

BS3-Artikel zum Vortrag
“Paketmanagement”

Wolfram Wagner (s52609)

30. Januar 2007

Inhaltsverzeichnis

1	Einleitung	3
1.1	Zu diesem Dokument	3
2	Paketformate	3
2.1	Prominentes Beispiel: <i>RPM</i>	4
2.2	Pakete unter <i>Debian: deb</i>	4
2.3	Das <i>ebuild</i> Konzept unter <i>Gentoo</i>	4
3	Paketmanager im Überblick	5
3.1	Überblick	5
3.2	Der Aufsteiger <i>yum</i>	7
3.3	Gute Wahl: <i>apt</i>	8
3.4	Quellen im Griff mit <i>portage</i>	8
4	Fazit	9

1 Einleitung

Dieser Artikel zum Thema “Paketemanager” ist im Rahmen der Lehrveranstaltung “Betriebssysteme3” (BS3) im siebenten Fachsemester (allgemeine Informatik) an der “Hochschule für Technik und Wirtschaft” (HTW) in Dresden entstanden.

Ziel ist es einen Überblick über verschiedenen Systeme zu geben und Unterschiede zwischen verschiedenen Ansätzen aufzuzeigen. Dabei kann keinerlei Anspruch auf Vollständigkeit erhoben werden. Dies ist allerdings kein großer Nachteil, da sich viele Systeme prinzipbedingt ähneln.

Die beiden Hauptansätze sind darin zusehen, dass man Paketmanager mit Abhängigkeitsauflösung und welche ohne unterscheidet. Mein Vortrag sollte dabei herausstellen, dass nur erstere “richtige” Paketmanager sind. In Kapitel 2 wird die Grundlage für Paketmanager angesprochen, die Pakete. In Kapitel 3 werden schließlich die drei betrachteten Systeme vorgestellt.

1.1 Zu diesem Dokument

Ich möchte an dieser Stelle nicht verschweigen, dass sowohl diese Dokumentation als auch die Folien zum Vortrag mittels des Textsatzsystems $\text{T}_{\text{E}}\text{X}$ in Verbindung mit $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ und BEAMER erstellt wurden.

Ich wollte diese Gelegenheit nutzen, um mich weiter auf die bevorstehende Diplomarbeit vorzubereiten, da ich diese höchstwahrscheinlich ebenfalls in $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ anfertigen werde. Der Ansatz und die Ergebnisse haben mich überzeugt, weswegen ich diese Möglichkeit gern auch meinen Kommilitonen vermitteln möchte.

2 Paketformate

In diesem Kapitel werden drei sehr gebräuchliche Paketformate und damit verbundene Werkzeuge vorgestellt. Grundsätzlich sind (Software-)Pakete die Grundlage dafür, dass ein Paketverwaltungssystem seine Arbeit verrichten kann. In einigen Fällen setzen Paketmanager auf “low-level” Programmen wie *RPM* (siehe 2.1) oder *dpkg* (siehe 2.2) auf. Ich möchte hier allerdings nicht auf den konkreten Aufbau der Pakete eingehen, da dies nicht dem allgemeinen Verständnis, um das es mir in diesem Beitrag geht, dient. Umfassende Dokumentation ist auf den Entwicklerseiten der entsprechenden Projekte zu finden und sollte konsultiert werden, wenn man sich in eines der Paketformate einarbeiten möchte.

2.1 Prominentes Beispiel: *RPM*

Das wohl bekannteste Format bzw. Werkzeug ist wahrscheinlich *RPM*. Bei *RPM*-Paketen handelt es sich um binäre Paket in einem eigens erdachten Format. Ursprünglich wurde *RPM* von RedHat entwickelt, weswegen der Name von *RedHat Package Manager* abgeleitet ist. *RPM* erfreut sich großer Beliebtheit und Verbreitung z.B. unter RedHat, SuSE, Fedora und auch Unix-Derivaten wie z.B. AIX.

Installiert und Deinstalliert werden diese Pakete mittels des *RPM*-Kommandos und verschiedenen Argumenten. Bspw.:

rpm -i package_1.2-1.i386.rpm zur Installation und

rpm -e package zur Deinstallation

2.2 Pakete unter *Debian*: *deb*

Die Linux-Distribution *Debian* kommt mit einem eigenen Paketformat daher. Dieses glänzt durch Einfachheit. Wer *deb*-Pakete schnüren möchte, muss sich nur an einige Konventionen für Dateinamen halten und kann diese mit dem Kommando *ar* zusammenpacken. Die drei Dateien, die sich in einem *deb*-Paket befinden sind:

debian-binary: enthält die Versionsnummer des Paketformates

control.tar.gz: enthält Abhängigkeitsbeschreibung, Checksummen, textuelle Beschreibung und verschiedene (optionale) Skripte zur Steuerung der Installation bzw. Deinstallation

data.tar.gz: enthält die eigentlichen Daten mit relativen Pfaden (von / aus)

Die Installation und Deinstallation erfolgt ähnlich *RPM*:

dpkg -i package1.2-1.i386.deb zur Installation und

dpkg -r package zur Deinstallation

2.3 Das *ebuild* Konzept unter *Gentoo*

Die sourcenbasierte Linux-Distribution *Gentoo* geht einen etwas ungewöhnlichen Weg, wenn man mit dem herkömmlichen Konzept von Softwarepaketen vertraut ist. Hier werden keine Pakete im eigentlichen Sinne erstellt, sondern Informationen zu Quellarchiven in einem speziellen Verzeichnisbaum verwaltet, dem s.g. *portage-tree*. Die Blätter des Verzeichnisbaumes sind die s.g. *ebuilds*. Das Entscheidende

hierbei ist, dass die eigentlichen Quellen nicht im *ebuild* (einfache Textdatei) untergebracht sind, sondern erst beim Wunsch ein entsprechendes Paket zu installieren von einem *Mirror* herunter geladen wird (z.B. via *rsync*, *http*, *ftp*). Die Informationen wo auf dem *Mirror* dieses Quellpaket zu finden ist, steht allerdings wieder im *ebuild*. Weiterhin sind im *ebuild* Abhängigkeiten und Unverträglichkeiten mit anderen Paketen eingetragen um die Stabilität des Systems zu gewährleisten.

Im Weiteren werden wir sehen, dass zu einer Paketverwaltung auch immer eine Datenbank (nicht im Sinne einer SQL-Datenbank) nötig ist um die Welten von *Mirror* und lokaler Installation im Einklang zu behalten. Interessant und erwähnenswert ist hier, dass bei *portage* die Datenbank bereits mit dem *portage-tree* abgebildet ist. Das heißt es ist lediglich eine Liste der installierten Pakete zu führen. Alle anderen Informationen zu verfügbaren Paketen sind im *portage-tree* eingetragen. Die Aktualisierung des Baumes erfolgt mit dem *emerge*-Kommando (siehe 3.4).

3 Paketmanager im Überblick

3.1 Überblick

Nachdem wir nun gesehen haben, was Paketmanagern zugrunde liegt, soll ein kleiner Überblick darüber gegeben werden, wie die einzelnen Komponenten zusammenspielen und welche Komponenten dies sind. Abbildung 1 soll verdeutlichen welche Welten miteinander in Verbindung stehen.

blauer Rahmen meint das z.B. Linux-System welches die Datenbank, die installierte Software und natürlich das Paketmanagementsystem (PMS) beinhaltet

System meint den eigentlichen Softwareinstallationsstand auf dem laufenden System

Datenbank hält Beschreibungen (Meta-Daten) welche Pakete auf welchen Medien verfügbar sind und wie diese voneinander abhängen

Medien durch eine gemeinsame Schnittstelle sind verschiedenen Medien zur Installation nutzbar (Bspw. CD/DVD/Netzinstallation/USB-Platte)

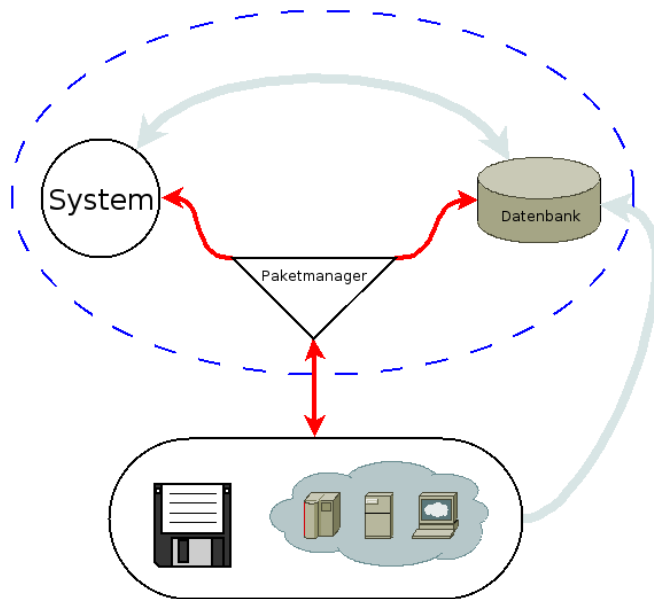


Abbildung 1: Übersicht über das Zusammenspiel der einzelnen Komponenten

Anforderungen:

1. Beachtung von Abhängigkeiten
2. Suchfunktion
3. flexibel
4. einfach
5. pflegbar

Wie aus dieser Liste erkennbar ist, muss ein Paketmanager einige Disziplinen beherrschen. Ich möchte die oben genannten Stichpunkte kurz erläutern.

Beachtung von Abhängigkeiten Sowohl bei der Installation als auch bei der Deinstallation können Abhängigkeiten zwischen verschiedenen Softwarepaketen entstehen. Diese zu beachten ist die Aufgabe eines "richtigen" Paketmanager wie *yum*, *apt* oder *emerge*. *RPM* oder *dpkg* können diesem nicht gerecht werden. Wünschenswert ist bei der Deinstallation, dass nicht mehr benötigte Pakete gleich mit deinstalliert werden, wenn das einzige Paket deinstalliert wird, welches von diesem abhängt. Allerdings sollte der Mechanismus

nicht bevormundend arbeiten, da der Nutzer vielleicht doch eine Bibliothek oder ähnliches benötigt, um eine eigene Implementierung linken zu können. Bei der Installation trifft man häufig *Dummy*-Pakete an, welche eine Art alias darstellen und gleich mehrere Pakete installieren die üblicherweise zusammen gehören (Bspw *xorg-x11* als *Dummy* könnte gleich noch wichtige Schriftarten oder ähnliches beinhalten, welche allerdings extra Pakete darstellen). Dies ist eine gute Lösung um dem Administrator Arbeit abzunehmen.

Suchfunktion Wichtig ist natürlich auch, dass man nach Softwarepaketen bequem suchen kann. Wer ein Linux-System nutzt weiß dass es sehr sehr viele solche Pakete gibt. Weiterhin bieten alle mir bekannten Systeme eine aussagekräftige Beschreibung zu den gefundenen Paketen, was die Suche oft sehr erleichtert.

flexibel Wie bereits angesprochen sollte es möglich sein, ein Installationsmedium zu wählen oder auch mehrere gleichzeitig zu nutzen. Hierzu ist eine flexible und einfache Konfiguration nötig. Diese wird durch Konfigurationsdateien gehandhabt oder mittels eines grafischen Aufsatzes für ein PMS. Im Falle von *Gentoo* bekommt die Flexibilität noch eine Dimension dazu. Hier kann man explizit durch USE-Flags angeben welche Features in die Programme mit ein-kompiliert werden sollen.

einfach Es sollten wenige Kommandos nötig sein um das PMS zu steuern und es sollte eine überschaubare Menge von Kommandozeilenargumenten nötig sein, um die alltäglichen Probleme zu lösen. Weiterhin sollten die Konfigurationsdateien zentral auffindbar und einfach editierbar sein.

pflegbar Die Paketdatenbank muss mit einfachen Mitteln auf einem aktuellen Stand gehalten werden können, so dass Sicherheitslücken schnell durch entsprechende Patches geschlossen werden können. Hier zeigt sich auch, wie wichtig ein PMS für ein öffentlich erreichbaren Server ist.

In den folgenden Punkten möchte ich auf die betrachteten PMS eingehen und aufzeigen wie die hier genannten Anforderungen erfüllt werden.

3.2 Der Aufsteiger *yum*

Yum stammt ursprünglich aus einer recht unbekanntem Linux-Distribution Namens *Yellow Dog Linux* und hieß dort *yup*. Da die Implementierung von *yup* recht unperformant war wurde schließlich eine Modifikation erarbeitet und in *yum* umgetauft (*Yellowdog Updater modified*). Diese Version wird nun z.B. von *RedHat*

und *Fedora* genutzt um die Paketverwaltung zu übernehmen. Auch *SuSE* ist eine *RPM*-basierte Distribution und soll in der neusten Version auf *yum* aufsetzen. Damit könnte ein Manko von *SuSE* behoben sein, dass man stets zur Installation *Yast* benötigt, wobei *Yast* den Service von *yum* verwendet um Pakete einzuspielen.

Yum wird über eine Konfigurationsdatei konfiguriert (*/etc/yum.conf*) und kann durch Pluggins erweitert werden. Eine sehr nützliche Fähigkeit ist, dass *yum* sich den zur Zeit schnellsten Mirror aussuchen kann, wenn dies entsprechend konfiguriert wurde. Die täglich wichtigen Aktionen können mittels

yum search search_string zum Suchen,

yum install package1 package2 ... zum Installieren und

yum remove package zum Deinstallieren ausgeführt werden.

3.3 Gute Wahl: *apt*

Das *Advanced Packaging Tool* der Linux-Distribution *Debian* ist ebenfalls ein leistungsfähiges Produkt. Wie auch *yum* ist es in der Lage alle Aktionen des PMS-Alltags zu meistern. Besondere Beliebtheit erlangt *apt* zur Zeit durch einige Endanwendersysteme wie *Knoppix* oder *Ubuntu*, die beide auf *Debian* aufsetzen und somit auch *apt* + GUI-Programme zur Softwareverwaltung nutzen. Auf der Kommandozeile lässt sich *apt* bequem mittels

apt-cache search search_string zum Suchen,

apt-get install package1 package2 ... zum Installieren und

apt-get remove package zum Deinstallieren

nutzen.

3.4 Quellen im Griff mit *portage*

Einen sehr schönen Ansatz bietet uns *Gentoo* mit seinem *portage* PMS. Man kann hier sehr feingranular einstellen, was in ein Paket einfließen soll und was man nicht unbedingt benötigt. Hierzu dienen die s.g. USE-Flags, die die entsprechenden *configure*-Argumente setzen. Ein Nachteil ist dass es natürlich recht lange dauert große Softwarepakete wie z.B. *openoffice* zu installieren. Einen Ausweg findet man hier durch bereits vor-kompilierte Pakete, die man explizit auswählen kann.

Zu nutzen ist diese PMS ebenfalls sehr einfach:

eix partofpackagename zum Suchen¹

emerge package1 package2 ... zum Installieren

emerge -unmerge package zum Deinstallieren

4 Fazit

Ich hoffe ich konnte mit diesem Beitrag die Funktionsweise von PMS näher bringen und die Wichtigkeit von diesen herausstellen. Ich selbst nutze seit Jahren *Debian* und seit einigen Monaten *Gentoo* und bin sehr angetan vom Ansatz des PMS. Allerdings gibt es zu jedem dieser Systeme extrem viel interessantes zu sagen, weswegen ich mich in diesem Artikel kurz gefasst habe, um die Übersichtlichkeit nicht zu gefährden. Ich hoffe ich konnte trotz Knappheit die interessanten Aspekte aufzeigen und möchte zur weitere Recherchen auf die Projektseiten der Distributoren verweisen.

¹*eix* ist *emerge -search* unbedingt vorzuziehen, da *emerge* beim Suchen sehr langsam arbeitet