

Debian Packaging Tutorial

Lucas Nussbaum
`lucas@debian.org`

version 0.6 – 2012-04-09



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 derivative distributions
 - ▶ That includes Ubuntu



Outline

- 1 Introduction
- 2 Creating source packages
- 3 Building and testing packages
- 4 Practical session 1: modifying the grep package
- 5 Advanced packaging topics
- 6 Maintaining packages in Debian
- 7 Conclusion
- 8 Practical session 2: packaging GNUjump
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem
- 11 Answers to practical sessions



Outline

- 1 Introduction
- 2 Creating source packages
- 3 Building and testing packages
- 4 Practical session 1: modifying the grep package
- 5 Advanced packaging topics
- 6 Maintaining packages in Debian
- 7 Conclusion
- 8 Practical session 2: packaging GNUjump
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem
- 11 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 package formats (with RPM)
- ▶ Universal:
 - ▶ 30,000 binary packages in Debian
→ 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 derivative 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



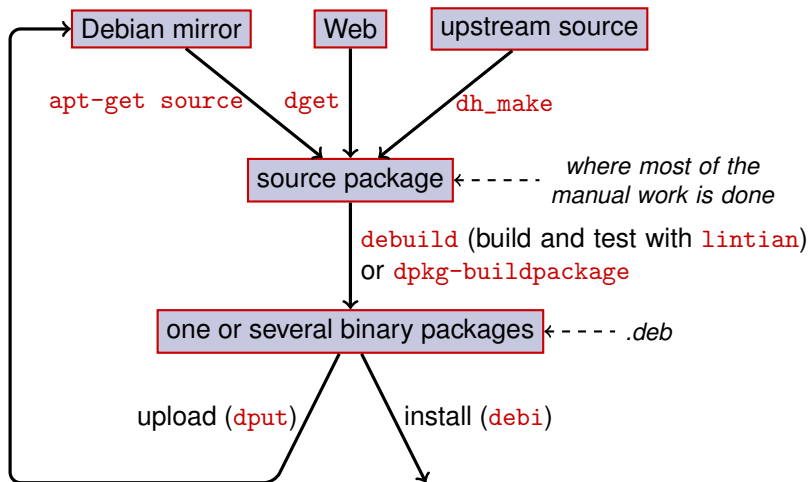
Tools you will need

- ▶ 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**, **cdb**s, **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 fakeroot
```
- 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



Outline

- 1 Introduction
- 2 **Creating source packages**
- 3 Building and testing packages
- 4 Practical session 1: modifying the grep package
- 5 Advanced packaging topics
- 6 Maintaining packages in Debian
- 7 Conclusion
- 8 Practical session 2: packaging GNUjump
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem
- 11 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 Kothé <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

1.2.1.1-5
Upstream Debian
version 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/packageName
    $(MAKE) DESTDIR=$(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/cdb/`
 - ▶ 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/cdb/1/rules/debhelper.mk
include /usr/share/cdb/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 $@

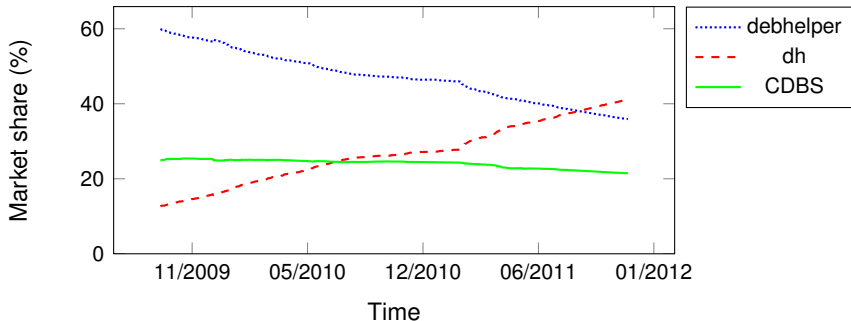
override_dh_auto_configure:
    dh_auto_configure -- --with-kitchen-sink

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)



Outline

- 1 Introduction
- 2 Creating source packages
- 3 Building and testing packages**
- 4 Practical session 1: modifying the grep package
- 5 Advanced packaging topics
- 6 Maintaining packages in Debian
- 7 Conclusion
- 8 Practical session 2: packaging GNUjump
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem
- 11 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 `sbuid`: 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/>



Outline

- 1 Introduction
- 2 Creating source packages
- 3 Building and testing packages
- 4 Practical session 1: modifying the grep package**
- 5 Advanced packaging topics
- 6 Maintaining packages in Debian
- 7 Conclusion
- 8 Practical session 2: packaging GNUjump
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem
- 11 Answers to practical sessions



Practical session 1: modifying the grep package

- ➊ 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`
- ➋ Look at the files in `debian/`.
 - ▶ How many binary packages are generated by this source package?
 - ▶ Which packaging helper does this package use?
- ➌ Build the package
- ➍ We are now going to modify the package. Add a changelog entry and increase the version number.
- ➎ Now disable perl-regexp support (it is a `./configure` option)
- ➏ Rebuild the package
- ➐ Compare the original and the new package with `debdiff`
- ➑ Install the newly built package
- ➒ Cry if you messed up ;)



Outline

- 1 Introduction
- 2 Creating source packages
- 3 Building and testing packages
- 4 Practical session 1: modifying the grep package
- 5 Advanced packaging topics**
- 6 Maintaining packages in Debian
- 7 Conclusion
- 8 Practical session 2: packaging GNUjump
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem
- 11 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* (*cdb*s), *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.alieth.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://bazaar.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/>



Outline

- 1 Introduction
- 2 Creating source packages
- 3 Building and testing packages
- 4 Practical session 1: modifying the grep package
- 5 Advanced packaging topics
- 6 Maintaining packages in Debian**
- 7 Conclusion
- 8 Practical session 2: packaging GNUjump
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem
- 11 Answers to practical sessions



Several ways to contribute to Debian

▶ **Worst** way to contribute:

- ❶ Package your own application
- ❷ Get it into Debian
- ❸ 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:
 - ▶ **Orphaned**: the package is unmaintained
Feel free to adopt it
 - ▶ **RFA: Request For Adopter**
Maintainer looking for adopter, but continues work in the meantime
Feel free to adopt it. A mail to the current maintainer is polite
 - ▶ **ITA: Intent To Adopt**
Someone intends to adopt the package
You could propose your help!
 - ▶ **RFH: Request For Help**
The maintainer is looking for help
- ▶ Some unmaintained packages not detected → 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
 - ➊ Prepare a source package
 - ➋ 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



Outline

- 1 Introduction
- 2 Creating source packages
- 3 Building and testing packages
- 4 Practical session 1: modifying the grep package
- 5 Advanced packaging topics
- 6 Maintaining packages in Debian
- 7 Conclusion**
- 8 Practical session 2: packaging GNUjump
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem
- 11 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

Feedback: **lucas@debian.org**



Legal stuff

Copyright ©2011 Lucas Nussbaum – lucas@debian.org

This document is free software: you can redistribute it and/or modify it under either (at your option):

- ▶ The terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
<http://www.gnu.org/licenses/gpl.html>
- ▶ The terms of the Creative Commons Attribution-ShareAlike 3.0 Unported License.
<http://creativecommons.org/licenses/by-sa/3.0/>



Latest version & source code

- ▶ Latest version:

`http://git.debian.org/?p=collab-maint/packaging-tutorial.git;a=blob_plain;f=packaging-tutorial.pdf;hb=refs/heads/pdf`

- ▶ Contribute:

- ▶ `git clone`

- `git://git.debian.org/collab-maint/packaging-tutorial.git`

- ▶ `apt-get source packaging-tutorial`

- ▶ `http://git.debian.org/?p=collab-maint/packaging-tutorial.git`

- ▶ Feedback: `lucas@debian.org`



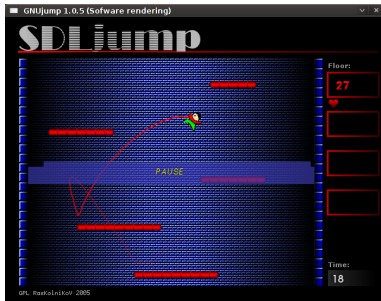
Outline

- 1 Introduction
- 2 Creating source packages
- 3 Building and testing packages
- 4 Practical session 1: modifying the grep package
- 5 Advanced packaging topics
- 6 Maintaining packages in Debian
- 7 Conclusion
- 8 Practical session 2: packaging GNUjump**
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem
- 11 Answers to practical sessions



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



Outline

- 1 Introduction
- 2 Creating source packages
- 3 Building and testing packages
- 4 Practical session 1: modifying the grep package
- 5 Advanced packaging topics
- 6 Maintaining packages in Debian
- 7 Conclusion
- 8 Practical session 2: packaging GNUjump
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem
- 11 Answers to practical sessions



Practical session 3: packaging a Java library

❶ 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>

❷ Download IRCLib from <http://moepii.sourceforge.net/>

❸ Package it



Outline

- 1 Introduction
- 2 Creating source packages
- 3 Building and testing packages
- 4 Practical session 1: modifying the grep package
- 5 Advanced packaging topics
- 6 Maintaining packages in Debian
- 7 Conclusion
- 8 Practical session 2: packaging GNUjump
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem**
- 11 Answers to practical sessions



Practical session 4: packaging a Ruby gem

- ❶ 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)
- ❷ Create a basic Debian source package from the `net-ssh` gem:
`gem2deb net-ssh`
- ❸ Improve it so that it becomes a proper Debian package



Outline

- 1 Introduction
- 2 Creating source packages
- 3 Building and testing packages
- 4 Practical session 1: modifying the grep package
- 5 Advanced packaging topics
- 6 Maintaining packages in Debian
- 7 Conclusion
- 8 Practical session 2: packaging GNUjump
- 9 Practical session 3: packaging a Java library
- 10 Practical session 4: packaging a Ruby gem
- 11 Answers to practical sessions



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

- ➊ 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 ;)

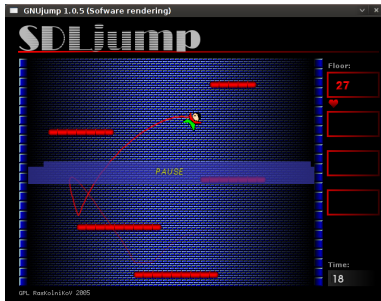
Or not: reinstall the previous version of the package:

- ▶ `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
emacs-en-install.ex manpage.xml.ex      source
emacs-en-remove.ex menu.ex             watch.ex
emacs-en-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 `debnc` 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 (→ 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

❶ 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>

❷ Download IRCLib from <http://moepii.sourceforge.net/>

❸ 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 `libirc-lib-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

