Debian-Paketier-Anleitung

Lucas Nussbaum
packaging-tutorial@packages.debian.org

version 0.29 – 2021-11-03

Über diese Anleitung

- Ziel: **Ihnen mitzuteilen, was Sie wirklich über das Paketieren für Debian wissen müssen**
  - Bestehende Pakete verändern
  - Eigene Pakete erstellen
  - Mit der Debian-Gemeinschaft arbeiten
  - Werden Sie ein versierter Debian-Benutzer
- Die wichtigsten Punkte werden abgedeckt, es ist aber nicht vollständig
  - Sie werden weitere Dokumentation lesen müssen
- Die meisten Inhalte passen auch auf von Debian abgeleitete Distributionen
  - Dazu gehört Ubuntu

Gliederung

Gliederung

Debian

- **GNU/Linux-Distribution**
- 1. größere Distribution, die »offen im Geiste von GNU« entwickelt wurde
- **Nicht kommerziell**, gemeinsam von über 1000 Freiwilligen gebaut
- 3 Hauptfunktionalitäten:
  - **Qualität** – Kultur der technischen Exzellenz
    Wir veröffentlichen, wenn es fertig ist
  - **Freiheit** – Entwickler und Benutzer sind durch den *Gesellschaftsvertrag* gebunden
    Fördern der Kultur der Freien Software seit 1993
  - **Unabhängigkeit** – keine (einzelne) Firma beaufsichtigt Debian
    Und offener Entscheidungsfindungsprozess (*do-ocracy* + *Demokratie*)
- **Amateur** im besten Sinne: Mit Liebe erstellt

Debian-Pakete

- **.deb**-Dateien (Binärpakete)
- Ein sehr mächtiger und bequemer Weg, Software an Benutzer zu verteilen
- Eines der beiden häufigsten Paketformate (mit RPM)
- Universell:
  - 30.000 Binärpakete in Debian
    $\rightarrow$ die meiste verfügbare freie Software ist für Debian paketiert!
  - Für 12 Portierungen (Architekturen), darunter 2 neben Linux (Hurd; KFreeBSD)
  - Wird auch von 120 von Debian abgeleiteten Distributionen verwandt

Das Deb-Paketformat

- `.deb`-Dateien: ein `ar`-Archiv

```
$ 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 des deb-Dateiformates, `"2.0\n"`
  - `control.tar.gz`: Metadaten über das Paket
    `control, md5sums, (pre|post)(rm|inst), triggers, shlibs,...`
  - `data.tar.gz`: Datendateien des Pakets

- Sie könnten Ihre `.deb`-Dateien manuell erstellen
  `http://tldp.org/HOWTO/html_single/Debian-Binary-Package-Building-HOWTO/`

- Die meisten Leute machen das aber nicht so

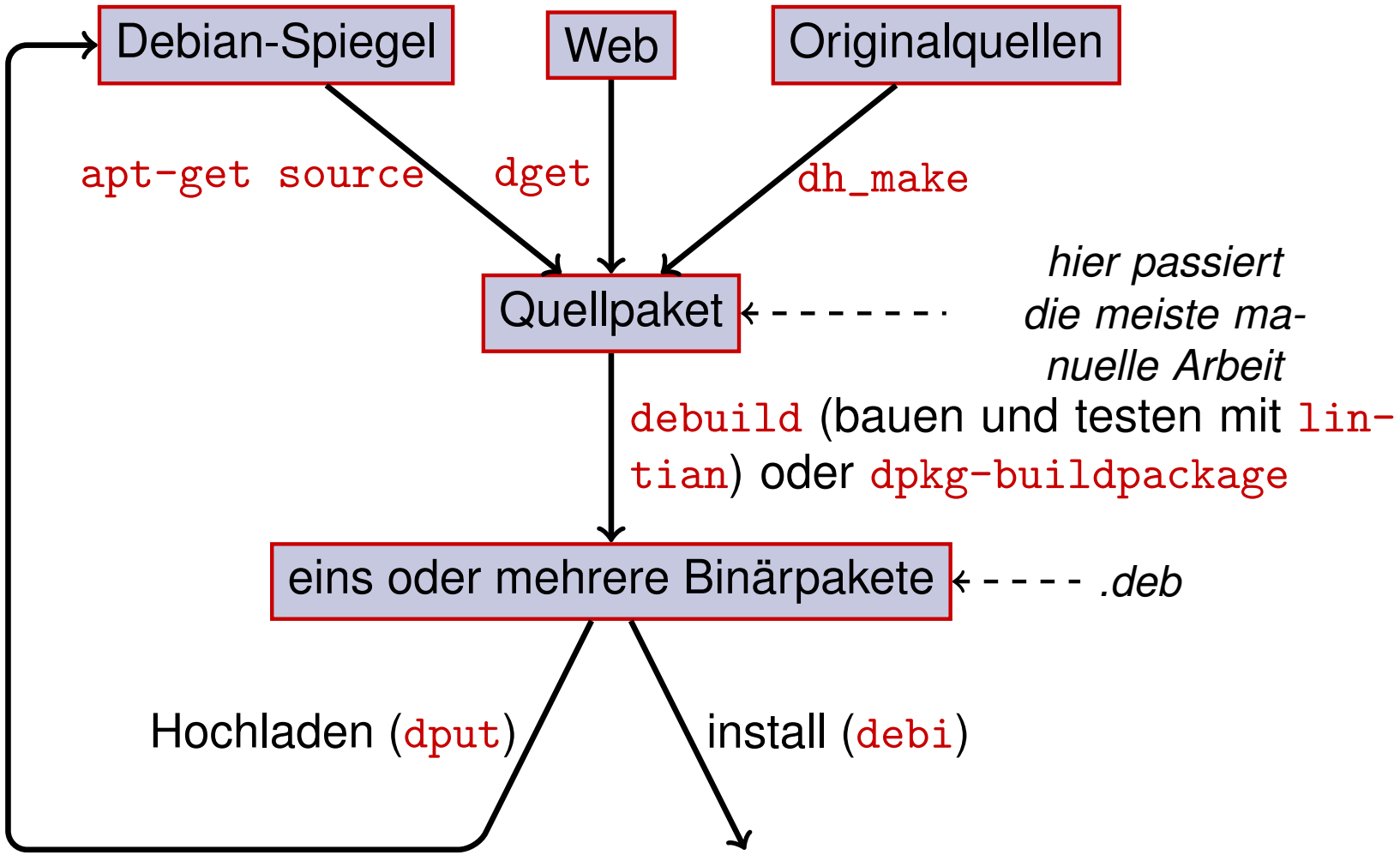
Diese Anleitung: Erstellen von Debian-Paketen, auf die Debian-Art

Folgende Werkzeuge benötigen Sie

- Ein Debian- (oder Ubuntu-)System (mit root-Zugang)
- Einige Pakete:
  - **build-essential**: hat Abhängigkeiten auf Pakete, von denen angenommen wird, dass sie auf der Maschine eines Entwicklers vorhanden sind (sie müssen nicht im Steuerfeld `Build-Depends:` Ihres Paketes aufgeführt werden)
    - enthält eine Abhängigkeit von **dpkg-dev**, das einige grundlegende Debian-spezifische Werkzeuge zum Erstellen von Paketen enthält
  - **devscripts**: Enthält viele nützliche Skripte für Debian-Betreuer

Viele weitere Werkzeuge werden später erwähnt, wie **debhelper**, **cdbs**, **quilt**, **pbuilder**, **sbuild**, **lintian**, **svn-buildpackage**, **git-buildpackage**, ...
Installieren Sie diese, wenn Sie sie benötigen.

Allgemeiner Paketierungsablauf



Beispiel: Dash neu bauen

1. Installieren Sie die zum Bau von Dash benötigten Pakete und Devscripts
   `sudo apt-get build-dep dash`
   (benötigt `deb-src`-Zeilen in `/etc/apt/sources.list`)
   `sudo apt-get install --no-install-recommends devscripts fakeroot`
2. Erstellen Sie ein Arbeitsverzeichnis und holen sie es:
   `mkdir /tmp/debian-tutorial ; cd /tmp/debian-tutorial`
3. Holen Sie das `dash`-Quellpaket
   `apt-get source dash`
   (Dies setzt voraus, dass Sie `deb-src`-Zeilen in Ihrer `/etc/apt/sources.list` haben)
4. Bauen Sie das Paket
   `cd dash-*`
   `debuild -us -uc` (`-us -uc` deaktiviert die Paketsignatur mit GPG)
5. Überprüfen Sie, dass es funktioniert hat
   - Im übergeordneten Verzeichnis sind einige neue `.deb`-Dateien
6. Schauen Sie auf das `debian/`-Verzeichnis
   - Hier passiert die Paketierungsarbeit

Gliederung

Quellpaket

- Ein Quellpaket kann mehrere Binärpakete erstellen
  z.B. erstellen die Quellen `libtar` die Binärpakete `libtar0` und `libtar-dev`.
- Zwei Arten von Paketen: (falls unsicher, verwenden Sie *nicht* native)
  - Native Pakete: Normalerweise für Debian-spezifische Software (*dpkg*, *apt*)
  - Nicht native Pakete: Software, die außerhalb von Debian entwickelt wird
- Hauptdatei: `.dsc` (Metadaten)
- Andere Dateien, abhängig von der Version des Quellformats
  - 1.0 oder 3.0 (nativ): `Paket_version.tar.gz`
  - 1.0 (nicht nativ):
    - `pkg_ver.orig.tar.gz`: Originalquellen
    - `Pkt_Debver.diff.gz`: Patch, um Debian-spezifische Änderungen hinzuzufügen
  - 3.0 (quilt):
    - `pkg_ver.orig.tar.gz`: Originalquellen
    - `pkg_debver.debian.tar.gz`: Tarball mit den Debian-Änderungen

(siehe `dpkg-source(1)` für exakte Details)

Quellpaketbeispiel (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
```

Ein existierendes Quellpaket holen

- Aus dem Debian-Archiv:
  - `apt-get source` *`Paket`*
  - `apt-get source` *`Paket=Version`*
  - `apt-get source` *`Paket/Veröffentlichung`*

  (Sie benötigen `deb-src`-Zeilen in der `sources.list`)

- Aus dem Internet:
  - `dget` *`url-zu.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 stellt alle Pakete aus Debian seit 2005 bereit)`

- Aus dem (angegebenen) Versionskontrollsystem:
  - `debcheckout` *`Paket`*

- Sobald es heruntergeladen ist, mittels `dpkg-source -x` *`Datei.dsc`* extrahieren

Ein einfaches Quellpaket erstellen

- Laden Sie die Originalquellen herunter
  (*Originalquellen* = die Quellen von den ursprünglichen Entwicklern der Software)
- Bennen Sie sie in `<`*`Quellpaket`*`>_<`*`Originalversion`*`>.orig.tar.gz` um
  (Beispiel: `simgrid_3.6.orig.tar.gz`)
- Entpacken Sie sie
- Bennen Sie das Verzeichnis in `<`*`Quellpaket`*`>-<`*`Originalversion`*`>` um
  (Beispiel: `simgrid-3.6`)
- `cd <`*`Quellpaket`*`>-<`*`Originalversion`*`> && dh_make`
  (aus dem Paket **dh-make**)
- Es gibt einige Alternativen zu `dh_make` für bestimmte Mengen von Paketen: **dh-make-perl**, **dh-make-php**, ...
- `debian/`-Verzeichnis wird erstellt, mit vielen Dateien darin

Dateien in debian/

Die gesamte Paketierungsarbeit sollte darin bestehen, Dateien unter `debian/` zu verändern

- Hauptdateien:
  - **control** – Metadaten über das Paket (Abhängigkeiten, usw.)
  - **rules** – gibt an, wie das Paket gebaut wird
  - **copyright** – Copyright-Informationen für das Paket
  - **changelog** – Änderungsverlauf des Debian-Pakets
- Andere Dateien
  - compat
  - watch
  - dh_install*-Ziele

    *.dirs, *.docs, *.manpages, . . .
  - Betreuer-Skripte

    *.postinst, *.prerm, . . .
  - source/format
  - patches/ – falls Sie die Originalquellen verändern müssen
- Verschiedene Dateien verwenden ein auf RFC 822 (E-Mail-Kopfzeilen) basierendes Format

debian/changelog

- Führt die Debian-Paketierungsänderungen auf
- Stellt die aktuelle Version des Pakets bereit

$$\underbrace{1.2.1.1}_{\text{Original-version}}\text{-}\underbrace{5}_{\text{Debian-Revision}}$$

- Manuell oder mit `dch` bearbeiten
  - Changelog-Eintrag für die neue Veröffentlichung erzeugen: `dch -i`
- Spezielles Format, um automatisch Debian- oder Ubuntu-Fehler zu schließen:
  Debian: `Closes: #595268`; Ubuntu: `LP: #616929`
- Installiert als `/usr/share/doc/`*`Paket`*`/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

- Paketmetadaten
  - für das Quellpaket selbst
  - für jedes von diesen Quellen gebaute Binärpaket
- Paketname, Abschnitt, Priorität, Betreuer, Uploaders, Bauabhängigkeiten, Abhängigkeiten, Beschreibung, Homepage, ...
- Dokumentation: Debian-Richtlinien Kapitel 5
  `https://www.debian.org/doc/debian-policy/ch-controlfields`

```
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 oder any

Es gibt zwei Arten von Binärpaketen:

- Pakete, mit Inhalten, die für jede Architektur anders sind
  - Beispiel: C-Programm
  - `Architecture: any` in `debian/control`
    - Oder, falls es nur auf gewissen Architekturen funktioniert
      `Architecture: amd64 i386 ia64 hurd-i386`
  - buildd.debian.org: Baut alle anderen Architekturen für Sie nach einem Upload
  - Benannt *Paket*`_`*Version*`_`*Architektur*`.deb`
- Pakete mit den gleichen Inhalten auf allen Architekturen
  - Beispiel: Perl-Bibliothek
  - `Architecture: all` in `debian/control`
  - Benannt *Paket*`_`*Version*`_all.deb`

Ein Quellpaket kann eine Mischung aus `Architecture: any`- und `Architecture: all`-Binärpaketen erstellen

debian/rules

- Makefile
- Schnittstelle zum Bau von Debian-Paketen
- Dokumentiert in den Debian-Richtlinien, Kapitel 4.8
  `https://www.debian.org/doc/debian-policy/ch-source#s-debianrules`
- Benötigte Ziele:
  - `build, build-arch, build-indep`: Sollte die gesamte Konfiguration und Übersetzung durchführen
  - `binary, binary-arch, binary-indep`: baut das Binärpaket
    - `dpkg-buildpackage` wird `binary` aufrufen, um alle Pakete zu bauen oder `binary-arch`, um nur die `Architecture: any`-Pakete zu bauen
  - `clean`: bereinigt das Quellverzeichnis

Paketierungshelfer – Debhelper

- Sie könnten in `debian/rules` direkt Shell-Code schreiben
- Besseres Vorgehen (wird von den meisten Paketen verwandt): verwenden Sie einen *Paketierungshelfer*
- Beliebtester: **Debhelper** (von 98% der Pakete verwandt)
- Ziele:
  - Die häufigen Aufgaben in Standardwerkzeuge, die von allen Paketen verwandt werden, zusammenfassen
  - Einige Paketierungsfehler einmal für alle Pakete beheben

  dh_installdirs, dh_installchangelogs, dh_installdocs, 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, ...
  - Aus `debian/rules` heraus aufgerufen
  - Mittels Parametern oder Dateien in `debian/` konfigurierbar

  Paket`.docs`, *Paket*`.examples`, *Paket*`.install`, *Paket*`.manpages`, ...
- Hilfsprogramme Dritter für Gruppen von Paketen: **python-support**, **dh_ocaml**, ...
- `debian/compat`: Debhelper-Kompatibilitätsversion
  - Definiert das genaue Verhalten von dh_*
  - Neue Syntax: `Build-Depends: debhelper-compat (= 13)`

debian/rules mittels debhelper (1/2)

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

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

build:
        $(MAKE)
        #docbook-to-man debian/Paketename.sgml > Paketname.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 mittels 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

- Mit Debhelper, immer noch eine Menge an Redundanz zwischen Paketen
- Nachrangige Hilfsprogramme, die gemeinsam genutzte Funktionalität aufnehmen
  - Z.B. Bauen mit `./configure && make && make install` oder CMake
- CDBS:
  - 2005 eingeführt, basierend auf fortgeschrittener *GNU make*-Magie
  - Dokumentation: `/usr/share/doc/cdbs/`
  - Unterstützung für Perl, Python, Ruby, GNOME, KDE, Java, Haskell, ...
  - Aber manche Leute hassen es:
    - Manchmal schwer, Paketbau anzupassen: "*Verzwicktes Labyrinth von Makefiles und Umgebungsvariablen*"
    - Langsamer als einfacher Debhelper (viele unnütze Aufrufe von `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
```

Dh (lang Debhelper 7 oder dh7)

- Eingeführt in 2008 als ein *CDBS-Mörder*
- **dh**-Befehl, der `dh_*` aufruft
- Einfache *debian/rules*, enthält nur Aufhebungen
- Einfacher als CDBS anzupassen
- Dokumentation: Handbuchseiten (`debhelper(7)`, `dh(1)`) + Folien vom DebConf9-Vortrag
  `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
```

Klassischer Debhelper vs. CDBS vs. dh

- Marktanteil:
  Klassischer Debhelper: 15% CDBS: 15% Dh: 68%
- Welchen soll ich lernen?
  - Wahrscheinlich ein bisschen von allen
  - Sie müssen Debhelper kennen, um Dh und CDBS zu benutzen
  - Es könnte sein, dass Sie CDBS-Pakete ändern müssen
- Welches sollte ich für ein neues Paket verwenden?
  - **dh** (einzige Lösung mit zunehmenden Marktanteil)
  - Siehe `https://trends.debian.net/#build-systems`

Gliederung

Pakete bauen

- `apt-get build-dep MeinPaket`
  Installiert die *Bauabhängigkeiten* (für ein Paket, das bereits im Debian-Archiv ist)
  Oder `mk-build-deps -ir` (für ein noch nicht hochgeladenes Paket)

- `debuild`: bauen, testen mit `lintian`, unterschreiben mit GPG

- Es ist auch möglich, `dpkg-buildpackage` direkt aufzurufen
  - Normalerweise mittels `dpkg-buildpackage -us -uc`

- Besser: Pakete in einer sauberen und minimalen Umgebung bauen
  - `pbuilder` – Helfer, um Pakete in einer *Chroot* zu bauen
    Gute Dokumentation: `https://wiki.ubuntu.com/PbuilderHowto`
    (Optimierung: `cowbuilder ccache distcc`)
  - `schroot` und `sbuild`: von den Debian-Build-Daemons verwandt
    (nicht so einfach wie `pbuilder`, erlaubt aber LVM-Schnappschüsse
    siehe: `https://help.ubuntu.com/community/SbuildLVMHowto`)

- Erstellt `.deb`-Dateien und eine `.changes`-Datei
  - `.changes`: beschreibt, was gebaut wurde; beim Hochladen verwandt

Installieren und Testen von Paketen

- Installieren Sie das Paket lokal: `debi` (wird `.changes` verwenden, um zu wissen, was installiert werden soll)
- Zeigen Sie den Inhalt des Pakets: `debc ../meinPaket<TAB>.changes`
- Vergleichen Sie das Paket mit der vorherigen Version:
  `debdiff ../meinPaket_1_*.changes ../meinPaket_2_*.changes`
  oder vergleichen Sie die Quellen:
  `debdiff ../meinPaket_1_*.dsc ../meinPaket_2_*.dsc`
- Überprüfen Sie das Paket mit `lintian` (statische Analyse):
  `lintian ../meinPaket<TAB>.changes`
  `lintian -i`: gibt weitere Informationen über die Fehler
  `lintian -EviIL +pedantic`: zeigt weitere Probleme
- Laden Sie das Paket nach Debian hoch (`dput`) (benötigt Konfiguration)
- Betreiben Sie ein privates Debian-Archiv mit `reprepro` oder `aptly`
  Dokumentation:
  `https://wiki.debian.org/HowToSetupADebianRepository`

Gliederung

Praktische Sitzung 1: Anpassen des Grep-Pakets

1. Laden Sie Version 2.12-2 des Pakets von `http://ftp.debian.org/debian/pool/main/g/grep/` herunter.
   - Falls das Quellpaket nicht automatisch entpackt wird, entpacken Sie es mit `dpkg-source -x grep_*.dsc`
2. Schauen Sie sich die Dateien in `debian/` an.
   - Wie viele Binärpakete werden aus diesem Quellpaket erstellt?
   - Welche Paketierungshelfer verwendet dieses Paket?
3. Bauen Sie das Paket
4. Wir werden das Paket jetzt anpassen. Fügen Sie einen Changelog-Eintrag hinzu und erhöhen Sie die Versionsnummer.
5. Deaktivieren Sie jetzt die Perl-Regexp-Unterstützung (dies ist eine `./configure`-Option)
6. Bauen Sie das Paket erneut
7. Vergleichen Sie das ursprüngliche und das neue Paket mit Debdiff
8. Installieren Sie das neu gebaute Paket

Gliederung

debian/copyright

- Urheberrecht- und Lizenzinformationen für diese Quellen und die Paketierung
- Traditionell als Textdatei geschrieben
- Neues, maschinenlesbares Format:
  `https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/`

```
Format: https://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 Max Mustermann <max.mustermann@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 Jana Meier <jmeierh@example.net>
License:
 [LIZENZTEXT]
```

Ändern der Originalquellen

Oft benötigt:

- Fehler beheben oder Debian-spezifische Anpassungen vornehmen
- Korrekturen aus einer neueren Veröffentlichung der Originalautoren rückportieren

Es gibt mehrere Methoden, dies durchzuführen:

- Die Dateien direkt anpassen
  - Einfach
  - Allerdings gibt es keine Möglichkeit, die Änderungen zu dokumentieren und nachzuvollziehen
- Verwendung von Patch-Systemen
  - Erleichtert die Weitergabe der Änderungen an die Originalautoren
  - Hilft beim gemeinsamen Nutzen der Korrekturen mit abgeleiteten Distributionen
  - Gibt den Änderungen mehr Aufmerksamkeit
    `http://patch-tracker.debian.org/` (derzeit nicht erreichbar)

Patch-Systeme

- Prinzip: Änderungen werden als Patches in `debian/patches/` gespeichert
- Sie werden während des Baus angewandt und entfernt
- Früher gab es mehrere Implementierungen – *simple-patchsys* (*cdbs*), *dpatch*, ***quilt***
  - Alle unterstützen zwei Ziele in `debian/rules`:
    - `debian/rules patch`: alle Patches anwenden
    - `debian/rules unpatch`: alle Patches entfernen
  - Weitere Dokumentation:
    `https://wiki.debian.org/debian/patches`
- **Neues Quellformat mit eingebautem Patch-System: 3.0 (quilt)**
  - Empfohlene Lösung
  - Sie müssen *quilt* lernen
    `https://perl-team.pages.debian.net/howto/quilt.html`
  - Patch-System-unabhängiges Werkzeug in `devscripts`: `edit-patch`

Dokumentation der Patches

- Standardkopfzeilen am Anfang des Patches
- Dokumentiert 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: Max Mustermann <mmustermann-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 {
```

Beim Installieren und Entfernen etwas machen

- Entpacken des Pakets ist manchmal nicht genug
- Benutzer erstellen/entfernen, Dienste starten/stoppen, *alternatives* verwalten
- Wird in *Betreuerskripten* erledigt

  `preinst, postinst, prerm, postrm`
  - Schnipsel für häufige Aktionen können durch Debhelper erstellt werden
- Dokumentation:
  - Debian-Richtlinien-Handbuch, Kapitel 6

    `https://www.debian.org/doc/debian-policy/ch-maintainerscripts`
  - Debian-Entwicklerreferenz, Kapitel 6.4

    `https://www.debian.org/doc/developers-reference/best-pkging-practices.html`
  - `https://people.debian.org/~srivasta/MaintainerScripts.html`
- Benutzer um Eingaben bitten:
  - Muss mit **debconf** erfolgen
  - Dokumentation: `debconf-devel(7)` (`debconf-doc`-Paket)

Version der Originalautoren überwachen

- Geben Sie in `debian/watch` (siehe `uscan(1)`) an, wo geschaut werden soll

```
version=3

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

- Es gibt automatische Nachverfolgungssysteme für neue Versionen der Originalautoren, die den Betreuer auf verschiedenen Armaturenbrettern, darunter `https://tracker.debian.org/` und `https://udd.debian.org/dmd/`, informieren
- `uscan`: eine manuelle Überprüfung durchführen
- `uupdate`: versucht, Ihr Paket auf den neusten Stand der Originalautoren zu aktualisieren

Mit einem Versionskontrollsystem paketieren

- Werkzeuge zur Verwaltung von Zweigen und Markierungen für Ihre Paketierungsarbeit: `svn-buildpackage`, `git-buildpackage`
- Beispiel: `git-buildpackage`
  - `upstream`-Zweig: die Arbeit der Originalautoren nachvollziehen mit `upstream/`*`Version`*-Markierungen
  - `master`-Zweig folgt dem Debian-Paket
  - `debian/`*`Version`*-Markierungen für jedes Hochladen
  - `pristine-tar`-Zweig, ermöglicht Neubau des Originalautoren-Tarballs

  Dokumentation: `http://honk.sigxcpu.org/projects/git-buildpackage/manual-html/gbp.html`
- `Vcs-*`-Felder in `debian/control`, um das Depot anzugeben
  - `https://wiki.debian.org/Salsa`

```
Vcs-Browser: https://salsa.debian.org/debian/devscripts
Vcs-Git: https://salsa.debian.org/debian/devscripts.git

Vcs-Browser: https://salsa.debian.org/perl-team/modules/packages/libwww-perl
Vcs-Git: https://salsa.debian.org/perl-team/modules/packages/libwww-perl.git
```

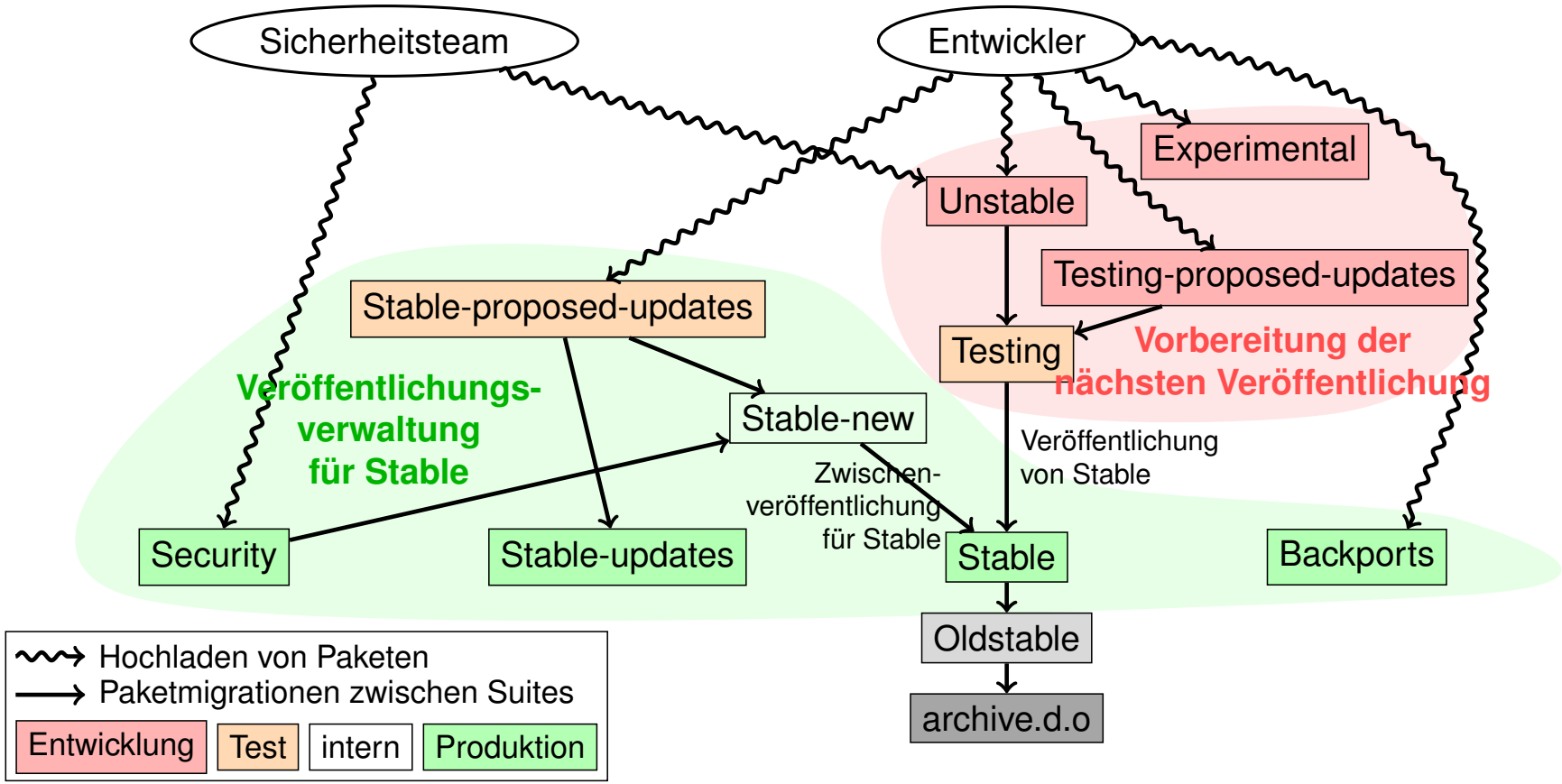
- VCS-unabhängige Schnittstelle: `debcheckout`, `debcommit`, `debrelease`
  - `debcheckout grep` → checkt das Quellpaket aus Git aus

Pakete rückportierern

- Ziel: Eine neuere Version eines Paketes auf einem älteren System verwenden
  z.B. *mutt* aus Debian-*Unstable* auf Debian-*Stable* verwenden
- Prinzipielle Idee:
  - Nehmen Sie das Quellpaket aus Debian Unstable
  - Passen Sie es an, so dass es auf Debian-Stable baut und gut funktioniert
    - Manchmal trivial (keine Änderungen notwendig)
    - Manchmal schwierig
    - Manchmal unmöglich (viele nicht verfügbare Abhängigkeiten)
- Einige Rückportierungen werden von Debian bereitgestellt und unterstützt
  `http://backports.debian.org/`

Gliederung

Debian-Archiv und -Suites



Basierend auf einer Darstellung von Antoine Beaupré. `https://salsa.debian.org/debian/package-cycle`

Suites für die Entwicklung

- Neue Versionen von Paketen werden nach **Unstable** (**Sid**) hochgeladen
- Pakete migrieren basierend auf einigen Kriterien von **Unstable** nach **Testing** (z.B. dass sie seit 10 Tagen in Unstable waren und keine Regressionen auftraten)
- Neue Pakete können auch in folgende Suites hochgeladen werden:
  - **Experimental** (für *experimentellere* Pakete, wie neue Versionen, die noch nicht geeignet sind, die Version aus Unstable zu ersetzen)
  - **Testing-proposed-updates**, um die Version in **Testing** zu aktualisieren, ohne durch **Unstable** zu gehen (dies wird selten benutzt)

Einfrieren und Veröffentlichen

- Zu einem Zeitpunkt während des Veröffentlichungszyklus entscheidet das Veröffentlichungsteam, Testing *einzufrieren*; automatische Migrationen von **Unstable** nach **Testing** werden gestoppt und durch manuelle Begutachtung ersetzt
- Wenn das Veröffentlichungsteam **Testing** für die Veröffentlichung als bereit betrachtet:
  - Die Suite **Testing** wird die neue Suite **Stable**
  - Ähnlich wird **Stable Oldstable**
  - Nicht mehr unterstützte Veröffentlichungen werden nach `archive.debian.org` verschoben
- Siehe `https://release.debian.org/`

Verwaltung stabiler Suites und Veröffentlichungen

- Mehrere Suites stellen Pakete der stabilen Veröffentlichungen bereit
  - **Stable**: die Haupt-Suite
  - **Security**: Aktualisierungs-Suite, auf `security.debian.org` bereitgestellt, verwandt vom Sicherheitsteam. Aktualisierungen werden auf der Mailingliste `debian-security-announce` bekanntgegeben
  - **Stable-updates**: Aktualisierungen, die nicht sicherheitsbezogen sind, aber die dringend installiert werden sollten (ohne auf die nächste Zwischenveröffentlichung zu warten): Antivierendatenbanken, Zeitzonen-bezogene Pakete usw. Ankündigungen auf der Mailingliste `debian-stable-announce`
  - **Backports**: neue Versionen der Originalautoren, basierend auf der Version in **Testing**
- Die Suite **Stable** wird alle paar Monate durch *Zwischenveröffentlichungen für Stable* (die nur Fehlerkorrekturen enthält) aktualisiert
  - Pakete, die für die nächste Zwischenveröffentlichung gedacht sind, werden nach **Stable-proposed-updates** hochgeladen und durch das

Es gibt viele Möglichkeiten, zu Debian beizutragen

- **Schlechteste** Art, beizutragen:
  1. Paketieren Sie Ihre eigene Anwendung
  2. Schaffen Sie diese nach Debian
  3. Verschwinden Sie

- **Bessere** Art, beizutragen:
  - Machen Sie bei einem Paketier-Team mit
    - Viele Teams konzentrieren sich auf eine Gruppe von Paketen und benötigen Hilfe
    - Liste verfügbar unter `https://wiki.debian.org/Teams`
    - Dies ist eine exzellente Art, um von erfahreneren Beitragenden zu lernen
  - Adoptieren Sie existierende, nicht betreute Pakete (*verwaiste Pakete*)
  - Bringen Sie neue Software in Debian
    - Bitte nur, falls diese interessant / nützlich genug ist
    - Sind die Alternativen bereits für Debian paketiert?

Verwaiste Pakete adoptieren

- Es gibt viele nicht betreute Pakete in Debian
- Komplette Liste und Prozess: `https://www.debian.org/devel/wnpp/`
- Installiert auf Ihrer Maschine: `wnpp-alert`
  Oder besser: `how-can-i-help`
- Verschiedene Zustände:
  - **O**rphaned: das Paket ist verwaist (es wird nicht mehr betreut)
    Adoptieren Sie es ruhig
  - **RFA**: **R**equest **F**or **A**dopter
    Der Betreuer sucht nach einem Adoptierer, arbeitet aber zwischenzeitlich weiter dran
    Adoptieren Sie es einfach. Eine E-Mail an den aktuellen Betreuer wäre nett
  - **ITA**: **I**ntent **T**o **A**dopt
    Jemand plant, das Pakete zu adoptieren – Sie könnten Hilfe anbieten!
  - **RFH**: **R**equest **F**or **H**elp
    Der Paketbetreuer sucht Hilfe
- Einige nicht betreute Pakete werden nicht erkannt → noch nicht verwaist
- Im Zweifelsfall fragen Sie auf `debian-qa@lists.debian.org`

Ein Paket adoptieren: Beispiel

```
Von: Sie <Sie@IhreDomain>
An: 640454@bugs.debian.org, control@bugs.debian.org
Cc: Francois Marier <francois@debian.org>
Betreff: 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,

Sie
```

- Es ist höflich, den vorhergehenden Betreuer zu kontaktieren (insbesondere wenn das Paket RFAt und nicht verwaist war)
- Es ist eine sehr gute Idee, die Originalautoren zu kontaktieren

Schaffen Sie Ihr Paket nach Debian

- Sie benötigen keinen offiziellen Status, um Ihr Paket in Debian zu bekommen
  1. Reichen Sie mit `reportbug wnpp` einen **ITP**-Fehler (**I**ntent **T**o **P**ackage) ein
  2. Bereiten Sie ein Quellpaket vor
  3. Finden Sie einen Debian-Entwickler, der Ihr Paket sponsern wird
- Offizieller Status (wenn Sie ein erfahrener Paketbetreuer sind):
  - **Debian-Betreuer (DM):**
    Recht, Ihre eigenen Pakete hochzuladen
    Siehe `https://wiki.debian.org/DebianMaintainer`
  - **Debian-Entwickler (DD):**
    Debian-Projektmitglied; darf abstimmen und jedes Paket hochladen

Zu Prüfendes, bevor Sie nach Sponsoren fragen:

- Debian legt **viel Wert auf Qualität**
- Typischerweise sind **Sponsoren schwer zu finden und beschäftigt**
  - Stellen Sie sicher, dass Ihr Paket bereit ist, bevor Sie Sponsoren fragen
- Dinge, die geprüft werden sollten:
  - Vermeiden Sie fehlende build-dependencies: Stellen Sie sicher, dass Ihr Paket problemlos in einer sauberen *sid chroot* baut
    - Verwendung von `pbuilder` wird empfohlen
  - Führen Sie für Ihr Paket `lintian -EviIL +pedantic` aus
    - Fehler müssen, alle anderen Probleme sollten behoben werden
  - Testen Sie natürlich Ihr Paket ausführlich
- Im Zweifelsfall fragen Sie nach Hilfe

Wo können Sie Hilfe finden?

Folgende Hilfe benötigen Sie:

- Ratschläge und Antworten auf Ihre Fragen, Code-Begutachtungen
- Unterstützung für Ihr Paket, sobald Ihr Paket fertig ist

Sie können Hilfe bekommen von:

- **Anderen Mitgliedern eines Paketierungsteams**
  - Liste von Teams: `https://wiki.debian.org/Teams`
- Der **Debian-Mentors-Gruppe** (falls Ihr Paket in kein Team passt)
  - `https://wiki.debian.org/DebianMentorsFaq`
  - Mailingliste: `debian-mentors@lists.debian.org`
    (auch eine gute Art, nebenbei was zu lernen)
  - IRC: `#debian-mentors` auf `irc.debian.org`
  - `http://mentors.debian.net/`
  - Dokumentation: `http://mentors.debian.net/intro-maintainers`
- **Lokalisierte Mailinglisten** (Hilfe in Ihrer Sprache erhalten)
  - `debian-devel-{french,italian,portuguese,spanish}@lists.d.o`
  - Komplette Liste: `https://lists.debian.org/devel.html`
  - Unsere Benutzerlisten: `https://lists.debian.org/users.html`

Weitere Dokumentation

- Debians Entwickler-Ecke
  `https://www.debian.org/devel/`
  Links auf viele Ressourcen über Debian-Entwicklung
- Leitfaden für Debian-Betreuer
  `https://www.debian.org/doc/manuals/debmake-doc/`
- Debian-Entwicklerreferenz
  `https://www.debian.org/doc/developers-reference/`
  Hauptsächlich über Debian-Prozeduren, aber auch einige goldene Regeln der Paketierung (Teil 6)
- Debian-Richtlinien
  `https://www.debian.org/doc/debian-policy/`
  - Alle Anforderungen, die jedes Paket erfüllen muss
  - Spezielle Richtlinen für Perl, Java, Python, ...
- Ubuntu-Paketierungsleitfaden
  `http://developer.ubuntu.com/resources/tools/packaging/`

Debian-Armaturenbrett für Betreuer

- **Quellpaket zentriert**:
  `https://tracker.debian.org/dpkg`
- **Betreuer/Team zentriert**: Paketüberblick für Entwickler (DDPO)
  `https://qa.debian.org/developer.php?login=pkg-ruby-extras-maintainers@lists.alioth.debian.org`
- **TODO-Listen-orientiert**: Debian Maintainer Dashboard (DMD)
  `https://udd.debian.org/dmd/`

Die Fehlerdatenbank (BTS) benutzen

- Eine recht einzigartige Art, mit Fehlern umzugehen
  - Web-Oberfläche zum Betrachten von Fehlern
  - E-Mail-Schnittstelle, um Fehler zu verändern
- Informationen zu Fehlern hinzufügen:
  - Schreiben Sie an `123456@bugs.debian.org` (geht nicht an Einreichenden, Sie müssen `123456-submitter@bugs.debian.org` hinzufügen)
- Fehlerstatus ändern:
  - Schicken Sie Befehle an `control@bugs.debian.org`
  - Befehlszeilen-Schnittstelle: `bts`-Befehl in `devscripts`
  - Dokumentation: `https://www.debian.org/Bugs/server-control`
- Fehler berichten: verwenden Sie `reportbug`
  - Normalerweise mit lokalem E-Mail-Server verwandt: installieren Sie `ssmtp` oder `nullmailer`
  - Alternativ verwenden Sie `reportbug --template`, schicken Sie dann (manuell) an `submit@bugs.debian.org`

Das BTS verwenden: Beispiele

- Eine E-Mail an den Fehler und den Einreichenden senden:
  `https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=680822#10`
- Markieren und Schweregrad ändern:
  `https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=680227#10`
- Neuzuweisen, Schweregrad ändern, umbenennen ...:
  `https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=680822#93`
  - `notfound`, `found`, `notfixed`, `fixed` sind für **Version-Nachverfolgung**
    Siehe `https://wiki.debian.org/HowtoUseBTS#Version_tracking`
- Usertags verwenden: `https://bugs.debian.org/cgi-bin/bugreport.cgi?msg=42;bug=642267`
  Siehe `https://wiki.debian.org/bugs.debian.org/usertags`
- BTS-Dokumentation:
  - `https://www.debian.org/Bugs/`
  - `https://wiki.debian.org/HowtoUseBTS`

Mehr an Ubuntu interessiert?

- Ubuntu verwaltet hauptsächlich die Abweichungen von Debian
- Kein echter Fokus auf spezielle Pakete
  stattdessen Kollaboration mit Debian-Teams
- Normalerweise wird empfohlen, neue Pakete zuerst nach Debian hochzuladen
  `https://wiki.ubuntu.com/UbuntuDevelopment/NewPackages`
- Eventuell ein besserer Plan:
  - Machen Sie bei einem Debian-Team mit und agieren Sie als Brücke zu Ubuntu
  - Helfen Sie bei der Reduktion der Unterschiede, sichten Sie Fehler in Launchpad
  - Viele Debian-Werkzeuge können helfen:
    - Ubuntu-Spalte auf dem Entwickler-Paketüberblick
    - Ubuntu-Kasten in der Paketdatenbank
    - Erhalten Sie Launchpad-Fehler-E-Mails über das PTS

Gliederung

Fazit

- Sie haben jetzt einen kompletten Überblick über die Debian-Paketierung
- Sie werden aber weitere Dokumentation lesen müssen
- Goldene Regeln entwickelten sich im Laufe der Jahre
  - Falls Sie sich unsicher sind, verwenden Sie die **dh**-Paketierungshelfer und das Format **3.0 (quilt)**

Rückmeldungen: **packaging-tutorial@packages.debian.org**

Rechtliches Zeug

Zur Anleitung beitragen

- Beitragen:
  - `apt-get source packaging-tutorial`
  - `debcheckout packaging-tutorial`
  - `git clone https://salsa.debian.org/debian/packaging-tutorial.git`
  - `https://salsa.debian.org/debian/packaging-tutorial`
  - Offene Fehler: `bugs.debian.org/src:packaging-tutorial`
- Rückmeldung geben:
  - `mailto:packaging-tutorial@packages.debian.org`
    - Was sollte zu dieser Anleitung hinzugefügt werden?
    - Was sollte verbessert werden?
  - `reportbug packaging-tutorial`

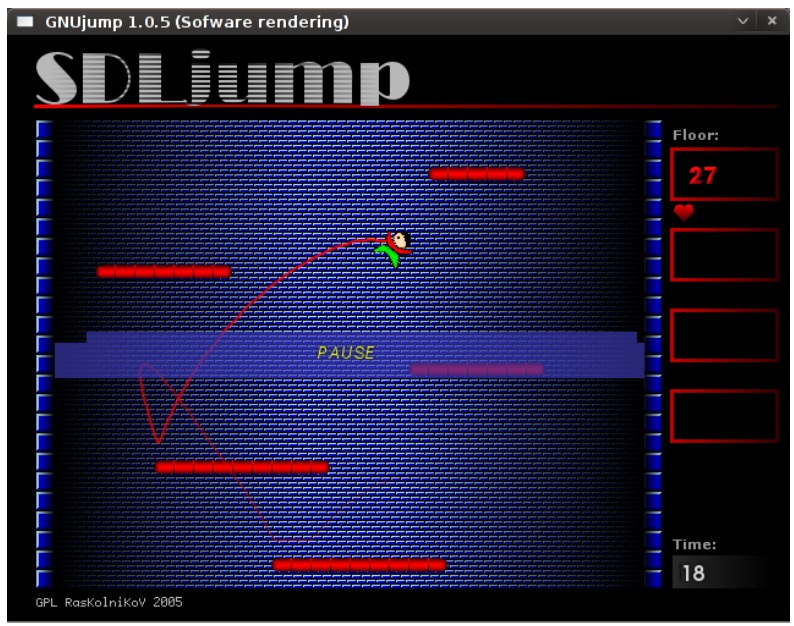
Gliederung

Praktische Sitzung 2: GNUjump paketieren

1. Laden Sie GNUjump 1.0.8 von `http://ftp.gnu.org/gnu/gnujump/gnujump-1.0.8.tar.gz` herunter

2. Erstellen Sie ein Debian-Paket dafür
   - Installieren Sie die Bauabhängigkeiten, so dass Sie das Paket bauen können
   - Korrigieren Sie Fehler
   - Erstellen Sie ein grundlegendes, funktionierendes Paket
   - Zum Schluss füllen Sie `debian/control` und andere Dateien aus

3. Viel Spaß



Praktische Sitzung 2: GNUjump paketieren (Tipps)

- Um ein grundlegendes, funktionierendes Quellpaket zu erhalten, verwenden Sie: `dh_make`
- Am Anfang ist ein Quellpaket *1.0* einfacher als ein *3.0 (quilt)* (ändern Sie das in `debian/source/format`)
- Um nach fehlenden Bauabhängigkeiten zu suchen, finden Sie eine fehlende Datei und verwenden Sie `apt-file`, um das fehlende Paket zu finden
- Falls Sie auf den folgenden Fehler stoßen:

```
/usr/bin/ld: SDL_rotozoom.o: undefined reference to symbol 'ceil@@GLIBC_2.2.5'
//lib/x86_64-linux-gnu/libm.so.6: error adding symbols: DSO missing from command line
collect2: error: ld returned 1 exit status
Makefile:376: die Regel für Ziel "gnujump" scheiterte
```

  Müssen Sie `-lm` zur Linker-Befehlszeile hinzufügen:
  Bearbeiten Sie `src/Makefile.am` und ersetzen Sie

```
gnujump_LDFLAGS = $(all_libraries)
```

  durch

```
gnujump_LDFLAGS = -Wl,--as-needed
gnujump_LDADD = $(all_libraries) -lm
```

  Führen Sie dann `autoreconf -i` aus

Praktische Sitzung 3: Eine Java-Bibliothek paketiere

1. Schauen Sie kurz mal auf die Dokumentation zur Java-Paketierung:
   - `https://wiki.debian.org/Java`
   - `https://wiki.debian.org/Java/Packaging`
   - `https://www.debian.org/doc/packaging-manuals/java-policy/`
   - `/usr/share/doc/javahelper/tutorial.txt.gz`
2. Laden Sie IRClib von `http://moepii.sourceforge.net/` herunter
3. Paketieren Sie es

Praktische Sitzung 4: Ein Ruby-Gem paketieren

1. Schauen Sie kurz auf einige Dokumentation über Ruby-Paketierung:
   - `https://wiki.debian.org/Ruby`
   - `https://wiki.debian.org/Teams/Ruby`
   - `https://wiki.debian.org/Teams/Ruby/Packaging`
   - `gem2deb(1)`, `dh_ruby(1)` (im Paket `gem2deb`)
2. Erstellen Sie ein grundlegendes Debian-Quellpaket aus dem `peach`-Gem: `gem2deb peach`
3. Verbessern Sie es, so dass es ein richtiges Debian-Paket wird

Praktische Sitzung 5: Ein Perl-Modul paketieren

1. Schauen Sie kurz mal auf die Dokumentation zur Perl-Paketierung:
   - `https://perl-team.pages.debian.net`
   - `https://wiki.debian.org/Teams/DebianPerlGroup`
   - `dh-make-perl(1)`, `dpt(1)` (im Paket `pkg-perl-tools`)
2. Erstellen Sie ein grundlegendes Debian-Quellpaket aus der `Acme`-CPAN-Distribution:
   `dh-make-perl --cpan Acme`
3. Verbessern Sie es, so dass es ein richtiges Debian-Paket wird

Gliederung

Antworten zu den praktische Sitzungen

Praktische Sitzung 1: Anpassen des Grep-Pakets

1. Laden Sie Version 2.12-2 des Pakets von `http://ftp.debian.org/debian/pool/main/g/grep/` herunter.
2. Schauen Sie sich die Dateien in `debian/` an.
   - Wie viele Binärpakete werden aus diesem Quellpaket erstellt?
   - Welche Paketierungshelfer verwendet dieses Paket?
3. Bauen Sie das Paket
4. Wir werden das Paket jetzt anpassen. Fügen Sie einen Changelog-Eintrag hinzu und erhöhen Sie die Versionsnummer.
5. Deaktivieren Sie jetzt die Perl-Regexp-Unterstützung (dies ist eine `./configure`-Option)
6. Bauen Sie das Paket erneut
7. Vergleichen Sie das ursprüngliche und das neue Paket mit Debdiff
8. Installieren Sie das neu gebaute Paket

Holen der Quellen

1. Laden Sie Version 2.12-2 des Pakets von `http://ftp.debian.org/debian/pool/main/g/grep/` herunter.

- Verwenden Sie dget, um die Datei `.dsc` herunterzuladen:
  `dget http://cdn.debian.net/debian/pool/main/g/grep/grep_2.12-2.dsc`
- Falls Sie `deb-src` für eine Debian-Veröffentlichung, die `grep` Version 2.12-2 hat, haben (schauen Sie auf `https://tracker.debian.org/grep`), können Sie folgendes verwenden: `apt-get source grep=2.12-2` oder `apt-get source grep/release` (z.B. `grep/stable`) oder, falls Sie es auf Gut Glück versuchen wollen: `apt-get source grep`
- Das Quellpaket `grep` besteht aus drei Dateien:
  - `grep_2.12-2.dsc`
  - `grep_2.12-2.debian.tar.bz2`
  - `grep_2.12.orig.tar.bz2`

  Dies ist für das Format »3.0 (quilt)« typisch.
- Falls notwendig, dekomprimieren Sie die Quellen mit
  `dpkg-source -x grep_2.12-2.dsc`

Rumschauen und Paket bauen

2. Schauen Sie sich die Dateien in `debian/` an.
   - Wie viele Binärpakete werden aus diesem Quellpaket erstellt?
   - Welche Paketierungshelfer verwendet dieses Paket?

- Laut `debian/control` erstellt dieses Paket nur ein Binärpaket namens `grep`.
- Laut `debian/rules` verwendet dieses Paket typisches *klassisches* Debhelper, ohne *CDBS* oder *dh*. Die verschiedenen Aufrufe an die `dh_*`-Befehle können in `debian/rules` gesehen werden.

3. Bauen Sie das Paket

- Verwenden Sie `apt-get build-dep grep`, um die Bauabhängigkeiten zu holen
- Dann `debuild` oder `dpkg-buildpackage -us -uc` (benötigt rund eine Minute)

Das Änderungsprotokoll (Changelog) bearbeiten

4. Wir werden das Paket jetzt anpassen. Fügen Sie einen Changelog-Eintrag hinzu und erhöhen Sie die Versionsnummer.

- `debian/changelog` ist eine Textdatei. Sie könnten sie manuell bearbeiten und einen Eintrag hinzufügen.
- Oder Sie können `dch -i` verwenden, das einen Eintrag hinzufügen und einen Editor öffnen wird.
- Der Name und die E-Mail kann mittels der Umgebungsvariablen `DEBFULLNAME` und `DEBEMAIL` definiert werden.
- Danach bauen Sie das Paket neu; eine neue Version des Paketes ist gebaut
- Paket-Versionierung wird im Detail in Abschnitt 5.6.12 der Debian-Richtlinien dargestellt
  `https://www.debian.org/doc/debian-policy/ch-controlfields`

Perl-Regex-Unterstützung deaktivieren und neu bau

5. Deaktivieren Sie jetzt die Perl-Regexp-Unterstützung (dies ist eine `./configure`-Option)
6. Bauen Sie das Paket erneut

- Prüfen Sie mit `./configure --help`: Die Option, um reguläre Perl-Ausdrücke zu deaktivieren, ist `--disable-perl-regexp`
- Bearbeiten Sie `debian/rules` und suchen Sie die Zeile mit `./configure`
- Fügen Sie `--disable-perl-regexp` hinzu
- Bauen Sie mit `debuild` oder `dpkg-buildpackage -us -uc` neu

Vergleichen und Testen des Pakets

7. Vergleichen Sie das ursprüngliche und das neue Paket mit Debdiff
8. Installieren Sie das neu gebaute Paket

- Vergleichen der Binärpakete: `debdiff ../*changes`
- Vergleichen der Quellpakete: `debdiff ../*dsc`
- Installieren Sie das neu gebaute Paket: `debi`
  Oder `dpkg -i ../grep_<TAB>`
- `grep -P foo` funktioniert nicht mehr!

Installieren Sie wieder die vorherige Version des Pakets:

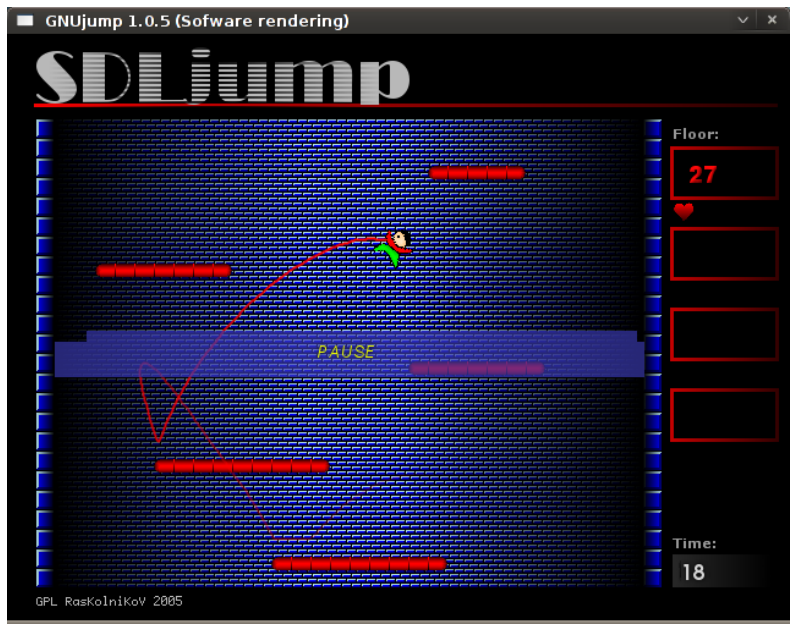
- `apt-get install --reinstall grep=2.6.3-3` *(= vorherige Version)*

Praktische Sitzung 2: GNUjump paketieren

1. Laden Sie GNUjump 1.0.8 von `http://ftp.gnu.org/gnu/gnujump/gnujump-1.0.8.tar.gz` herunter

2. Erstellen Sie ein Debian-Paket dafür
   - Installieren Sie die Bauabhängigkeiten, so dass Sie das Paket bauen können
   - Erstellen Sie ein grundlegendes, funktionierendes Paket
   - Zum Schluss füllen Sie `debian/control` und andere Dateien aus

3. Viel Spaß



Schritt für Schritt...

- `wget http://ftp.gnu.org/gnu/gnujump/gnujump-1.0.8.tar.gz`
- `mv gnujump-1.0.8.tar.gz gnujump_1.0.8.orig.tar.gz`
- `tar xf gnujump_1.0.8.orig.tar.gz`
- `cd gnujump-1.0.8/`
- `dh_make -f ../gnujump-1.0.8.tar.gz`
  - Pakettyp: Einzelnes Programm (derzeit)

```
gnujump-1.0.8$ 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
```

Schritt für Schritt. . . (2)

- Schauen Sie in `debian/changelog`, `debian/rules`, `debian/control` (durch **dh_make** automatisch ausgefüllt)
- In `debian/control`:
  `Build-Depends: debhelper (>= 7.0.50 ), autotools-dev`
  Führt die *build-dependencies* auf = Pakete, die zum Bau des Pakets benötigt werden
- Versuchen Sie, das Paket so mit `debuild` zu bauen (dank der **dh**-Magie)
  - Fügen Sie Bauabhängigkeiten hinzu, bis das Paket baut
  - Tipp: Verwenden Sie `apt-cache search` und `apt-file`, um die Pakete zu finden
  - Beispiel:

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

    → **libsdl1.2-dev** zu den Build-Depends hinzufügen und installieren.
  - Besser: **pbuilder** verwenden, um in einer sauberen Umgebung zu bauen

Schritt für Schritt. . . (3)

- Benötigte Bauabhängigkeiten sind `libsdl1.2-dev`, `libsdl-image1.2-dev`, `libsdl-mixer1.2-dev`
- Wahrscheinlich tritt danach ein anderer Fehler auf:

```
/usr/bin/ld: SDL_rotozoom.o: undefined reference to symbol 'ceil@@GLIBC_2.2.5'
//lib/x86_64-linux-gnu/libm.so.6: error adding symbols: DSO missing from command line
collect2: error: ld returned 1 exit status
Makefile:376: die Regel für Ziel "gnujump" scheiterte
```

- Diese Problem entsteht durch vernachlässigte Software: Gnujump wurde nicht auf die Änderungen beim Linker angepasst.
- Falls Sie das Quellformat Version **1.0** verwenden, können Sie direkt die Quellen der Originalautoren ändern.
  - Bearbeiten Sie `src/Makefile.am` und ersetzen Sie

    ```
    gnujump_LDFLAGS = $(all_libraries)
    ```

    durch

    ```
    gnujump_LDFLAGS = -Wl,--as-needed
    gnujump_LDADD = $(all_libraries) -lm
    ```

  - Führen Sie dann `autoreconf -i` aus

Schritt für Schritt. . . (4)

- Falls Sie das Quellformat Version **3.0 (quilt)** verwenden, verwenden Sie `quilt`, um einen Patch zu erstellen. (siehe `https://wiki.debian.org/UsingQuilt`)
  - `export QUILT_PATCHES=debian/patches`
  - `mkdir debian/patches`
    `quilt new linker-fixes.patch`
    `quilt add src/Makefile.am`
  - Bearbeiten Sie `src/Makefile.am` und ersetzen Sie

    ```
    gnujump_LDFLAGS = $(all_libraries)
    ```

    durch

    ```
    gnujump_LDFLAGS = -Wl,--as-needed
    gnujump_LDADD = $(all_libraries) -lm
    ```

  - `quilt refresh`
  - Da `src/Makefile.am` geändert wurde, muss während des Baus Autoreconf aufgerufen werden. Um das mit `dh` automatisch durchzuführen, ändern Sie den Aufruf von `dh` in `debian/rules` von `dh $ --with autotools-dev` in `dh $ --with autotools-dev --with autoreconf`

Schritt für Schritt. . . (5)

- Das Paket sollte jetzt korrekt bauen.
- Verwenden Sie `debc`, um den Inhalt des erstellten Pakets aufzulisten und `debi`, um es zu installieren und zu testen
- Testen Sie das Pakets mit `lintian`
  - Nicht zwingend gefordert, aber für nach Debian hochgeladene Pakete wird *lintian-clean* empfohlen
  - Weitere Probleme können mit `lintian -EviIL +pedantic` aufgelistet werden
  - Einige Tipps:
    - Entfernen Sie nicht benötigte Dateien aus `debian/`
    - Ausfüllen von `debian/control`
    - Installation des Programms nach `/usr/games`, außer Kraft setzen von `dh_auto_configure`
    - Verwendung von *Härtungs*-Compiler-Schalter für bessere Sicherheit. Siehe `https://wiki.debian.org/Hardening`

Schritt für Schritt. . . (6)

- Vergleichen Sie Ihr Paket mit dem bereits in Debian paketierten:
  - Es verschiebt die Datendateien in ein zweites Paket, das über alle Architekturen hinweg identisch ist ($\rightarrow$ spart Platz im Debian-Archiv)
  - Es installiert eine .desktop-Datei (für die GNOME-/KDE-Menüs) und integriert sich auch in das Debian-Menü
  - Es korrigiert ein paar kleinere Probleme mit Patches

Praktische Sitzung 3: Eine Java-Bibliothek paketiere

1. Schauen Sie kurz mal auf die Dokumentation zur Java-Paketierung:
   - `https://wiki.debian.org/Java`
   - `https://wiki.debian.org/Java/Packaging`
   - `https://www.debian.org/doc/packaging-manuals/java-policy/`
   - `/usr/share/doc/javahelper/tutorial.txt.gz`
2. Laden Sie IRClib von `http://moepii.sourceforge.net/` herunter
3. Paketieren Sie es

Schritt für Schritt...

- `apt-get install javahelper`
- Ein grundlegendes Quellpaket erstellen: `jh_makepkg`
  - Bibliothek
  - Keine
  - Standard Freier Compiler/Laufzeitumgebung
- `debian/*` anschauen und korrigieren
- `dpkg-buildpackage -us -uc` oder `debuild`
- `lintian`, `debc`, usw.
- Vergleichen Sie Ihr Ergebnis mit dem Quellpaket `libirclib-java`

Praktische Sitzung 4: Ein Ruby-Gem paketieren

1. Schauen Sie kurz auf einige Dokumentation über Ruby-Paketierung:
   - `https://wiki.debian.org/Ruby`
   - `https://wiki.debian.org/Teams/Ruby`
   - `https://wiki.debian.org/Teams/Ruby/Packaging`
   - `gem2deb(1)`, `dh_ruby(1)` (im Paket `gem2deb`)
2. Erstellen Sie ein grundlegendes Debian-Quellpaket aus dem `peach`-Gem: `gem2deb peach`
3. Verbessern Sie es, so dass es ein richtiges Debian-Paket wird

Schritt für Schritt. . .

`gem2deb peach`:

- Lädt Gem von rubygems.org herunter
- Erstellt ein geeignetes .orig.tar.gz-Archiv und entpackt es
- Initialisiert ein Debian-Quellpaket, basierend auf den Gem-Metadaten.
  - Namens `ruby-`*`Gem-Name`*
- Versucht, das Debian-Paket zu bauen (kann fehlschlagen)

`dh_ruby` (Teil von *gem2deb*) erledigt die Ruby-spezfischen Aufgaben:

- C-Erweiterungen für jede Ruby-Version bauen
- Dateien in ihr Zielverzeichnis kopieren
- Shebangs in ausführbaren Skripten aktualisieren
- Die in `debian/ruby-tests.rb`, `debian/ruby-tests.rake` oder `debian/ruby-test-files.yaml` definierten Tests ausführen sowie weitere Prüfungen

Schritt für Schritt. . . (2)

Verbessern des erstellten Paketes:

- `debclean` ausführen, um den Quellbaum zu bereinigen. Schauen Sie in `debian/`.
- `changelog` und `compat` sollten korrekt sein
- Bearbeiten Sie `debian/control`: verbessern Sie `Description`
- Schreiben Sie eine vernünftige `copyright`-Datei, basierend auf den Dateien der Originalautoren
- Bauen Sie das Paket
- Vergleichen Sie Ihr Paket mit dem Paket `ruby-peach` im Debian-Archiv

Praktische Sitzung 5: Ein Perl-Modul paketieren

1. Schauen Sie kurz mal auf die Dokumentation zur Perl-Paketierung:
   - `https://perl-team.pages.debian.net`
   - `https://wiki.debian.org/Teams/DebianPerlGroup`
   - `dh-make-perl(1)`, `dpt(1)` (im Paket `pkg-perl-tools`)
2. Erstellen Sie ein grundlegendes Debian-Quellpaket aus der `Acme`-CPAN-Distribution:
   `dh-make-perl --cpan Acme`
3. Verbessern Sie es, so dass es ein richtiges Debian-Paket wird

Schritt für Schritt...

`dh-make-perl --cpan Acme`:

- Lädt den Tarball von CPAN herunter
- Erstellt ein geeignetes .orig.tar.gz-Archiv und entpackt es
- Initialisiert ein Debian-Quellpaket, basierend auf den Metadaten der Distribution.
  - Namens `lib`*`Distname`*`-perl`

Schritt für Schritt. . . (2)

Verbessern des erstellten Paketes:

- `debian/changelog`, `debian/compat`, `debian/libacme-perl.docs` und `debian/watch` sollten korrekt sein
- Bearbeiten Sie `debian/control`: Verbessern Sie `Description` und entfernen Sie den Textbaustein am Ende
- Bearbeiten Sie `debian/copyright`: Entfernen Sie den Textbaustein am Anfang und fügen Sie Copyright-Jahre zum Block `Files: *` hinzu