of compilation
  - on all supported platforms

Trade-off: distribution work vs user (machine) work

Packages — a closer look

```
zack@usha:~% ls -al apache2-bin_2.4.2-2_amd64.deb
-rw-r--r-- 1 zack zack 1288852 mag 28 19:17 apache2-bin_2.4.2-2_amd64.deb
```

Contains:

1. the actual files that come with the Apache HTTP server
   - `/usr/sbin/apache2`
   - `/usr/share/doc/apache2/README`
   - ...
2. configuration programs that get executed before/after installation
   - start automatically Apache after installation
   - stop automatically Apache before removal

   ⇒ so that Apache is down during upgrade and up again immediately after
3. metadata, lots of...

Package metadata

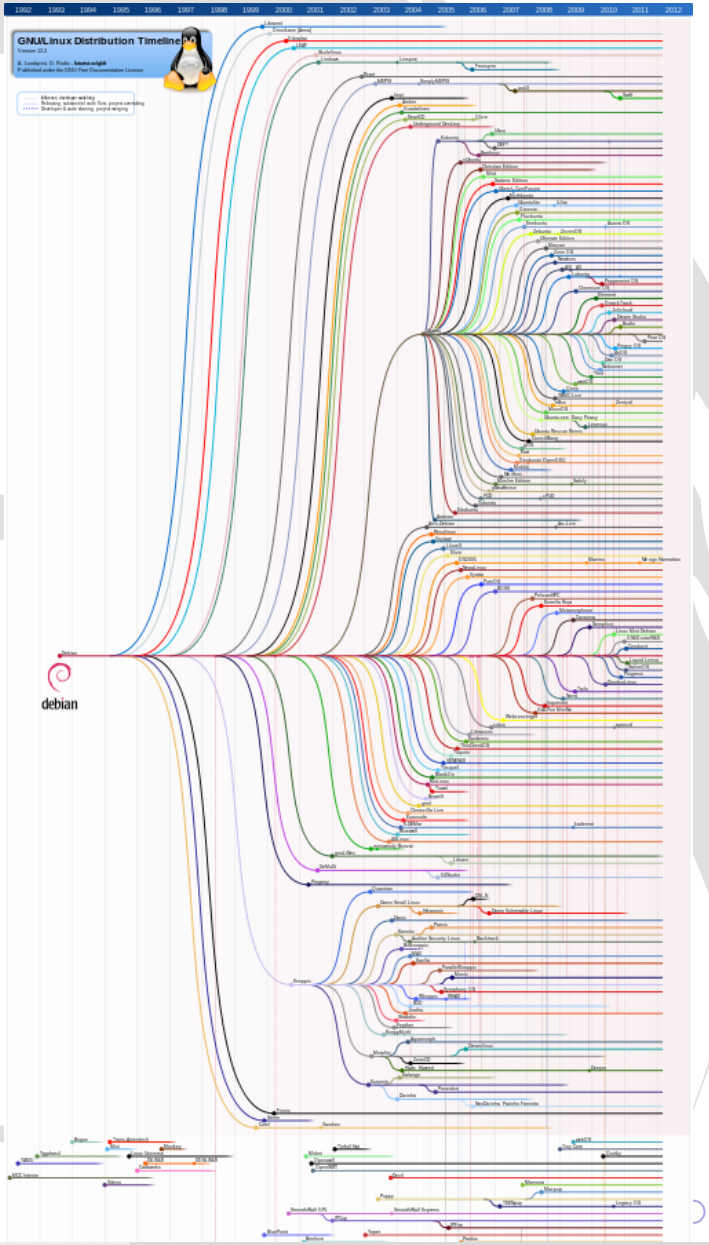
```
zack@usha:~% apt-cache show apache2-bin
Package: apache2-bin
Version: 2.4.2-2
Installed-Size: 3321
Maintainer: Debian Apache Maintainers <debian-apache@lists.debian.org>
Architecture: amd64
Replaces: apache2.2-bin (<< 2.3~), apache2.2-common
Provides: apache2-api-20120211, httpd, httpd-cgi
Depends: libapr1 (>= 1.4.2), libaprutil1-dbd-sqlite3 | libaprutil1-dbd-mysql
  libaprutil1-ldap, libc6 (>= 2.4), libldap-2.4-2 (>= 2.4.7), liblua5.1-0,
  libpcre3 (>= 8.10), libssl1.0.0 (>= 1.0.1), libxml2 (>= 2.7.4),
  zlib1g (>= 1:1.1.4), perl
Suggests: www-browser, apache2-doc, apache2-suexec-pristine | apache2-suexec
Conflicts: apache2.2-bin (<< 2.3~), apache2.2-common
Description-en: Apache HTTP Server (binary files and modules)
 The Apache Software Foundation's goal is to build a secure, efficient and
 extensible HTTP server as standards-compliant open source software. The
 result has long been the number one web server on the Internet. [...]
Homepage: http://httpd.apache.org/
Section: httpd
Priority: optional
Size: 1288852
MD5sum: 0f2988d78c7653ed9f967437f477059a
SHA1: 31f3015b2b94dd8f9fc2f573784fe98178ccbadc
```

Distributions

- easing software distribution: major concern in the early Free Software days
- distribution: a “somewhat” successful idea
- ≈ 300+ *active* distribution nowadays

  source: `http://distrowatch.com`

- a distribution timeline: `http://futurist.se/gldt/`



Distributions (cont.)

Some axes to compare distributions:

- package management
  - source vs binary distribution
  - low-level package format (most notably: deb vs rpm)
  - high-level package managers
- other technical features
  - included/default software
  - architectures
  - ...
- organization
  - manpower: company vs volunteer
  - decision making
  - community
- availability of commercial support / expertise

Review of major distributions:
`http://lwn.net/Distributions/#lead`

This class

In-depth case study of how a major volunteer Free Software project—and in particular a distribution: Debian—is organized.

Points of view:

- philosophical (volunteer motivation, social structure, etc.)
- management (decision making, project structure, etc.)
- technical

Outline

1. FOSS concepts
2. Debian overview
3. Philosophy
4. Organization
5. Processes
6. Derivatives
7. Appendix: packaging tutorial
8. Appendix: contribute

Debian: once upon a time

Fellow Linuxers,

This is just to announce the imminent completion of a ***brand-new Linux release****, which I'm calling the* ***Debian Linux Release****. [. . .]*

Ian A Murdock, 16/08/1993
comp.os.linux.development
http://deb.li/bigbang

- make GNU/Linux competitive with commercial OS
- easy to install
- built collaboratively by software experts
- 1st major distro developed "openly in the spirit of GNU" FSF-supported for a while

trivia: named after **DEB**ra Lynn and **IAN** Ashley Murdock

Since then — 15 releases

1993 development snapshots
1994 0.91
1995 0.93r5, 0.93r6, ~~1.0~~
1996 1.1 (Buzz), 1.2 (Rex)
1997 1.3 (Bo)
1998 2.0 (Hamm)
1999 2.1 (Slink)
2000 2.2 (Potato)
2002 3.0 (Woody)
2005 3.1 (Sarge)
Apr 2007 4.0 (Etch)
Feb 2009 5.0 (Lenny)
Feb 2011 6.0 (Squeeze)
Q4 2012 (?) 6.0 (Wheezy)

trivia:
why does Buzz have a (Debian) swirl on his chin?

Since then — 12 Debian Project Leaders (DPL)

1993–1996 Ian Murdock
1996–1997 Bruce Perens
1997–1998 Ian Jackson
1999–2001 Wichert Akkerman
2001–2002 Ben Collins
2002–2003 Bdale Garbee
2003–2005 Martin Michlmayr
2005–2006 Branden Robinson
2006–2007 Anthony Towns
2007–2008 Sam Hocevar
2008–2010 Steve McIntyre
2010–2013 *yours truly*

What is Debian?

3 aspects, interlinked:

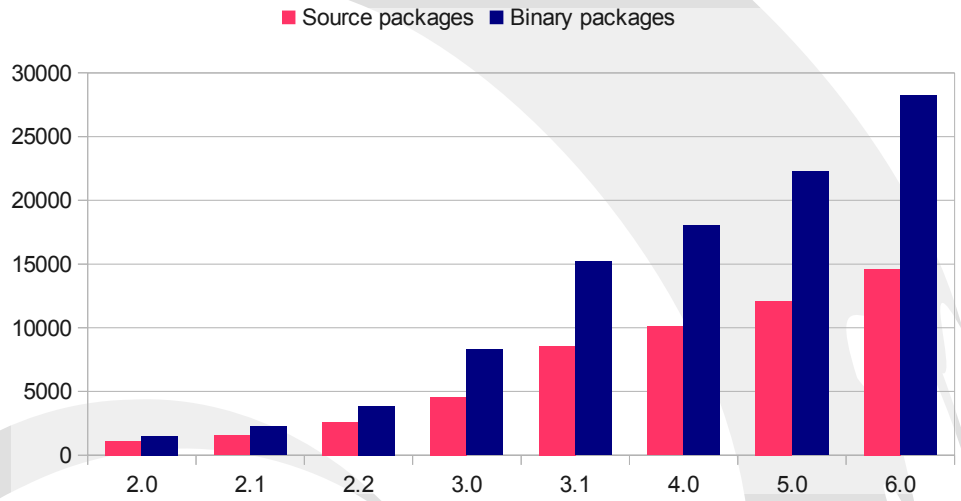
1. an operating system
2. a project
3. a community

Debian: the operating system

flagship product: Debian stable

- binary distribution
- completely Free (DFSG)
  - DFSG
  - ~~contrib~~, ~~non-free~~
- released every 24 months (≈)
- a dozen architectures

  amd64, armel, armhf, ia64, mips, mipsel, powerpc, s390, s390x, sparc
- archive-wide security support



one of the largest GNU/Linux porting platforms

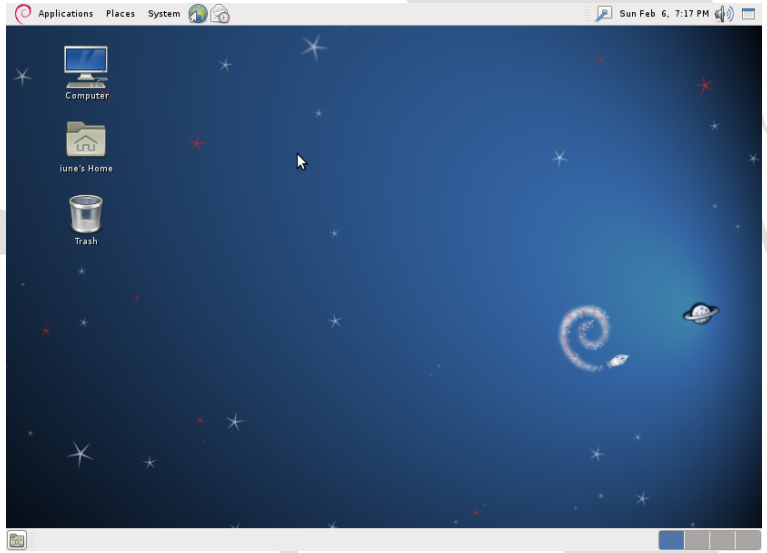
renowned for

ports, stability, packaging system, old hardware support, documentation, smooth upgrades, i18n/l10n, the `testing` suite, runs anywhere, technical policy, package choice, ...

Debian 6.0 “Squeeze” — highlights

What could happen in a release cycle?

- dependency-based boot system (faster, more robust)
- completely Free Linux kernel, firmware included
- GNU/kFreeBSD as technology preview
- improved debian-installer
  - ext4, btrfs
  - ZFS (kFreeBSD)
  - better support for complex setups e.g. LVM + RAID + encryption

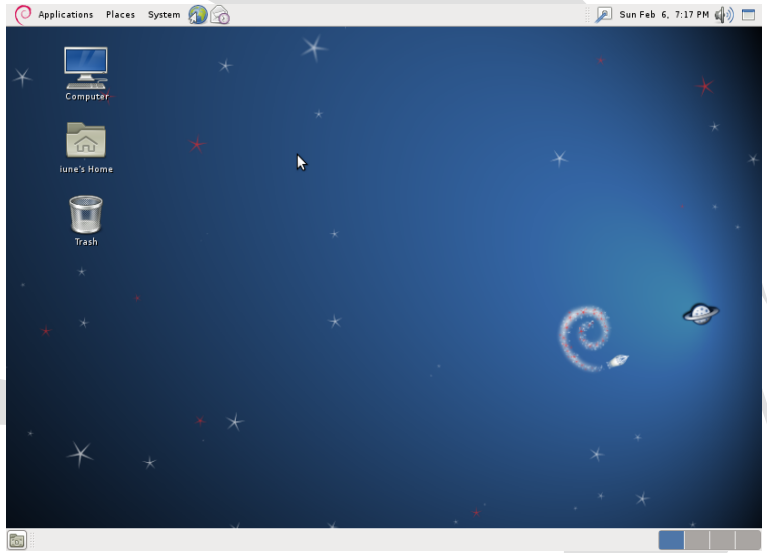


get Squeeze

http://deb.li/squeeze

Debian 6.0 “Squeeze” — highlights (cont.)

- Debian Pure Blends
  - DebianEdu, Debian Med, Debian Science, Debian Accessibility, DebiChem, Debian EzGo, Debian GIS, Debian Multimedia, ...
  - `blends.alioth.debian.org/`
- new services
  - `snapshot.debian.org`
  - `backports.debian.org`
  - `squeeze-updates` suite (ex-volatile)
  - `screenshots.debian.net`
  - `ask.debian.net`
- updates throughout the archive
- choice: GNOME, KDE Plasma, Xfce, LXDE, ...



get Squeeze

`http://deb.li/squeeze`

Debian: the Project

Common goal:

Create the best, Free operating system.

Debian Social Contract (1997)

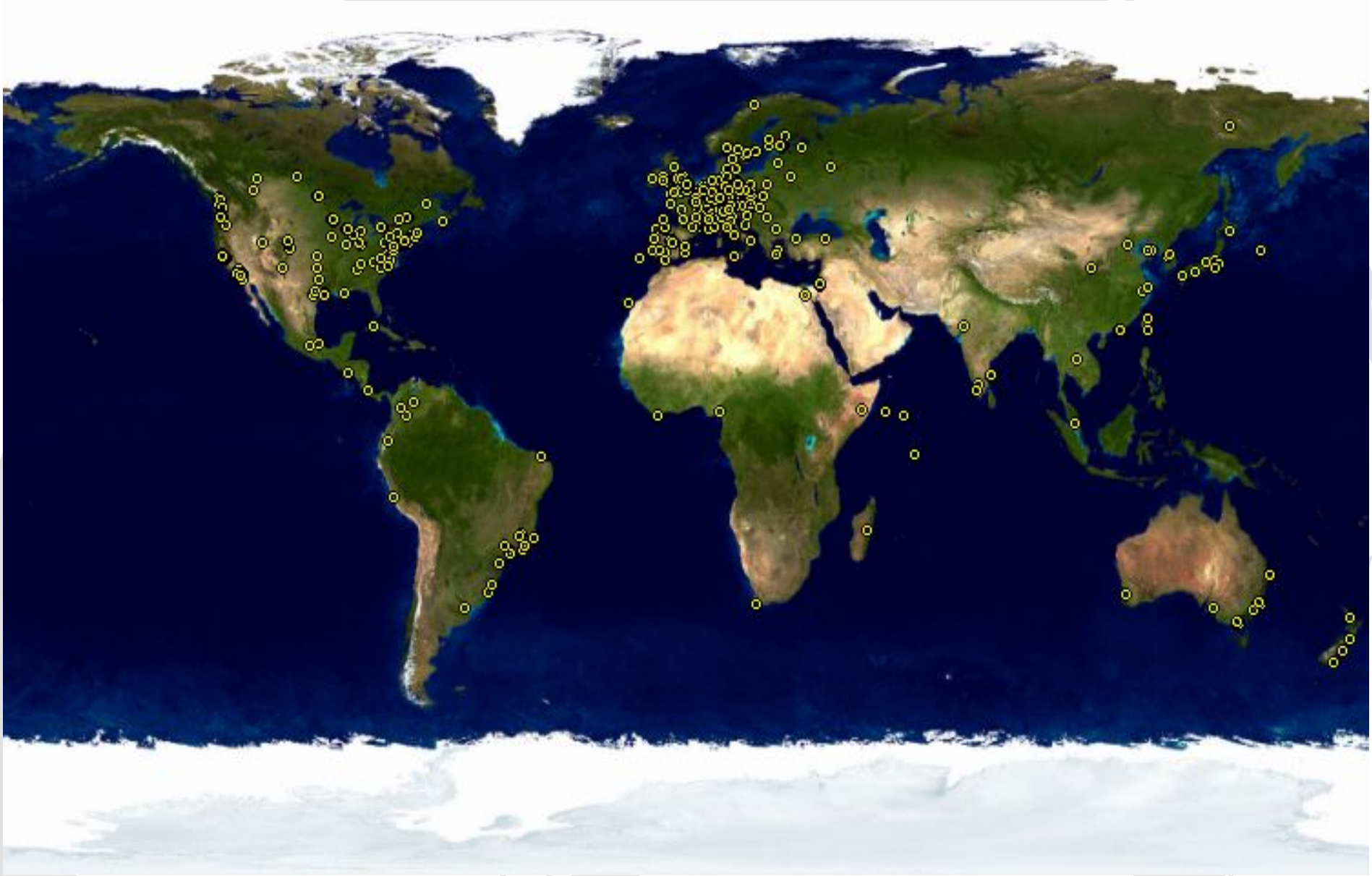
- 100% Free Software
- give back
- don't hide problems
- priorities: users & Free Software

Debian Constitution (1998)

Structures and rules of a Free-Software-compatible democracy

Strong motive to join: $\approx$ 1'000 volunteers, world-wide

Demography



Demography (cont.)

Developer's per country

2012 statistics:
http://www.perrier.eu.org/weblog/2012/06/06#devel-countries-201206

Take a guess: your country's position?

Demography (cont.)

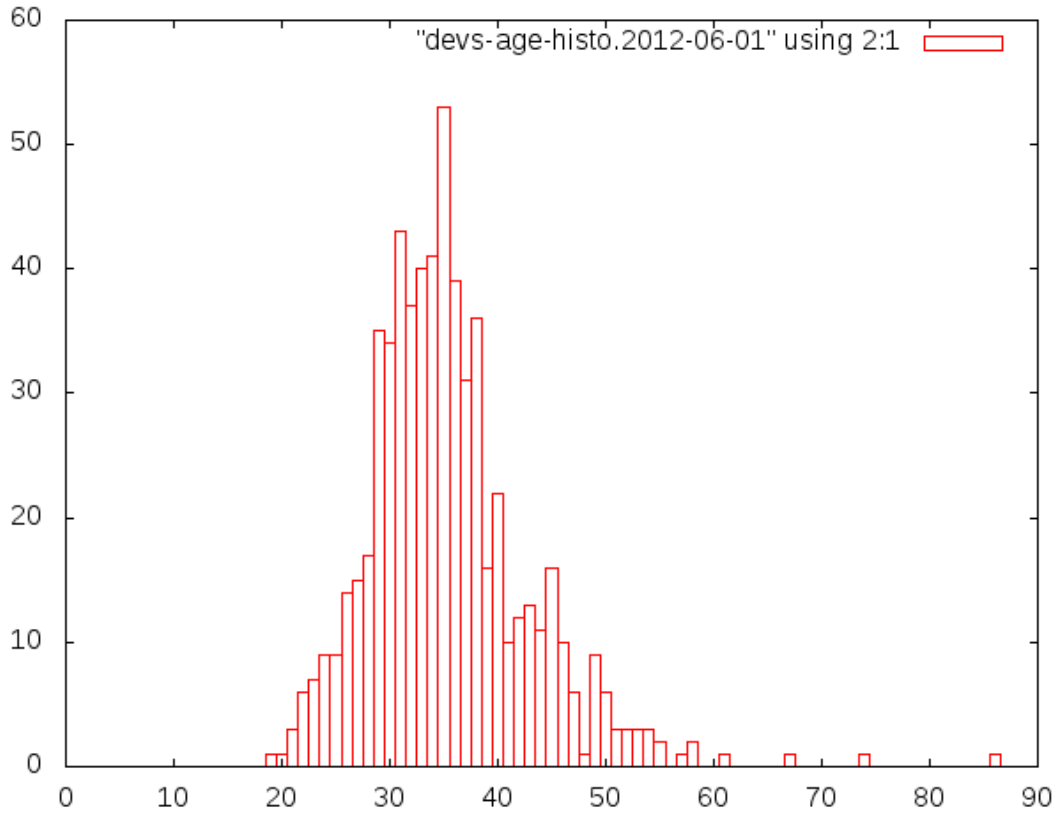


Figure: age histogram

Bottom line: very diverse, international and inter-generation project.

Debian: the community

Open development

- we don't hide problem
- easy to have an impact (just "show me the code!")

Large amounts of communication

- mailing lists
- IRC
- (a few) social media (growing)
  - social: `@debian`, `!debian` on identi.ca

Large number of tech-savvy users

- users help each other, contribute patches, get involved

Debian: one of a kind?

1993 — not many distros back then
19 years later — *lots* of other distros

openSUSE, Linux Mint, PCLinuxOS, Slackware, Gentoo Linux, CentOS, FreeBSD, Arch, Sabayon, Puppy, Lubuntu, MEPIS, Ultimate, NetBSD, Tiny Core, Zenwalk, CrunchBang, Dreamlinux, Vector, Kubuntu, Maemo, Red Hat, aptosid, Peppermint, PC-BSD, Chakra, Salix, ClearOS, KNOPPIX, Xubuntu, Super OS, BackTrack, gOS, TinyMe, Zentyal, EasyPeasy, Frugalware, Clonezilla, Pardus, Meego, OpenBSD, Quirky, PC/OS, Zorin, **Debian**, SystemRescue, Element, Unity, SliTaz, Macpup, wattOS, Scientific, Mythbuntu, Slax, DragonFLY, Elive, linux-gamers, 64 Studio, Ubuntu, mageia, Nexenta, Parisx, NuTyX, GhostBSD, Kongoni, moonOS, LFS, Lunar, Imagineos, Untangle, Fedora, Yellow Dog, aLinux, Yoper, IPFire, BlankOn, Mandriva, PureOS, FreeNAS, Moblin, Linpus, TurboLinux, blackPanther, ...

with many differences:

- technical choices
- release management
- release schedule
- target user
- community
- support
- packaging system
- user base
- look & feel
- ...

How is Debian different?

Debian's special #1: package quality

" Culture of technical excellence "

- package design: Policy
  i.e. "how a package should look like"
- package testing: `lintian`, `piuparts`,
  archive rebuilds (FTBFS), ...
- package maintainers are software experts
- no 2^{nd} class packages, all are equal

Debian release mantra

we release when it's ready

Debian's special #2: freedom

Firm principles: developers and users bound by the *Social Contract*

1. promoting the "culture of Free Software" since 1993
2. Free the bottom up
   - in its software
     firmware included !
   - in its infrastructure
     no non-free web services (for users)
     no non-free services (for developers)

Community awareness

- users know
- users trust Debian not to betray Free Software principles
- high bar for software freedom advocates

Debian's special #3: independence

Debian is an independent project

- no (single) company babysitting us
- living up on:
  1. donations (money & hardware)
  2. gift-economy

. . . truly remarkable in today "big" distro world

people trust Debian choices not to be "profit-driven"

Debian's special #4: decision making

1. do-ocracy

   An individual Developer may make any technical or nontechnical decision with regard to their own work;

   — Debian Constitution, §3.3.1.1

2. democracy

   Each decision in the Project is made by one or more of the following:

   1. The Developers, by way of General Resolution [...]

   — Debian Constitution, §2

that means:

- reputation follows work
- no benevolent dictator, no oligarchy
- no imposed decisions

  by who has money, infrastructure, people, ...

Outline

1. FOSS concepts
2. Debian overview
3. Philosophy
4. Organization
5. Processes
6. Derivatives
7. Appendix: packaging tutorial
8. Appendix: contribute

Social Contract

History

1996 early worries about companies involvement in distro development "*will RedHat always remain Free?*"

- FOSS licenses are not enough to guarantee that
- Debian solution: a "contract" on project commitments

1996 drafted by Bruce Perens (as DPL) + discussion

1997 ratified version 1.0

2004 ratified version 1.1

- one of Debian Foundation Documents
- tacit agreement between Debian and the FOSS community
- volunteer project → very important for developers' motivation

`http://www.debian.org/social_contract`

Social Contract — details

We declare that:

1. ***Debian will remain 100% free***
   We provide the guidelines that we use to determine if a work is "free" in the document entitled "The Debian Free Software Guidelines". We promise that the Debian system and all its components will be free according to these guidelines. We will support people who create or use both free and non-free works on Debian. We will never make the system require the use of a non-free component.

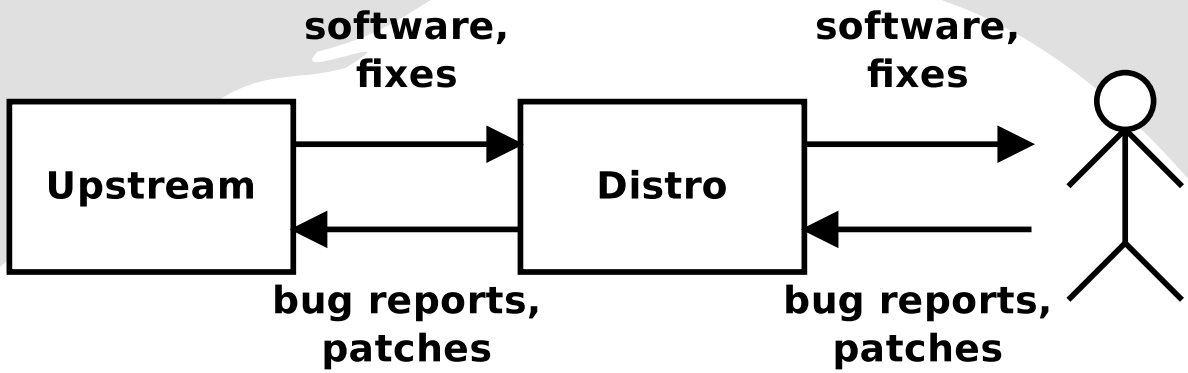
- Depends: DFSG
- allow users to live 100% Free digital life
- . . . but does not *force* them to

Social Contract — details (cont.)

2. ***We Will Give Back to the Free Software Community***
   When we write new components of the Debian system, we will license them in a manner consistent with the Debian Free Software Guidelines. We will make the best system we can, so that free works will be widely distributed and used. We will communicate things such as bug fixes, improvements and user requests to the upstream authors of works included in our system.

- dogfooding
- commitment to the Free Software ecosystem
- encourage ecosystem sustainability, rather than Debian's only
- principle: Free Software success is more important than Debian's

Implied distribution ecosystem



- shortcut the 1-to-many user-upstream relationship
- does not *hide* upstream existence (they're in the social contract!)
- encourages:
  - contributing distro-originated bugs and fixes upstream
  - software selection and integration at the distro level
- centralizes user trust on distro editors

Social Contract — details (cont.)

3. ***We Won't Hide Problems***
   We will keep our entire bug report database open for public view at all times. Reports that people file online will promptly become visible to others.

- historically relevant: first community distro
  - possibly the single greatest Debian contribution to Free Software
- induced a culture of project-wide transparency
  - folklore: "*social contract 3 is not only for bugs*"
  - folklore: "*there is no cabal*"

con: it is hard, really; makes harder interaction with actors that have different values

pro: fundamental for volunteer motivation

Social Contract — details (cont.)

4. ***Our priorities are our users and free software***
   We will be guided by the needs of our users and the free software community. We will place their interests first in our priorities. [...] We will not object to non-free works that are intended to be used on Debian systems, or attempt to charge a fee to people who create or use such works. We will allow others to create distributions containing both the Debian system and other works, without any fee from us. [...]

- attempted balance: user and free software interests
  - often at stake: non-free graphic/network drivers in Debian?
- first glimpses of pragmatism: the real world is what it is

Social Contract — details (cont.)

5. ***Works that do not meet our free software standards***
   We acknowledge that some of our users require the use of works that do not conform to the Debian Free Software Guidelines. We have created `contrib` and `non-free` areas in our archive for these works. The packages in these areas are not part of the Debian system, although they have been configured for use with Debian. We encourage CD manufacturers to read the licenses of the packages in these areas and determine if they can distribute the packages on their CDs. Thus, although non-free works are not a part of Debian, we support their use and provide infrastructure for non-free packages (such as our bug tracking system and mailing lists).

- Debian pragmatism at its peek
- enabled to attract a vast non ideological community

Debian Free Software Guidelines (DFSG)

the Social Contract relies on a “definition” of Free Software
the other Debian Foundation Document

- guidelines only — not hard rules
- used to help decide what is part of Debian
- apply to the “freedoms” attached to a given package
  - usually: copyright license
  - but also: trademark license, and your favorite `$monopoly`
- have *de facto* made Debian a renowned authority about software free-ness
  - together with major authorities: FSF, OSI

trivia: basis for Open Source Definition / Initiative

`http://www.debian.org/social_contract#guidelines`

DFSG — details

1. ***Free Redistribution***
   The license of a Debian component may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license may not require a royalty or other fee for such sale.

≈ Free Software freedom 2: redistribute copies

DFSG — details

2. **Source Code**
   The program must include source code, and must allow distribution in source code as well as compiled form.

- explicit the “open source” requirement of freedoms 1 (study) and 3 (distribute modifications)
- “open source” heritage of those days, then become part of its definition

3. **Derived Works**
   The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

≈ Free Software freedom 3 (distribute modifications)

DFSG — details

2. **Source Code**
   The program must include source code, and must allow distribution in source code as well as compiled form.

- explicit the “open source” requirement of freedoms 1 (study) and 3 (distribute modifications)
- “open source” heritage of those days, then become part of its definition

3. **Derived Works**
   The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

$\approx$ Free Software freedom 3 (distribute modifications)

DFSG — details (cont.)

4. ***Integrity of The Author's Source Code***
   The license may restrict source-code from being distributed in modified form only if the license allows the distribution of patch files with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

- trademarks are not (necessarily) incompatible with software freedoms

But:

(This is a compromise. The Debian group encourages all authors not to restrict any files, source or binary, from being modified.)

DFSG — details (cont.)

5. ***No Discrimination Against Persons or Groups***
   The license must not discriminate against any person or group of persons.
6. ***No Discrimination Against Fields of Endeavor***
   The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

Exercise (Free Software ethics)

Why a license that states the software cannot be used for warfare purposes is unacceptable from the Free Software point of view? What would be the loss for the Free Software ecosystem?

DFSG — details (cont.)

5. ***No Discrimination Against Persons or Groups***
   The license must not discriminate against any person or group of persons.
6. ***No Discrimination Against Fields of Endeavor***
   The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

Exercise (Free Software ethics)

Why a license that states the software cannot be used for warfare purposes is unacceptable from the Free Software point of view? What would be the loss for the Free Software ecosystem?

DFSG — details (cont.)

7. ***Distribution of License***
   The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

- i.e. license applies by default, no need to “execute” anything

DFSG — details (cont.)

8. ***License Must Not Be Specific to Debian***
   The rights attached to the program must not depend on the program's being part of a Debian system. If the program is extracted from Debian and used or distributed without Debian but otherwise within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the Debian system.

- no special casing: it is free for us only if it is free for everybody
- coherent with the "Free Software first" view

DFSG — details (cont.)

9. ***License Must Not Contaminate Other Software***
   The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be free software.

- license modularity
- ancillary clause to ensure Debian can be distributed on media

Debian Free *Software* (?) Guidelines

Spot the differences!

Debian Will Remain 100% Free Software *(SC 1.0)*

We promise to keep the Debian GNU/Linux Distribution entirely free software. As there are many definitions of free software, we include the guidelines we use to determine if software is "free" below. We will support our users who develop and run non-free software on Debian, but we will never make the system depend on an item of non-free software.

vs

Debian will remain 100% free *(SC 1.1)*

We provide the guidelines that we use to determine if a work is "free" in the document entitled "The Debian Free Software Guidelines". We promise that the Debian system and all its components will be free according to these guidelines. We will support people who create or use both free and non-free works on Debian. We will never make the system require the use of a non-free component.

Debian Free *Software* (?) Guidelines (cont.)

- SC “editorial” changes of 2004
  `http://www.debian.org/vote/2004/social_contract_reform.3`
- all content is equal and subject to DFSG
- no double standard for
  - software
  - documentation
  - firmware
  - data collections
  - ...
- radical position at the time
  - increasingly popular today in the free culture movement
  - objected by major actors (e.g. FSF for “political” texts)

Source of ethical doubts and technical issues within the community for Debian releases up to Squeeze. Today: no non-free firmware in main and no (new) non-free firmware in Linux upstream.

DFSG in practice

- DFSG are no laws, but guidelines
- no judges, no tribunals, just judgement (with responsibles in charge)

For complex cases, a series of thought experiments have been developed and are used as “benchmark” for *some* DFSG features

DFSG thought experiments

Example (The Desert Island test)

Imagine a castaway on a desert island with a solar-powered computer.

This would make it impossible to fulfill any requirement to make changes publicly available or to send patches to some particular place. This holds even if such requirements are only upon request, as the castaway might be able to receive messages but be unable to send them. To be free, software must be modifiable by this unfortunate castaway, who must also be able to legally share modifications with friends on the island.

DFSG thought experiments (cont.)

Example (The Desert Island test)

Imagine a castaway on a desert island with a solar-powered computer.
This would make it impossible to fulfill any requirement to make changes publicly available or to send patches to some particular place. This holds even if such requirements are only upon request, as the castaway might be able to receive messages but be unable to send them. To be free, software must be modifiable by this unfortunate castaway, who must also be able to legally share modifications with friends on the island.

DFSG thought experiments (cont.)

Example (The Dissident test)

Consider a dissident in a totalitarian state who wishes to share a modified bit of software with fellow dissidents, but does not wish to reveal the identity of the modifier, or directly reveal the modifications themselves, or even possession of the program, to the government.

Any requirement for sending source modifications to anyone other than the recipient of the modified binary—in fact any forced distribution at all, beyond giving source to those who receive a copy of the binary—would put the dissident in danger. For Debian to consider software free it must not require any such excess distribution.

DFSG thought experiments (cont.)

Example (The Dissident test)

Consider a dissident in a totalitarian state who wishes to share a modified bit of software with fellow dissidents, but does not wish to reveal the identity of the modifier, or directly reveal the modifications themselves, or even possession of the program, to the government.
Any requirement for sending source modifications to anyone other than the recipient of the modified binary—in fact any forced distribution at all, beyond giving source to those who receive a copy of the binary—would put the dissident in danger. For Debian to consider software free it must not require any such excess distribution.

DFSG thought experiments (cont.)

Example (The Tentacles of Evil test)

Imagine that the author is hired by a large evil corporation and, now in their thrall, attempts to do the worst to the users of the program: to make their lives miserable, to make them stop using the program, to expose them to legal liability, to make the program non-free, to discover their secrets, etc. The same can happen to a corporation bought out by a larger corporation bent on destroying free software in order to maintain its monopoly and extend its evil empire.

The license cannot allow even the author to take away the required freedoms!

DFSG thought experiments (cont.)

Example (The Tentacles of Evil test)

Imagine that the author is hired by a large evil corporation and, now in their thrall, attempts to do the worst to the users of the program: to make their lives miserable, to make them stop using the program, to expose them to legal liability, to make the program non-free, to discover their secrets, etc. The same can happen to a corporation bought out by a larger corporation bent on destroying free software in order to maintain its monopoly and extend its evil empire.
The license cannot allow even the author to take away the required freedoms!

DFSG vs common licenses

some DFSG-free licenses:

- strong copyleft licenses: GPL, AGPL, (CC BY-SA 3.0,) ...
- weak copyleft licenses: LGPL, MPL
- liberal license: BSD, MIT/X11, Apache, ...

some non-DFSG-free licenses:

- all the “bad” ones

DFSG — exercises

Exercise

Which of the following Creative Commons licenses is DFSG-free?

- *CC BY 3.0* `http://creativecommons.org/licenses/by/3.0/`
- *CC BY-ND 3.0* `http://creativecommons.org/licenses/by-nd/3.0/`
- *CC BY-NC-SA 3.0*
  `http://creativecommons.org/licenses/by-nc-sa/3.0/`

Exercise

Is the TeX license DFSG-free?

`http://en.wikipedia.org/wiki/TeX#License`

Exercise

Is the GNU Free Documentation License DFSG-free?

`http://www.gnu.org/copyleft/fdl.html`

DFSG — exercises

Exercise

Which of the following Creative Commons licenses is DFSG-free?

- *CC BY 3.0* `http://creativecommons.org/licenses/by/3.0/`
- *CC BY-ND 3.0* `http://creativecommons.org/licenses/by-nd/3.0/`
- *CC BY-NC-SA 3.0*
  `http://creativecommons.org/licenses/by-nc-sa/3.0/`

Exercise

Is the TeX license DFSG-free?

`http://en.wikipedia.org/wiki/TeX#License`

Exercise

Is the GNU Free Documentation License DFSG-free?

`http://www.gnu.org/copyleft/fdl.html`

DFSG — exercises

Exercise

Which of the following Creative Commons licenses is DFSG-free?

- *CC BY 3.0* `http://creativecommons.org/licenses/by/3.0/`
- *CC BY-ND 3.0* `http://creativecommons.org/licenses/by-nd/3.0/`
- *CC BY-NC-SA 3.0*
  `http://creativecommons.org/licenses/by-nc-sa/3.0/`

Exercise

Is the TeX license DFSG-free?

`http://en.wikipedia.org/wiki/TeX#License`

Exercise

Is the GNU Free Documentation License DFSG-free?

`http://www.gnu.org/copyleft/fdl.html`

Outline

1. FOSS concepts
2. Debian overview
3. Philosophy
4. Organization
5. Processes
6. Derivatives
7. Appendix: packaging tutorial
8. Appendix: contribute

Constitution

structure and rules for decision making in a Free Software-compatible democracy

- volunteers
- minimal “people management”
- “do-ocracy”
  - anybody can decide how to do their job
  - nobody can impose to others what to do
- relationships with the ~~real~~ “fiscal” world

`http://www.debian.org/devel/constitution`

Constitution — changelog

Need: project-wide decisions
Solution: project-wide voting (AKA general resolution)

- 1998 drafted by Ian Jackson (as DPL) + discussion
- 1998 v1.0: ratified
- 2003 v1.1: clarify voting method: Condorcet/clone proof SSD
- 2003 v1.2: clearly define foundation documents
- 2006 v1.3: generalize asset management
- 2007 v1.4: reduce the length of DPL *election*

Note: equipped with typical constitutional self-defense mechanisms.
All changes above needed to pass, and obtained, 3:1 majority

Constitution — bodies

- individual “developers” (or, better, project members)
- Debian Project Leader (DPL) elected each year
- technical committee (tech-ctte)
- secretary
- trusted organizations

Project members

- akin to Debian Project citizens
- everybody can work on Debian without being a project member...
- but project members do have specific rights:
  - voting (and being voted)
  - right to use project technical infrastructure
  - upload access to the official archive (for packagers)

Project leader

- represents Debian
- delegates “area of ongoing responsibility” to developers
  - AKA appoint delegates
- coordinate project activities, “lead discussions”
- decide upon project assets
  - money
  - hardware
  - “IP”, e.g. trademarks
- decision “garbage collector”
  - urgency
  - lack of other responsibles

Technical committee

- "tribunal" for *technical* disputes, 4–8 members
- the only formalized dispute resolution body in Debian
  - everything else (e.g. social issues) dealt with via mediation
  - often by the DPL
- members: skilled, (project-)elderly, well-respected developers
  - appointed by DPL
- formal voting process that mimics project-wide votes

Example (some recent tech-ctte issues)

- #614907 node: name conflicts with node.js interpreter
- #552688 Please decide how Debian should enable hardening build flags
- #665851 GNU parallel, name conflict with moreutils
- #573745 Please decide on Python interpreter packages maintainership

Constitution — decision making

golden rule

do-ocracy, no formal process

formally, decisions are taken by:[1]

1. developers as a whole
   - with general resolutions or elections
2. the DPL
3. the technical committee (CTTE)
4. individual developers working on some task ← **default**
5. DPL delegates
6. the project secretary

[1] overruling from top to bottom

General resolutions

decision making heavy weapon, not to be abused

- used for project-wide decisions and position statement
- folklore: "*thou shalt not use GRs for technical decisions*"

1. initial proposal
   - post to the `debian-vote` mailing list
   - requires seconds, depend on n. of developer
2. discussion period
   - might lead to alternative proposals
   - can put "on hold" decisions of any body
3. vote with Condorcet-based method
4. single winner
   - super majority (3:1) required to change Foundation Documents and Constitution

Voting method

Definition (Condorcet winner)

A candidate that would win majority against any *single* other candidate.

- If there is a Condorcet winner, it will win in any Condorcet method election
- Debian: Schulze method (most popular Condorcet method)

Sample ballot:

```
[4] bar
[2] baz
[1] foo
[2] quux
[3] None Of The Above
```

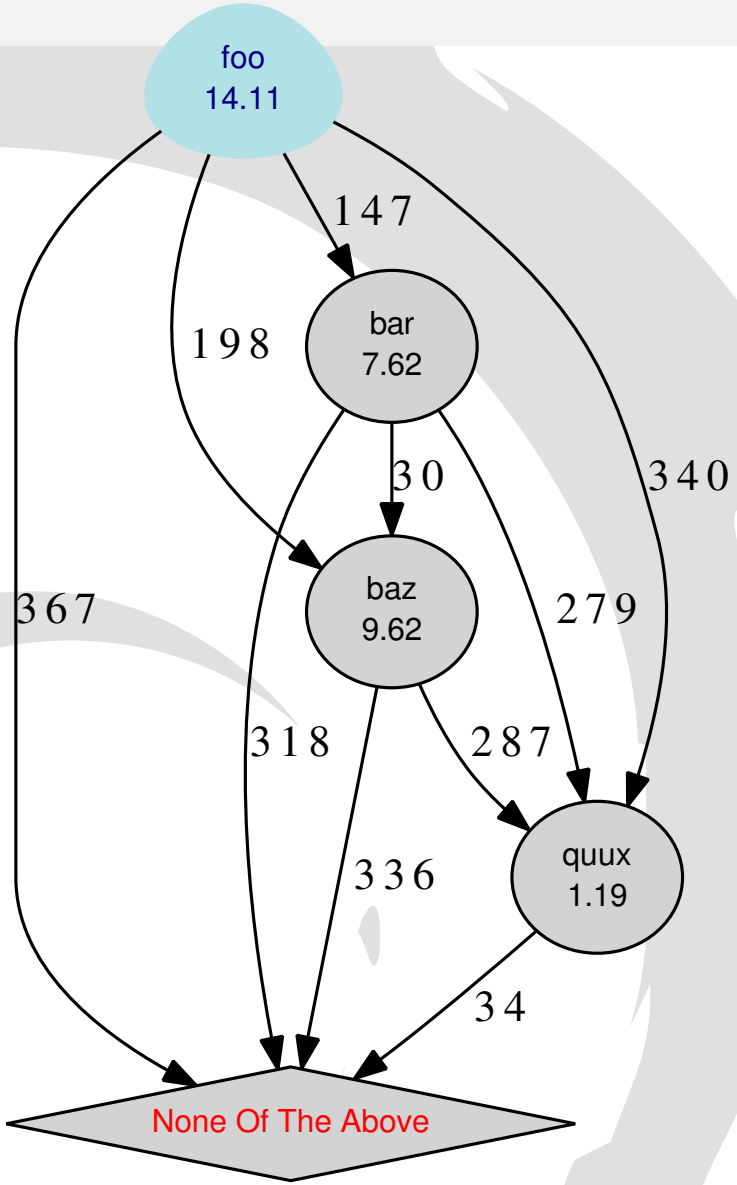


Figure: foo has won!

Secretary

- appointed conjointly by DPL and incumbent secretary
- responsible for election procedures
  - *de facto* authority for Constitution interpretation in electoral matters
- maintains and run the voting software devotee
  - voting artifacts (software, ballots, etc.) available for review
  - software allows to rerun and verify election results, e.g.:
    - `http://www.debian.org/vote/2010/vote_001_tally.txt`
    - `http://www.debian.org/vote/2010/vote_002_tally.txt`

Pop quiz: e-voting

- would you run your country's election this way?

Secretary

- appointed conjointly by DPL and incumbent secretary
- responsible for election procedures
  - *de facto* authority for Constitution interpretation in electoral matters
- maintains and run the voting software devotee
  - voting artifacts (software, ballots, etc.) available for review
  - software allows to rerun and verify election results, e.g.:
    - `http://www.debian.org/vote/2010/vote_001_tally.txt`
    - `http://www.debian.org/vote/2010/vote_002_tally.txt`

Pop quiz: e-voting

- would you run your country's election this way?

Fiscal sponsorship

Do Free Software projects exist in the "real world",
the one made of money, laws (and lawyers), taxes, etc?

They do have needs that relate them to it, e.g.:

- receive (tax exempt) donations
  - ...and provide (tax deductible) receipts
- own hardware, potentially expensive
  - Debian hardware cost per year: 29'000 USD
- own copyright and trademarks
  - that might want/need to enforce...
- use donated money to reimburse or pay developers
- developers might get sued
  - `$evil_proprietary_software_company`
  - patent trolls
  - ...

Fiscal sponsorship

Do Free Software projects exist in the "real world",
the one made of money, laws (and lawyers), taxes, etc?

They do have needs that relate them to it, e.g.:

- receive (tax exempt) donations
  - ...and provide (tax deductible) receipts
- own hardware, potentially expensive
  - Debian hardware cost per year: 29'000 USD
- own copyright and trademarks
  - that might want/need to enforce...
- use donated money to reimburse or pay developers
- developers might get sued
  - `$evil_proprietary_software_company`
  - patent trolls
  - ...

Fiscal sponsorship (cont.)

Definition (Fiscal sponsorship)

Fiscal sponsorship is the practice of non-profit organizations (NPO) to offer legal and tax-exempt status to groups related to the organization's missions.

By extension, in Free Software it commonly refers to providing all the "real world"-related needs that a project needs.

- high-profile FOSS projects have set up their own NPO
- but it is a lot of work!
  ...and hackers are not necessarily good at it
- umbrella organizations that do fiscal sponsorship for Free Software projects are more and more common, e.g.:
  - Software Freedom Conservancy, `http://sfconservancy.org/`
  - Software in the Public Interest (SPI), `http://spi-inc.org/`
  - (Apache Software Foundation, `http://apache.org/`)

Trusted Organization

1997 Debian founds SPI for the needs of Free Software projects
- including Debian itself, but with the usual “give back” intent

1998 the Constitution entrusts SPI to handle Debian assets

2006 Constitution amended to not special case SPI introducing the notion of. . .

Trusted Organizations (TO):
- hold assets “in trust” for the Project
  - DPL as liaison / decision maker
- link with the ~~real~~ bureaucratic world
  - donations, legal advice, tax exemption, reimbursements, . . .
- SPI (us), FFIS (de), debian.ch (ch), Assoli (it), ASL (br), . . .

Day to day organization: teams!

Luckily, day to day organization is much easier and more informal:

`http://wiki.debian.org/Teams/`

- teams grow as jobs get bigger
- some “core teams” are DPL delegates, most are not
- examples
  - packaging teams for related packages
  - ftp-master
  - release team
  - security team
  - kernel team
  - debian-installer
  - debian-cd
  - ...

Joining — an ethical moment

1993 as most FOSS projects, Debian incubated as 1-man-show

1994 Debian manifesto to explain Debian values

1995–1997 easy to join: send a mail!

- small numbers, project members in the tens

1998-1999 ethic crisis

- we need manpower!
- new developers accepted too quickly
- disagreement on core values

  to be more competitive with other distros, we should accept non-free components

- (lack of needed technical skills)

Debian Account Manager (DAM) stops accepting new members

Joining — an ethical moment (cont.)

1999 creation of the NM (New Maintainer) process and NM team to accept new members

DPL stated requirements to be on the NM team (excerpt):

```
- needs to have a *strong* opinion for free software
- needs to have a *strong* opinion for free software
- he needs to know what he's doing,
  that new people need some guidance,
  we have to prevent ourselves from trojans, etc.
- we need to trust him more than we trust *any* other
  active person
- he *has to* understand that new-maintainer is *more*
  than just creating dumb accounts on N machines
```

References

Gabriella Coleman, *Three Ethical Moments in Debian*,
http://papers.ssrn.com/sol3/papers.cfm?abstract_id=805287

NM Process

`http://www.debian.org/devel/join/newmaint`

1. identification
   - via GPG key, available in the Web of Trust (WoT)
   - signed by at least 2 project members
   - correspondence: Internet identity ↔ real person
   - Debian people: largest connected group in the WoT
2. assignment of an Application Manager (AM)
   - both mentoring and examination
   - requirement: not a newbie project member
3. philosophy & procedures
   - adherence to project core values
   - license/legal knowledge
   - knowledge of common procedures
   - Q&A via email

NM Process (cont.)

`http://www.debian.org/devel/join/newmaint`

4. tasks and skills
   - technical (packaging or other) ability
   - with evidence of previous work → trivial
5. DAM review & approval
   - DAMs are DPL delegates, (indirect) formal blessing of new members by the Project as a whole
   - special casing in the Constitution:

     > *Leader's Delegates [. . .] may make certain decisions which the Leader may not make directly, including approving or expelling Developers*

6. account creation
   - and setup of related permissions

Diversity

The Debian Project is an association of individuals who have made common cause to create a free operating system.

- but you have the New (Package) Maintainer process
- is that a problem?

Yes

- technical: there's much more than packaging to a Free OS
  - translation, infrastructure, porting, bug triaging, artwork, communication, management, testing, legal advice, QA, ...
- ethical: first/second class citizen split
  - no sense of belonging for non-packagers results in lack of motivation

Diversity

The Debian Project is an association of individuals who have made common cause to create a free operating system.

- but you have the New (Package) Maintainer process
- is that a problem?

Yes

- technical: there's much more than packaging to a Free OS
  - translation, infrastructure, porting, bug triaging, artwork, communication, management, testing, legal advice, QA, ...
- ethical: first/second class citizen split
  - no sense of belonging for non-packagers results in lack of motivation

Diversity (cont.)

2010 GR “Debian project members”
to become a Project member, all contributions count
http://www.debian.org/vote/2010/vote_002

2011 rename: New Maintainer (NM) Process → New Member

2012 GR “Diversity statement”

The Debian Project welcomes and encourages participation by everyone.
No matter how you identify yourself or how others perceive you: we welcome you. We welcome contributions from everyone as long as they interact constructively with our community.
While much of the work for our project is technical in nature, we value and encourage contributions from those with expertise in other areas, and welcome them into our community.

Diversity (cont.)

2010 GR “Debian project members”

to become a Project member, all contributions count

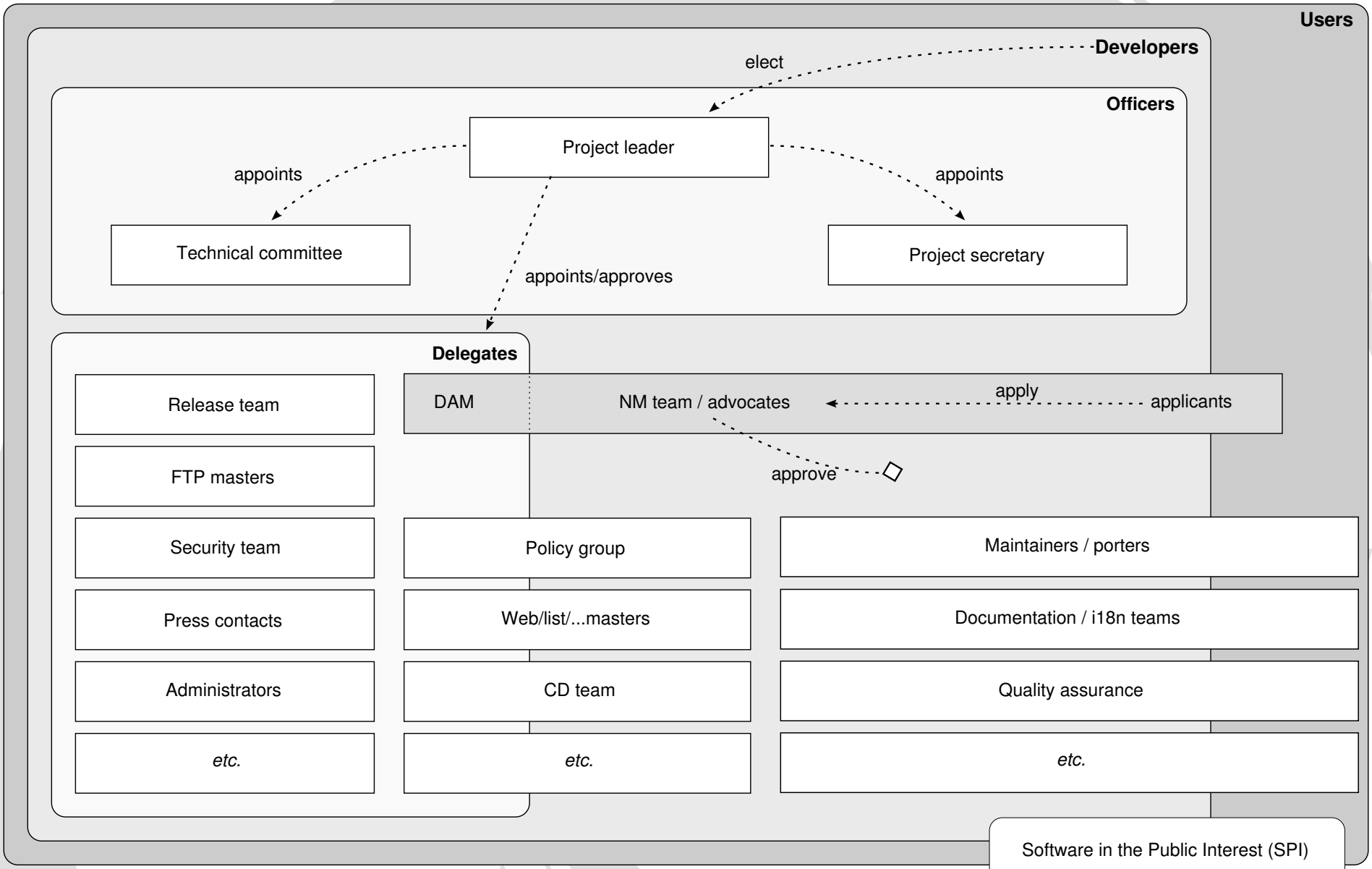
`http://www.debian.org/vote/2010/vote_002`

2011 rename: New Maintainer (NM) Process → New Member

2012 GR “Diversity statement”

> *The Debian Project welcomes and encourages participation by everyone.*
> *No matter how you identify yourself or how others perceive you: we welcome you. We welcome contributions from everyone as long as they interact constructively with our community.*
> *While much of the work for our project is technical in nature, we value and encourage contributions from those with expertise in other areas, and welcome them into our community.*

Organization — putting it all together



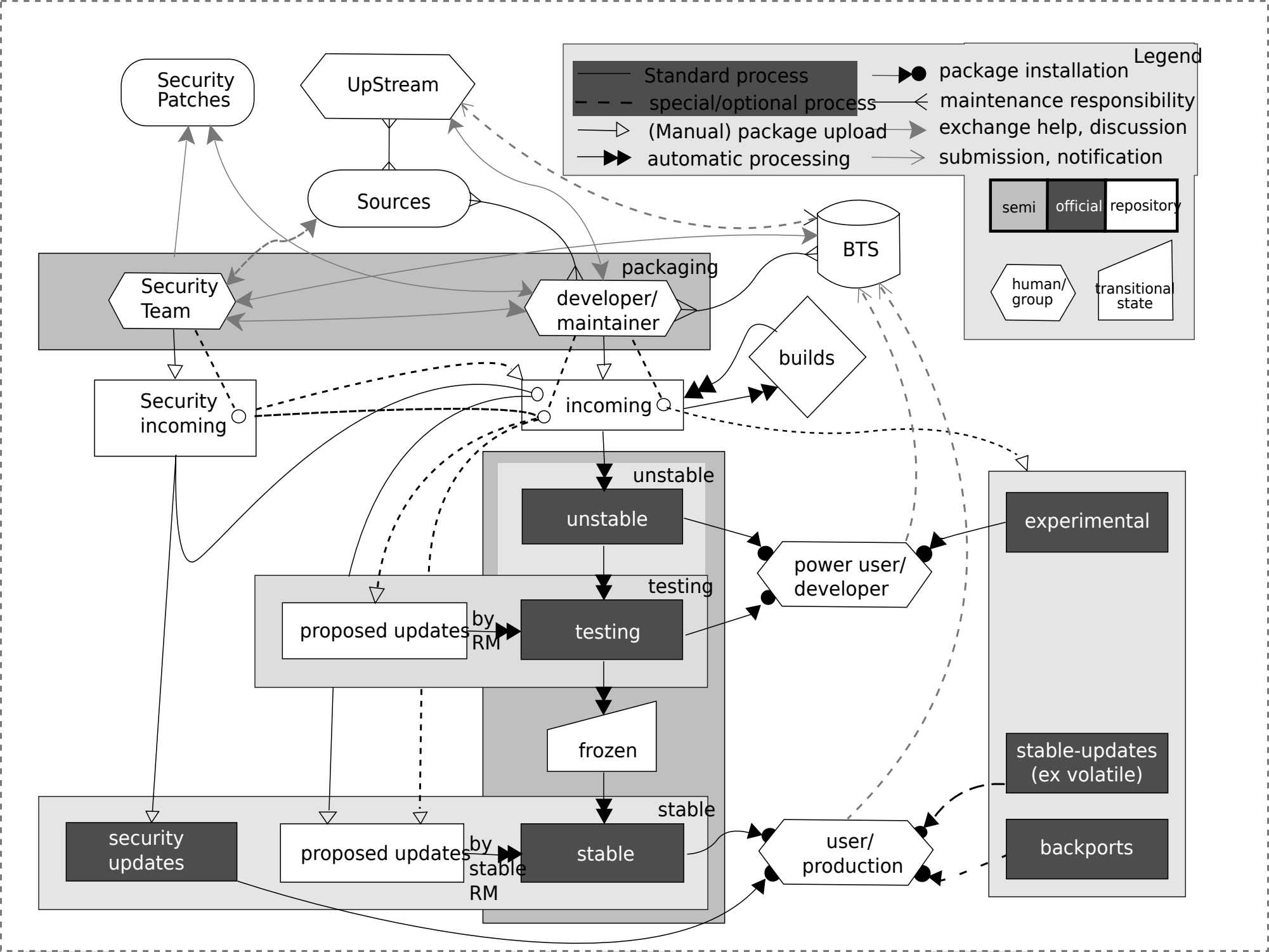
Outline

1. FOSS concepts
2. Debian overview
3. Philosophy
4. Organization
5. Processes
6. Derivatives
7. Appendix: packaging tutorial
8. Appendix: contribute

Package lifecycle

The most important processes in Debian derive from the package lifecycle, which follows packages from upstream software creation, through packaging, to final users of one or more of Debian suites (or *package repositories*).

Related human processes are documented extensively in the Debian Developer's Reference:
`http://www.debian.org/doc/manuals/developers-reference/`


Legend
Standard process
special/optional process
(Manual) package upload
automatic processing
package installation
maintenance responsibility
exchange help, discussion
submission, notification
semi
official
repository
human/ group
transitional state
Security Patches
UpStream
Sources
BTS
packaging
Security Team
developer/ maintainer
builds
Security incoming
incoming
unstable
unstable
experimental
power user/ developer
testing
proposed updates
by RM
testing
frozen
stable-updates (ex volatile)
backports
stable
security updates
proposed updates
by stable RM
stable
user/ production

Quality Assurance (QA)

How do you do quality in a large volunteer project?

Pitfalls:

- no “I’ll fire you” or monetary levers
- people might (and will) disappear without notice
- strong opinions
- “thou shalt not touch *my* package”
- a pinch of anarchy

The Debian Policy Manual

AKA “the Policy”

- specification-like manual describing expectations on Debian artifacts (in particular: packages)
  - structure of the archive, source, and binary packages
  - semantics of package metadata
  - expectations on package installation and *maintainer scripts*
  - file system logical structure (FHS)
  - OS design issues: service invocations, shared libraries, . . .
- maintained by the policy auditors, who are DPL delegates
- failure to respect the Policy → RC bugs

`http://www.debian.org/doc/debian-policy/`

Testing packages

How do you *enforce* policy?

- user testing → bug reports (fundamental contribution in FOSS!)
- `testing` suite
  - lack of compliance, among other reasons, keep packages out
- automated testing
  - lintian — automated policy compliance checker
    `http://lintian.debian.org/`
  - piuparts — stress testing of package installation expectations
    `http://piuparts.debian.org/`
  - periodic archive-wide rebuilds

References

Lucas Nussbaum, *Rebuilding Debian using distributed computing*, CLADE'09, `http://dl.acm.org/citation.cfm?id=1552318`

No automated bug filing (false positive will waste volunteer time and upset people), rather manual review + bug report.

QA Team

`http://qa.debian.org`

Loosely defined team: “every DD is in the QA Team”
Rather, a discussion place for people interested in distro-wide QA
More generally: interested in the distro as a whole

Maintainers of the QA infrastructure:

- DDPO: `http://qa.debian.org/developer.php?login=zack`
- PTS: `http://packages.qa.debian.org/o/ocaml.html`
- `http://lintian.debian.org/`
- `http://piuparts.debian.org/`
- rebuild scripts (now in “the cloud”!)
- ...

The Maintainer field

in the beginning, it was the base system
then the Maintainer field

```
Package: git
Version: 1:1.7.10-1
Maintainer: Gerrit Pape <pape@smarden.org>
Architecture: amd64
```

Pros:

- it gives pride and motivations
- "*wow, you maintain THAT!*"

Cons:

- create islands, increase barriers to contributions
- agile methods say "fight strong ~~code~~ package ownership"
- what if a maintainer disappears?

Missing In Action (MIA)

Given enough volunteers,
someone will eventually disappear without notice.

You can't *assume* they are gone; might be simply busy with RL.
You can't do nothing: blocks others, and frustrates volunteers.
MIA process / MIA team:

- big brother like tracking of developers activities
- periodic, cadenced pings
- package orphaning in the end
- still much more work than in the responsible leave scenario

References

Martin Michlmayr, *Managing volunteer activity in free software projects*, USENIX 2004 (Freenix track),
`http://static.usenix.org/publications/library/proceedings/usenix04/tech/freenix/full_papers/michlmayr/michlmayr_html/`

Non Maintainer Uploads (NMU)

You've found a serious issue in a package and a fix is available. How do you deploy it?

NMU: upload performed by people other than the official maintainer

via maintainer:

1. report bug+patch
2. wait maintainer reacts
3. eventually: upload

What if the maintainer is MIA?
What is the community impact?

- very effective in reducing volunteer inertia
- lot of care needed to avoid upset people and bad publicity to the process
  - principle: NMU to help fellow developers
  - make it easy to integrate your work
  - DELAYED/XX uploads

Non Maintainer Uploads (NMU) (cont.)

References:

- http://www.debian.org/doc/manuals/developers-reference/pkgs.html#nmu
  - note the care in avoiding to upset and/or undermine the authority of the legitimate maintainer
  - volunteer work is to be cherished, until it gets in the way of the work of other volunteers
- a (successful) experiment in dispelling NMU's bad publicity: http://upsilon.cc/~zack/hacking/debian/rcbw/

Exercise

Where does the need for the NMU process come from?
Why is it not needed in other large Free Software projects?

Outline

1 FOSS concepts

2 Debian overview

3 Philosophy

4 Organization

5 Processes

6 Derivatives

7 Appendix: packaging tutorial

8 Appendix: contribute

Interlude — derivatives how to

Free Software 101 — reminder

3. The freedom to redistribute copies so you can help your neighbor.
4. The freedom to distribute copies of your modified versions to others.

When applied to distros: derived distributions, AKA derivatives

How?
1. take existing packages and add your extras
2. patch & rebuild packages as needed
3. sync periodically

Derivatives are game changers

Derivatives have changed the way in which distros are made

- derivatives' focus is on customization
- people power is needed "only" for that

everybody wins (if done properly)

- derivative: massive reuse of packaging work
- "mother" distro: reach out to new public
  - users *and* contributors

Debian derivatives

Debian: a base for ≈140 active derivatives — distrowatch.com

- Tucunare, LinEx, Inquisitor, Grml, UniventionCorporateServer, Vanillux, Emdebian, Crunchbang, PureOS, StormOS, Ubuntu, GNUSTEP, gNewSense, Debathena, Maemo, LMDE, SPACEflight, BCCD, Bayanihan, semplice, ArchivistaBox, Knoppix, Tails, BlankOn, AlienVault-OSSIM, DoudouLinux, Vyatta, Symbiosis, VoyageLinux, Lihuen, LinuxAdvanced, Aptosid, Canaima, siduction, ZevenOS-Neptune, BOSSlinux, Parsix, AstraLinux, ProgressLinux, Finnix, SprezzOS, CoreBiz, Epidemic-Linux, MetamorphoseLinux , . . .

Why?

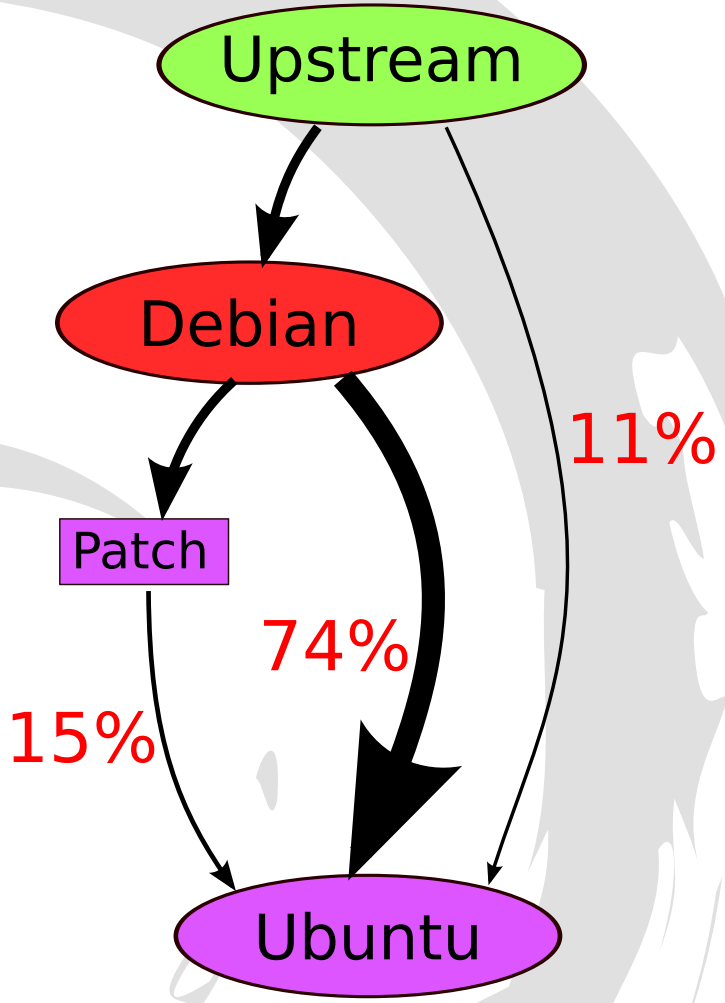
- quality & licensing assurances
- solid base system
- huge package base
- the "*universal* OS", perfect for customizations

A Debian derivative example: Ubuntu

- started in 2004 by Canonical
  target: desktop
- Debian derivative
- very popular (15–20x Debian?)
- historical/past correlations

  main ↔ corporate
  universe ↔ community

  - heavily customized/forked in main
  - very close to Debian elsewhere
- sprouting its own derivatives (≈80)
  - . . . as Debian *transitive* derivatives



Data for Oneiric Ocelot, main + universe

Do you Debian?

- Ubuntu appears to be the most customized Debian derivative
- other derivs. ⇒ much larger amount of *pristine* Debian packages

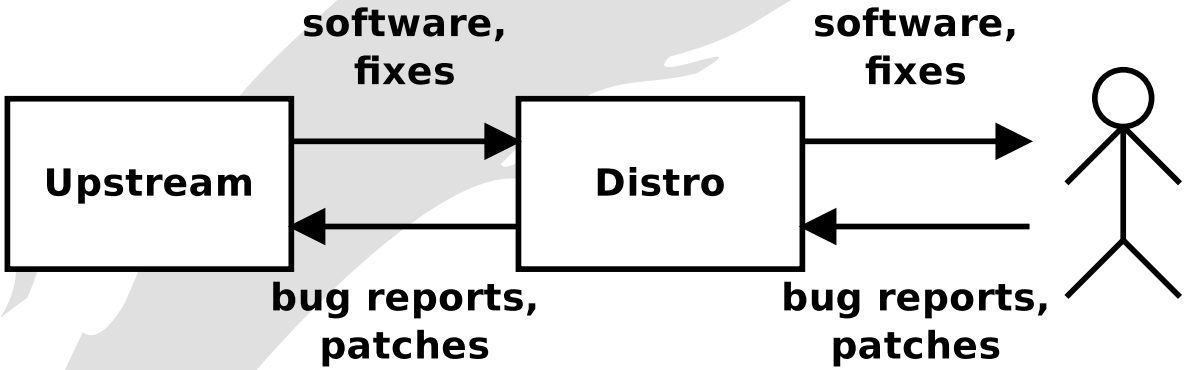
Tucunare, LinEx, Inquisitor, Grml, UniventionCorporateServer, Vanillux, Emdebian, Crunchbang, PureOS, StormOS, Ubuntu, GNUSTEP, gNewSense, Debathena, Maemo, LMDE, SPACEflight, BCCD, Bayanihan, semplice, ArchivistaBox, Knoppix, Tails, BlankOn, AlienVault-OSSIM, DoudouLinux, Vyatta, Symbiosis, VoyageLinux, Lihuen, LinuxAdvanced, Aptosid, Canaima, siduction, ZevenOS-Neptune, BOSSlinux, Parsix, AstraLinux, ProgressLinux, Finnix, SprezzOS, CoreBiz, Epidemic-Linux, MetamorphoseLinux , Debian, Xubuntu, Linux Mint, Ubuntu Studio, Mythbuntu, ArtistX, Asturix, Peppermint OS, TurnKey Linux, Kubuntu, Caixa Mágica, Lubuntu , . . .

if you are running a Debian (transitive) derivative, chances are you heavily depend on Debian and on its well-being

even if your distro hasn't told you

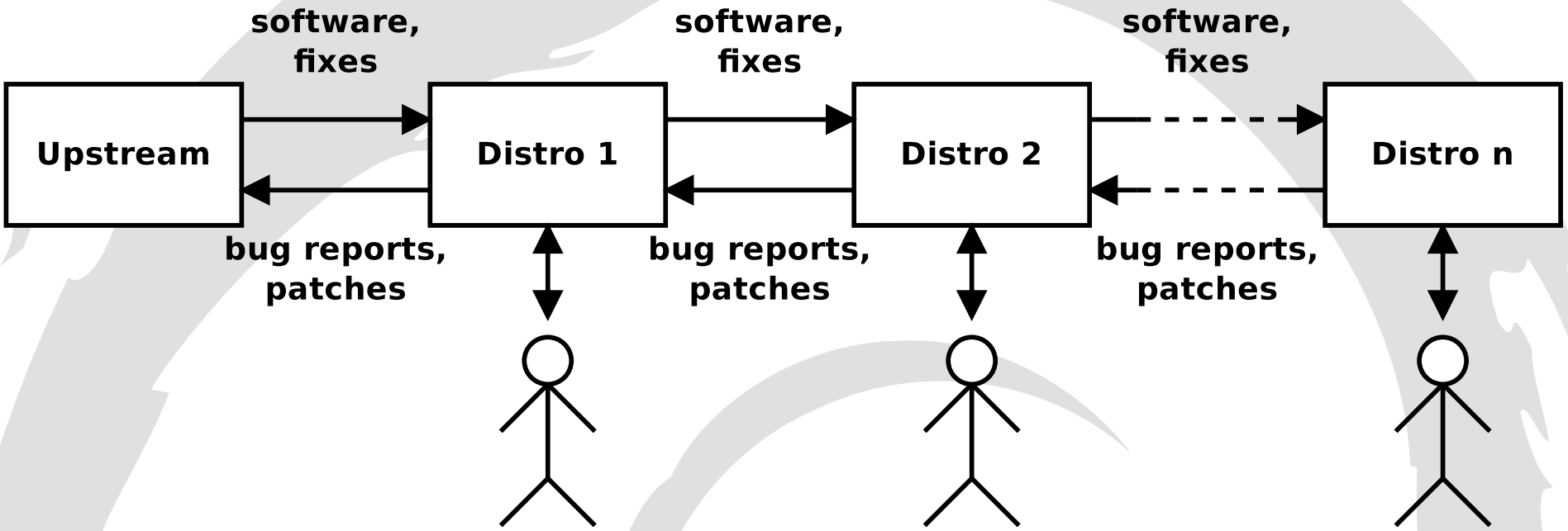
The distribution pipeline

yesterday . . .



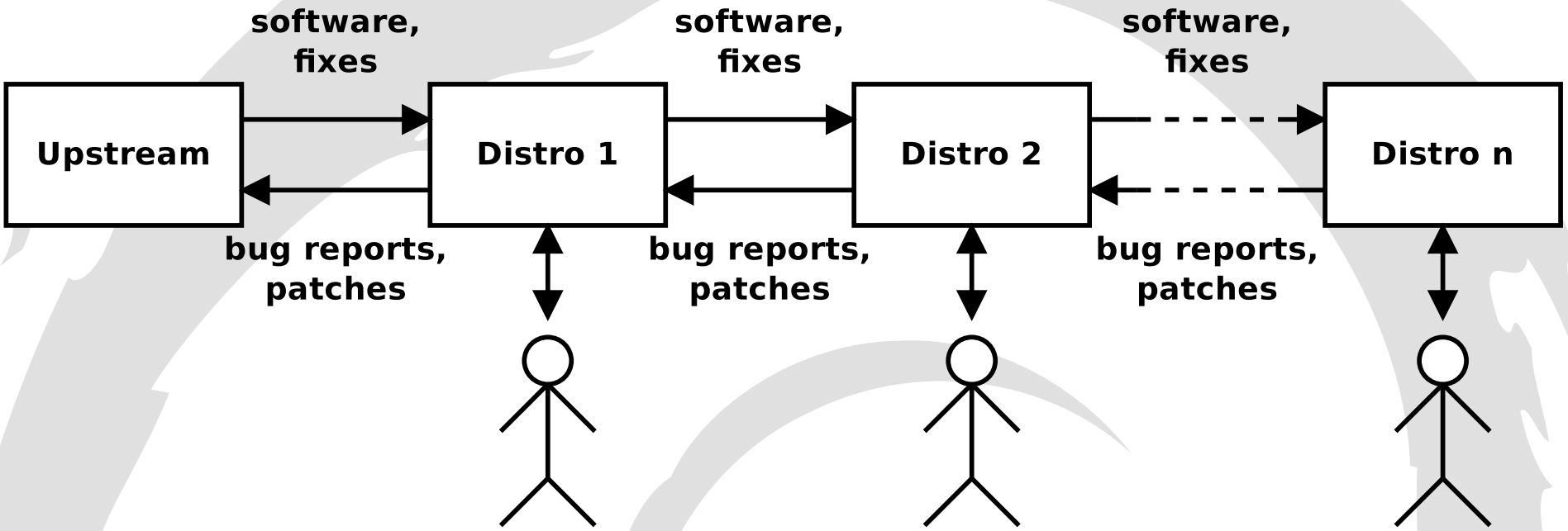
The *new* distribution pipeline

... today



The *new* distribution pipeline

... today



That's wonderful!

- freedom spreads
- more eyeballs swallow more bugs
- more potential contributors

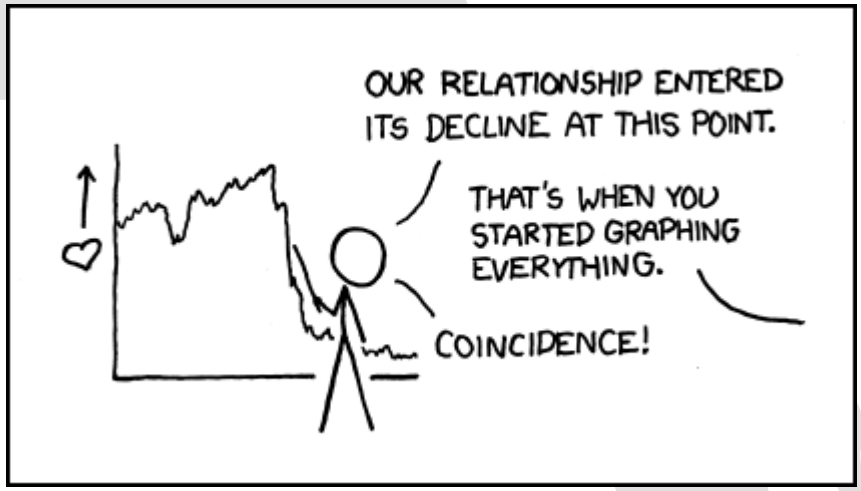
But.

- should be sustainable
- to everybody's benefit

Derivative pitfalls

- are derivatives always useful?
  - similar to: are *fork* useful in Free Software?
- what could possibly go wrong...

Looking back: a derivatives crisis



http://xkcd.com/523/

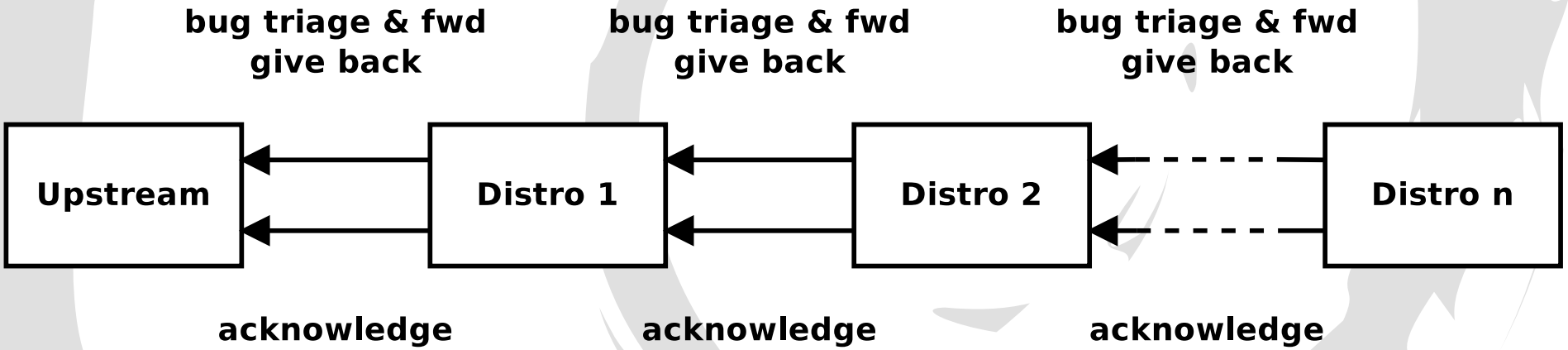
- Aug 1993 Debian birth
- Jul 1997 Debian Social Contract
- Mar 2004 Canonical birth
- Oct 2004 Ubuntu Warty release
- Apr 2005 Ubuntu Hoary release
- Jun 2005 Debian Sarge release (after a long delay)
- 2006–2007 The Big Crisis™
  - Debian: *"Ubuntu is not giving back!"*
  - Debian: *"Ubuntu is taking all the credit!"*
  - Ubuntu: *"Debian is not easy to work with"*
  - Ubuntu: *"Debian is hostile to us"*
- 2008 getting better, signs of mutual interest in collaboration
- 2009 failed release coordination results in new crisis
  - due to communication issues

A Debian-ic vision of derivatives

2010 propose a Debian-compatible vision of derivatives
present it to *both* (making headlines)

Free Software is bigger and more important
than Debian and any other distro or project

1. give back, i.e. reduce patch flow viscosity
2. give credit where credit is due



Implementing that vision

Derivatives Front Desk `wiki.debian.org/DerivativesFrontDesk`

- contact point and discussion place
- make emerge a critical mass of DDs interested in collaboration

Debian dErivatives eXchange (DEX) `dex.alioth.debian.org`

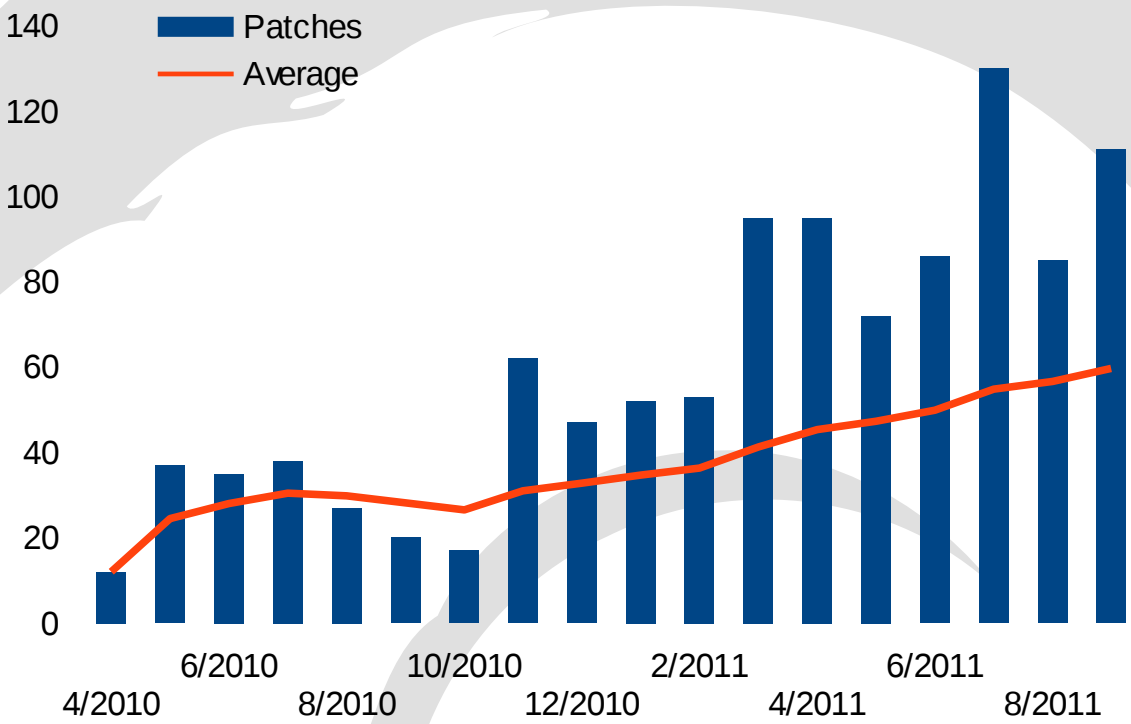
- short-lived cross-distro projects to merge back changes
- visible progress

Derivatives Census `wiki.debian.org/Derivatives/Census`

- gather detailed information about derivatives
- useful for QA and for relationship development
- “patches.ubuntu.com” equivalent for all derivatives

Solutions for *all* derivatives, obtained generalizing lessons learned from the Debian ↔ Ubuntu experience.

Early results



forwarded Ubuntu→Debian patches per month; source: Debian BTS

- increase in forwarded patches
- new “upstream first” guidelines for new packages (in Universe)
- more Ubuntu people getting involved in Debian as DMs/DDs

Outline

1. FOSS concepts
2. Debian overview
3. Philosophy
4. Organization
5. Processes
6. Derivatives
7. Appendix: packaging tutorial
8. Appendix: contribute

About this tutorial

- Goal: **tell you what you really need to know about Debian packaging**
  - Modify existing packages
  - Create your own packages
  - Interact with the Debian community
  - Become a Debian power-user
- Covers the most important points, but is not complete
  - You will need to read more documentation
- Most of the content also applies to Debian derivatives distributions
  - That includes Ubuntu

Packaging tutorial — outline

1. Introduction
2. Creating source packages
3. Building and testing packages
4. Advanced packaging topics
5. Maintaining packages in Debian
6. Conclusion
7. Answers to practical sessions

Packaging tutorial — outline

1. Introduction
2. Creating source packages
3. Building and testing packages
4. Advanced packaging topics
5. Maintaining packages in Debian
6. Conclusion
7. Answers to practical sessions

Debian

- **GNU/Linux distribution**
- 1st major distro developed “openly in the spirit of GNU”
- **Non-commercial**, built collaboratively by over 1,000 volunteers
- 3 main features:
  - **Quality** – culture of technical excellence
    We release when it’s ready
  - **Freedom** – devs and users bound by the *Social Contract*
    Promoting the culture of Free Software since 1993
  - **Independence** – no (single) company babysitting Debian
    And open decision-making process (*do-ocracy* + *democracy*)
- **Amateur** in the best sense: done for the love of it

Debian packages

- **.deb** files (binary packages)
- A very powerful and convenient way to distribute software to users
- One of the two most common packages format (with RPM)
- Universal:
  - 30,000 binary packages in Debian
    $\rightarrow$ most of the available free software is packaged in Debian!
  - For 12 ports (architectures), including 2 non-Linux (Hurd; KFreeBSD)
  - Also used by 120 Debian derivatives distributions

The Deb package format

- `.deb` file: an `ar` archive

```
$ ar tv wget_1.12-2.1_i386.deb
rw-r--r-- 0/0       4 Sep  5 15:43 2010 debian-binary
rw-r--r-- 0/0    2403 Sep  5 15:43 2010 control.tar.gz
rw-r--r-- 0/0  751613 Sep  5 15:43 2010 data.tar.gz
```

  - `debian-binary`: version of the deb file format, `"2.0\n"`
  - `control.tar.gz`: metadata about the package
    `control, md5sums, (pre|post)(rm|inst), triggers, shlibs, ...`
  - `data.tar.gz`: data files of the package

- You could create your `.deb` files manually
  `http://tldp.org/HOWTO/html_single/Debian-Binary-Package-Building-HOWTO/`
- But most people don't do it that way

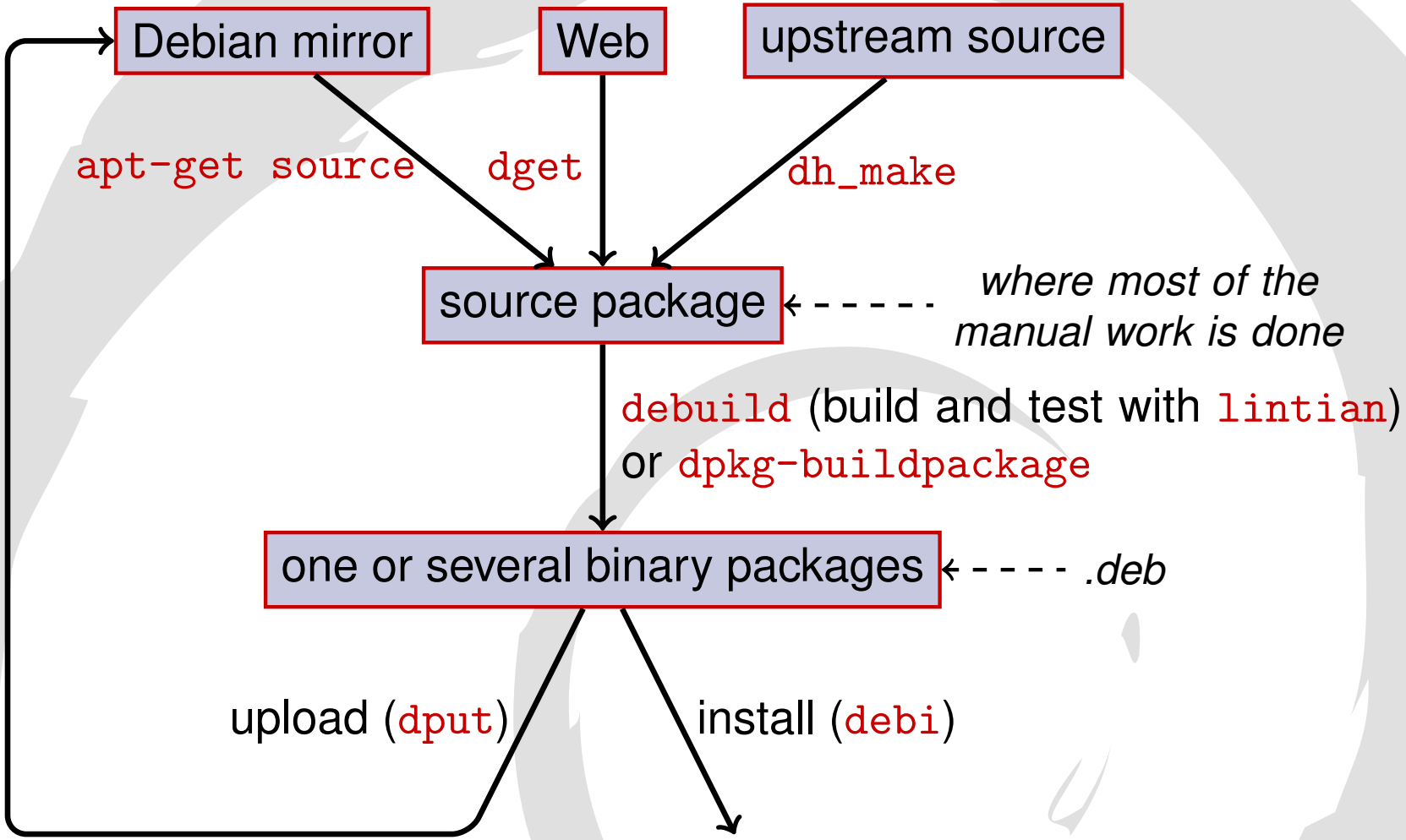
This tutorial: create Debian packages, the Debian way

Tools you will need

- A Debian (or Ubuntu) system (with root access)
- Some packages:
  - **build-essential**: has dependencies on the packages that will be assumed to be available on the developers' machine (no need to specify them in the `Build-Depends:` control field of your package)
    - includes a dependency on **dpkg-dev**, which contains basic Debian-specific tools to create packages
  - **devscripts**: contains many useful scripts for Debian maintainers

Many other tools will also be mentioned later, such as **debhelper**, **cdbs**, **quilt**, **pbuilder**, **sbuild**, **lintian**, **svn-buildpackage**, **git-buildpackage**, ... Install them when you need them.

General packaging workflow



Example: rebuilding dash

1. Install packages needed to build dash, and devscripts
   `sudo apt-get build-dep dash`
   (requires `deb-src` lines in `/etc/apt/sources.list`)
   `sudo apt-get install --no-install-recommends devscripts`
2. Create a working directory, and get in it:
   `mkdir /tmp/debian-tutorial ; cd /tmp/debian-tutorial`
3. Grab the `dash` source package
   `apt-get source dash`
   (This needs you to have `deb-src` lines in your `/etc/apt/sources.list`)
4. Build the package
   `cd dash-*`
   `debuild -us -uc` (`-us -uc` disables signing the package with GPG)
5. Check that it worked
   - There are some new `.deb` files in the parent directory
6. Look at the `debian/` directory
   - That's where the packaging work is done

Packaging tutorial — outline

1. Introduction
2. Creating source packages
3. Building and testing packages
4. Advanced packaging topics
5. Maintaining packages in Debian
6. Conclusion
7. Answers to practical sessions

Source package

- One source package can generate several binary packages
  e.g the `libtar` source generates the `libtar0` and `libtar-dev` binary packages
- Two kinds of packages: (if unsure, use non-native)
  - Native packages: normally for Debian specific software (*dpkg*, *apt*)
  - Non-native packages: software developed outside Debian
- Main file: `.dsc` (meta-data)
- Other files depending on the version of the source format
  - 1.0 – native: `package_version.tar.gz`
  - 1.0 – non-native:
    - `pkg_ver.orig.tar.gz` : upstream source
    - `pkg_debver.diff.gz` : patch to add Debian-specific changes
  - 3.0 (quilt):
    - `pkg_ver.orig.tar.gz` : upstream source
    - `pkg_debver.debian.tar.gz` : tarball with the Debian changes

(See `dpkg-source(1)` for exact details)

Source package example (wget_1.12-2.1.dsc)

```
Format: 3.0 (quilt)
Source: wget
Binary: wget
Architecture: any
Version: 1.12-2.1
Maintainer: Noel Kothe <noel@debian.org>
Homepage: http://www.gnu.org/software/wget/
Standards-Version: 3.8.4
Build-Depends: debhelper (>> 5.0.0), gettext, texinfo,
 libssl-dev (>= 0.9.8), dpatch, info2man
Checksums-Sha1:
 50d4ed2441e67[..]1ee0e94248 2464747 wget_1.12.orig.tar.gz
 d4c1c8bbe431d[..]dd7cef3611 48308 wget_1.12-2.1.debian.tar.gz
Checksums-Sha256:
 7578ed0974e12[..]dcba65b572 2464747 wget_1.12.orig.tar.gz
 1e9b0c4c00eae[..]89c402ad78 48308 wget_1.12-2.1.debian.tar.gz
Files:
 141461b9c04e4[..]9d1f2abf83 2464747 wget_1.12.orig.tar.gz
 e93123c934e3c[..]2f380278c2 48308 wget_1.12-2.1.debian.tar.gz
```

Retrieving an existing source package

- From the Debian archive:
  - `apt-get source` *package*
  - `apt-get source` *package=version*
  - `apt-get source` *package/release*

  (You need `deb-src` lines in `sources.list`)
- From the Internet:
  - `dget` *url-to.dsc*
  - `dget http://snapshot.debian.org/archive/debian-archive/20090802T004153Z/debian/dists/bo/main/source/web/wget_1.4.4-6.dsc`

    (`snapshot.d.o` provides all packages from Debian since 2005)
- From the (declared) version control system:
  - `debcheckout` *package*
- Once downloaded, extract with `dpkg-source -x` *file.dsc*

Creating a basic source package

- Download the upstream source
  (*upstream source* = the one from the software's original developers)
- Rename to `<source_package>_<upstream_version>.orig.tar.gz`
  (example: `simgrid_3.6.orig.tar.gz`)
- Untar it
- `cd upstream_source && dh_make` (from the **dh-make** package)
- There are some alternatives to `dh_make` for specific sets of packages:
  dh-make-perl, **dh-make-php**, ...
- `debian/` directory created, with a lot of files in it

Files in debian/

All the packaging work should be made by modifying files in `debian/`

- Main files:
  - **control** – meta-data about the package (dependencies, etc)
  - **rules** – specifies how to build the package
  - **copyright** – copyright information for the package
  - **changelog** – history of the Debian package
- Other files:
  - compat
  - watch
  - dh_install* targets
    *.dirs, *.docs, *.manpages, . . .
  - maintainer scripts
    *.postinst, *.prerm, . . .
  - source/format
  - patches/ – if you need to modify the upstream sources
- Several files use a format based on RFC 822 (mail headers)

debian/changelog

- Lists the Debian packaging changes
- Gives the current version of the package

$$\underbrace{1.2.1.1}_{\text{Upstream version}}\text{-}\underbrace{5}_{\text{Debian revision}}$$

- Edited manually or with `dch`
  - Create a changelog entry for a new release: `dch -i`
- Special format to automatically close Debian or Ubuntu bugs
  Debian: `Closes: #595268`; Ubuntu: `LP: #616929`
- Installed as `/usr/share/doc/`*`package`*`/changelog.Debian.gz`

```
mpich2 (1.2.1.1-5) unstable; urgency=low

  * Use /usr/bin/python instead of /usr/bin/python2.5. Allow
    to drop dependency on python2.5.  Closes: #595268
  * Make /usr/bin/mpdroot setuid. This is the default after
    the installation of mpich2 from source, too. LP: #616929
    + Add corresponding lintian override.

 -- Lucas Nussbaum <lucas@debian.org>  Wed, 15 Sep 2010 18:13:44 +0200
```

debian/control

- Package metadata
  - For the source package itself
  - For each binary package built from this source
- Package name, section, priority, maintainer, uploaders, build-dependencies, dependencies, description, homepage, ...
- Documentation: Debian Policy chapter 5
  `http://www.debian.org/doc/debian-policy/ch-controlfields.html`

```
Source: wget
Section: web
Priority: important
Maintainer: Noel Kothe <noel@debian.org>
Build-Depends: debhelper (>> 5.0.0), gettext, texinfo,
 libssl-dev (>= 0.9.8), dpatch, info2man
Standards-Version: 3.8.4
Homepage: http://www.gnu.org/software/wget/

Package: wget
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: retrieves files from the web
 Wget is a network utility to retrieve files from the Web
```

Architecture: all or any

Two kinds of binary packages:

- Packages with different contents on each Debian architecture
  - Example: C program
  - `Architecture: any` in `debian/control`
    - Or, if it only works on a subset of architectures:
      `Architecture: amd64 i386 ia64 hurd-i386`
  - buildd.debian.org: builds all the other architectures for you on upload
  - Named *package*`_`*version*`_`*architecture*`.deb`
- Packages with the same content on all architectures
  - Example: Perl library
  - `Architecture: all` in `debian/control`
  - Named *package*`_`*version*`_all.deb`

A source package can generate a mix of `Architecture: any` and `Architecture: all` binary packages

debian/rules

- Makefile
- Interface used to build Debian packages
- Documented in Debian Policy, chapter 4.8
  `http://www.debian.org/doc/debian-policy/ch-source.html#s-debianrules`
- Five required targets:
  - `build`: should perform all the configuration and compilation
  - `binary`, `binary-arch`, `binary-indep`: build the binary packages
    - `dpkg-buildpackage` will call `binary` to build all the packages, or `binary-arch` to build only the `Architecture: any` packages
  - `clean`: clean up the source directory

Packaging helpers – debhelper

- You could write shell code in `debian/rules` directly
  - See the `adduser` package for example
- Better practice (used by most packages): use a *Packaging helper*
- Most popular one: **debhelper** (used by 98% of packages)
- Goals:
  - Factor the common tasks in standard tools used by all packages
  - Fix some packaging bugs once for all packages

  dh_installdirs, dh_installchangelogs, dh_installdocs, dh_installexamples, dh_install, dh_installdebconf, dh_installinit, dh_link, dh_strip, dh_compress, dh_fixperms, dh_perl, dh_makeshlibs, dh_installdeb, dh_shlibdeps, dh_gencontrol, dh_md5sums, dh_builddeb, ...

  - Called from `debian/rules`
  - Configurable using command parameters or files in `debian/`

  package`.docs`, *package*`.examples`, *package*`.install`, *package*`.manpages`, ...
- Third-party helpers for sets of packages: **python-support**, **dh_ocaml**, ...
- Gotcha: `debian/compat`: Debhelper compatibility version (use "7")

debian/rules using debhelper (1/2)

```
#!/usr/bin/make -f

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

build:
        $(MAKE)
        #docbook-to-man debian/packagename.sgml > packagename.1

clean:
        dh_testdir
        dh_testroot
        rm -f build-stamp configure-stamp
        $(MAKE) clean
        dh_clean

install: build
        dh_testdir
        dh_testroot
        dh_clean -k
        dh_installdirs
        # Add here commands to install the package into debian/package
        $(MAKE) DESTDIR=$(CURDIR)/debian/packagename install
```

debian/rules using debhelper (2/2)

```
# Build architecture-independent files here.
binary-indep: build install

# Build architecture-dependent files here.
binary-arch: build install
        dh_testdir
        dh_testroot
        dh_installchangelogs
        dh_installdocs
        dh_installexamples
        dh_install
        dh_installman
        dh_link
        dh_strip
        dh_compress
        dh_fixperms
        dh_installdeb
        dh_shlibdeps
        dh_gencontrol
        dh_md5sums
        dh_builddeb

binary: binary-indep binary-arch
.PHONY: build clean binary-indep binary-arch binary install configure
```

CDBS

- With debhelper, still a lot of redundancy between packages
- Second-level helpers that factor common functionality
  - E.g building with `./configure && make && make install` or CMake
- CDBS:
  - Introduced in 2005, based on advanced *GNU make* magic
  - Documentation: `/usr/share/doc/cdbs/`
  - Support for Perl, Python, Ruby, GNOME, KDE, Java, Haskell, ...
  - But some people hate it:
    - Sometimes difficult to customize package builds: "*twisty maze of makefiles and environment variables*"
    - Slower than plain debhelper (many useless calls to `dh_*`)

```
#!/usr/bin/make -f
include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/autotools.mk

# add an action after the build
build/mypackage::
	/bin/bash debian/scripts/foo.sh
```

Dh (aka Debhelper 7, or dh7)

- Introduced in 2008 as a *CDBS killer*
- **dh** command that calls `dh_*`
- Simple *debian/rules*, listing only overrides
- Easier to customize than CDBS
- Doc: manpages (`debhelper(7)`, `dh(1)`) + slides from DebConf9 talk
  `http://kitenet.net/~joey/talks/debhelper/debhelper-slides.pdf`

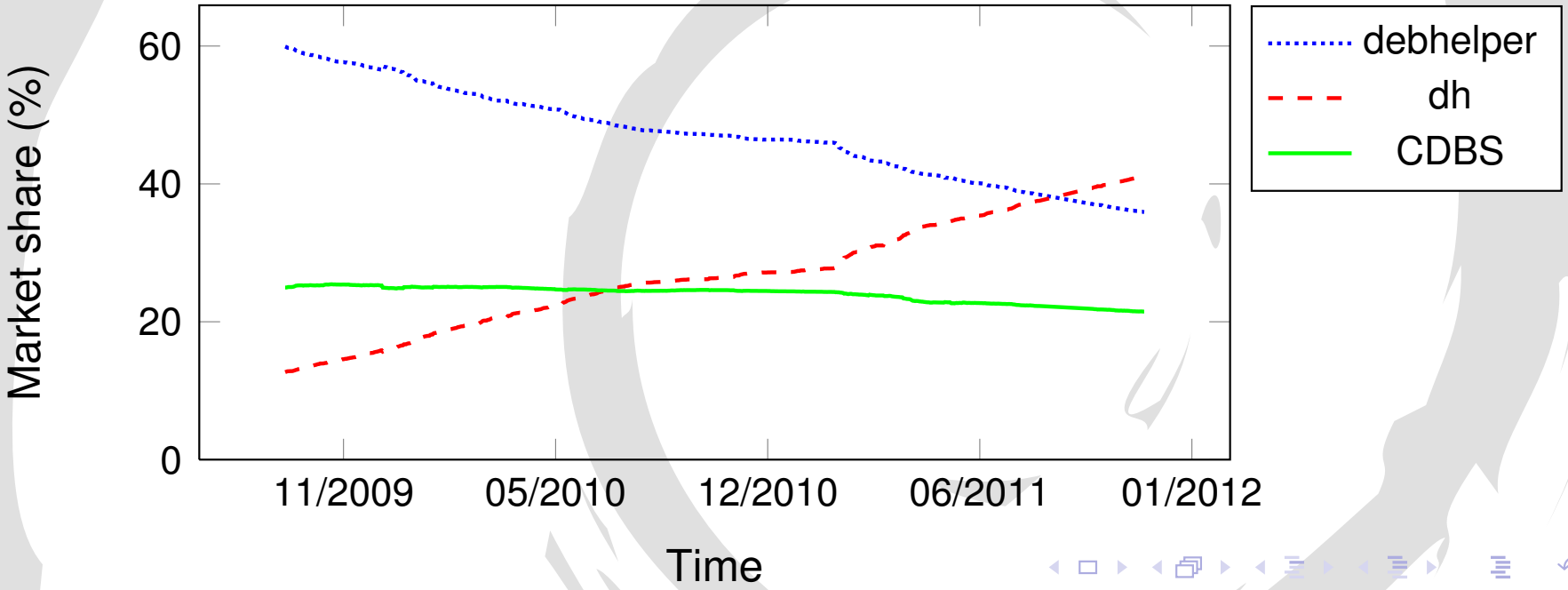
```
#!/usr/bin/make -f
%:
	dh $@

override_dh_auto_configure:
	dh_auto_configure -- --with-kitchen-sink

override_dh_auto_build:
	make world
```

Classic debhelper vs CDBS vs dh

- Mind shares:
  Classic debhelper: 36% CDBS: 21% dh: 41%
- Which one should I learn?
  - Probably a bit of all of them
  - You need to know debhelper to use dh and CDBS
  - You might have to modify CDBS packages
- Which one should I use for a new package?
  - **dh** (only solution with an increasing mind share)



Packaging tutorial — outline

1. Introduction
2. Creating source packages
3. Building and testing packages
4. Advanced packaging topics
5. Maintaining packages in Debian
6. Conclusion
7. Answers to practical sessions

Building packages

- `apt-get build-dep mypackage`
  Installs the *build-dependencies* (for a package already in Debian)
  Or `mk-build-deps -ir` (inside the package source tree)

- `debuild`: build, test with `lintian`, sign with GPG

- Also possible to call `dpkg-buildpackage` directly
  - Usually with `dpkg-buildpackage -us -uc`

- It is better to build packages in a clean & minimal environment
  - `pbuilder` – helper to build packages in a *chroot*
    Good documentation: `https://wiki.ubuntu.com/PbuilderHowto`
    (optimization: `cowbuilder ccache distcc`)
  - `schroot` and `sbuild`: used on the Debian build daemons
    (not as simple as `pbuilder`, but allows LVM snapshots
    see: `https://help.ubuntu.com/community/SbuildLVMHowto`)

- Generates `.deb` files and a `.changes` file
  - `.changes`: describes what was built; used to upload the package

Installing and testing packages

- Install the package locally: `debi` (will use `.changes` to know what to install)
- List the content of the package: `debc ../mypackage<TAB>.changes`
- Compare the package with a previous version:
  `debdiff ../mypackage_1_*.changes ../mypackage_2_*.changes`
  or to compare the sources:
  `debdiff ../mypackage_1_*.dsc ../mypackage_2_*.dsc`
- Check the package with `lintian` (static analyzer):
  `lintian ../mypackage<TAB>.changes`
  `lintian -i`: gives more information about the errors
- Upload the package to Debian (`dput`) (needs configuration)
- Manage a private Debian archive with `reprepro`
  Documentation: `http://mirrorer.alioth.debian.org/`

Practical session 1: modifying the grep package

1. Go to `http://ftp.debian.org/debian/pool/main/g/grep/` and download version 2.6.3-3 of the package (if you use Ubuntu 11.10 or later, or Debian testing or unstable, use version 2.9-1 or 2.9-2 instead)
   - If the source package is not unpacked automatically, unpack it with `dpkg-source -x grep_*.dsc`
2. Look at the files in `debian/`.
   - How many binary packages are generated by this source package?
   - Which packaging helper does this package use?
3. Build the package
4. We are now going to modify the package. Add a changelog entry and increase the version number.
5. Now disable perl-regexp support (it is a `./configure` option)
6. Rebuild the package
7. Compare the original and the new package with debdiff
8. Install the newly built package
9. Cry if you messed up ;)

Packaging tutorial — outline

1. Introduction
2. Creating source packages
3. Building and testing packages
4. Advanced packaging topics
5. Maintaining packages in Debian
6. Conclusion
7. Answers to practical sessions

debian/copyright

- Copyright and license information for the source and the packaging
- Traditionally written as a text file
- New machine-readable format:
  `http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/`

```
Format: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: X Solitaire
Source: ftp://ftp.example.com/pub/games

Files: *
Copyright: Copyright 1998 John Doe <jdoe@example.com>
License: GPL-2+
 This program is free software; you can redistribute it
 [...]
 .
 On Debian systems, the full text of the GNU General Public
 License version 2 can be found in the file
 ‘/usr/share/common-licenses/GPL-2’.

Files: debian/*
Copyright: Copyright 1998 Jane Smith <jsmith@example.net>
License:
 [LICENSE TEXT]
```

Modifying the upstream source

Often needed:

- Fix bugs or add customizations that are specific to Debian
- Backport fixes from a newer upstream release

Several methods to do it:

- Modifying the files directly
  - Simple
  - But no way to track and document the changes
- Using patch systems
  - Eases contributing your changes to upstream
  - Helps sharing the fixes with derivatives
  - Gives more exposure to the changes
    `http://patch-tracker.debian.org/`

Patch systems

- Principle: changes are stored as patches in `debian/patches/`
- Applied and unapplied during build
- Past: several implementations – *simple-patchsys* (*cdbs*), *dpatch*, ***quilt***
  - Each supports two `debian/rules` targets:
    - `debian/rules patch`: apply all patches
    - `debian/rules unpatch`: de-apply all patches
  - More documentation: `http://wiki.debian.org/debian/patches`
- **New source package format with built-in patch system: 3.0 (quilt)**
  - Recommended solution
  - You need to learn *quilt*
    `http://pkg-perl.alioth.debian.org/howto/quilt.html`
  - Patch-system-agnostic tool in `devscripts`: `edit-patch`

Documentation of patches

- Standard headers at the beginning of the patch
- Documented in DEP-3 - Patch Tagging Guidelines
  `http://dep.debian.net/deps/dep3/`

```
Description: Fix widget frobnication speeds
 Frobnicating widgets too quickly tended to cause explosions.
Forwarded: http://lists.example.com/2010/03/1234.html
Author: John Doe <johndoe-guest@users.alioth.debian.org>
Applied-Upstream: 1.2, http://bzr.foo.com/frobnicator/revision/123
Last-Update: 2010-03-29

--- a/src/widgets.c
+++ b/src/widgets.c
@@ -101,9 +101,6 @@ struct {
```

Doing things during installation and removal

- Decompressing the package is sometimes not enough
- Create/remove system users, start/stop services, manage *alternatives*
- Done in *maintainer scripts*
  `preinst, postinst, prerm, postrm`
  - Snippets for common actions can be generated by debhelper
- Documentation:
  - Debian Policy Manual, chapter 6
    `http://www.debian.org/doc/debian-policy/ch-maintainerscripts.html`
  - Debian Developer's Reference, chapter 6.4
    `http://www.debian.org/doc/developers-reference/best-pkging-practices.html`
  - `http://people.debian.org/~srivasta/MaintainerScripts.html`
- Prompting the user
  - Must be done with **debconf**
  - Documentation: `debconf-devel(7)` (`debconf-doc` package)

Monitoring upstream versions

- Specify where to look in `debian/watch` (see `uscan(1)`)

```
version=3

http://tmrc.mit.edu/mirror/twisted/Twisted/(\d\.\d)/ \
  Twisted-([\d\.]*)\.tar\.bz2
```

- Debian infrastructure that makes use of `debian/watch`:
  Debian External Health Status
  `http://dehs.alioth.debian.org/`
- Maintainer warned by emails sent to the Package Tracking System
  `http://packages.qa.debian.org/`
- `uscan`: run a manual check
- `uupdate`: try to update your package to the latest upstream version

Packaging with a Version Control System

- Several tools to help manage branches and tags for your packaging work: `svn-buildpackage`, `git-buildpackage`
- Example: `git-buildpackage`
  - `upstream` branch to track upstream with `upstream/`*`version`* tags
  - `master` branch tracks the Debian package
  - `debian/`*`version`* tags for each upload
  - `pristine-tar` branch to be able to rebuild the upstream tarball
- `Vcs-*` fields in `debian/control` to locate the repository
  - `http://wiki.debian.org/Alioth/Git`
  - `http://wiki.debian.org/Alioth/Svn`

```
Vcs-Browser: http://git.debian.org/?p=devscripts/devscripts.git
Vcs-Git: git://git.debian.org/devscripts/devscripts.git

Vcs-Browser: http://svn.debian.org/viewsvn/pkg-perl/trunk/libwww-perl/
Vcs-Svn: svn://svn.debian.org/pkg-perl/trunk/libwww-perl
```

- VCS-agnostic interface: `debcheckout`, `debcommit`, `debrelease`
  - `debcheckout grep` → checks out the source package from Git

Backporting packages

- Goal: use a newer version of a package on an older system
  e.g use *mutt* from Debian *unstable* on Debian *stable*
- General idea:
  - Take the source package from Debian unstable
  - Modify it so that it builds and works fine on Debian stable
    - Sometimes trivial (no changes needed)
    - Sometimes difficult
    - Sometimes impossible (many unavailable dependencies)
- Some backports are provided and supported by the Debian project
  `http://backports.debian.org/`

Packaging tutorial — outline

1. Introduction
2. Creating source packages
3. Building and testing packages
4. Advanced packaging topics
5. Maintaining packages in Debian
6. Conclusion
7. Answers to practical sessions

Several ways to contribute to Debian

- **Worst** way to contribute:
  1. Package your own application
  2. Get it into Debian
  3. Disappear
- **Better** ways to contribute:
  - Get involved in packaging teams
    - Many teams that focus on set of packages, and need help
    - List available at `http://wiki.debian.org/Teams`
    - An excellent way to learn from more experienced contributors
  - Adopt existing unmaintained packages (*orphaned packages*)
  - Bring new software to Debian
    - Only if it's interesting/useful enough, please
    - Are there alternatives already packaged in Debian?

Adopting orphaned packages

- Many unmaintained packages in Debian
- Full list + process: `http://www.debian.org/devel/wnpp/`
- Installed on your machine: `wnpp-alert`
- Different states:
  - **O**rphaned: the package is unmaintained
    Feel free to adopt it
  - **RFA**: **R**equest **F**or **A**dopter
    Maintainer looking for adopter, but continues work in the meantime
    Feel free to adopt it. A mail to the current maintainer is polite
  - **ITA**: **I**ntent **T**o **A**dopt
    Someone intends to adopt the package
    You could propose your help!
  - **RFH**: **R**equest **F**or **H**elp
    The maintainer is looking for help
- Some unmaintained packages not detected $\rightarrow$ not orphaned yet
- When in doubt, ask `debian-qa@lists.debian.org`
  or `#debian-qa` on `irc.debian.org`

Adopting a package: example

```
From: You <you@yourdomain>
To: 640454@bugs.debian.org, control@bugs.debian.org
Cc: Francois Marier <francois@debian.org>
Subject: ITA: verbiste -- French conjugator

retitle 640454 ITA: verbiste -- French conjugator
owner 640454 !
thanks

Hi,

I am using verbiste and I am willing to take care of the package.

Cheers,

You
```

- Polite to contact the previous maintainer (especially if the package was RFAed, not orphaned)
- Very good idea to contact the upstream project

Getting your package in Debian

- You do not need any official status to get your package into Debian
  1. Prepare a source package
  2. Find a Debian Developer that will sponsor your package
- Official status (when you are already experienced):
  - **Debian Maintainer (DM):**
    Permission to upload your own packages
    See `http://wiki.debian.org/DebianMaintainer`
  - **Debian Developer (DD):**
    Debian project members; can vote and upload any package

Where to find help?

Help you will need:

- Advice and answers to your questions, code reviews
- Sponsorship for your uploads, once your package is ready

You can get help from:

- **Other members of a packaging team**
  - They know the specifics of your package
  - You can become a member of the team
- The Debian Mentors group (if your package doesn't fit in a team)
  - `http://wiki.debian.org/DebianMentorsFaq`
  - Mailing list: `debian-mentors@lists.debian.org` (also a good way to learn by accident)
  - IRC: `#debian-mentors` on `irc.debian.org`
  - `http://mentors.debian.net/`

Official documentation

- Debian Developers' Corner
  `http://www.debian.org/devel/`
  Links to many resources about Debian development
- Debian New Maintainers' Guide
  `http://www.debian.org/doc/maint-guide/`
  An introduction to Debian packaging, but could use an update
- Debian Developer's Reference
  `http://www.debian.org/doc/developers-reference/`
  Mostly about Debian procedures, but also some best packaging practices (part 6)
- Debian Policy
  `http://www.debian.org/doc/debian-policy/`
  - All the requirements that every package must satisfy
  - Specific policies for Perl, Java, Python, ...
- Ubuntu Packaging Guide
  `https://wiki.ubuntu.com/PackagingGuide`

Debian dashboards for maintainers

- **Source package centric**: Package Tracking System (PTS)
  `http://packages.qa.debian.org/dpkg`
- **Maintainer/team centric**: Developer's Packages Overview (DDPO)
  `http://qa.debian.org/developer.php?login=pkg-ruby-extras-maintainers@lists.alioth.debian.org`

More interested in Ubuntu?

- Ubuntu mainly manages the divergence with Debian
- No real focus on specific packages
  Instead, collaboration with Debian teams
- Usually recommend uploading new packages to Debian first
  `https://wiki.ubuntu.com/UbuntuDevelopment/NewPackages`
- Possibly a better plan:
  - Get involved in a Debian team and act as a bridge with Ubuntu
  - Help reduce divergence, triage bugs in Launchpad
  - Many Debian tools can help:
    - Ubuntu column on the Developer's packages overview
    - Ubuntu box on the Package Tracking System
    - Receive launchpad bugmail via the PTS

Packaging tutorial — outline

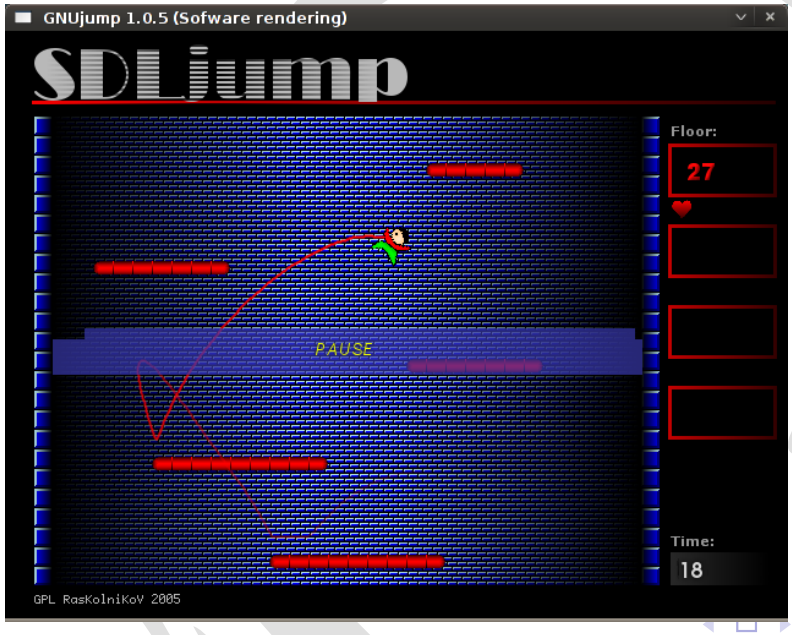
1. Introduction
2. Creating source packages
3. Building and testing packages
4. Advanced packaging topics
5. Maintaining packages in Debian
6. Conclusion
7. Answers to practical sessions

Conclusion

- You now have a full overview of Debian packaging
- But you will need to read more documentation
- Best practices have evolved over the years
  - If not sure, use the **dh** packaging helper, and the **3.0 (quilt)** format
- Things that were not covered in this tutorial:
  - UCF – manage user changes to configuration files when upgrading
  - dpkg triggers – group similar maintainer scripts actions together
  - Debian development organization:
    - Bug Tracking System (BTS)
    - Suites: stable, testing, unstable, experimental, security, *-updates, backports, . . .
    - Debian Blends – subsets of Debian targeting specific groups

Practical session 2: packaging GNUjump

1. Download GNUjump 1.0.6 from
   `http://ftp.gnu.org/gnu/gnujump/1.0.6/gnujump-1.0.6.tar.gz`
2. Create a Debian package for it
   - Install build-dependencies so that you can build the package
   - Get a basic working package
   - Finish filling `debian/control` and other files
3. Enjoy



Practical session 3: packaging a Java library

1. Take a quick look at some documentation about Java packaging:
   - `http://wiki.debian.org/Java`
   - `http://wiki.debian.org/Java/Packaging`
   - `http://www.debian.org/doc/packaging-manuals/java-policy/`
   - `http://pkg-java.alioth.debian.org/docs/tutorial.html`
   - Paper and slides from a Debconf10 talk about javahelper:
     `http://pkg-java.alioth.debian.org/docs/debconf10-javahelper-paper.pdf`
     `http://pkg-java.alioth.debian.org/docs/debconf10-javahelper-slides.pdf`
2. Download IRClib from `http://moepii.sourceforge.net/`
3. Package it

Practical session 4: packaging a Ruby gem

1. Take a quick look at some documentation about Ruby packaging:
   - `http://wiki.debian.org/Ruby`
   - `http://wiki.debian.org/Teams/Ruby`
   - `http://wiki.debian.org/Teams/Ruby/Packaging`
   - `gem2deb(1)`, `dh_ruby(1)` (in the `gem2deb` package)
2. Create a basic Debian source package from the `net-ssh` gem: `gem2deb net-ssh`
3. Improve it so that it becomes a proper Debian package

Packaging tutorial — outline

1. Introduction
2. Creating source packages
3. Building and testing packages
4. Advanced packaging topics
5. Maintaining packages in Debian
6. Conclusion
7. Answers to practical sessions

Practical session 1: modifying the grep package

1. Go to `http://ftp.debian.org/debian/pool/main/g/grep/` and download version 2.6.3-3 of the package (if you use Ubuntu 11.10 or later, or Debian testing or unstable, use version 2.9-1 or 2.9-2 instead)
2. Look at the files in `debian/`.
   - How many binary packages are generated by this source package?
   - Which packaging helper does this package use?
3. Build the package
4. We are now going to modify the package. Add a changelog entry and increase the version number.
5. Now disable perl-regexp support (it is a `./configure` option)
6. Rebuild the package
7. Compare the original and the new package with debdiff
8. Install the newly built package
9. Cry if you messed up ;)

Fetching the source

1. Go to `http://ftp.debian.org/debian/pool/main/g/grep/` and download version 2.6.3-3 of the package
   - Use dget to download the `.dsc` file:
     `dget http://cdn.debian.net/debian/pool/main/g/grep/grep_2.6.3-3.dsc`
   - According to `http://packages.qa.debian.org/grep`, `grep` version 2.6.3-3 is currently in *stable* (*squeeze*). If you have `deb-src` lines for *squeeze* in your `/etc/apt/sources.list`, you can use:
     `apt-get source grep=2.6.3-3`
     or `apt-get source grep/stable`
     or, if you feel lucky: `apt-get source grep`
   - The `grep` source package is composed of three files:
     - `grep_2.6.3-3.dsc`
     - `grep_2.6.3-3.debian.tar.bz2`
     - `grep_2.6.3.orig.tar.bz2`

     This is typical of the "3.0 (quilt)" format.
   - If needed, uncompress the source with
     `dpkg-source -x grep_2.6.3-3.dsc`

Looking around and building the package

2. Look at the files in `debian/`.
   - How many binary packages are generated by this source package?
   - Which packaging helper does this package use?

- According to `debian/control`, this package only generates one binary package, named `grep`.
- According to `debian/rules`, this package is typical of *classic* debhelper packaging, without using *CDBS* or *dh*. One can see the various calls to `dh_*` commands in `debian/rules`.

3. Build the package

- Use `apt-get build-dep grep` to fetch the build-dependencies
- Then `debuild` or `dpkg-buildpackage -us -uc` (Takes about 1 min)

Editing the changelog

4. We are now going to modify the package. Add a changelog entry and increase the version number.

- `debian/changelog` is a text file. You could edit it and add a new entry manually.
- Or you can use `dch -i`, which will add an entry and open the editor
- The name and email can be defined using the `DEBFULLNAME` and `DEBEMAIL` environment variables
- After that, rebuild the package: a new version of the package is built
- Package versioning is detailed in section 5.6.12 of the Debian policy `http://www.debian.org/doc/debian-policy/ch-controlfields.html`

Disabling Perl regexp support and rebuilding

5. Now disable perl-regexp support (it is a `./configure` option)
6. Rebuild the package

- Check with `./configure --help`: the option to disable Perl regexp is `--disable-perl-regexp`
- Edit `debian/rules` and find the `./configure` line
- Add `--disable-perl-regexp`
- Rebuild with `debuild` or `dpkg-buildpackage -us -uc`

Comparing and testing the packages

7. Compare the original and the new package with debdiff
8. Install the newly built package

- Compare the binary packages: `debdiff ../*changes`
- Compare the source packages: `debdiff ../*dsc`
- Install the newly built package: `debi`
  Or `dpkg -i ../grep_<TAB>`
- `grep -P foo` no longer works!

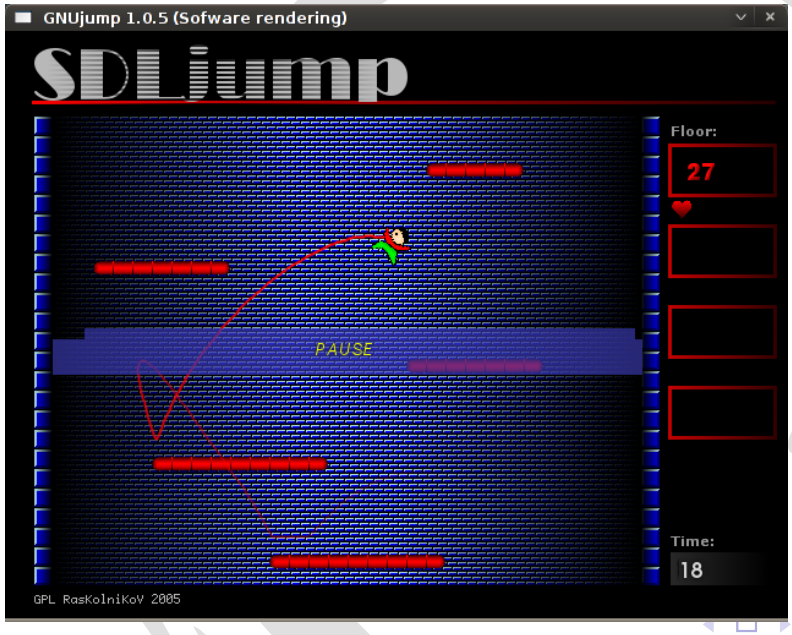
9. Cry if you messed up ;)

Or not: reinstall the previous version of the package:

- `apt-get install --reinstall grep=2.6.3-3` *(= previous version)*

Practical session 2: packaging GNUjump

1. Download GNUjump 1.0.6 from
   `http://ftp.gnu.org/gnu/gnujump/1.0.6/gnujump-1.0.6.tar.gz`
2. Create a Debian package for it
   - Install build-dependencies so that you can build the package
   - Get a basic working package
   - Finish filling `debian/control` and other files
3. Enjoy



Step by step...

- `wget http://ftp.gnu.org/gnu/gnujump/1.0.6/gnujump-1.0.6.tar.gz`
- `mv gnujump-1.0.6.tar.gz gnujump_1.0.6.orig.tar.gz`
- `tar xf gnujump_1.0.6.orig.tar.gz`
- `cd gnujump-1.0.6/`
- `dh_make`
  - Type of package: single binary (for now)

```
gnujump-1.0.6$ ls debian/
changelog              gnujump.default.ex   preinst.ex
compat                 gnujump.doc-base.EX  prerm.ex
control                init.d.ex            README.Debian
copyright              manpage.1.ex         README.source
docs                   manpage.sgml.ex      rules
emacsen-install.ex     manpage.xml.ex       source
emacsen-remove.ex      menu.ex              watch.ex
emacsen-startup.ex     postinst.ex
gnujump.cron.d.ex      postrm.ex
```

Step by step. . . (2)

- Look at `debian/changelog`, `debian/rules`, `debian/control` (auto-filled by **dh_make**)
- In `debian/control`:
  `Build-Depends: debhelper (>= 7.0.50 ), autotools-dev`
  Lists the *build-dependencies* = packages needed to build the package
- Try to build the package as-is (thanks to **dh** magic)
  - And add build-dependencies, until it builds
  - Hint: use `apt-cache search` and `apt-file` to find the packages
  - Example:

```
checking for sdl-config... no
checking for SDL - version >= 1.2.0... no
[...]
configure: error: *** SDL version 1.2.0 not found!
```

    → Add **libsdl1.2-dev** to Build-Depends and install it.
  - Better: use **pbuilder** to build in a clean environment

Step by step. . . (3)

- After installing `libsdl1.2-dev`, `libsdl-image1.2-dev`, `libsdl-mixer1.2-dev`, the package builds fine.
- Use `debc` to list the content of the generated package.
- Use `debi` to install it and test it.
- Fill in `debian/control` using `http://www.debian.org/doc/debian-policy/ch-controlfields.html`
- Test the package with `lintian`
- Remove the files that you don't need in `debian/`
- Compare your package with the one already packaged in Debian:
  - It splits the data files to a second package, that is the same across all architectures ($\rightarrow$ saves space in the Debian archive)
  - It installs a .desktop file (for the GNOME/KDE menus) and also integrates into the Debian menu
  - It fixes a few minor problems using patches

Practical session 3: packaging a Java library

1. Take a quick look at some documentation about Java packaging:
   - `http://wiki.debian.org/Java`
   - `http://wiki.debian.org/Java/Packaging`
   - `http://www.debian.org/doc/packaging-manuals/java-policy/`
   - `http://pkg-java.alioth.debian.org/docs/tutorial.html`
   - Paper and slides from a Debconf10 talk about javahelper:
     `http://pkg-java.alioth.debian.org/docs/debconf10-javahelper-paper.pdf`
     `http://pkg-java.alioth.debian.org/docs/debconf10-javahelper-slides.pdf`
2. Download IRClib from `http://moepii.sourceforge.net/`
3. Package it

Step by step...

- `apt-get install javahelper`
- Create a basic source package: `jh_makepkg`
  - Library
  - None
  - Default Free compiler/runtime
- Look at and fix `debian/*`
- `dpkg-buildpackage -us -uc` or `debuild`
- `lintian`, `debc`, etc.
- Compare your result with the `libirclib-java` source package

Practical session 4: packaging a Ruby gem

1. Take a quick look at some documentation about Ruby packaging:
   - `http://wiki.debian.org/Ruby`
   - `http://wiki.debian.org/Teams/Ruby`
   - `http://wiki.debian.org/Teams/Ruby/Packaging`
   - `gem2deb(1)`, `dh_ruby(1)` (in the `gem2deb` package)
2. Create a basic Debian source package from the `net-ssh` gem: `gem2deb net-ssh`
3. Improve it so that it becomes a proper Debian package

Step by step. . .

`gem2deb net-ssh`:

- Downloads the gem from rubygems.org
- Creates a suitable .orig.tar.gz archive, and untar it
- Initializes a Debian source package based on the gem's metadata
  - Named `ruby-`*`gemname`*
- Tries to build the Debian binary package (this might fail)

`dh_ruby` (included in *gem2deb*) does the Ruby-specific tasks:

- Build C extensions for each Ruby version
- Copy files to their destination directory
- Update shebangs in executable scripts
- Run tests defined in `debian/ruby-tests.rb` or `debian/ruby-test-files.yaml`, as well as various other checks

Step by step. . . (2)

Improve the generated package:

- Run `debclean` to clean the source tree. Look at `debian/`.
- `changelog` and `compat` should be correct
- Edit `debian/control`: uncomment `Homepage`, improve `Description`
- Write a proper `copyright` file based on the upstream files
- `ruby-net-ssh.docs`: install `README.rdoc`
- `ruby-tests.rb`: run the tests. In that case, it is enough to do:

```
$: << 'test' << 'lib' << '.'
require 'test/test_all.rb'
```

Step by step. . . (3)

Build the package. It fails to build. There are two problems:

- You need to disable the `gem` call in the test suite.
  In `test/common.rb`, remove the `gem "test-unit"` line:
  - `edit-patch disable-gem.patch`
  - Edit `test/common.rb`, remove the `gem` line. Exit the sub-shell
  - Describe the changes in `debian/changelog`
  - Document the patch in `debian/patches/disable-gem.patch`
- The package lacks a build-dependency on `ruby-mocha`, which is used by the test suite (you might need to build your package in a clean environment, using `pbuilder`, to reproduce that problem)
  - Add `ruby-mocha` to the package's `Build-Depends`
  - *gem2deb* copies the dependencies documented in the *gem* as comments in `debian/control`, but *mocha* is not listed as a development dependency by the gem (that's a bug in the gem)

Compare your package with the `ruby-net-ssh` package in the Debian archive

Outline

1. FOSS concepts
2. Debian overview
3. Philosophy
4. Organization
5. Processes
6. Derivatives
7. Appendix: packaging tutorial
8. Appendix: contribute

Contributing — donate to Debian

even if completely volunteer-driven, Debian uses resources

- hardware for essential services
  - archive, buildds, devel. machines, ...
- money for hw-related services
  - guarantees, shipments, hosting, ...
- money to sponsor developer meetings
  - strengthen the community
  - get work done

Donations

- donations: `http://www.debian.org/donations`
- partners program: `http://www.debian.org/partners`

Contributing — work with Debian

- test, report, triage, fix bugs
  - `reportbug` on your Debian
  - `http://bugs.debian.org`
- translation (e.g.: in Italian)
  - `http://wiki.debian.org/it/DebianWiki`
  - `http://wiki.debian.org/L10n/Italian`
  - `http://lists.debian.org/debian-l10n-italian/`
- documentation
- help with packaging

`http://wiki.debian.org/HelpDebian`

Contributing — join Debian

choose your commitment:

package maintainer maintain packages, via sponsoring

Debian Maintainer (DM) upload your own packages

- advocacies required

Debian Project Member (DD) become a Debian "citizen"

- `http://nm.debian.org`
- upload access to all the archive *for packagers*
- voting rights
- all kinds of contributions are equally welcome!

Zack's tips for wannabe Debianers

1. choose a team: `http://wiki.debian.org/Teams`
2. stay on their mailing list and IRC channel
3. triage bugs, test patches, etc. *for packagers*
4. ... the rest will come!

Want to know more?

- web starting points:
  - `http://www.debian.org`
  - `http://wiki.debian.org`
- mailing lists: `http://lists.debian.org`
- IRC: `#debian-*` channels on `irc.debian.org`
- ask me!

Thanks!

Questions?

Stefano Zacchiroli
zack@pps.univ-paris-diderot.fr

http://upsilon.cc/zack
http://identi.ca/zack