

# Debian-Paketbau für Fortgeschrittene



- Grundkenntnisse im Debianpaketbau werden erwartet, z.B. das `debian/rules` ein Makefile ist.

## Kurzer Sprachunterricht

- falsches deutsches Wort: Packet; richtig: Paket oder Packung
- englisches packet: abzählbare Etwas, z.B. network packet, Gegensatz: network stream
- englisches package: Sammlung von ‚Etwassen‘
- daher: Debian package (englisch) oder Debianpaket (deutsch)

# Versionskontrolle mit Subversion (svn)

- checkout (co)
- update (up)
- revert
- copy (cp) für tagging
- diff (di), diff -rBASE:HEAD → lesen
- mkdir
- add
- delete (del, remove, rm)
- commit (ci) → lesen
- propset (ps), z.B.: svn:executable, svn:ignore, svn:keywords, mergeWithUpstream
- Buch: "Version Control with Subversion":  
<http://svnbook.red-bean.com/>
- SVK: <http://svk.bestpractical.com/view/HomePage> ,  
<http://svkbook.elixus.org/>
- svnadmin create /tmp/chemnitz
- SVN=<file:///tmp/chemnitz>; export \$SVN
- script log

- Vorteile: alle vergangenen Versionen leicht verfügbar; mehr caching als CVS
- Nachteile: nichts löscher; Verzeichnisse .svn in den Arbeitsverzeichnissen; weniger caching als svk, arch, git, ...

## Hostingdienste im Internet

- der erste: <http://sourceforge.net>
- GForge basiert:
  - <http://alioth.debian.org>
  - <http://savannah.gnu.org>
  - <http://developer.berlios.de> (svn über https)
- <http://code.google.com/hosting/> :
  - svn über https
  - Mailinglisten über <http://groups.google.com>
  - einfacher Einstieg
  - Google-Konto notwendig

## svn-buildpackage

- --svn-ignore
- --svn-tag, --svn-retag
- --svn-export

- mergeWithUpstream (svn ps ...1 debian)
- debuild-Optionen wie -a, -S, -uc, -us
- svn-inject [-o] paketname.dsc
- ~/.svn-buildpackage.conf :

```
svn-builder=debuild -uc -us
svn-no-links
svn-dont-clean
```

- Vorteile: keine Build-Dateien im Arbeitsverzeichnis;  
nur Unterschiede zu Upstream im Repository

## debhelper

- #DEBHELPER# in debian/{pre,post}{inst,rm}
- dh\_install
- Verzeichnisse automatisch oder optional über  
dh\_installdirs
- dh\_installinit
- dh\_installdocs
- dh\_installexamples
- dh\_installman

# common debian build system (cdb)

- fertige Schnipsel für debian/rules
- per include einbinden
- sehr guter debhelper-Support
- Beispiele: Makefile, autotools, python, ...
- debian/rules; debian/control Build-Depends; Makefile-Probleme (CFLAGS=-O2 -g)

## Patches verwalten mit quilt

- in Kombination mit diff, vimdiff, kdiff3 oder meld
- quilt kommando:
  - new / delete
  - add / edit / remove
  - push / pop
  - refresh
  - import / fold
- Einbindung in cdb: include /usr/share/cdb/rules/patchsys-quilt.mk
- debian/rules; debian/control Build-Depends
- Vorteile: kleine übersichtliche Patches; passend auch

nach upstream-Updates; für Upstream-Autoren leichter verdaulich als große Patches

- Beispiel: debian/rules apply-patches
  - cflags.diff: Makefile
  - typo.diff: acpi\_listen.8
  - run-parts.diff: acpid.8, event.c
  - logrotate.diff: acpid.c
  - portability.diff: ud\_socket.c

## **Konfigurationsmanagement mit debconf**

- Management von Konfigurationsdaten (nicht -dateien)
- Depends:  $\${misc:Depends}$
- debian/config
- debian/templates
- debian/postinst
- debian/postm automatisch über debhelper (dh\_installdebconf)
- preseeding und alternative Quellen für Konfigurationsdaten möglich
- Unterstützung durch debhelper

# debconf-i18n

- Übersetzung von debconf-Meldungen
- erstmalige Umstellung: debconf-gettextize
- Build-Depends: po-debconf
- Verzeichnis debian/po/; eine Datei pro Sprache
- Updates: debconf-updatepo
- ungenaue oder fehlende Übersetzungen: podedebconf-report-po
- Übersetzerteams arbeiten meist automatisch und öffnen Bugsreports

# update configuration file (ucf)

- flexible Alternative zu dpkgs conffile-Mechanismus
- conffile-Behandlung von dpkg ist komplex und unvollständig dokumentiert
- ucf ist grundsätzlich ähnlich dazu
- debconf-Unterstützung
- Depends: ucf

# devscripts

- debchange (dch)

- debuild
- debsign; kann auch remote signieren
- in ~/.bashrc:

```
DEBEMAIL="Torsten Werner  
<twerner@debian.org>"; export DEBEMAIL
```

## automatische Abhängigkeiten mit `${*:Depends}`

- Vorbild: `${shlibs:Depends}`
- Beispiele: python-central, octave-depends, erlang-depends
- Depends: `${misc:Depends}`, `${perl:Depends}`, `${python:Depends}`, `${erlang-nox:Depends}`
- in debian/rules aufrufen
- python-central macht noch mehr (automatisches Kompilieren von \*.py-Dateien)

## chroot-Umgebungen mit pbuilder

- saubere sid-chroot-Umgebung zum Paketbau; findet falsche Build-Depends
- pbuilder create
- pbuilder update --autocleanaptcache
- pbuilder build paketname.dsc

installiert automatisch Build-Depends; unterhält einen Paketcache

- sudo notwendig
- mögliche Alternative zu pbuilder: dchroot (nicht automatisch)
- Konfiguration in /etc/pbuilderrc:  
BUILDUSERID=1234  
BUILDUSERNAME=pbuilder  
DEBEMAIL="..." s.u.
- Vorteil: sid-Umgebung auch wenn Host nicht mit sid läuft; Backports usw.

## Weitere Tipps

- lintian oder linda benutzen
- Target get-orig-source in debian/rules
- ev. debian/watch
- Kopfzeile X-XS-Vcs-Svn in debian/control
- Homepage: in der Paketbeschreibung
- LSB Initskripte
- "bashisms" vermeiden
- Shellskripte mit 'dash -n' immer auf Syntaxfehler prüfen