

Paketverwaltung von DEB Paketen

Johannes Knopp

29.04.2008

1 Einleitung

Zu den Grundlagen jeder Linux Distribution gehört die Paketverwaltung. Im folgenden werden zunächst die Idee hinter dem Konzept von Softwareverwaltung mithilfe von Paketen beschrieben und danach der Aufbau von .deb Paketen und der Umgang mit ihnen im Detail vorgestellt.

Inhaltsverzeichnis

1	Einleitung	1
2	Pakete allgemein	2
2.1	Pakete - Wozu?	2
3	Aufbau eines .deb Pakets I	3
4	Arbeiten mit .deb Paketen I	3
4.1	dpkg	3
4.2	Installations-/Updatevorgang	4
4.3	Deinstallationsvorgang	4
4.4	Paketzustand	4
5	Aufbau eines .deb Pakets II	5
5.1	Abhängigkeiten	5
5.2	Paketprioritäten	6
5.3	Installationsinformationen	6
6	Arbeiten mit Paketen II	7
6.1	Bezugsquellen von Paketen	7
6.2	Probleme bei Abhängigkeiten	7
6.3	apt - Advanced packaging tool	7
6.4	aptitude	8
6.5	Grafische Paketverwaltung mit Synaptic	8

7 Zusammenfassung	10
8 Quellen	10

2 Pakete allgemein

Im Normalfall sind jedem Betriebssystem standardmäßig eine Menge Programme beigelegt wie ein Browser (z.B. Firefox, Internet Explorer, Safari...), ein E-Mail Programm (z.B. Thunderbird, Outlook...) und Textverarbeitungssoftware (z.B. Open Office Writer, MS Word...). Für den Anfang reichen diese Programme aus, um arbeiten zu können, aber wer mehr mit dem Computer arbeiten möchte mit Sicherheit noch weitere Programme installieren und die bereits aufgespielten auf dem neuesten Stand halten. Unter Linux werden alle Programme im Normalfall über Pakete verwaltet. Diese bestehen im Wesentlichen aus der Software selbst und noch einigen zusätzlichen Informationen über das Paket wie Versionsnummer und Abhängigkeiten(s. 3).

2.1 Pakete - Wozu?

Die Idee hinter Paketen folgt dem grundlegendem Konzept des *Baukastenprinzips* von Linux-Systemen: Probleme werden in möglichst kleine Teilprobleme heruntergebrochen und auf dieser Ebene gelöst, woraus sich kleine Programmbausteine (Tools/Kommandos) ergeben, die wieder in größeren Teilen Verwendung finden können. Jedes dieser Programme wird in einem Paket verwaltet, das installiert werden kann. Dieses Vorgehen bringt einige Vorteile mit sich:

- **Robustheit:** Es ist möglichst schwer, das System ungewollt zu korrumpieren
- **Einfache Bedienung:** Anfänger sollen keine große Hürden haben, um mit dem System umzugehen – Fortgeschrittene sollen aber auch keine Funktionalität vermissen.
- **Upgrade-Fähigkeit:** Durch die dynamische Entwicklung von Linux erscheinen häufig neue Versionen von Programmen. Diese sollen problemlos integriert werden können und kein neues Booten erfordern.
- **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 upgradable ist." ¹

¹Zitat und Auflistung von <http://www.schlittermann.de/deb-intern/dpkg/>

Am weitesten verbreitet sind zwei verschiedene Paketformate: *RPMs* (Redhat Package Manager) und *DEBs*. In diesem Vortrag wird nur das *.deb*-Format behandelt.

3 Aufbau eines *.deb* Pakets I

Die Pakete im *.deb* Format finden Anwendung in *Debian* und darauf basierenden Distributionen (Ubuntu, MEPIS, Dreamlinux, Damn Small Linux, Xandros, Knoppix, Linspire, sidux, Kanotix...). Sie stehen in der Regel unter einer freien Lizenz wie GPL, LGPL, BSD, oder Artistic.

Deb-Pakete sind mit **ar** gepackt und haben einen Namen der Form

```
<foo>_<VersionNumber>-<DebianRevisionNumber>_<DebianArchitecture>.deb
```

wobei foo den Paketnamen beschreibt. Entpackt ergeben sich drei Dateien:

1. **debian-binary**: Die Versionsnummer des verwendeten Paketformats (momentan 2.0)
2. **control.tar.gz**: Ein Archiv mit Dateien, die zur Installation dienen oder Abhängigkeiten auflisten.(s. Abschnitt 5)
3. **data.tar.gz**: enthält die eigentlichen Programmdateien mit relativen, beim root-Verzeichnis beginnenden Pfaden.

4 Arbeiten mit *.deb* Paketen I

Wir wissen nun, dass *.deb*-Pakete installiert werden können, doch wie funktioniert das in der Praxis? Zu diesem Zweck gibt es verschiedene Möglichkeiten, die in den nächsten Abschnitten vorgestellt werden.

4.1 dpkg

Ein Low-Level Werkzeug zum Installieren von *.deb*-Paketen ist dpkg. Der Befehl zum Installieren von Paketen lautet (als root)

```
dpkg --install <paketname>
```

Analog dazu funktioniert das Entfernen von Paketen:

```
dpkg --remove <paketname>
```

Dabei ist zu bemerken, dass *remove* nur das Paket, nicht aber die Konfigurationsdateien entfernt. Wer das Paket restlos vom System verschwinden lassen will benutzt statt *remove* *purge*.

4.2 Installations-/Updatevorgang

Beim Installieren werden sechs Phasen durchlaufen, um sicheres und störfreies Installieren von Paketen zu ermöglichen:

1. *Prerm*: Wird vor dem Entfernen vom System aufgerufen. Damit kann das alte Paket z.B. Logdateien aufräumen oder bestimmte Dienste beenden
2. Die Dateien des neuen Paketes werden ausgepackt und überschreiben die alten Originale. Backups der Originale werden angelegt.
3. *Preinst*: Wird vor dem Entpacken auf die Festplatte aufgerufen. Hier ist die letzte Chance für das neue Paket, Dinge zu prüfen, die durch die normalen Abhängigkeitsbeziehungen nicht erschlagen werden können.
4. *Postrm*: Letzte Aufräumarbeiten des dahinscheidenden Pakets.
5. Die Backups der alten Originale werden entfernt. Das ist der Point of No Return.
6. *Postinst*: Wird nach dem Entpacken aufgerufen. Letzte Installationsarbeiten, z.B. die endgültige Integration in das SysV Init-System, das Abfeuern von Daemons. . .

4.3 Deinstallationsvorgang

Eine Deinstallation ist einfacher.

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

Konfigurationsdateien spielen eine Sonderrolle, da sie nicht ungewollt gelöscht werden sollen, da möglicherweise eine Menge Handarbeit in sie eingeflossen ist.

4.4 Paketzustand

Pakete können im Prinzip installiert sein oder nicht. Es sind allerdings noch mehr Zustände möglich, wie diese Übersicht zeigt:

- *Not installed n*: nicht installiert
- *Installed i*: installiert

- *Config-Files c*: nicht mehr installiert, aber die Config-Dateien sind noch vorhanden
- *Unpacked u*: ausgepackt, aber noch nicht konfiguriert
- *Failed-Config f*: Fehler bei Konfiguration aufgetreten
- *Half-installed h*: Paket unvollständig installiert

5 Aufbau eines .deb Pakets II

Nachdem nun der Überblick über Pakete und dessen einfache Funktion klar ist, betrachten wir nun die Details. In 3 war die Rede davon, dass das Archiv **control.tar.gz** in einem .deb Paket Abhängigkeiten auflistet und der Installation dient.

5.1 Abhängigkeiten

Diese beschreiben welche anderen Pakete vorausgesetzt sind, um das gewählte Paket zu installieren. Beispielsweise hat es keinen Sinn einen Videoplayer zu installieren, wenn man das Grafikpaket *X Window System* nicht installiert hat. Hier eine Übersicht über die möglichen Arten von Abhängigkeiten:

- **Pre-Depends**: Die aufgeführten Pakete müssen bereits fertig installiert sein. Die meisten Pakete setzen z. B. *libc* voraus.
- **Depends**: Das Paket kann nur installiert werden, wenn die hier aufgeführten Pakete auch installiert sind bzw. werden. Die Reihenfolge der Installation ist aber 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. 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**: Die Installation wird abgelehnt, falls eines der hier aufgeführten Pakete installiert ist. Auf einem System darf es z. B. nur ein Print-System geben

- **Replaces:** Das Paket ersetzt ein anderes. Zum Beispiel die Release-Version eine zuvor installierte Betaversion eines Programms
- **Provides:** Hier werden Paketnamen aufgeführt, dessen Funktionalität das jeweilige Paket alternativ auch zur Verfügung stellt. 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 **irgendeines** Mail-Readers abhängig zu machen. Mehr dazu findet man unter dem Stichwort *virtuelle Pakete/virtual packages* im Internet

5.2 Paketprioritäten

In der *control*-Datei gibt es auch die Information **priority**, die dazu dient zu beschreiben, wie wichtig das Paket für ein Linux-System ist:

- *Required*-Pakete sind nötig, damit das System ordentlich läuft (z.B. Programme zum Reparieren des Systems). Eine Installation nur bestehend aus Required Paketen ist vielleicht noch nicht nützlich, ist aber in der Lage zu booten und neue Pakete zu installieren
- *Important*-Pakete sollten auf jedem Unix-like System gefunden werden. Keine großen Anwendungen wie *X11*, eher Pakete für die Infrastruktur
- *Standard*: Wie der Name sagt sind das die Standardpakete eines Linux-Systems. Für den Anwender absolut grundlegende Pakete sind so markiert (*OpenSSH* für remote Administration, *exim* zum Email-senden, *mutt* zum Empfangen...)
- *Optional* sind Pakete, von denen man wahrscheinlich einige installieren möchte. Unter anderem *X11* oder eine *TEX* Distribution
- Mit *Extra* sind Pakete gekennzeichnet, die z.B. im Konflikt mit Paketen höherer Priorität stehen oder spezielle Anforderungen haben. Sollten nur installiert werden, wenn der Anwender auch weiß, was es damit auf sich hat

5.3 Installationsinformationen

In 4.2 und 4.3 werden Pre- und Postrm/inst Vorgängen gesprochen. Diese sind von den Paketen definiert, meist durch *BASH* oder *PERL* Programme. Diese sind auch in **control.tar.gz** enthalten.

6 Arbeiten mit Paketen II

Mit `dpkg` sind wir nun also in der Lage Pakete zu installieren. Es stellen sich zwei Fragen:

1. Woher kann ich Pakete bekommen?
2. Was mache ich, wenn die Abhängigkeiten von einem Paket nicht erfüllt sind?

6.1 Bezugsquellen von Paketen

Quellen für Pakete sind in `/etc/apt/sources.list` aufgelistet. Die Datei sieht auf meinem Debian-System folgendermaßen aus:

```
deb http://sunsite.informatik.rwth-aachen.de/ftp/pub/Linux/debian/ lenny main contrib non-free
deb-src http://sunsite.informatik.rwth-aachen.de/ftp/pub/Linux/debian/ lenny main contrib non-free

deb http://security.debian.org/ lenny/updates main contrib
deb-src http://security.debian.org/ lenny/updates main contrib

deb ftp://debian.netcologne.de/debian-multimedia.org testing main
deb-src ftp://debian.netcologne.de/debian-multimedia.org testing main
```

Mit `deb` und `deb-src` wird unterschieden, ob es sich um kompilierte Binärpakete (`.deb`) handelt oder um Quellpakete, die aus dem Quellcode, einer Debian control file (`.dsc`) und der `diff.gz` zum Debianisieren des Pakets enthalten. Nach der Art der Quelle folgt der Ort. Verschiedene Protokolle zur Dateiübertragung werden unterstützt (`http`, `ftp`, `ssh`, ...). Die Argumente hinter dem Quellort teilen `apt` mit, nach welchen Arten von Paketen es suchen soll. Diese können nach Stabilität und Lizenzen gefiltert werden.

6.2 Probleme bei Abhängigkeiten

Was passiert, wenn es Probleme mit den Abhängigkeiten gibt bei der Installation gibt? `dpkg` gibt an dieser Stelle auf und gibt das Problem an den Anwender zurück. Der Benutzer muss sich selbst darum kümmern, die Abhängigkeiten aufzulösen, also die fehlenden Pakete herunterladen und installieren. Diese haben eventuell wieder eigene Abhängigkeiten, die wiederum Abhängigkeiten haben. ... man landet in der sprichwörtlichen **dependency hell**. Die Lösung für das Problem des Auflöserns von Abhängigkeit ist ein weiteres Programm zur Paketverwaltung - `apt`.

6.3 apt - Advanced packaging tool

Das Advanced packaging tool `apt` ist in der Lage Abhängigkeiten eines Pakets zu erkennen und die entsprechenden fehlenden Pakete automatisch inklusive deren Abhängigkeiten zu installieren. Im folgenden gibt es einen Überblick über die wichtigsten Befehle:

- *apt-get update* aktualisiert die Liste der verfügbaren Pakete. Sollte immer gestartet werden bevor man ein Paket installiert, um ausschließlich die neuesten Pakete zu installieren.
- *apt-get install <paketname>* schaut in der Datenbank nach der letzten Version des Pakets lädt es von der entsprechenden Quelle. Falls das Paket von anderen abhängig ist werden diese auch nach Rückfrage an den Benutzer installiert.
- *apt-get remove <paketname>* entfernt sowohl das Paket selbst als auch alle Pakete, die von diesem abhängen. Mit der Option *--purge* werden alle entsprechenden Konfigurationsdateien mit gelöscht.
- *apt-get -u upgrade* bringt die installierten Programme auf den neuesten Stand. Ohne das Argument *-u* zeigt apt-get nicht an, welche Pakete im speziellen erneuert werden. Es kann sein, dass Pakete als "kept back" gekennzeichnet werden und trotz neuerer Versionen vom Update ausgenommen sind, da sich Abhängigkeiten verändert haben.
- *apt-get -u dist-upgrade* bringt das System auf den Stand eines neuen größeren Releases. Kann die Lösung für "kept back" Pakete sein.

Außerdem kann man mit *apt-cache search <begriff>* nach Paketnamen suchen und mit *apt-cache show <paketname>* Details zu dem gewählten Paket erfahren, wie mit *dpkg --info <paketname>*.

6.4 aptitude

Mit *apt* ist uns jetzt ein mächtiges Werkzeug an die Hand gegeben worden, das die Paketverwaltung deutlich vereinfacht. Allerdings löst *apt* nur die direkten Abhängigkeiten und ignoriert z.B. empfohlene Pakete. Außerdem vergisst *apt-get* beim Entfernen von Paketen auch die Pakete vom System zu werfen, die nur aufgrund von Abhängigkeiten zu diesem Paket installiert wurden. Diesem Problem kommt man mit Benutzung von **aptitude** bei, das ein Front-End zu apt darstellt (inklusive der Fähigkeiten von *dselect*²), aber mit den genannten Problemen umgehen kann. Aus diesem Grund ist in der Regel aptitude der Benutzung von apt-get vorzuziehen. Des Weiteren hat aptitude viele kleine Vorteile, nachzulesen in Sektion 8.1.3. auf

<http://www.debian.org/doc/manuals/debian-faq/ch-pkgtools.en.html#s-aptitude>

6.5 Grafische Paketverwaltung mit Synaptic

Synaptic ist auch ein Front-End zu apt, stellt aber im Gegensatz zu aptitude keine weiteren Fähigkeiten zur Verfügung. Linux Neueinsteiger und weniger konsolenorientierte Benutzer finden sich hier wahrscheinlich leichter zurecht.

²Ein Programm das eine Menü-Schnittstelle zu apt hat und ein paar Vorteile gegenüber apt-get beinhaltet, z.B. trägt es Sorge dafür, dass keine Pakete in Konflikt geraten, stellt die Reihenfolge fest, in der Pakete installiert werden müssen und mehr. . .

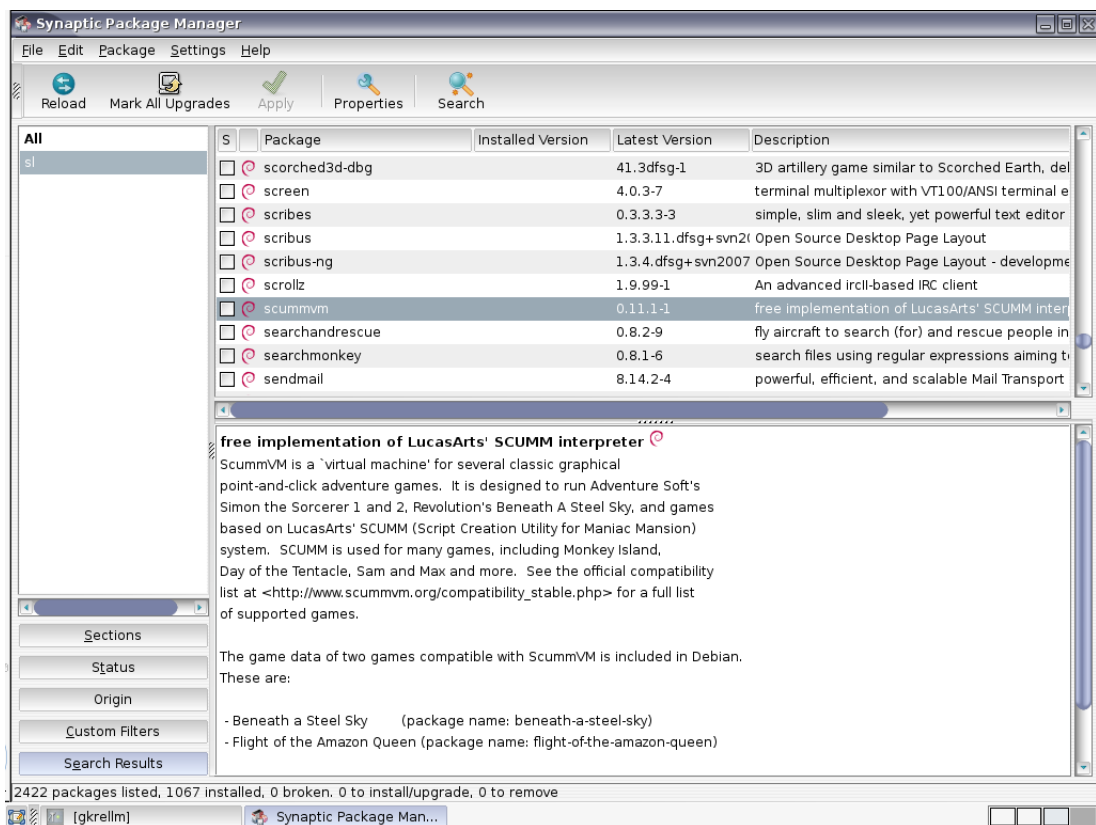


Abbildung 1: Der grafische Paketmanager Synaptic

7 Zusammenfassung

Wir haben gesehen, dass das Paketsystem von Linux eine sehr dynamische Verwaltung von Programmen ermöglicht. Das hat den Vorteil, dass in der Regel nur benötigte Pakete installiert werden, allerdings müssen dabei die Abhängigkeiten beachtet werden. Zum Umgang mit Konflikten und Abhängigkeiten gibt es fortgeschrittene Programme wie apt-get und aptitude, die es dem Anwender erleichtern gute Entscheidungen beim Installieren von Programmen zu treffen.

8 Quellen

http://www.debian.org/doc/FAQ/ch-pkg_basics
<http://www.debian.org/doc/manuals/debian-faq/ch-pkgtools.en.html>
<http://nixdoc.net/man-pages/Linux/dpkg.8.html>
<http://www.schlittermann.de/deb-intern/dpkg/>
<http://www.debian.org/doc/manuals/apt-howto/>
<http://www.youtube.com/watch?v=1FzPrzY2KFM&v3>
http://en.wikipedia.org/wiki/Dependency_hell
http://en.wikipedia.org/wiki/Aptitude_%28program%29
<http://en.wikipedia.org/wiki/Dpkg>
http://en.wikipedia.org/wiki/List_of_Linux_distributions#Debian-based
<http://de.wikipedia.org/wiki/Debian-Bin%C3%A4rpaket>
<http://www.newlinuxuser.com/howto-use-dpkg-to-install-deb-files/>
<http://pthree.org/2007/08/12/aptitude-vs-apt-get/>