

Univention Paketerstellung

Thema:	Beschreibung der Paketerstellung für und mit Univention Corporate Server	
Datum:	11. April 2007	
Seitenzahl:	19	
Versionsnummer:	639	
Autoren:	Janis Meybohm Stefan Gohmann	meybohm@univention.de gohmann@univention.de

Inhaltsverzeichnis

1	Einführung	3
2	Voraussetzungen	3
3	Erzeugen eigener Pakete	3
4	Kontrolldateien	4
4.1	control	6
4.2	copyright	8
4.3	changelog	8
4.4	rules	9
4.5	preinst, prein, postinst, postrm	14
4.6	conffiles, dirs	17
5	Erzeugen des Paketes	17
6	Weiterführende Informationen	18

1 Einführung

Univention Corporate Server basiert auf der Debian-Distribution. Diese verfügt über das ausgereifte Paketverwaltungssystem `apt` (Advanced Packaging Tool). Es ermöglicht Pakete unter Berücksichtigung von Abhängigkeiten zu installieren und zu entfernen.

Univention Corporate Server besteht aus über 1000 Paketen. Diese Dokumentation soll helfen, eigene Pakete für UCS zu erstellen, um beispielsweise lokale Software oder Konfigurationen effizienter einzubinden.

Das unterliegende Debian-Paketformat bietet mächtige Methoden um komplexe Paketstrukturen abzubilden. Beispielsweise kann ein Paket zur Verwendung eines anderen vorausgesetzt werden, welches dann automatisch mitinstalliert werden kann.

2 Voraussetzungen

Vorausgesetzt wird, dass die folgenden Pakete auf Ihrem System installiert sind:

- `dh-make`
- `devscripts`
- `build-essential`

Sollte dies nicht der Fall sein, können die Pakete mit folgendem Befehl nachinstalliert werden:

```
apt-get install dh-make devscripts build-essential
```

3 Erzeugen eigener Pakete

Im Folgenden wird ein Paket erstellt, das nur ein Python-Skript enthält. Das Skript liegt unterhalb von `/tmp` eine Datei namens `testdeb-`, gefolgt von Datum und der Zeit der Ausführung des Skripts im Format `JJTTMMHHmm` an.

Zunächst sollte ein Arbeitsverzeichnis erstellt werden. Der Übersicht halber trägt es den Namen des zukünftigen Paketes (ohne Versionsnummer). Darin wird erneut ein Verzeichnis erstellt, das den Namen des Paketes inklusive der Versionsnummer trägt.

```
mkdir testdeb
mkdir testdeb/testdeb-0.1
cd testdeb/testdeb-0.1
```

Im untersten Verzeichnis des soeben erstellten Verzeichnisbaums wird die Datei `testdeb.py` erstellt, die das nachfolgende Python-Skript enthält, welches paketiert werden soll:

```
#!/usr/bin/env python
#
# Beispielskript Univention-Paketerstellung

import time, os

file='/tmp/testdeb-%s' % time.strftime('%y%m%d%H%M', time.localtime())
f=open(file,'a')
f.close()
```

Für die Erstellung des Paketes wird das Unterverzeichnis `debian/` benötigt, welches einige Kontrolldateien enthält, die nachfolgend genauer eingeführt werden. Das Erstellen des Verzeichnisses sowie der Entwürfe der benötigten Kontrolldateien kann über den Befehl `dh_make` durchgeführt werden:

```
dh_make -e ihre@emailadresse.de
```

`dh_make` fragt nach dem Start, um was für eine Art von Paket es sich handelt. Hier stehen vier Varianten zur Auswahl: **single binary**, **multiple binary**, **library** sowie **kernel module**. Anhand der vom Benutzer gemachten Eingabe werden einige Einstellungen in den Kontrolldateien vorgenommen. So wird z.B. bei einem "single binary"-Paket der Name des Quellverzeichnisses verwendet, um den Paketnamen (hier `testdeb`) sowie die Versionsnummer (hier `0.1`) zu setzen.

In dem hier aufgeführten Beispiel wird ein "single binary"-Paket erstellt. Nach der Eingabe von "s" gefolgt von einem ENTER, sollte eine Ausgabe, ähnlich folgender, erscheinen:

```
Type of package: single binary, multiple binary, library, or kernel module?
[s/m/l/k] s
```

```
Maintainer name : Max Muster
Email-Address   : ihre@emailadresse.de
Date            : Mon, 21 Mar 2005 13:46:39 +0100
Package Name    : testdeb
Version         : 0.1
Type of Package : Single
Hit <enter> to confirm:
Currently there is no top level Makefile. This may require additional tuning.
Done. Please edit the files in the debian/ subdirectory now. You should also
check that the testdeb Makefiles install into \${DESTDIR} and not in / .
```

Nach der Ausführung sollte in dem aktuellen Verzeichnis ein Unterverzeichnis namens `debian` vorhanden sein, welches Entwürfe für Kontrolldateien enthält. Sie werden im nächsten Kapitel genauer beschrieben.

4 Kontrolldateien

Nachfolgend wird ein kleiner Überblick über die möglichen Kontrolldateien und ihre Funktionen gegeben.

Datei	Funktion
README.Debian	Debian Readme-Datei. Wird nach /usr/share/doc/<Paket> kopiert. →dh_installdocs
changelog	Änderungshistorie des Paketes. Achtung: Diese Datei unterliegt strengen Syntax-Regeln.
conffiles.ex	Eine Liste der im Paket verwendeten Konfigurationsdateien. Konfigurationsdateien werden von dpkg während der Installation gesondert behandelt und nicht automatisch überschrieben. In Problemfällen wird der Benutzer um eine Lösung des Konfliktes gefragt. (<i>optional</i>)
control	Enthält Informationen über ein Paket wie z.B. den Paketnamen oder Paketabhängigkeiten. Diese Informationen können mittels dpkg -info abgefragt werden. Achtung: Diese Datei unterliegt strengen Syntax-Regeln.
copyright	Enthält Copyright des Paketerstellers und des Autors der Upstream-Version sowie weitere Lizenzinformationen.
cron.d.ex	Automatisch zu erstellende Crontab-Einträge. →dh_installcron (<i>optional</i>)
dirs	Eine Liste von Verzeichnissen, die ggf. von dpkg erstellt werden müssen. →dh_files (<i>optional</i>)
docs	Eine Liste von Dateien, die nach /usr/share/doc/<Paket> kopiert werden. →dh_installdocs (<i>optional</i>)
emacsen-install.ex	Emacs-spezifische Kontrolldatei, die die Installation, automatische Übersetzung in Binärformate, automatisches Laden sowie das Entfernen von Emacs-Lisp-Dateien steuert. →dh_installemacsen (<i>optional</i>)
emacsen-remove.ex	siehe emacsen-install.ex (<i>optional</i>)
emacsen-startup.ex	siehe emacsen-install.ex (<i>optional</i>)
ex.doc-base.package	Wird benötigt, um HTML-Dokumentationen bei doc-base zu registrieren. (<i>optional</i>)
init.d.ex	Kann ein Startup-Skript enthalten, welches beim Starten und Herunterfahren des Rechners ausgeführt wird. (<i>optional</i>)
manpage.1.ex	Beispielrumpf einer Manpage. →dh_installman (<i>optional</i>)
manpage.sgml.ex	Beispielrumpf einer Manpage (SGML-Format). →dh_installman (<i>optional</i>)
menu.ex	Ermöglicht Einträge in das Debian-Menü. (<i>optional</i>)
postinst.ex	Nach der Installation des Paketes auszuführendes Skript. (<i>optional</i>)
postrm.ex	Nach dem Entfernen des Paketes auszuführendes Skript. (<i>optional</i>)
preinst.ex	Vor der Installation des Paketes auszuführendes Skript. (<i>optional</i>)
prerm.ex	Vor dem Entfernen des Paketes auszuführendes Skript. (<i>optional</i>)
rules	Enthält das zentrale Regelwerk für dpkg-buildpackage zum Bauen des Paketes.

watch.ex	Überwacht den Server der Upstream-Version auf Neuerungen. (<i>optional</i>)
----------	---

Bei einigen Dateien wurde mit `→prgname` der Hinweis gegeben, daß die betreffende Datei während der Paketerstellung vom Programm bzw. Befehl `prgname` interpretiert oder kopiert wird. Dateien, welche auf `.ex` enden, sind Entwürfe für optionale Kontrolldateien. Werden sie benötigt, können sie angepasst und anschließend umbenannt werden, so daß die Endung `.ex` entfällt. Dateien mit der Endung `.ex` im Verzeichnis `debian/` werden beim Bau eines Paketes ignoriert.

4.1 control

Die `control`-Datei enthält verschiedene Informationen, die `dpkg` benötigt, um das Paket bearbeiten zu können. Die von `dh_make` erstellte `control`-Datei sollte in etwa dem Folgenden gleichen:

```
Source: testdeb
Section: unknown
Priority: optional
Maintainer: Max Muster <ihre@emailadresse.de>
Build-Depends: debhelper (>= 4.0.0)
Standards-Version: 3.6.1

Package: testdeb
Architecture: any
Depends: \${shlibs:Depends}, \${misc:Depends}
Description: <insert up to 60 chars description>
<insert long description, indented with spaces>
```

Der obere Block enthält Informationen für die Erstellung des Quell-Pakets, der untere für das Binär-Paket. Ein Quell-Paket kann mehrere Binär-Pakete erzeugen.

In der **Source**-Zeile wird der Name des Quellcode-Pakets angegeben. **Section**: und **Priority**: dienen der Kategorisierung und Priorisierung des Paketes (z.B. durch grafische Paketmanager). **Maintainer**: gibt Namen und Email-Adresse des Paketerstellers an.

Die Zeile **Build-Depends**: enthält die Namen von Paketen, die zur Übersetzung dieses Paketes zwingend benötigt werden. **Depends**: beschreibt eine Abhängigkeit zur Laufzeit des Paketes. Überdies gibt es noch weitere Paketrelationen:

- **Recommends**: — Empfehlung für optionale Pakete, die häufig zusammen mit dem Ursprungspaket eingesetzt werden und z.B. seine Funktion ergänzen.
- **Suggests**: — Vorschlag für optionale Pakete, die mit dem Ursprungspaket eingesetzt werden können.
- **Pre-Depends**: — Voraussetzung: die angegebenen Pakete müssen installiert und konfiguriert sein, bevor das Ursprungspaket installiert werden kann.

- **Conflicts:** – Konflikt: das Ursprungspaket kann nicht zusammen mit den hier aufgeführten Paketen installiert werden.
- **Provides:** – stellt zur Verfügung
- **Replaces:** – Ersetzung: das Ursprungspaket ersetzt die hier aufgeführten Pakete

Die Angabe der Paketnamen in diesen Feldern erfolgt nacheinander, durch Kommata getrennt. Ein Paketname kann aber auch aus einer Liste alternativer Pakete bestehen, die dann durch das Pipe-Zeichen "|" getrennt werden müssen.

Die Angabe eines Paketes kann auch mit einer exakten Versionsnummer erfolgen. Dafür stehen Ihnen folgende Vergleichsoperatoren zur Verfügung:

Zeichen	Bedeutung
<<	niedriger
<=	niedriger oder gleich
=	gleich
>=	höher oder gleich
>>	höher

Diese werden, zusammen mit der gewünschten Versionsnummer, direkt hinter den Paketnamen in Klammern gefasst angegeben. Ein Beispiel:

```
Depends: libc6 (>= 2.3.2.ds1-4), exim4 | mail-transport-agent
Conflicts: libgg0, libggi
Recommends: libncurses5 (>> 5.3)
Suggests: libgii0-target-x (= 1:0.8.5-2)
Replaces: vim-python (<< 6.0), vim-tcl (<= 6.0)
Provides: www-browser, news-reader
```

Standards-Version: enthält die Version des Debian-Policy-Standards und **Package:** den Namen eines Binär-Paketes. **Architecture:** gibt die CPU-Architektur an, für die dieses Paket gebaut werden soll. Belassen Sie es bei **any**, wird es von `dpkg-gencontrol` entsprechend ersetzt. Handelt es sich bei Ihrem Paket ausschließlich um Skripte oder Dokumentationen, die unabhängig der Architektur sind, können Sie **any** durch **all** ersetzen.

In der Zeile **Depends:** wird angegeben, welche zusätzlichen Pakete dieses Paket benötigt, um funktionstüchtig zu sein. Die Zeile **Description:** muss eine max. 60 Zeichen lange Kurzbeschreibung des Paketes enthalten. Darunter folgt, durch Leerzeichen am Zeilenanfang eingerückt, eine beliebig lange, ausführliche Beschreibung. Leerzeilen in der Beschreibung können Sie mit einem Leerzeichen und einem Punkt erzeugen.

Ergänzen Sie die `control`-Datei entsprechend dem folgenden Beispiel:

```
Source: testdeb
Section: univention
Priority: optional
Maintainer: Max Muster <ihre@emailadresse.de>
Build-Depends: debhelper (>= 4.0.0)
```

```
Standards-Version: 3.6.1

Package: testdeb
Architecture: all
Depends: \${shlibs:Depends}, \${misc:Depends}, \${python:Depends}
Description: Ein Beispieldokumentation.
Dieses Paket soll die Struktur von Debian Paketen beschreiben.
Außerdem erläutert die Dokumentation:
.
+ Grundsetzliches zu Debian/Univention Paketen.
+ Installationsverlauf.
+ Aufbau eines Paketes.
+ Inhalt und Funktion der Kontroll Dateien.
.
For more information about UCS, refer to:
http://www.univention.de
```

Es wurde eine zusätzliche Abhängigkeit auf **`python:Depends`** hinzugefügt. Diese Variable wird während der Paketerstellung von dem Debhelper-Programm `dh_python` durch die Namen der benutzten Python-Pakete ersetzt. `dh_python` untersucht dabei den Inhalt Ihres Paketes nach Python-Skripten und -Modulen und generiert aus diesen Informationen Abhängigkeiten für Ihr Paket.

4.2 copyright

Die `copyright`-Datei, die sich unterhalb des `debian`-Verzeichnisses befindet, enthält sowohl Lizenzinformationen des Paketes als auch des Original-Quellcodes. Dazu zählen auch die Kontaktadressen der Entwickler. Die von `dh_make` erstellte Version sollte ungefähr so aussehen:

```
This package was debianized by Max Muster <ihre@emailadresse.de> on
Mon, 21 Mar 2005 13:46:39 +0100.

It was downloaded from <fill in ftp site>

Copyright:
Upstream Author(s): <put author(s) name and email here>

License:
<Must follow here>
```

Sie können diese Datei nach Ihren Bedürfnissen anpassen. Syntax-Regeln müssen hier nicht beachtet werden.

4.3 changelog

Die `changelog`-Datei enthält eine Historie der Änderungen, die an diesem Paket vorgenommen wurden. Die von `dh_make` erzeugte Datei sollte ungefähr den folgenden Inhalt aufweisen:

```
testdeb (0.1-1) unstable; urgency=low

* Initial Release.

-- Max Muster <ihre@emailadresse.de> Mon, 21 Mar 2005 13:46:39 +0100
```

Wird eine neue Paketversion gebaut, muss zuvor die Versionsnummer erhöht und ein Kommentar zu den erfolgten Änderungen in dieser Datei eingetragen werden. Am einfachsten ist dies mit dem Programm `debchange` bzw. dem Alias `dch` zu erledigen, da es auch gleich die `changelog`-Datei in einem Editor öffnet und auf die Eingabe der Änderungen wartet. Vor dem Laden kann von `debchange` automatisch die Versionsnummer erhöht werden. Speichert man das Dokument und schließt den Texteditor wieder, werden die soeben angegebenen Informationen in der `changelog`-Datei gespeichert. Folgender Befehl, ausgeführt in Ihrem Arbeitsverzeichnis, löst diese Aktion aus:

```
dch -i
```

Danach sollte die `changelog`-Datei ungefähr so aussehen:

```
testdeb (0.1-2) unstable; urgency=low

* Nur eine Versionsnummer höher!

-- Max Muster <ihre@emailadresse.de> Mon, 21 Mar 2005 17:55:47 +0100

testdeb (0.1-1) unstable; urgency=low

* Initial Release.

-- Max Muster <ihre@emailadresse.de> Mon, 21 Mar 2005 13:46:39 +0100
```

Achtung:

Das Datums- und Zeitformat muss RFC822-konform sein. Manuell kann die aktuelle Uhrzeit/Datum mit dem Befehl `date -R` im RFC822-Format ausgegeben werden.



4.4 rules

Die `rules`-Datei enthält die eigentlichen Befehlssequenzen, mit denen `dpkg-buildpackage` das Paket schließlich erstellt. Die Datei ist ein Makefile. Die Datei enthält mehrere Regeln (Rules), die bestimmen, wie mit dem Quellcode verfahren werden soll. Generell ist eine Regel folgendermaßen aufgebaut:

```
ZIEL: VORAUSSETZUNG
      BEFEHL
      ...
```

Jede Regel besteht aus mindestens einem **ZIEL** (Target), das der Name einer Aktion (z.B. **build**: oder **install**:) oder ein Dateiname sein kann. Wildcards (z.B. `"*"`) in den Dateinamen sind ebenfalls erlaubt. Die **BEFEHL**-Zeile muß mit einem Tab beginnen und wird durch die Shell ausgeführt.

Diese Regel steuert das Verhalten von `make`. Sie gibt an, wann das Ziel veraltet ist und wie `make` es erneuern kann. Das Kriterium, nach dem `make` entscheidet, ob das Ziel veraltet ist, ist in der **VORAUSSETZUNG** beschrieben. Sie kann eine oder mehrere Dateien, ein anderes Ziel oder auch eine Datei innerhalb eines Archivs enthalten. Wie bei **ZIEL** sind auch hier Wildcards erlaubt.

Ein **ZIEL** ist veraltet, sobald es nicht existiert oder mindestens eine der Dateien der **VORAUSSETZUNG** neuer als das **ZIEL** ist. In einem solchen Fall wird mittels des unter **BEFEHL** angegebenen Kommandos das **ZIEL** neu erzeugt.

Zu beachten ist, dass `dh_make` lediglich ein Beispiel der `rules`-Datei erzeugt, das bei kleineren Paketen ohne Anpassungen funktionieren kann. Dennoch sollte immer der Inhalt dieser Datei überprüft werden. Die von `dh_make` generierte Datei sollte in etwa den folgenden Inhalt aufweisen:

```
#!/usr/bin/make -f
# -*- makefile -*-
# Sample debian/rules that uses debhelper.
# This file was originally written by Joey Hess and Craig Small.
# As a special exception, when this file is copied by dh-make into a
# dh-make output file, you may use that output file without restriction.
# This special exception was added by Craig Small in version 0.37
# of dh-make.

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

CFLAGS = -Wall -g
ifneq (,$(findstring noopt,$(DEB_BUILD_OPTIONS)))
    CFLAGS += -O0
else
    CFLAGS += -O2
endif

configure: configure-stamp
configure-stamp:
    dh_testdir
    # Add here commands to configure the package.

    touch configure-stamp

build: build-stamp

build-stamp: configure-stamp
    dh_testdir
    # Add here commands to compile the package.
    $(MAKE)
    #docbook-to-man debian/testdeb.sgml > testdeb.1
    touch build-stamp

clean:
    dh_testdir
    dh_testroot
    rm -f build-stamp configure-stamp
```


meinen vorzugehen ist. Sie müssen ggf. herausfinden, wie das Einfügen und Entfernen von Debugging-Informationen im Upstream-Quellcode gehandhabt wird und es selbst integrieren.

Die Regeln **configure** und **configure-stamp** sind für die Konfiguration des Paketes zuständig. Wenn das Quellpaket autoconf/automake verwendet, sind dort evtl. schon Schritte angegeben, die dh_make aus der config-Datei des Quellpaketes generiert hat. **build** und **build-stamp** steuern den eigentlichen Erstellungsprozess des Paketes. Die Regel **clean** verwaltet die notwendigen Schritte zum Entfernen temporärer Dateien aus dem Paketbaum.

binary-indep erstellt, falls vorhanden, architekturunabhängige Pakete. Ist das bei Ihrem Paket nicht der Fall, ist hier keine Änderung notwendig, **binary-arch** dementsprechend architekturabhängige Paket-Bestandteile.

Da in diesem Paket in der control-Datei **Architecture:** auf **all** gesetzt wurde, sollten alle Anweisungen zum Erzeugen des Paketes in der Regel **binary-indep** platziert werden. **binary-arch** bleibt in diesem Fall leer.

Unterhalb der **binary-arch**-Regel finden sich eine Menge mit dh_ beginnende Debhelper-Programme. In den meisten Fällen gibt der Name ersten Aufschluss über ihre Funktion. Nähere Informationen finden Sie in den Manpages des jeweiligen Programms. Sie können sich diese mit dem allgemein gültigen Befehl man anzeigen lassen. Zum Beispiel:

```
man dh_testdir
```

Da eine Beschreibung aller Debhelper-Programme einen enormen Umfang annehmen würde, werden im Folgenden nur die wichtigsten Programme kurz aufgeführt:

Programmname	Beschreibung
dh_testdir	Überprüft, ob Sie sich im obersten Verzeichnis des Quellcode-Verzeichnissesbaums befinden.
dh_testroot	Überprüft, ob Sie die für die Ziele binary-arch , binary-indep und clear benötigten root-Rechte besitzen.
dh_installmanpages	Installiert die Manpages. Es muss der Pfad zu den Manpages relativ zum obersten Verzeichnis angegeben werden.
dh_installdocs	Installiert copyright, README.Debian und TODO.Debian.
dh_installdeb	Installiert die Maintainer-Skripte in debian/<Paket>/DEBIAN.
dh_strip	Entfernt nicht benötigte Debug-Header aus Bibliotheken und Programmen. (wird nur bei Binärprogrammen benötigt)
dh_gencontrol	Kopiert eine angepasste control-Datei nach debian/<Paket>/DEBIAN.
dh_md5sums	Generiert MD5-Prüfsummen für jede Datei in dem Paket.
dh_fixperms	Korrigiert Dateirechte.
dh_builddeb	Erstellt mittels dpkg das Paket.
dh_installinfo	Installiert Info-Seiten.
dh_examples	Kopiert Beispieldateien.
dh_shlibdeps	Prüft Abhängigkeiten auf Bibliotheken und Binär-Dateien.

dh_python	Durchsucht Python-Skripte und -Module und generiert aus gefundenen Informationen Abhängigkeiten für das Paket. <i>{python:Depends}</i> wird in der control-Datei ersetzt. (Siehe 4.1.)
-----------	---

Erweitern Sie die rules-Datei entsprechend dem Beispiel. Es sind nicht viele Veränderungen notwendig, da lediglich das Python-Skript nach /usr/bin kopiert werden soll. Zusätzlich wird eine Manpage angelegt, die besagt, dass dieses Paket nicht dokumentiert ist.

```
#!/usr/bin/make -f
# -- makefile --
# Sample debian/rules that uses debhelper.
# This file was originally written by Joey Hess and Craig Small.
# As a special exception, when this file is copied by dh-make into a
# dh-make output file, you may use that output file without restriction.
# This special exception was added by Craig Small in version 0.37
# of dh-make.

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

CFLAGS = -Wall -g
ifneq (,$(findstring noopt,$(DEB_BUILD_OPTIONS)))
    CFLAGS += -O0
else
    CFLAGS += -O2
endif

configure: configure-stamp
configure-stamp:
    dh_testdir
    # Add here commands to configure the package.

    touch configure-stamp

build: build-stamp

build-stamp: configure-stamp
    dh_testdir
    # Add here commands to compile the package.
    $(MAKE)
    #docbook-to-man debian/testdeb.sgml > testdeb.1
    touch build-stamp

clean:
    dh_testdir
    dh_testroot
    rm -f build-stamp configure-stamp
    # Add here commands to clean up after the build process.
    -$(MAKE) clean
    dh_clean

install: build
```

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

install -m 0755 testdeb.py debian/testdeb/usr/bin/

# Build architecture-independent files here.
binary-indep: build install
dh_shlibdeps
dh_python
dh_md5sums
dh_gencontrol
dh_installdeb
dh_builddeb

# Build architecture-dependent files here.
binary-arch: build install

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

Mit dem Befehl

```
install -m 0755 testdeb.py debian/testdeb/usr/bin/
```

wird die in Ihrem Arbeitsverzeichnis befindliche Datei `testdeb.py` relativ zum Paketverzeichnis in `debian/testdeb/usr/bin/` kopiert und bekommt die Dateizugriffsrechte 0755 (rwxr-xr-x) zugewiesen.

4.5 preinst, prerm, postinst, postrm

Es existieren vier so genannte Maintainer-Skripte, die zu unterschiedlichen Zeitpunkten während der Paketinstallation bzw. -deinstallation aufgerufen werden:

- `preinst` — vor der Installation
- `prerm` — vor dem Entfernen
- `postinst` — nach der Installation
- `postrm` — nach dem Entfernen

Sie sind im Grunde einfache Shell-Skripte, die von `dh_make` schon mit einem Grundgerüst gefüllt wurden. Um das im Paket befindliche Python-Skript nach erfolgreicher Installation zu starten, müssen Anpassungen in der `postinst`-Datei vorgenommen werden. Die von `dh_make` generierte Version sollte folgenden Inhalt haben:

```
#!/bin/sh
# postinst script for testdeb
#
# see: dh_installdeb(1)

set -e

# summary of how this script can be called:
# * <postinst> 'configure' <most-recently-configured-version>
# * <old-postinst> 'abort-upgrade' <new version>
# * <conflictor's-postinst> 'abort-remove' 'in-favour' <package>
#   <new-version>
# * <deconfigured's-postinst> 'abort-deconfigure' 'in-favour'
#   <failed-install-package> <version> 'removing'
#   <conflicting-package> <version>
# for details, see http://www.debian.org/doc/debian-policy/ or
# the debian-policy package
#

case "$1" in
    configure)

        ;;

        abort-upgrade|abort-remove|abort-deconfigure)

        ;;

        *)
            echo "postinst called with unknown argument \`$1'" >&2
            exit 1
        ;;
esac

# dh_installdeb will replace this with shell code automatically
# generated by other debhelper scripts.

#DEBHELPER#

exit 0
```

Grundsätzlich kann ein Aufruf, der nach einer erfolgreichen Installation ausgeführt werden soll, innerhalb des **configure**-Blocks erfolgen, da das Paket auf jeden Fall während der Installation konfiguriert wird. Weitere Blöcke mit Übergabeparametern wurden bereits von `dh_make` erzeugt und brauchen in diesem Fall nicht beachtet werden.

Der Eintrag **#DEBHELPER#** wird, wie in den darüber stehenden Kommentaren beschrieben, automatisch von den Debhelper-Programmen ersetzt. Er sollte nicht entfernt werden.

Im nachfolgenden Beispiel ist lediglich ein Aufruf zum Starten des Skripts `testdeb.py` einzufügen. Zusätzlich wird die Baseconfig-Variable **update/server** auf **<http://bitz150.bitz.briteline.de>** gesetzt. Das Fragezeichen nach der Variable weist Baseconfig an, die Variable nur zu setzen, wenn sie noch keinen Wert enthält. Ist bereits ei-

ne Version kleiner der "0.1-2" installiert, wird die Baseconfig-Variable **timeserver1** auf **time.fu-berlin.de** gesetzt.

```
#!/bin/sh
# postinst script for testdeb
#
# see: dh_installdeb(1)

set -e

# summary of how this script can be called:
# * <postinst> 'configure' <most-recently-configured-version>
# * <old-postinst> 'abort-upgrade' <new version>
# * <conflictor's-postinst> 'abort-remove' 'in-favour' <package>
#   <new-version>
# * <deconfigured's-postinst> 'abort-deconfigure' 'in-favour'
#   <failed-install-package> <version> 'removing'
#   <conflicting-package> <version>
# for details, see http://www.debian.org/doc/debian-policy/ or
# the debian-policy package
#

case "$1" in
    configure)
        /usr/bin/testdeb.py
        univention-baseconfig set \
            update/server?http://bitz150.bitz.briteline.de
        if [ -n "$2" ] && dpkg --compare-versions "$2" lt 0.1-2; then
            univention-baseconfig set timeserver1?time.fu-berlin.de
        fi
        ;;

    abort-upgrade|abort-remove|abort-deconfigure)
        ;;

    *)
        echo "postinst called with unknown argument \`$1'" >&2
        exit 1
        ;;
esac

# dh_installdeb will replace this with shell code automatically
# generated by other debhelper scripts.

#DEBHELPER#

exit 0
```

Die anderen Maintainer-Skripte verhalten sich analog zu postinst, werden aber an anderer Stelle von dpkg ausgeführt. Nicht benötigte Skripte können gefahrlos entfernt bzw. auskommentiert werden.

4.6 conffiles, dirs

Da dieses Paket keine Konfigurationsdateien enthält, die installiert werden müssen, sind Änderungen an der `conffiles`-Datei nicht erforderlich. Das von `dh_make` erzeugte Beispiel weist auf das benötigte Format der Datei hin:

```
#
# If you want to use this conffile, remove all comments and put files that
# you want dpkg to process here using their absolute pathnames.
# See the policy manual
#
# for example:
# /etc/testdeb/testdeb.conf
```

Die Konfigurationsdateien müssen also mit absoluten Pfadangaben in der Datei angegeben werden. Falls, wie in diesem Fall, die Datei nicht verwendet wird, kann sie entfernt werden. Die `dirs`-Datei enthält evtl. von `dpkg` zu erstellende Verzeichnisse. Das können alle Verzeichnisse sein, in die das Paket schreibt, da nicht gewährleistet ist, dass das entsprechende Verzeichnis auf dem Zielsystem existiert. Üblicherweise ist diese Datei von `dh_make` mit folgendem Inhalt erzeugt worden:

```
usr/bin
usr/sbin
```

Geben Sie weitere Verzeichnisse immer ohne den führenden Schrägstrich `"/` an.

5 Erzeugen des Paketes

Bevor der eigentliche Erstellungsprozess des Paketes beginnen kann, sollte das Paketverzeichnis von nicht verwendeten Dateien "gesäubert" werden. Grundsätzlich können Dateien unterhalb von `debian/`, die auf `.ex` enden (`.ex` steht für `example`, d.h. Beispiel) und nicht benötigt werden, gelöscht werden. Dateien die die Endung `.ex` tragen, aber im Paket verwendet werden sollen, müssen spätestens jetzt umbenannt werden, so dass sie keine Endung `.ex` mehr aufweisen. Dies kann z.B. mit dem Befehl

```
mv preinst.ex preinst
```

durchgeführt werden.

Sind alle nicht verwendeten Dateien entfernt, kann der Erstellungsprozess gestartet werden. Dazu muss folgender Befehl im Paket-Quellverzeichnis (`testdeb/testdeb-0.1/`) ausgeführt werden:

```
dpkg-buildpackage -rfakeroot
```

Es sollte eine Ausgabe ähnlich der folgenden erscheinen:

```
pkg-buildpackage: source package is testdeb
dpkg-buildpackage: source version is 0.1-1
dpkg-buildpackage: source maintainer is Max Muster <ihre@emailadresse.de>
dpkg-buildpackage: host architecture is i386
```

```
fakeroot debian/rules clean
dh_testdir
dh_testroot
rm -f build-stamp configure-stamp
# Add here commands to clean up after the build process.
dh_clean
  dpkg-source -b testdeb-0.1
dpkg-source: building testdeb in testdeb_0.1.orig.tar.gz
dpkg-source: building testdeb in testdeb_0.1-1.diff.gz
dpkg-source: building testdeb in testdeb_0.1-1.dsc
  debian/rules build
dh_testdir
# Add here commands to configure the package.
touch configure-stamp
dh_testdir
# Add here commands to compile the package.
#docbook-to-man debian/testdeb.sgml > testdeb.1
touch build-stamp
  fakeroot debian/rules binary
dh_testdir
dh_testroot
dh_clean -k
dh_installdirs
# Add here commands to install the package into debian/testdeb.
install -m 0755 testdeb.py debian/testdeb/usr/bin/
dh_shlibdeps
dh_python
dh_md5sums
dh_gencontrol
dpkg-gencontrol: warning: unknown substitution variable ${shlibs:Depends}
dpkg-gencontrol: warning: unknown substitution variable ${misc:Depends}
dh_installdeb
dh_builddeb
dpkg-deb: building package 'testdeb' in './testdeb_0.1-1_all.deb'.
  dpkg-genchanges
dpkg-genchanges: not including original source code in upload
dpkg-buildpackage: binary and diff upload (original source NOT included)
```

Das fertig erstellte Paket liegt nun eine Verzeichnisebene höher. Es kann mittels

```
dpkg -i ../testdeb_0.1-1_all.deb
```

installiert werden. Nach der Installation sollte sich im /tmp-Verzeichnis eine Datei befinden, die mit testdeb- beginnt und mit dem aktuellen Datum plus Uhrzeit endet. Diese Datei ist der Beweis dafür, dass das Skript nach Beendigung des Installationsprozesses vom postinst-Skript aufgerufen wurde.

6 Weiterführende Informationen

- “Debian New Maintainers’ Guide”, Josip Rodin, Jaldhar Vyas, Will Lowe
<http://www.debian.de/doc/manuals/maint-guide/index.de.html>

- “Debian Policy Manual”; Ian Jackson, Christian Schwarz
<http://www.debian.org/doc/debian-policy/>
- “Debian Developer’s Reference”; Andreas Barth, Adam Di Carlo, Raphaël Hertzog, Christian Schwarz, **<http://www.debian.org/doc/developers-reference/>**