Gut verpackt ist halb gewonnen

von Sven Rudolph und Heiko Schlittermann

Juli 1997

Die Verwaltung von Software-Paketen ist ein Highlight von Debian GNU/Linux. Der Artikel gibt einen Überblick über Anwendung, Struktur und Arbeitsweise des Systems. An einem Beispiel wird gezeigt, wie man selbst in den Quelltexten Fehler korrigieren und ein neues Paket bauen kann.

1 Die ersten Worte

1.1 Prähistorische Zeiten

Es war einmal vor langer Zeit, da installierte man Linux von zwei Disketten, und die Versionsnummer war 0.96b. Wer den Kern selbst übersetzen wollte, mußte den gcc und die binutils zu Fuß installieren. War das geschafft, konnte man sich einen eigenen Kern bauen oder andere Programme übersetzen.

Wer es nicht geschafft hat, mußte sich mit *cp*, *rm* und ein paar anderen Programmen zufriedengeben. Bald erschien die erste Distribution, nach ihrem Entstehungsort, dem Manchester Computing Centre, MCC genannt. Die ersten beiden Disketten entsprachen obigem Paar, drei weitere Disketten enthielten das langersehnte Entwicklungssystem.

Die erste Distribution, die in nennenswertem Umfang Anwendungen enthielt, war die SLS (von Softlanding Systems). Die gesamte Distribution belegte 20 Disketten, und man konnte auswählen, welche Programme installiert werden sollten.

Heutzutage gibt es (erfreulicherweise) eine große Auswahl an Software für Linux. Auf einem Linux-System können über tausend Programme installiert sein, die insgesamt über hunderttausend Dateien mitbringen. Selbst für einen erfahrenen Linux-Anwender ist es ein immenser Aufwand, alle diese Programme aus den Quelltexten zu installieren. Den meisten Linux-Anwendern fehlen ohnehin Zeit oder Erfahrung, ein System dieser Größe vollständig selbst zu warten.

1.2 Paketdienste

Deshalb werden Linux-Distributionen erstellt, die dem einzelnen diese Arbeit abnehmen, indem sie Programme in kompilierter und leicht installierbarer Form in Software-Paketen zusammenfassen. Ein Paket umfaßt im Normalfall alle Dateien, die zu einem bestimmten Programmpaket gehören (Programme, Bibliotheken, Handbuchseiten ...), und zusätzlich Steuerinformationen für das Paket-System. Da selten wirklich alle Pakete einer Distribution benötigt werden, sollte es möglich sein, bei der Installation nur bestimmte Pakete auszuwählen. Ebenso sollte es möglich sein, auch nach der Installation Pakete zu installieren bzw. zu entfernen. Ebenso muß ein verlustfreies Update bereits installierter Pakete möglich sein.

1.3 Wertschöpfung

Den Wert eines Paket-Systems machen aber erst die vielen Dinge aus, die über die Grundfunktionalität hinausgehen:

Abhängigkeiten Wenn ein Programmpaket für seine Arbeit auf andere Pakete angewiesen ist (z.B. benötigt der Editor Vim die NCurses-Bibliothek), dann sollten diese Abhängigkeiten bei der Installation berücksichtigt werden.

„Bootless" Install Wenn man einen Daemon installiert (z.B. Sendmail oder einen WWW-Server), dann soll der Daemon bei der Installation, und nicht erst nach dem nächsten Booten, automatisch gestartet werden. Das Paket-System muß also im Rahmen der Installation Pakte-spezifische Skripte ausführen.

Konfigurationsdateien Wer zum Beispiel seine WWW-Server-Konfiguration mühevoll angepaßt hat, will diese bei einem Update des WWW-Servers erhalten wissen. Das Paket-System muß beachten, daß es diese Konfigurationsdatei nicht einfach überschreiben darf.

1.4 Anforderungen an die Paketverwaltung in Debian

Systeme zur Verwaltung von Software-Paketen sind in der Linux-Welt nicht neu. SLS und Slackware hatten bereits ein einfaches System, und es gibt auch Portierungen des in kommerziellen Unix-Systemen genutzten System V Paket-Systems (pkgadd etc.). In kommerziellen Unix-Systemen gibt es auch andere Paket-Systeme, die aber unter Linux nicht verfügbar sind.

Welche einzelnen Anforderungen muß das Paket-System erfüllen ?

Robustheit Linux wird inzwischen erfolgreich von Leuten genutzt und auch administriert, die vorher nie mit Unix zu tun hatten. Deshalb ist es wichtig, es möglichst schwer zu machen, eine funktionierende Installation zu beschädigen. Wenn ein Paket ein anderes benötigt (z.B. läuft *ghostview* nur, wenn *ghostscript* installiert ist), soll das benötigte Programm automatisch mit installiert werden. Analog muß die Löschung eines Pakets verhindert werden, solange es von einem anderen installierten Paket benötigt wird. Außerdem soll es unmöglich sein, ein unbedingt benötigtes Paket (z.B. *init* oder die Shell) gelöscht wird.

Einfache Bedienung Debian soll ohne großen Lernaufwand nutzbar sein. Dabei ist es schwierig, die Bedienung für den Neuling einfach zu gestalten, ohne daß sich fortgeschrittene Anwender über fehlende Funktionalität ärgern müssen. Ein Ausweg ist, verschiedene Ebenen von Werkzeugen bereitzustellen, um sowohl die einfache Bedienung zu ermöglichen als auch vollständigen Zugriff auf die unvermeidbar vorhandene Komplexität bereitzustellen.

Upgrade-Fähigkeit Die Linux-Entwicklung ist sehr dynamisch. Von praktisch allen Programmen eines gängigen Linux-System erscheinen mehrmals im Jahr neue Versionen. Darüber hinaus ist es unvermeidbar, daß Pakete Fehler enthalten. Somit kann es schnell passieren, daß in einem Jahr über zehn neue Versionen eines Debian-Pakets erscheinen.

Deshalb müssen Upgrades einfach und ohne neues Booten des Rechners möglich sein. Es soll sichergestellt werden, daß ein Programm nach dem Upgrade korrekt läuft.

Komplexe Updates, wie z.B. der Umstieg von a.out auf ELF (so geschehen mit Debian 0.93 $\rightarrow$ 1.1) sollen möglich sein *ohne* Neuinstallation.

Inkrementelle Upgrades Bei kommerziellen Unix-Systemen ist es meist so, daß einmal im Jahr (oder noch seltener) eine neue Release erscheint. In der Zwischenzeit erscheinen bei einschneidenden Problemen hinsichtlich Funktionalität oder Rechnersicherheit sogenannte Patches. Wenn ein Problem nicht „groß genug“ ist, wird es also unter Umständen erst nach einem Jahr vom Hersteller gelöst. (Sicher gibt es für viel Geld Wartungsverträge ...)

Aufgrund der hohen Dynamik im Linux-Umfeld ist es für Debian wichtig, daß Fehler schnell korrigiert werden und diese Änderungen einfach verteilt und auf den Systemen eingespielt werden können. Wenn man nur einen bestimmten Fehler korrigieren will, soll man nicht gezwungen sein, alle anderen bis dahin erschienenen Änderungen ebenfalls einzuspielen. Aktualisierungen in Paketen sollen also voneinander möglichst unabhängig sein, um beim Lösen eines bestimmten Problems die Wahrscheinlichkeit unerwarteter Probleme zu minimieren. (Wenn man das Paket über ein Modem nach Hause übertragen muß, spart man so auch noch Telefonkosten.)

Verwaltung des gesamten Systems Das Paket-System soll in der Lage sein, das Debian-System vollständig zu verwalten. Insbesondere soll kein extra Basis-System erstellt werden, daß sich der Paketverwaltung entzieht und dessen Inhalt deshalb nicht einfach upgradebar ist.

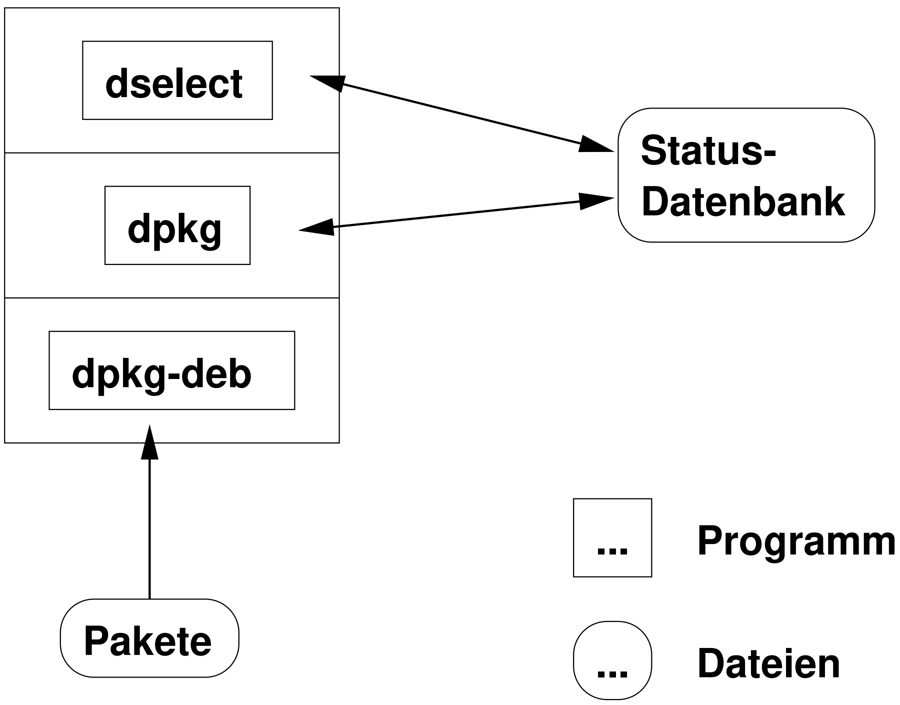
Verschiedene Rechnerarchitekturen Linux ist für verschiedene Architekturen verfügbar, und Debian ist für einige dieser Architekturen verfügbar. Das Paket-System muß mindestens sicherstellen, daß in jedem Paket die Information enthalten ist, auf welcher Architektur es installiert werden kann. Diese Information wird dann beim Installieren von Paketen geprüft.

Verwaltung der Quelltexte, Verteilte Entwicklung Im Unterschied zu kommerziellen Unix-Systemen sind in Debian zu allen Paketen auch die Quellen verfügbar. Diese Quellen werden in Source-Paketen zusammengefaßt und verteilt. Es soll jedem möglich sein, aus diesen Source-Paketen die Binär-Pakete zu erstellen. Die Verteilung der Entwicklung wird in Debian so gestaltet, daß für jedes Paket ein Maintainer zuständig ist. Ein Paket stellt damit eine relativ selbständige Einheit dar. Dadurch können die Maintainer unabhängig voneinander arbeiten, was die Verteilung der Entwicklungsarbeit ermöglicht.

Die Anforderungen der Debian-Entwicklung unterscheiden sich insbesondere in der Handhabung der Quelltexte, der verteilten Entwicklung und der Upgrade-Häufigkeit stark gegenüber kommerziellen Unix-Systemen und auch anderen freien Unix-ähnlichen Betriebssystem-Projekten. Es existierte kein freies Paket-System, das diese Anforderungen erfüllte. Deshalb war es nötig, für Debian ein eigenes Paket-System zu entwickeln.

1.5 Architektur

Um die geforderte Einfachheit der Bedienung und den Zugriff auf die komplexe Funktionalität zu realisieren, wurde die Debian-Paketverwaltung modular aufgebaut Sie besteht aus den drei Programmen *dpkg-deb*, *dpkg* und *dselect*, die jeweils Operationen verschiedener Abstraktions-Ebenen und unterschiedlicher Komplexität bereitstellen. Die von den Programmen benötigten Informationen über den Zustand installierter und verfügbarer Pakete werden in einer Statusdatenbank gespeichert.



dpkg-deb stellt auf der untersten Ebene den Zugriff auf das Debian-Paketformat bereit. Es kann Informationen über das Paket anzeigen und Daten aus dem Paket auspacken. Durch diese Abstraktion müssen die anderen Programme den internen Aufbau eines Debian-Pakets nicht kennen. (*dpkg-deb* wird auch genutzt, um Pakete zu erstellen, dazu später.)

Die Arbeitsteilung zwischen *dpkg* und *dselect* ist weniger streng und weniger offensichtlich. *dselect* ist das primäre Nutzerinterface, und es dient dem Auswählen von zu installierenden oder zu löschenden Paketen. *dpkg* dient dem Installieren, Entfernen und Upgraden von Paketen, und es kann zur Recherche in der Statusdatenbank genutzt werden.

dpkg und *dselect* greifen auf die Statusdatenbank zu. Diese speichert unter anderem für jedes Paket den aktuellen Zustand und die gewünschte Aktion. (Der Zustand kann u.a. die Werte *Not installed* oder *Installed* haben, die Aktion kann u.a. *Install* oder *Remove* sein.)

dselect kann nur die gewünschte Aktion modifizieren, nie den aktuellen Zustand! Wenn mittels *dselect* ein Paket zum Installieren ausgewählt wird, setzt *dselect* die Aktion in der Datenbank auf *Install*. Anschließend kann *dpkg* (mit entsprechenden Optionen) aufgerufen werden, und *dpkg* führt diese Aktion durch; im Ergebnis ist das Paket im Zustand *Installed*.

Wird ein Paket direkt mit *dpkg* bearbeitet, setzt *dpkg* sowohl die Aktion, die es gerade durchführt, als auch den Zustand. *dpkg* kann also im Unterschied zu *dselect* beide Werte ändern.

2 dpkg anwenden

dpkg deckt mit einer Unmenge von Optionen weite Bereiche der Paketverwaltung ab. Wir werden uns hier auf die Grundfunktionen beschränken.

2.1 Installieren, Upgraden, Löschen

Die erste Funktion, die man von einem Paket-System erwartet, ist das Installieren von Paketen. Man nehme *dpkg -i* (Das Paket `hello` ist noch nicht installiert) :

```
# dpkg -i hello_1.3-12.deb
Selecting previously deselected package hello.
(Reading database ... 111404 files and directories currently installed.)
Unpacking hello (from hello_1.3-12.deb) ...
Setting up hello (1.3-12) ...
```

Nun sei eine neuere Version des Pakets `hello` erschienen, die ein paar Bugs behebt. Ein Upgrade wird ebenfalls mit der Option *-i* durchgeführt. *dpkg* hat in der Status-Datenbank vermerkt, daß `hello` bereits installiert ist, und führt deshalb ein Upgrade durch:

```
# dpkg -i hello_1.3-13.deb
(Reading database ... 111593 files and directories currently installed.)
Preparing to replace hello 1.3-12 (using hello_1.3-13.deb) ...
Unpacking replacement hello ...
Setting up hello (1.3-13) ...
```

Wenn einem das im Paket `hello` enthaltene Programm zu uninteressant sein sollte, kann es gelöscht werden:

```
# dpkg -r hello
(Reading database ... 111594 files and directories currently installed.)
Removing hello ...
```

Beim Löschen bleiben die Konfigurationsdateien erhalten. Sollen sie auch gelöscht werden, muß man *dpkg –purge* aufrufen.

Weitere Optionen sind wichtig, wenn man mehrere Pakete auf einmal installieren oder updaten möchte:

-R	*--recursive <dir>*	alle Dateien unterhalb <dir> durchlaufen
-O	*--selected-only*	nur Pakete mit gewünschtem Zustand gleich Install bearbeiten
-E	*--skip-same-version*	Datei überspringen, wenn diese Version bereits installiert ist
-G	*--refuse-downgrade*	Datei überspringen, wenn diese Version oder eine höhere bereits installiert ist

2.2 Recherchieren

Die Status-Datenbank enthält viele Informationen über die installierten Pakete. Mit *dpkg* kann man sich diese Informationen anzeigen lassen und in dieser Datenbank suchen.

Anzeigen der allgemeinen Informationen zu einem Paket `$ dpkg -s hello`

```
Package: hello
Status: install ok installed
Priority: optional
Section: misc
Installed-Size: 31
Maintainer: Ian Jackson <ian@chiark.greenend.org.uk>
Version: 1.3-13
Depends: libc5 (>= 5.2.18)
Description: The classic greeting, and a good example
 The GNU hello program produces a familiar, friendly greeting.  It
 allows nonprogrammers to use a classic computer science tool which
 would otherwise be unavailable to them.
 .
 Seriously, though: this is an example of how to do a Debian package.
 It is the Debian version of the GNU Project's ‘hello world’ program
 (which is itself an example for the GNU Project).
```

Übersicht über den Zustand von Paketen Mit *dpkg -l* kann ein Überblick über installierte Pakete angegeben werden. Gibt man keinen Paketnamen an, werden alle Pakete ausgegeben; im Beispiel lassen wir uns nur die Daten zu `hello` ausgeben:

```
$ dpkg -l hello
Desired=Unknown/Install/Remove/Purge
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
||/ Name            Version        Description
+++-===============-==============-============================================
ii  hello           1.3-13         The classic greeting, and a good example
```

In der ersten Spalte steht die gewünschte Aktion:

u	Unknown	unbekannt (keine Aktion)
i	Install	installieren
r	Remove	löschen
p	Purge	löschen (einschließlich Konfigurationsdaten)

In der zweiten Spalte steht der aktuelle Zustand:

n	Not installed	nicht installiert
i	Installed	installiert
c	Config-files	nicht mehr installiert, aber die Konfigurationsdateien sind noch installiert
u	Unpacked	ausgepackt, aber noch nicht konfiguriert
f	Failed-Config	Fehler bei Konfiguration aufgetreten
h	Half-installed	Paket unvollständig installiert

In der dritten Spalte folgt gegebenenfalls eine nähere Beschreibung des Fehlers, darauf folgen der Paket-Name, die Versionsnummer und die Kurzbeschreibung des Pakets.

Anzeigen aller Dateien eines Pakets Mit *dpkg -L* werden alle Dateien des entsprechenden Pakets angezeigt.

```
$ dpkg -L hello
/.
/usr
/usr/doc
/usr/doc/hello
/usr/doc/hello/copyright
/usr/doc/hello/changelog.gz
/usr/doc/hello/changelog.Debian.gz
/usr/bin
/usr/bin/hello
/usr/info
/usr/info/hello.info.gz
```

Suchen des Pakets, in dem eine Datei enthalten ist Wenn man in irgendeiner Datei einen Fehler gefunden hat oder die Dokumentation dazu sucht, braucht man oft den Namen des Pakets, zu dem die Datei gehört. *dpkg -S* hilft:

```
$ dpkg -S bin/hello
Motif: /usr/X11R6/bin/hellomotif
Motif: /usr/X11R6/bin/helloint
hello: /usr/bin/hello
```

Das Argument wird als *glob(3)*-Pattern behandelt, wie sie von der Shell bekannt sind. Deshalb werden alle Dateien ausgegeben, in denen die Zeichenkette enthalten ist. Man kann als Argument auch Wildcards angeben, dann muß man sie aber entsprechend vor der Shell schützen:

```
$ dpkg -S bin/\*llo
Motif: /usr/X11R6/bin/hellomotif
mush: /usr/bin/maillock
Motif: /usr/X11R6/bin/helloint
kbd: /usr/bin/disalloc
dmalloc: /usr/bin/dmalloc
hello: /usr/bin/hello
login: /usr/bin/faillog
```

2.3 Installations-Skripte

Wird ein Paket installiert, so können vor und nach dem Auspacken Skripte ausgeführt werden (das Preinst- und das Postinst-Skript).

Wenn Pakete extra konfiguriert werden müssen, geschieht das normalerweise im Postinst-Skript. Da die Skripte so untelepathisch sind, daß sie nicht einmal rauskriegen, welche Maus man angeschlossen hat und ob man A4-Papier im Drucker liegen hat, fragen sie den Anwender. Das Paket `libpaper` dient zum Beispiel der Einstellung des bevorzugten Papierformats. In Deutschland wird üblicherweise Papier im Format DIN A4 genutzt:

```
# dpkg -i libpaper_1.0.3-3.deb
Selecting previously deselected package libpaper.
(Reading database ... 32533 files and directories currently installed.)
Unpacking libpaper (from libpaper_1.0.3-3.deb) ...
Setting up libpaper (1.0.3-3) ...

The default (also known as system) paper can be chosen from many known
papers that are currently recognized by programs using the libpaper
library (with libpaper, paper names are case insensitive; if you use
programs that use the system paper size but do not rely on the libpaper
library, this may not be true and some of the papers listed below may
not be known by these programs):

    a3 a4 letter note legal executive halfletter halfexecutive 11x17
    statement folio quarto 10x14 ledger tabloid a0 a1 a2 a5 a6 a7 a8 a9
    a10 b0 b1 b2 b3 b4 b5 c5 DL Comm10 Monarch archE archD archC archB
    archA flsa flse csheet dsheet esheet

Default paper name? [letter] a4
```

Vielen Pakete kann man auch nachträglich neu konfigurieren, bei `libpaper` muß man dafür *paperconfig* aufrufen.

2.4 Konfigurationsdateien

Wenn man ein Paket upgraded, ist es ärgerlich, wenn die mühsam zusammengebastelte Konfiguration überschrieben wird. Wenn man zum Beispiel sendmail upgraded, sollen die in */etc/aliases* eingetragenen Aliase nicht überschrieben werden. Auch die Dateien */etc/passwd* und */etc/group* werden als Konfigurationsdatei verwaltet!

dpkg versucht Konflikte automatisch zu lösen. Ist dies nicht möglich, wird der Anwender gefragt:

```
Configuration file '/etc/passwd'
 ==> File on system created by you or by a script.
 ==> File also in package provided by package maintainer.
   What would you like to do about it ?  Your options are:
    Y or I  : install the package maintainer's version
    N or O  : keep your currently-installed version
      Z     : background this process to examine the situation
 The default action is to keep your current version.
*** /etc/passwd (Y/I/N/O/Z) [default=N] ?
```

Die alte bzw. die neue Version erhält die Endung *dpkg-old* bzw. *dpkg-new*. Wenn man die Konfiguration von Hand anpaßt, kann man somit noch mal nachlesen ...

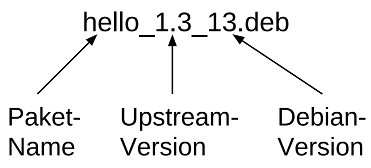
3 Interna

Nachdem die Funktion des .deb-Systems beschrieben ist, geht's nun an's Eingemachte. Wir werden mit dem Paket `hello` ein Exempel statuieren ;-).

Da Debian strengen Wert auf Nachvollziehbarkeit legt, finden sich in den Debian-Archiven nicht nur Binär-Pakete, sondern auch die Source-Archive zu den Paketen und alle für Debian gemachten Änderungen. Zum `hello`-Paket gehören:

hello_1.3-13.deb	Das Binär-Paket
hello_1.3-13.diff.gz	Änderungen, die am Original gemacht wurden
hello_1.3-13.dsc	Beschreibung der Source-Files
hello_1.3.orig.tar.gz	Original-Quelle

Die Versionsnummern spiegeln sowohl die Upstream-Version als auch die Debian-Version wider:



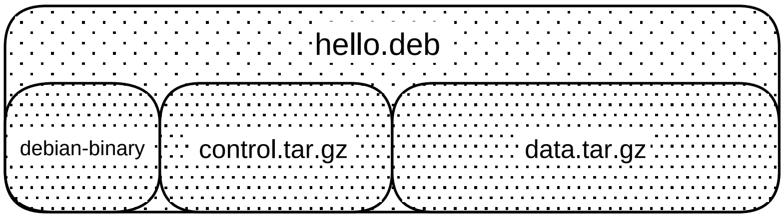
In unserem Beispiel handelt es sich um Hello-1.3 (wie auf vielen GNU-Mirrors zu finden). Für Debian ist es die 13. Version dieser 1.3 Release. Die Debian-Version ändert sich, wenn z.B. durch den Debian-Maintainer Bugfixes gemacht werden, die ihren Weg noch nicht bis zum Originalautor gefunden haben, oder wenn Änderungen für die Integration in Debian notwendig werden.

Es sei kurz erwähnt, daß es auch „debian only" Pakete gibt (`dpkg`, der Package-Manager z.B.), diese führen dann nur eine Version, also z.B. `dpkg_1.4.0.9.deb`.

3.1 Paket-Aufbau

Der interne Aufbau von Debian-Paketen selbst für den die Debian Maintainer nicht relevant. Alle Manipulationen an den Paketen werden von *dpkg-deb* vorgenommen, da nur *dpkg-deb* über Paketstruktur informiert ist. Um trotzdem nicht den Eindruck der schwarzen Magie aufkommen zu lassen, werden wir uns zur Übung der UNIX-Bordmittel bedienen. Alle diese Beispiele beziehen sich auf das Package-Format 2.0.

Träger der verschiedenen Komponenten eines Debian-Paketes ist ein Archiv im *ar(1)*-Format.In diesem Archiv sind gegenwärtig 3 Teile verpackt:



Wie versprochen, mit Bordmitteln aus der UNIX-Kiste:

```
$ ar tv hello_1.3-13.deb
rw-r--r-- 0/0      4 Sep 12 02:31 1996 debian-binary
rw-r--r-- 0/0    646 Sep 12 02:31 1996 control.tar.gz
rw-r--r-- 0/0  16699 Sep 12 02:31 1996 data.tar.gz
```

Der Inhalt der einzelnen Bestandteile soll nun kurz erklärt werden. Es muß nochmals betont werden, daß diese Details selbst für das Erstellen und Warten eines Debian Paketes nicht zum Grundwissen gehören (ich selbst habe lange Zeit einige Pakete gewartet, ohne wirklich über die Archivstruktur Bescheid zu wissen. Zwischenzeitlich hatte sich sogar die interne Struktur der Pakete geändert ...).

debian-binary ist ein Text-File mit der Versions-Nummer des jeweiligen Package-Standards. Damit ist für die Zukunft sichergestellt, daß auch ältere Pakete ausgepackt und installiert werden können.

control.tar.gz enthält alle Debian-spezifischen Informationen:

- Skripte, die vor und nach dem Entfernen bzw. Installieren eines Paketes bearbeitet werden
- Kurzbeschreibung des Paketes
- Liste der Konfigurationsdateien
- Liste von Shared Libraries, die von diesem Paket zur Verfügung gestellt werden

data.tar.gz ist das Archiv mit

- Programmen, Bibliotheken
- Konfigurationsdateien
- Manualseiten
- zusätzlicher Dokumentation

Um weiter hineinzusteigen, werden wir ein *.deb*-File auspacken. Der offizielle Weg wäre *dpkg-deb -e* für die Steuerinformationen und *dpkg-deb -x* für die Files.

Aber, wegen der schwarzen Magie, packen wir mit *ar(1)* aus:

```
$ ar xv hello_1.3-13.deb
x - debian-binary
x - control.tar.gz
x - data.tar.gz
```

ar hinterläßt uns drei Dateien im Verzeichnis. Diese Dateien werden wir jetzt etwas genauer betrachten.

3.1.1 Versionsinfo *debian-binary*

In diesem File liegt lediglich eine Versionsnummer, mit der es möglich ist, auch unterschiedliche Formate von Debian-Archiven sicher voneinander zu trennen.

```
$ cat debian-binary
2.0
```

3.1.2 Hilfsdateien in *control.tar.gz*

Die in diesem Archiv verpackten Files enthalten debian-spezifische Informationen. Weil wir es etwas genauer wissen wollen, zerlegen wir *control.tar.gz*:

```
$ tar zxvf control.tar.gz
./
postinst
prerm
control
```

Diese Files werden vom Paket-Management-System übernommen und dienen nur diesem System, zur Funktion des jeweiligen Paketes sind sie nicht notwendig. Gegenwärtig werden {*pre,post*}-{*inst,rm*}-Skripte nach */var/lib/dpkg/info/<package_name>.** installiert. In unserem Falle also z.B. *postinst* nach */var/lib/dpkg/info/hello.postinst*. Das File *control* wird in die Status-Datenbank */var/lib/dpkg/status* übernommen.

{*pre,post*}{*inst,rm*} Diese vier möglichen Files sind Skripte, die, wie die Namen suggerieren, vor und nach der Installation bzw. vor und nach dem Entfernen des Paketes ausgeführt werden. Das sind beispielsweise das Starten von Daemons nach der Installation, das Beenden von Daemons vor der Deinstallation, und das Beseitigen von Logfiles nach erfolgter Deinstallation des Packages.

control In diesem File sind alle Informationen enthalten, die der Package-Manager braucht. Einige Felder dieses Files werden vom Package-Maintainer gefüllt (Description, ...), einige Felder (Depends, Installed-Size, ...) werden automatisch beim Zusammenbau des Packages eingetragen.

Sehen wir uns den Inhalt an, wieder mit Bordmitteln:

```
$ cat control
Package: hello
Version: 1.3-13
Architecture: i386
Depends: libc5 (>= 5.2.18)
Installed-Size: 31
Maintainer: Ian Jackson <ian@chiark.greenend.org.uk>
Description: The classic greeting, and a good example
The GNU hello program produces a familiar, friendly greeting.  It
allows nonprogrammers to use a classic computer science tool which
would otherwise be unavailable to them.
.
Seriously, though: this is an example of how to do a Debian package.
It is the Debian version of the GNU Project's ‘hello world’ program
(which is itself an example for the GNU Project).
```

Die einzelnen Felder sind selbsterklärend. Besondere Aufmerksamkeit verdient das *Depends*-Feld. Mit diesem und ähnlichen Feldern wird die Abhängigkeit zwischen einzelnen Pakete überprüft, um möglichst konsistente Installationen zu verwirklichen. Im konkreten Fall läßt sich `hello` nur installieren, wenn es ein Paket gibt, das `libc5` in einer Version $\geq 5.2.18$ zur Verfügung stellt.

Etwas erklärungsbedürftig sind die Abhängigkeitsfelder, sie sollten näher erklärt werden:

Pre-Depends Hier werden Pakete aufgeführt, deren Installation bereits erfolgreich abgeschlossen sein muß. (Diese Paket müssen also sowohl die „unpack" als auch „configure"-Phase durchlaufen haben.) Ein Beispiel: Die meisten Pakete setzen die erfolgreiche Installation von libc voraus.

Depends Das Paket kann nur installiert werden, wenn auch alle in dieser Liste aufgeführten Pakete installiert werden, die Reihenfolge ist hier jedoch egal.

Recommends Das Paket empfiehlt sehr stark, die in dieser Liste aufgeführten Pakete zu installieren. Es werden hier Pakete genannt, die in einer üblichen Installation zusammengehören. Wird *dselect* genutzt, um Pakete auszuwählen, werden die empfohlenen Pakete genauso behandelt wie Abhängigkeiten. Bei der Verwendung von `dpkg` auf der Kommandozeile wird davon ausgegangen, daß der Installateur weiß, was er tut. Ein Beispiel: Das Paket `sendmail` geht davon aus, daß normalerweise auch ein „mail-reader" notwendig ist.

Suggests Das Paket schlägt die Installation der aufgeführten Pakete vor. Dieser Vorschlag ist nicht zwingend, sollte aber je nach gewünschter Funktionalität beachtet werden. Ein Beispiel: Der Mailreader `mutt` schlägt vor, auch `pgp` zu installieren. Funktionieren würde er auch ganz gut ohne ...

Conflicts Wie der Name schon nahelegt. So lange eines der aufgeführten Packages installiert ist, wird die Installation abgelehnt. Ein Beispiel: `lprng` gerät mit `lpr` in Konflikt, es sollte auf jedem System nur maximal *ein* Print-System vorhanden sein. Ein einfaches Ersetzen von `lpr` durch `lprng` ist nicht möglich, da es der Installateuse überlassen werden soll, sich für ein System zu entscheiden.

Replaces Wenn ein Paket der Meinung ist, ein anderes ersetzen zu können, wird das in diesem Feld aufgeführt. Ein Beispiel: `netscape` ersetzt `netscape-beta`.

Provides Hier können Paketnamen eingetragen werden, die das jeweilige Paket alternativ zur Verfügung stellt. Mit diesem Mechanismus werden *virtuelle* Pakete realisiert. Z.B. gibt es mehrere Pakete, die hier einen `mail-reader` aufführen (`elm`, `mutt`, `mh`, `emacs`, ...). Damit ist es für andere Pakete möglich, sich vom Vorhandensein *irgend*eines Mail-Readers abhängig zu machen.

3.1.3 Programmdateien in *data.tar.gz*

Dieses Archiv enthält alle Files, die zum eigentlichen Paket gehören, eventuell ergänzt um Files, die die Debian-Policy vorschreibt (Copyright, Changelog):

```
$ tar zxvf data.tar.gz
./
usr/
usr/doc/
usr/doc/hello/
usr/doc/hello/copyright
usr/doc/hello/changelog.gz
usr/doc/hello/changelog.Debian.gz
usr/bin/
usr/bin/hello
usr/info/
usr/info/hello.info.gz
```

Wir sehen, daß sich hier das Filesystem-Layout widerspiegelt. Auch die endgültigen Zugriffsrechte sind in diesem Archiv schon gesetzt.

3.2 Installations/Upgrade/Deinstallations-Prozeß

Der folgende Abschnitt umreißt *kurz* die einzelnen Schritte, die bei der Installation bzw. beim Upgrade eines Packages erfolgen. Generell gelten zu fast allen Schritten auch Regeln, die im Fehlerfall ein „Rollback“ ermöglichen. Weiterhin gibt es noch weitere Regeln, die sich um Konflikte und deren Abhängigkeiten kümmern. Hier sei einfach auf das *Debian Programmers Manual* verwiesen.

3.2.1 Intallation/Update

1. *Prerm* des alten Packages wird aufgerufen. Damit kann das alte Package z.B. Logfiles aufräumen oder bestimmte Dienste beenden
2. *Preinst* des neuen Packages wird aufgerufen. Hier ist die letzte Chance für das neue Paket, Dinge zu prüfen, die durch die normalen Abhängigkeitsbeziehungen nicht nicht erschlagen werden können. Ein Paket zum Sprengen der Netzwerkkarte könnte z.B. prüfen, ob noch eine Reservekarte im System installiert ist.
3. Die Files des neuen Packages werden ausgepackt und überschreiben die alten Originale. Backups der Originale werden angelegt.
4. *Postrm* der alten Package wird ausgeführt. Falls das alte Paket noch den Sargdeckel zuklappen möchte ...
5. Die Backups der alten Originale werden entfernt. Das ist der *Point of No Return*.
6. *Postinst* des neuen Pakets wird ausgeführt. Letzte Installationsarbeiten, z.B. die endgültige Integration in das SysV Init-System, das Abfeuern von Daemons ...

3.2.2 Deinstallation

Das Entfernen eines Paketes dagegen gestaltet sich relativ trivial. Es werden lediglich zwei Fälle unterschieden: *remove* and *purge*. Letzteres beseitigt das Paket restlos, also auch die (eventuell angepaßten) Konfigurationsdateien.

1. *Prerm* wird aufgerufen.
2. Alle zum Paket gehörenden Files werden entfernt, Konfigurationsdateien bleiben erhalten.
3. *Postrm* wird ausgeführt.
4. Konfigurationsdateien *postrm* werden entfernt (beim *purging*.

Nach diesen grundlegenden Regeln gestaltet sich jede Installation und jedes Upgrade. Wie die spezielle Erwähnung von Konfigurationsdateien vermuten läßt, werden diese gesondert behandelt.

3.2.3 Konfigurationsdateien beim Upgrade

Damit die Administrateuse die mühsam angepassten Konfigurationsdateien nicht bei jedem Upgrade verstecken muß, werden diese vom Package-Management gesondert behandelt.

Der jeweilige Package-Maintainer legt dem Package eine Liste mit Konfigurationsdateien bei. Bei der Installation dieser Konfigurationsdateien werden Prüfsummen in der dpkg-Datenbank festgehalten. Bei einem Upgrade wird an Hand dieser Prüfsummen entschieden, wie die Konfigurationsdateien zu behandeln sind.

- Hat sich das lokale Konfigurationsdatei nicht geändert, wird das neue installiert.
- Hat sich das lokale Konfigurationsdatei geändert und das des Pakets ist unverändert geblieben, passiert nichts.
- Hat sich das lokale Konfigurationsdatei *und* das des Paketes geändert, wird der Installateur vor die Wahl gestellt, welches von beiden zu installieren ist. Das jeweils andere wird als Backup-Version (*<config_file>.dpkg-new* bzw. *<config_file>.dpkg-old*) hinterlassen.

4 Source-Pakete und der Build-Prozeß

Die oben erwähnten Binär-Pakete entstehen nicht auf magischem Wege sondern in einem recht genau festgelegten „Build"-Prozeß aus den Originalquellen. Zum Aufbau der Source-Pakete gehört ein unvermeidlicher Hinweis auf das *Debian Policy Manual* und auf das *Debian Packaging Manual*. Diese Policy ist Garantie für die ausgezeichnete Nachvollziehbarkeit von Änderungen am Source-Code der Originale und für die Reproduzierbarkeit der Binär-Pakete.

Quellen zu einem Binary finden sich (bei der gegenwärtigen Struktur der Debian-FTP-Server) parallel zu den *binary*-Verzeichnissen. Für unser `hello`-Beispiel also als *source/misc/hello**:

hello_¡version¿.dsc Kurze Beschreibung der Source-Files, incl. Prüfsummen

hello_¡version¿.diff.gz Anpassungen, die die Originale ins Debian-System integrieren, also

- alles was für den Build-Prozeß notwendig ist
- Bug-Fixes, die noch nicht im Original enthalten sind
- zusätzliche Files (z.B. Beispiele ;-))

hello_¡version¿.orig.tar.gz Unveränderte Quellfiles, mit *hello-¡origversion¿.orig/* als Toplevel-Directory.

4.1 Übung am praktischen Beispiel

Durch diese klare Trennung zwischen Original und Debian-Änderungen ist auch eine lokale Anpassung einfach möglich. Damit bleibt die Konsistenz des Gesamtsystems erhalten. Durch Setzen den *hold*-Status (*dselect* bzw. *dpkg*) läßt sich verhindern, daß die lokale Version ersetzt wird.

Um die oben aufgestellen Behauptungen zu beweisen, werden wir zur Übung das `hello`-Package im einige fehlende (?) Features erweitern:

- Manual-Seite *hello(1)*
- Konfigurationsdatei */etc/hello.conf*
- Logfile */var/log/hello.log*

Im Verlauf dieser kurzen Übung werden wir etwas über den Paket-Zusammenbau erfahren. Danach sollte eigentlich jeder in der Lage sein, selbst lokale Änderungen an Paketen vorzunehmen und vielleicht sogar selbst Programme zu *debianisieren*.

4.1.1 Auspacken der Quellen

Zum Entpacken der Quellen genügt ein Aufruf von *dpkg-source*. Mit der Beschreibungsdatei *hello_1.3-13.dsc* wird das Patch- und das Original-File gefunden.

```
$ dpkg-source -x /home/ftp/pub/debian/bo/source/misc/hello*dsc
dpkg-source: extracting hello in hello-1.3
```

Nach dem Auspacken haben wir ein Verzeichnis *hello-1.3/* und das File *hello-1.3.orig.tar.gz* im aktuellen Verzeichnis. In *hello_1.3/* finden wir die bereits an Debian angepaßten Quellen einschließlich eines Verzeichnisses *debian/*, daß Debian-spezifische Dateien enthält. Paranoide Naturen sollten jetzt zuerst versuchen, das unveränderte Paket zu bauen.

```
# cd hello-1.3
# dpkg-buildpackage -us -uc
dpkg-buildpackage: source package is hello
dpkg-buildpackage: source version is 1.3-13
dpkg-buildpackage: build architecture is i386
debian/rules clean
test -f hello.c -a -f debian/rules
rm -f build
dpkg --build debian/tmp ..
dpkg-deb: building package ‘hello’ in ‘../hello_1.3-13_i386.deb’.
dpkg-genchanges
dpkg-genchanges: not including original source code in upload
dpkg-buildpackage: diff-only upload (original source NOT included)
```

(Wer PGP installiert hat, kann die Optionen *-us* und *-uc* auch weglassen. Sie verhindern das Erzeugen einer PGP-Signature.) Nach dem Durchlauf von *dpkg-buildpackage* haben wir jetzt das Binary-Paket .deb, die Änderungen .diff.gz, das Original .orig.tar.gz und die Beschreibung der Quellen .dsc. Das .changes-File ist lediglich für Uploads zum Debian FTP-Server notwendig. Für unseren Hausgebrauch können wir es ignorieren.

4.1.2 Lokale Änderungen

Manual-Seite Zuerst wollen wir uns die fehlende Manualseite vornehmen. Im konkreten Fall habe ich sie mit zu den Original-Quellen gelegt und auch das Original-Makefile modifiziert. Eine andere Möglichkeit wäre, es in das Verzeichnis *./debian/* zu legen und nur Änderungen im Debian-Makefile *./debian/rules* einzutragen. Da aber der normale Fall eher eine Modifikation der eigentlichen Quellen sein wird, werden wir auch diesen Fall hier simulieren.

1. Schreiben der Seite (hier hab' ich jetzt hauptsächlich die Info-Dokumentation abgeschrieben). (*pod2man(1)* mit *perlpod(1)* eignet sich gut für schnell erstellte Manualseiten.)

   ```
   =head1 NAME

   hello -- the classic greeting

   =head1 SYNOPSIS

   B<hello> [I<[options>]

   =head1 DESCRIPTION

   The GNU B<hello> program produces a familiar, friendly greeting.
   It allows nonprogrammers to use a classic computer science tool
   which
   ...
   ```

2. Das Vorhandensein von *Makefile.in* läßt auf die Verwendung von *autoconf* schließen. Änderungen am *Makefile* selbst würden also verloren gehen. Also tragen wir die Regel zum Erzeugen von *hello.1* aus *hello.1pod* in *Makefile.in* ein:

   ```
   # *.x.pod -> *.x
   %: %pod
   pod2man $< \
    --section=‘echo $@ | sed ’s/^.*(?)$$/$$&/’‘\
    --release="‘date -r $< ’+%d %b %Y’‘"\
    --center="Debian/GNU"\
    --date=’ ’\
    >$@,$$$$ && mv -f $@,$$$$ $@\
    || rm -f $@,$$$$
   ```

   Weiterhin sorgen wir noch an zwei weiteren Stellen in *Makefile.in* dafür, daß diese neue Manualseite auch installiert wird. Diese Änderungen sollten für jeden, der jemals fremde Programme geändert hat, leicht anzubringen sein.

3. Die erzeugte Manualseite soll im Debian-System komprimiert vorliegen. Dazu müssen wir wissen, wie das Paket zusammengebaut wird. Die meiste Magie wird von *./debian/rules* gesteuert. *./debian/rules* ist ein Makefile mit zwei wesentlichen Targets:

   build: Hier wird das Original übersetzt. Das entspricht etwa dem Ablauf *./configure; make* im Original. Dieses, wie auch die anderen Targets sollten automatisch elektrisch abarbeitbar sein, d.h., es sollen keine weitere Fragen gestellt werden. Es ist Aufgabe des Package-Maintainers eventuell lästig fragende *Configure*-Skripte zufriedenzustellen ...

   binary: Dieses Target erzeugt eine temporäre Verzeichnisstruktur *./debian/tmp/* und installiert darin das Paket und die Files für das Package-Management.

   Dieses *./debian/rules* muß jetzt also von der neuen Manualseite erfahren und noch so angepaßt werden, daß es diese Seite auch komprimiert.

   ```
   binary-arch:    checkroot build
        ...
        gzip -9v debian/tmp/usr/doc/$(package)/changelog{,.Debian}
        find debian/tmp/usr/man -type f -name ’*.?’ | xargs gzip -v9
        dpkg-shlibdeps hello
        dpkg-gencontrol
        ...
   ```

   Mit der *markierten* Zeile werden in der temporären Verzeichnisstruktur alle Manualseiten gesucht und komprimiert.

4. Die Änderungen sollten in *./debian/changelog* dokumentiert werden. Dabei ist auf die Version und das Format des Eintrags zu achten. Da wir hier eine *non-Maintainer*-Version erzeugen, wird die Version 1.3-13 um eine weitere Stelle ergänzt auf 1.3-13.1. Unser Changelog-Eintrag am Anfange des *./debian/changes* sieht nun so aus:

   ```
   hello␣(1.3-13.1)␣unstable;␣urgency=low

   ␣␣*␣Manpage␣added.

   ␣--␣Heiko␣Schlittermann␣<heiko@lotte.sax.de>␣␣Fri,␣25␣Jul␣1997␣15:46:38␣+0200
   ```

Wichtig sind die Leerzeichen am Zeilenanfang und in der den Eintrag abschließenden Zeile. Das Datum läßt sich mit *822-date* oder mit *date -R* erzeugen.

Damit ist es möglich, ein neues Paket zu bauen. Der erste Test sollte *./debian/rules build* heißen. Damit wird die Übersetzung gestartet. Sind hier keine Fehler aufgetreten, wird (als Root) mit *./debian/rules binary* das .deb-File erzeugt werden. Das Erzeugen sämtlicher, zu einen ordentlichen Debian-Paket gehörender Files erfolgt dann mit:

```
# dpkg-buildpackage -us -uc
dpkg-buildpackage: source package is hello
dpkg-buildpackage: source version is 1.3-13.1
dpkg-buildpackage: build architecture is i386
debian/rules clean
test -f hello.c -a -f debian/rules
rm -f build
...
dpkg --build debian/tmp ..
dpkg-deb: building package 'hello' in '../hello_1.3-13.1_i386.deb'.
dpkg-genchanges
dpkg-genchanges: not including original source code in upload
dpkg-buildpackage: diff-only upload (original source NOT included)
```

Als Ergegnis liegen uns jetzt das Binärpaket *hello_1.3-13.1_i386.deb*, das Original-Archiv *hello_1.3.orig.tar.gz*, das File mit allen Änderungen gegenüber dem Original *hello_1.3-13.1.diff.gz*, die Beschreibungs-Datei für die Quellen *hello_1.3-13.1.dsc* vor. Zusätzlich ist noch *hello_1.3-13.1.i386.changes* entstanden. Es würde zur Plazierung des einzelnen Files auf dem Debian FTP-Server dienen.

Konfigurationsdatei Nun, nachdem wir „Hello Classic" perfekt gemacht haben, wollen wir ein „Hello Pro" bauen. Dazu soll Hello um das Lesen einer Konfigurationsdatei erweitert werden.

Konfigurationsdateien liegen gemäß Debian-Policy und Filesystem-Standard unterhalb */etc/*. Für uns hieße dieses File */etc/hello.conf*.

Nach den entsprechenden Code-Änderungen in *hello.c* (denn die Konfigurationsdatei soll ja auch gelesen werden) ist vor allem dafür zu sorgen, daß das Package-Management von dieser Konfigurationsdatei weiß und ihm dann die entprechende Beachtung schenken kann. Das wird über ein File mit magischem Namen gesteuert. Wenn das Paket in seinen Hilfsinformationen ein File *conffiles* besitzt, wird in diesem File die Liste der Konfigurationsfiles erwartet.

Dazu wird zuerst *./debian/conffiles* angelegt:

```
$ echo /etc/hello.conf > ./debian/conffiles
```

Dieses File muß nun beim Bau des Binärpaketes in der Control-Area installiert werden. Dies geschieht in der *markierten* Zeile des *./debian/rules* Files:

```
binary:
    ...
    install -d debian/tmp/usr/doc/$(package)
```

```
    install -d debian/tmp/etc
    cp debian/{postinst,prerm,conffiles} debian/tmp/DEBIAN/.
...
```

Und natürlich muß auch noch die Konfigurationsdatei selbst installiert werden. Dazu ist eine weitere Änderung in *./debian/rules* notwendig:

```
binary:
    ...
    install -d debian/tmp/usr/doc/$(package)
    install -d debian/tmp/etc
    install -m644 debian/hello.conf debian/tmp/etc/hello.conf
...
```

Nach der Anpassung der Manualseite (sie sollte das Vorhandensein der Konfigurationsdatei erwähnen) sind nur noch entsprechenden Ergänzungen in *./debian/changelog* vorzunehmen. Dabei sollte sich die Versionsnummer auf 1.3-13.2 erhöhen.

Nach dem Aufruf von *dpkg-buildpackage -us -uc* steht uns im Verzeichnis *../* ein neues `hello`-Paket zur Verfügung.

Logfiles Nachdem wir nun mutig geworden sind, wollen wir „Hello Pro" erweitern zu „Hello Gold". Die Gold-Edition soll den Aufruf des von *hello* in einem Logfile dokumentieren.

Wie nicht anders zu erwarten, gibt es auch hierfür eine Policy. Logfiles liegen in */var/log/* und heißen nach Möglichkeit auch so, daß sich auf das zugehörige Paket schließen läst. Wir wollen also unser Logfile */var/log/hello.log* nennen. Die Policy und der gute Ton verlangen weiterhin, daß Logfiles nicht ins Unermeßliche wachsen.

Zuerst kommen wieder die Änderungen im Programmcode *hello.c*. Änderungen am *Makefile.in* sind nur im Installationstarget notwendig. Hier muß dafür gesorgt werden, daß das Verzeichnis */var/log/* angelegt wird. Die betroffenen Stellen sind offensichtlich. Mindestens für denjenigen, der in der Lage ist, „Hello Pro" zu „Hello Gold" zu machen.

Um das Wachstum des Logfiles zu verhindern, können wir es regelmäßig löschen. Durchgesetzt hat sich aber der Weg des Verschiebens der Logfiles: *hello.log* → *hello.log.0* → *hello.log.1.gz* → *hello.log.2.gz* → … → */dev/null*. Bei Debian gibt es, wie auf anderen Unixen auch, einen Cron-Daemon. Damit nicht jedes Paket Einträge in der Crontab vornehmen muß, unterhält ein Debian-System */etc/cron.daily/*, */etc/cron.weekly/* und */etc/cron.monthly/*. In diesen Verzeichnissen können Pakete ihre Aufräumskripte installieren.

Unser Aufräumskript:

```
#! /bin/sh
# /etc/cron.monthly for hello
LOG=/var/log/hello.log
test -f $LOG || exit 0
savelog -g adm -m 644 -u root -c 6 $LOG >/dev/null
```

Dieses Skript legen wir als *./debian/cleanup.sh* an. Das Debian-Makefile *./debian/rules* ergänzen wir um die Installation dieses Files:

```
binary-arch: checkroot build
$(checkdir)
-rm -rf debian/tmp
...
install -d debian/tmp/usr/doc/$(package)
install -d debian/tmp/etc
install -d debian/tmp/etc/cron.monthly
cp debian/postinst,prerm,conffiles debian/tmp/DEBIAN/.
...
find debian/tmp/usr/man -type f -name '*.?' | xargs gzip -v9
install -m644 debian/hello.conf debian/tmp/etc/hello.conf
install -m644 debian/cleanup.sh debian/tmp/etc/cron.monthly/hello
...
```

Da die Logfiles erst im laufenden Betrieb entstehen und nicht mit installiert werden, muß der *postrm*-Skript für das Aufräumen sorgen. Zu beachten ist hierbei, daß dieser Skript in jedem Falle gut enden muß, da ansonsten *dpkg* einen Fehler vermuten würde und das kundtäte.

```
#! /bin/bash
# postrm for hello
set -e
cd /var/log
rm -f hello.log*
```

Und dieses *postrm* muß natürlich installiert werden:

```
binary-arch: checkroot build
       $(checkdir)
        -rm -rf debian/tmp
        ...
        install -d debian/tmp/usr/doc/$(package)
        install -d debian/tmp/etc
        install -d debian/tmp/etc/cron.monthly
        cp debian/{postinst,postrm,prerm,conffiles} debian/tmp/DEBIAN/.
```

Weiterhin sollten wir */etc/cron.monthly/hello* auch in die Liste der Konfigurationsfiles *./debian/conffiles* aufnehmen, denn es ist ja denkbar, daß die Systemverwalterin statt monthly 6 lieber 7 hat …

Sind diese Änderungen vollbracht, gehen wir wieder zurück nach Los, ergänzen den ChangeLog *./debian/changelog* um einen netten Kommentar und neue Versionsnummern und lassen bauen.

Dann installieren wir `hello_1.3-13.3`, AKA „Hello Gold Edition“.

5 Debian-Policy

Mit dpkg kann man also Pakete bauen, die sich auf einem Debian-System installieren lassen. In obigem Beispiel war schon zu erkennen, daß es nicht genügt, mit den Debian-Werkzeugen ein Paket zu erstellen. Man muß auch bestimmte Dinge beachten, damit das Gesamtsystem etwas einheitlich aussieht.

An Debian arbeiten über 200 Entwickler mit, die meisten davon warten eines oder mehrere der über 1000 Pakete. Die Größe von Debian macht eine geordnete Zusammenarbeit erforderlich. Die Policy beschreibt all die Dinge, auf die sich die Debian-Entwickler in Diskussionen auf den entsprechenden Mailing-Listen geeinigt haben und die somit von allen Debian-Paketen erfüllt werden müssen. Erst das Einhalten der Policy macht aus einem Paket im .deb-Format ein (bugfreies) Debian-Paket.

5.1 FSSTND

Die Debian-Policy bezieht ausdrücklich den *Linux Filesystem Structure Standard* (FSSTND) ein. In ihm wird definiert, welche Verzeichnisse in einem Linux-System existieren müssen und welche Programme oder Dateien sie beinhalten müssen.

Ziel des FSSTND ist, ein Auseinanderlaufen des Aussehens der verschiedenen Linux-Distributionen zu vermeiden. Somit ist er notwendigerweise allgemeiner als die Debian-Policy, und er beschränkt sich auf das Dateisystem-Layout.

Der FSSTND sieht vor, daß */usr/* read-only gemountet sein darf, um z. B. */usr/* über NFS von einem Server zu mounten. Konfigurationsdateien müssen unterhalb */etc/* gespeichert werden. Variable Daten, also praktisch alle Daten, die automatisch von Programmen geschrieben werden, gehören deshalb nach */var/*.

5.2 Beispiele

In der Policy sind alle Fälle geklärt, die nach einer Vereinheitlichung rufen:

- Manual-Seiten
- Info-Dokumente
- zusätzliche Dokumente
- bevorzugte Dokumentationsformate
- Logfiles
- Copyright Information
- Beispiele
- Changelog Files

Zwei dieser Punkte seien in den folgenden Abschnitten beispielhaft erklärt.

5.2.1 /usr/doc/*/

Die Debian-Policy fordert, daß für jedes Binär-Paket ein Verzeichnis */usr/doc/<paketname>/* existiert, das mindestens folgende Dateien enthält:

copyright	Autor und Copyright beteiligte Debian-Maintainer Ort der Original-Quellen vollständige Lizenz
changelog.gz	Changelog (Liste der Änderungen) in den Original-Quellen
changelog.Debian.gz	Changelog Debian-spezifischer Änderungen in dem Paket (u. U. optional)

5.2.2 install-info

GNU Info ist das offizielle Dokumentations-System für GNU-Software. Emacs und fast allen anderen GNU-Programme haben ihre Dokumentation im Info-Format. Der Startpunkt für das Lesen dieser Dokumentation ist die Datei */usr/info/dir*. Sie enthält Verweise auf die einzelnen installierten Info-Dateien.

Wenn ein Programm seine Info-Dokumentation installiert, muß sie dort eingetragen werden; und beim Löschen des Pakets muß der Verweis wieder ausgetragen werden. Traditionell mußte der Systemadministrator diese Änderungen selbst vornehmen. In Debian wurde das Programm *install-info* geschrieben, das diese Modifikationen vornimmt.

Im Postinst-Skript des hello-Pakets wird der Eintrag installiert:

```
install-info --description='GNU Hello.  Prints a classic greeting.' \
             --quiet /usr/info/hello.info
```

und im Prerm-Skript wird er entfernt:

```
install-info --quiet --remove /usr/info/hello.info
```

6 Fremde Pakete

Kommerzielle Software für Debian ist praktisch nie im Debian-Format erhältlich. Stattdessen wird diese Software im RPM-Format und/oder als tar.gz-Datei vertrieben. Mit *alien* existiert ein Werkzeug, Pakete von einem Paketformat in ein anderes zu konvertieren.

Da nicht in allen Formaten die gleichen Informationen stehen, muß *alien* bei fehlenden Angaben „raten". Wunder kann *alien* dabei nicht vollbringen; es ist unerreichbar, daß ein konvertiertes Paket vollständig der Debian-Policy entspricht.

Testweise wird Wabi konvertiert:

```
# alien /cdrom/RPMS/Wabi-2.2B-8.i386.rpm
(...)

Generation of wabi_2.2B-9_i386.deb complete.
-- Successfully finished
# dpkg -i wabi_2.2B-9_i386.deb
Selecting previously deselected package wabi.
(Reading database ... 35175 files and directories currently installed.)
Unpacking wabi (from wabi_2.2B-9_i386.deb) ...
Setting up wabi (2.2B-9) ...
# dpkg -s wabi
Package: wabi
Status: install ok installed
Installed-Size: 9220
Maintainer: Sven Rudolph <sr1@os.inf.tu-dresden.de>
Version: 2.2B-9
Depends: ldso (>= 1.8.0-0), libc5 (>= 5.4.0-0), xlib6 (>= 3.3-0)
Description: Wabi - MS Windows Emulator
 Wabi - MS Windows Emulator
```

alien hat automatisch herausgefunden, in welchen Paketen die benutzten shared libraries liegen, und diese Pakete in das *Depends*-Feld aufgenommen. Die *Description* ist sehr kurz, aber *alien* hat eben nicht mehr gefunden. Als Maintainer wurde ich eingetragen.

```
# dpkg -L wabi
/.
/opt
/opt/bin
/opt/bin/dos2unix
/opt/bin/unix2dos
/opt/bin/wabi
/opt/bin/wabimakelower
/opt/man
/opt/man/man1
/opt/man/man1/wabi.1
(...)
/opt/wabi/wbin/windows/system/8514oem.fon
/usr
/usr/doc
```

```
/usr/doc/wabi
/usr/doc/wabi/copyright
/usr/doc/wabi/changelog.Debian
/usr/doc/wabi/buildinfo.Debian
/usr/man
/usr/man/man1
/usr/man/man1/wabi.1.gz
```

Das RPM-Archiv ist so organisiert, daß die Wabi-Dateien in /opt/wabi/ stehen. Laut Debian-Policy ist der FSSTND strikt einzuhalten, so existiert unter Debian kein Verzeichnis */opt/*, und die Programme würden nach */usr/X11R6/bin/* gehören. *alien* ist nicht in der Lage, die Dateien selbständig zu verschieben.

Wabi für Linux wird von Caldera produziert, und Caldera hat Wabi primär für die eigene Distribution konzipiert. In dieser ist */opt/* als Verzeichnis für Add-on-Programme zulässig, und somit ist das Paket dafür korrekt. (Es ist auch für einen Nachfolger des FSSTND im Gespräch, */opt/* einzuführen. Man kann nicht eindeutig sagen, daß Caldera etwas falsch macht.)

Die kommerziellen Programme für Linux sind üblicherweise große Anwenderprogramme, die nicht systemnah arbeiten. Sollen diese Programme unter Debian genutzt werden, lassen sie sich mit *alien* in brauchbare .deb-Pakete konvertieren.

Man sollte aber nicht versuchen, mittels alien beliebige RPM-Pakete auf seinem Debian-System zu installieren, insbesondere wenn ihre Funktionalität wichtig für das Gesamtsystem ist. Diese Pakete entsprechen mit Sicherheit nicht der Debian-Policy, und ihre Nutzung kann unvorhergesehene Folgen haben.

7 Letzte Worte

Wer jetzt erwartet hat, daß mit Hilfe des superben Package-Managements Schlachten um bessere oder schlechtere Distributionen geschlagen werden, muß leider enttäuscht werden. Entscheidend ist, wie die Philosophie des Gesamtsystems zu den Ansprüchen des potentiellen Anwenders paßt. Eins sollte klar geworden sein — Debian fährt noch nicht im roten Bereich ;-) (Maximum RPM – taking the red hat package manager to the limit).

Die Autoren
Sven Rudolph studiert Informatik an der TU Dresden. Am Lehrstuhl Betriebssysteme administriert er ein paar Linux-Rechner. Mit Linux fing es an, weil er keine Lust hatte, 20 Disketten für 386BSD zu kopieren. Als ihn die Langlebigkeit einiger Slackware-Bugs ärgerte, begegnete ihm Debians Bug-Report-System (http://www.debian.org/Bugs/). Seitdem also Debian ... Zu erreichen ist er unter sr1@debian.org.
Heiko Schlittermann ist Heiko Schlittermann und zu erreichen unter heiko@debian.org.

Literatur	
Debian Policy Manual	`http://www.debian.org/manuals/debian-policy/`
Debian Packaging Manual	`http://www.debian.org/manuals/packaging-manual/`
Aktuelle Version	`http://www.schlittermann.de/deb-intern/dpkg/`
Aktuelle Version (.ps)	`http://www.schlittermann.de/deb-intern/dpkg/dpkg.ps.gz`
FSSTND	`ftp://ftp.debian.org/debian/doc/package-developer/fsstnd-1.2.txt.gz`
Maximum RPM; Edward C. Bailey	Red Hat Software Inc.