

Debian-Paketier-Anleitung

Lucas Nussbaum
lucas@debian.org

Version 0.6 – 2012-04-09



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

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets
- 5 Fortgeschrittene Paketierungsthemen
- 6 Pakete in Debian betreuen
- 7 Fazit
- 8 Praktische Sitzung 2: GNUjump paketieren
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren
- 11 Antworten zu den praktischen Sitzungen



Gliederung

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets
- 5 Fortgeschrittene Paketierungsthemen
- 6 Pakete in Debian betreuen
- 7 Fazit
- 8 Praktische Sitzung 2: GNUjump paketieren
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren
- 11 Antworten zu den praktischen Sitzungen



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
→ 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



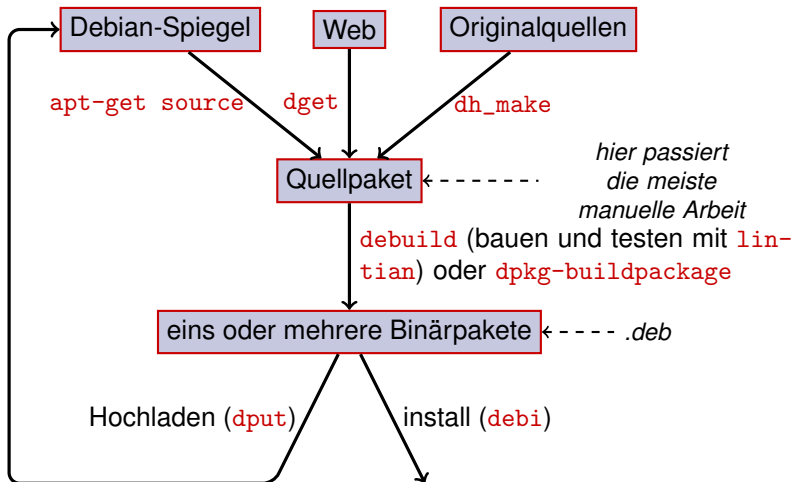
Folgende Werkzeuge benötigen Sie

- ▶ 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**, **cdb**s, **quilt**, **pbuilder**, **sbuid**, **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

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets
- 5 Fortgeschrittene Paketierungsthemen
- 6 Pakete in Debian betreuen
- 7 Fazit
- 8 Praktische Sitzung 2: GNUjump paketieren
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren
- 11 Antworten zu den praktischen Sitzungen



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 – 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 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
```

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)
- ▶ Benennen Sie sie in `<Quellpaket>_<Originalversion>.orig.tar.gz` um
(Beispiel: `simgrid_3.6.orig.tar.gz`)
- ▶ Entpacken Sie es
- ▶ `cd Originalquellen && 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

1.2.1.1-5

Original- Debian-
version Revision

- ▶ Manuell oder mit **dch** bearbeiten
 - ▶ Änderungsprotokolleintrag 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.
```



# 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  
<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 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 einer Teilmenge der 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  
<http://www.debian.org/doc/debian-policy/ch-source.html#s-debianrules>
- ▶ Es gibt fünf benötigte Ziele:
  - ▶ `build`: 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
  - ▶ Siehe beispielsweise das Paket `adduser`
- ▶ 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_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`, ...

- ▶ 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/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/package 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/cdb/`
  - ▶ 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/cdb/1/rules/debhelper.mk
include /usr/share/cdb/1/class/autotools.mk
```





# Dh (lang Debhelper 7 oder dh7)

- ▶ Eingeführt in 2008 als ein *CDBS-Mörder*
- ▶ **dh**-Befehl, der `dh_*` aufruft
- ▶ Einfache *debian/rules*, nur mit 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
```

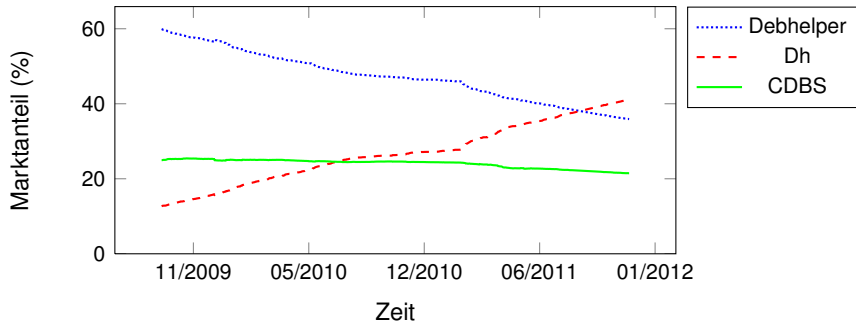
```
override_dh_auto_build:
```

```
make world
```



# Klassischer Debhelper vs. CDBS vs. dh

- ▶ Marktanteil:  
Klassischer Debhelper: 36%   CDBS: 21%   Dh: 41%
- ▶ 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)



# Gliederung

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen**
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets
- 5 Fortgeschrittene Paketierungsthemen
- 6 Pakete in Debian betreuen
- 7 Fazit
- 8 Praktische Sitzung 2: GNUjump paketieren
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren
- 11 Antworten zu den praktischen Sitzungen



# Pakete bauen

- ▶ `apt-get build-dep MeinPaket`  
Installiert die *Bauabhängigkeiten* (für ein Paket, das bereits im Debian-Archiv ist)  
Oder `mk-build-deps -ir` (innerhalb des Paket-Quellbaums)
- ▶ `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
- ▶ Laden Sie das Paket nach Debian hoch (`dput`) (benötigt Konfiguration)
- ▶ Betreiben Sie ein privates Debian-Archiv mit `reprepro`  
Dokumentation: <http://mirrorer.alioth.debian.org/>



# Gliederung

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets**
- 5 Fortgeschrittene Paketierungsthemen
- 6 Pakete in Debian betreuen
- 7 Fazit
- 8 Praktische Sitzung 2: GNUjump paketieren
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren
- 11 Antworten zu den praktischen Sitzungen



# Praktische Sitzung 1: Anpassen des Grep-Pakets

- ❶ Holen Sie Version 2.6.3-3 (bzw. Version 2.9-1 oder 2.9-2, falls Sie Ubuntu 11.10 oder neuer oder Debian Testing oder Unstable verwenden) des Pakets von <http://ftp.debian.org/debian/pool/main/g/grep/>
  - ▶ Falls das Quellpaket nicht automatisch entpackt wird, entpacken Sie es mit `dpkg-source -x grep_*.dsc`
- ❷ Schauen Sie sich die Dateien in `debian/` an.
  - ▶ Wie viele Binärpakete werden aus diesem Quellpaket erstellt?
  - ▶ Welche Paketierungshelfer verwendet dieses Paket?
- ❸ Bauen Sie das Paket
- ❹ Wir werden das Paket jetzt anpassen. Fügen Sie einen Changelog-Eintrag hinzu und erhöhen Sie die Versionsnummer.
- ❺ Deaktivieren Sie jetzt die Perl-Regex-Unterstützung (dies ist eine `./configure`-Option)
- ❻ Bauen Sie das Paket erneut
- ❼ Vergleichen Sie das ursprüngliche und das neue Paket mit `Debdiff`
- ❽ Installieren Sie das neu gebaute Paket
- ❾ Weinen Sie, falls es schief gegangen ist ;)



# Gliederung

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets
- 5 Fortgeschrittene Paketierungsthemen**
- 6 Pakete in Debian betreuen
- 7 Fazit
- 8 Praktische Sitzung 2: GNUjump paketieren
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren
- 11 Antworten zu den praktischen Sitzungen





# debian/copyright

- ▶ Urheberrecht- und Lizenzinformationen für diese Quellen und die Paketierung
- ▶ Traditionell als Textdatei geschrieben
- ▶ Neues, maschinenlesbares 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 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
  - ▶ Hilfte beim gemeinsamen Nutzen der Korrekturen mit derivativen Distributionen
  - ▶ Gibt den Änderungen mehr Aufmerksamkeit  
<http://patch-tracker.debian.org/>



# 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* (*cdb*s), *dpatch*, ***quilt***
  - ▶ Alle unterstützen zwei Ziele in `debian/rules`:
    - ▶ `debian/rules patch`: alle Patches anwenden
    - ▶ `debian/rules unpatch`: alle Patches entfernen
  - ▶ Weitere Dokumentation: <http://wiki.debian.org/debian/patches>
- ▶ **Neues Quellformat mit eingebautem Patch-System: 3.0 (quilt)**
  - ▶ Empfohlene Lösung
  - ▶ Sie müssen *quilt* lernen  
<http://pkg-perl.alioth.debian.org/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://bazaar.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  
<http://www.debian.org/doc/debian-policy/ch-maintainerscripts.html>
  - ▶ Debian-Entwicklerreferenz, Kapitel 6.4  
<http://www.debian.org/doc/developers-reference/best-pkging-practices.html>
  - ▶ <http://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
```

- ▶ Debian-Infrastruktur, die `debian/watch` verwendet:

## **Debian External Health Status**

<http://dehs.alioth.debian.org/>

- ▶ Betreuer werden durch E-Mails, die an die Paketdatenbank geschickt werden, gewarnt

<http://packages.qa.debian.org/>

- ▶ `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
- ▶ `Vcs-*`-Felder in `debian/control`, um das Depot anzugeben
  - ▶ `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-unabhängige Schnittstelle: `debcheckout`, `debcommit`, `debrelease`
  - ▶ `debcheckout grep` → checkt das Quellpaket aus Git aus



# Pakete rückportieren

- ▶ 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

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets
- 5 Fortgeschrittene Paketierungsthemen
- 6 Pakete in Debian betreuen
- 7 Fazit
- 8 Praktische Sitzung 2: GNUjump paketieren
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren
- 11 Antworten zu den praktischen Sitzungen



# Es gibt viele Möglichkeiten, zu Debian beizutragen

## ▶ **Schlechteste** Art, beizutragen:

- ➊ Paketieren Sie Ihre eigene Anwendung
- ➋ Schaffen Sie diese nach Debian
- ➌ 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 <http://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: <http://www.debian.org/devel/wnpp/>
- ▶ Installiert auf Ihrer Maschine: `wnpp-alert`
- ▶ Verschiedene Stati:
  - ▶ **Orphaned**: das Paket ist verwaist (es wird nicht mehr betreut)  
Adoptieren Sie es ruhig
  - ▶ **RFA: Request For Adopter**  
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: Intent To Adopt**  
Jemand plant, das Pakete zu adoptieren – Sie könnten Hilfe anbieten!
  - ▶ **RFH: Request For Help**  
Der Paketbetreuer sucht Hilfe
- ▶ Einige nicht betreute Pakete werden nicht erkannt → noch nicht verwaist
- ▶ Im Zweifelsfall fragen Sie auf `debian-qa@lists.debian.org`  
oder `#debian-qa` auf `irc.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
  - ❶ Ein Quellpaket vorbereiten
  - ❷ Finden Sie einen Debian-Entwickler, der Ihr Paket sponsern wird
- ▶ Offizieller Status (wenn Sie bereits erfahren sind):
  - ▶ **Debian-Betreuer (DM):**  
Recht, Ihre eigenen Pakete hochzuladen  
See <http://wiki.debian.org/DebianMaintainer>
  - ▶ **Debian-Entwickler (DD):**  
Debian-Projektmitglied; darf abstimmen und jedes Paket hochladen



# 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**
  - ▶ Sie kennen die Spezifika ihres Pakets
  - ▶ Sie können ein Mitglied des Teams werden
- ▶ Der Debian-Mentors-Gruppe (falls Ihr Paket in kein Team passt)
  - ▶ <http://wiki.debian.org/DebianMentorsFaq>
  - ▶ Mailingliste: [debian-mentors@lists.debian.org](mailto:debian-mentors@lists.debian.org)  
(auch eine gute Art, nebenbei was zu lernen)
  - ▶ IRC: #debian-mentors auf [irc.debian.org](http://irc.debian.org)
  - ▶ <http://mentors.debian.net/>



# Offizielle Dokumentation

- ▶ **Debian's Entwickler-Ecke**  
<http://www.debian.org/devel/>  
Links auf viele Ressourcen über Debian-Entwicklung
- ▶ **Leitfaden für neue Debian-Betreuer**  
<http://www.debian.org/doc/maint-guide/>  
Eine Einführung zur Debian-Paketierung, sollte mal aktualisiert werden
- ▶ **Debian-Entwicklerreferenz**  
<http://www.debian.org/doc/developers-reference/>  
Hauptsächlich über Debian-Prozeduren, aber auch einige goldene Regeln der Paketierung (Teil 6)
- ▶ **Debian-Richtlinien**  
<http://www.debian.org/doc/debian-policy/>
  - ▶ Alle Anforderungen, die jedes Paket erfüllen muss
  - ▶ Spezielle Richtlinien für Perl, Java, Python, ...
- ▶ **Ubuntu-Paketierungsleitfaden**  
<https://wiki.ubuntu.com/PackagingGuide>



# Debian-Armaturenbrett für Betreuer

- ▶ **Quellpaket zentriert:** Paketdatenbank (PTS)  
`http://packages.qa.debian.org/dpkg`
- ▶ **Betreuer/Team zentriert:** Paketüberblick für Entwickler (DDPO)  
`http://qa.debian.org/developer.php?login=`  
`pkg-ruby-extras-maintainers@lists.alioth.debian.org`





# 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

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets
- 5 Fortgeschrittene Paketierungsthemen
- 6 Pakete in Debian betreuen
- 7 Fazit**
- 8 Praktische Sitzung 2: GNUjump paketieren
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren
- 11 Antworten zu den praktischen Sitzungen



# 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)**
- ▶ Dinge, die in dieser Anleitung nicht betrachtet wurden:
  - ▶ UCF – Benutzeränderungen an Konfigurationsdateien beim Upgrade verwalten
  - ▶ Dpkg-Triggers - ähnliche Betreuerskriptaktionen zusammengruppieren
  - ▶ Debian-Entwicklungsorganisation:
    - ▶ Fehlerdatenbank (BTS)
    - ▶ Suites: Stable, Testing, Unstable, Experimental, Security, \*-updates, Backports, ...
    - ▶ Debian Blends – Teilmenge von Debian, gezielt für bestimmte Gruppen

Rückmeldungen: **lucas@debian.org**



# Rechtliches Zeug

Copyright ©2011 Lucas Nussbaum – lucas@debian.org

**Dieses Dokument ist freie Software;** Sie können es unter einer der folgenden Optionen (Ihrer Wahl) vertreiben und/oder verändern:

- ▶ Den Bedingungen der GNU General Public License, wie sie von der Free Software Foundation in Version 3 (oder nach Ihrer Wahl) einer neueren Version veröffentlicht wurden  
<http://www.gnu.org/licenses/gpl.html>
- ▶ Den Bedingungen der Creative Commons Attribution-ShareAlike 3.0 Unported License.  
<http://creativecommons.org/licenses/by-sa/3.0/>



# Neuste Version & Quellcode

## ▶ Neuste Version:

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

## ▶ Beitragen:

### ▶ `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`

## ▶ Rückmeldung: `lucas@debian.org`



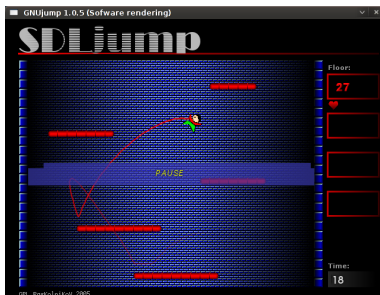
# Gliederung

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets
- 5 Fortgeschrittene Paketierungsthemen
- 6 Pakete in Debian betreuen
- 7 Fazit
- 8 Praktische Sitzung 2: GNUjump paketieren**
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren
- 11 Antworten zu den praktischen Sitzungen



# Praktische Sitzung 2: GNUjump paketieren

- 1 Laden Sie GNUjump 1.0.6 von <http://ftp.gnu.org/gnu/gnjump/1.0.6/gnjump-1.0.6.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



# Gliederung

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets
- 5 Fortgeschrittene Paketierungsthemen
- 6 Pakete in Debian betreuen
- 7 Fazit
- 8 Praktische Sitzung 2: GNUjump paketieren
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren**
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren
- 11 Antworten zu den praktischen Sitzungen





# Praktische Sitzung 3: Eine Java-Bibliothek paketieren

- ❶ Schauen Sie kurz mal auf die Dokumentation zur Java-Paketierung:
  - ▶ <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>
  - ▶ Veröffentlichungen und Folien von einem Debconf10-Vortrag über Javahelper:  
<http://pkg-java.alioth.debian.org/docs/debconf10-javahelper-paper.pdf>  
<http://pkg-java.alioth.debian.org/docs/debconf10-javahelper-slides.pdf>
- ❷ Laden Sie IRCLib von <http://moepii.sourceforge.net/> herunter
- ❸ Paketieren Sie es



# Gliederung

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets
- 5 Fortgeschrittene Paketierungsthemen
- 6 Pakete in Debian betreuen
- 7 Fazit
- 8 Praktische Sitzung 2: GNUjump paketieren
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren**
- 11 Antworten zu den praktischen Sitzungen



# Praktische Sitzung 4: Ein Ruby-Gem paketieren

---

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



# Gliederung

- 1 Einleitung
- 2 Ein Quellpaket erstellen
- 3 Pakete bauen und testen
- 4 Praktische Sitzung 1: Anpassen des Grep-Pakets
- 5 Fortgeschrittene Paketierungsthemen
- 6 Pakete in Debian betreuen
- 7 Fazit
- 8 Praktische Sitzung 2: GNUjump paketieren
- 9 Praktische Sitzung 3: Eine Java-Bibliothek paketieren
- 10 Praktische Sitzung 4: Ein Ruby-Gem paketieren
- 11 Antworten zu den praktischen Sitzungen**



# Antworten zu den praktische Sitzungen



# Praktische Sitzung 1: Anpassen des Grep-Pakets

- ➊ Holen Sie Version 2.6.3-3 (bzw. Version 2.9-1 oder 2.9-2, falls Sie Ubuntu 11.10 oder neuer oder Debian Testing oder Unstable verwenden) des Pakets von <http://ftp.debian.org/debian/pool/main/g/grep/>
- ➋ Schauen Sie sich die Dateien in `debian/` an.
  - ▶ Wie viele Binärpakete werden aus diesem Quellpaket erstellt?
  - ▶ Welche Paketierungshelfer verwendet dieses Paket?
- ➌ Bauen Sie das Paket
- ➍ Wir werden das Paket jetzt anpassen. Fügen Sie einen Changelog-Eintrag hinzu und erhöhen Sie die Versionsnummer.
- ➎ Deaktivieren Sie jetzt die Perl-Regex-Unterstützung (dies ist eine `./configure`-Option)
- ➏ Bauen Sie das Paket erneut
- ➐ Vergleichen Sie das ursprüngliche und das neue Paket mit `Debdiff`
- ➑ Installieren Sie das neu gebaute Paket
- ➒ Weinen Sie, falls es schief gegangen ist ;)



# Holen der Quellen

- ❶ Laden Sie Version 2.6.3-3 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_2.6.3-3.dsc`
- ▶ Laut `http://packages.qa.debian.org/grep` ist derzeit Grep Version 2.6.3-3 in *Stable* (*Squeeze*). Falls Sie `deb-src`-Zeilen für *squeeze* in Ihrer `/etc/apt/sources.list` haben, können Sie folgendes verwenden:  
`apt-get source grep=2.6.3-3`  
oder `apt-get source grep/stable`  
oder, falls Sie es auf Gut Glück versuchen wollen: `apt-get source grep`
- ▶ Das Quellpaket `grep` besteht aus drei Dateien:
  - ▶ `grep_2.6.3-3.dsc`
  - ▶ `grep_2.6.3-3.debian.tar.bz2`
  - ▶ `grep_2.6.3.orig.tar.bz2`Dies ist für das Format »3.0 (quilt)« typisch.
- ▶ Falls notwendig, dekomprimieren Sie die Quellen mit  
`dpkg-source -x grep_2.6.3-3.dsc`



# Rumschauen und Paket bauen

- ➊ 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.
- ➋ 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  
<http://www.debian.org/doc/debian-policy/ch-controlfields.html>



# Perl-Regex-Unterstützung deaktivieren und neu bauen

- 5 Deaktivieren Sie jetzt die Perl-Regex-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!

- 9 Weinen Sie, falls es schief gegangen ist ;)

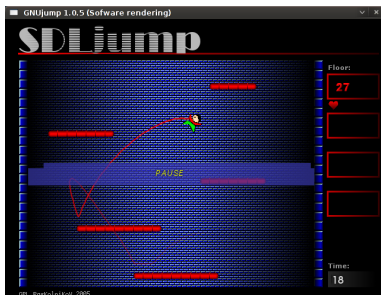
Oder nicht: Installieren Sie wieder die vorherige Version des Pakets:

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



# Praktische Sitzung 2: GNUjump paketieren

- 1 Laden Sie GNUjump 1.0.6 von <http://ftp.gnu.org/gnu/gnjump/1.0.6/gnjump-1.0.6.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/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`
  - ▶ Pakettyp: Einzelnes Programm (derzeit)

```
gnujump-1.0.6$ ls debian/
```

|                                 |                                  |                            |
|---------------------------------|----------------------------------|----------------------------|
| <code>changelog</code>          | <code>gnujump.default.ex</code>  | <code>preinst.ex</code>    |
| <code>compat</code>             | <code>gnujump.doc-base.EX</code> | <code>prerm.ex</code>      |
| <code>control</code>            | <code>init.d.ex</code>           | <code>README.Debian</code> |
| <code>copyright</code>          | <code>manpage.1.ex</code>        | <code>README.source</code> |
| <code>docs</code>               | <code>manpage.sgml.ex</code>     | <code>rules</code>         |
| <code>emacsen-install.ex</code> | <code>manpage.xml.ex</code>      | <code>source</code>        |
| <code>emacsen-remove.ex</code>  | <code>menu.ex</code>             | <code>watch.ex</code>      |
| <code>emacsen-startup.ex</code> | <code>postinst.ex</code>         |                            |
| <code>gnujump.cron.d.ex</code>  | <code>postrm.ex</code>           |                            |



## 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 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)

- ▶ Nach der Installation von `libSDL1.2-dev`, `libSDL-image1.2-dev`, `libSDL-mixer1.2-dev` baut das Paket problemlos.
- ▶ Verwenden Sie `debc`, um den Inhalt des erstellten Pakets aufzulisten
- ▶ Verwenden Sie `debi`, um es zu installieren und zu testen
- ▶ Füllen Sie `debian/control` mit Hilfe von <http://www.debian.org/doc/debian-policy/ch-controlfields.html> aus
- ▶ Testen Sie das Pakets mit `lintian`
- ▶ Entfernen Sie nicht benötigte Dateien aus `debian/`
- ▶ 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 (→ 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 paketieren

- ❶ Schauen Sie kurz mal auf die Dokumentation zur Java-Paketierung:
  - ▶ <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>
  - ▶ Veröffentlichungen und Folien von einem Debconf10-Vortrag über Javahelper:  
<http://pkg-java.alioth.debian.org/docs/debconf10-javahelper-paper.pdf>  
<http://pkg-java.alioth.debian.org/docs/debconf10-javahelper-slides.pdf>
- ❷ Laden Sie IRCLib von <http://moepii.sourceforge.net/> herunter
- ❸ 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 `libirc-lib-java`



# Praktische Sitzung 4: Ein Ruby-Gem paketieren

---

- ➊ Schauen Sie kurz auf einige Dokumentation über Ruby-Paketierung:
  - ▶ `http://wiki.debian.org/Ruby`
  - ▶ `http://wiki.debian.org/Teams/Ruby`
  - ▶ `http://wiki.debian.org/Teams/Ruby/Packaging`
  - ▶ `gem2deb(1)`, `dh_ruby(1)` (im Paket `gem2deb`)
- ➋ Erstellen Sie ein grundlegendes Debian-Quellpaket aus dem `net-ssh-Gem`:  
`gem2deb net-ssh`
- ➌ Verbessern Sie es, so dass es ein richtiges Debian-Paket wird



# Schritt für Schritt...

`gem2deb net-ssh:`

- ▶ 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-spezifischen Aufgaben:

- ▶ C-Erweiterungen für jede Ruby-Version bauen
- ▶ Dateien in ihr Zielverzeichnis kopieren
- ▶ Shebangs in ausführbaren Skripten aktualisieren
- ▶ Die in `debian/ruby-tests.rb` 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`: aktivieren Sie Homepage, verbessern Sie Description
- ▶ Schreiben Sie eine vernünftige `copyright`-Datei, basierend auf den Dateien der Originalautoren
- ▶ `ruby-net-ssh.docs`: Installieren Sie `README.rdoc`
- ▶ `ruby-tests.rb`: Tests ausführen. In diesem Fall reicht Folgendes aus:  

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



## Schritt für Schritt... (3)

Bauen Sie das Paket. Dies schlägt fehl. Es gibt zwei Probleme:

- ▶ Sie müssen den Aufruf `gem` in der Test-Suite deaktivieren. Entfernen Sie die Zeile `gem "test-unit"` in `test/common.rb`
  - ▶ `edit-patch disable-gem.patch`
  - ▶ Bearbeiten Sie `test/common.rb`, entfernen Sie die `gem`-Zeile. Verlassen Sie die Unter-Shell
  - ▶ Beschreiben Sie die Änderungen in `debian/changelog`
  - ▶ Dokumentieren Sie den Patch in `debian/patches/disable-gem.patch`
- ▶ Dem Paket fehlt eine Bauabhängigkeit auf `ruby-mocha`, das in der Test-Suite verwandt wird (es könnte sein, dass Sie das Paket in einer sauberen Umgebung mittels `pbuilder` bauen müssen, um das Problem zu reproduzieren)
  - ▶ Fügen Sie `ruby-mocha` zu den Build-Depends des Pakets hinzu
  - ▶ `gem2deb` kopiert die Abhängigkeiten, die in `gem` als Kommentare dokumentiert sind, in `debian/control`, allerdings wird `mocha` nicht als Entwicklungsabhängigkeit von dem Gem aufgeführt (das ist ein Fehler in dem Gem)

Vergleichen Sie Ihr Paket mit dem Paket `ruby-net-ssh` im Debian-Archiv

