Linux RPM HOWTO

Donnie Barnes (djb@redhat.com) und Tilo Wenzel

v2.0.1-2, 25. September 1997

In diesem HOWTO wird beschrieben, wie man den Red Hat Packagemanager `rpm` anwendet und eigene Pakete erstellt.

Inhaltsverzeichnis

1 Copyright

1.1 Copyright der englischen Version

1.2 Bemerkungen zur deutschen Übersetzung

Dieses Dokument ist die deutsche Übersetzung des englischen Originals von Donnie Barnes. In der deutschen Übersetzung wurden mit Zustimmung des Autors des Originals ein paar Korrekturen und Änderungen vom Übersetzer Tilo Wenzel vorgenommen, die das Eine oder Andere hoffentlich etwas klarer machen.

2 Einführung

RPM ist der **R**ed Hat **P**ackage **M**anager. Obwohl der Name die Bezeichnung Red Hat enthält, ist er dennoch als ein vollständig offenes System angelegt, welches jeder benutzen kann. Es gibt dem Benutzer die Möglichkeit, aus dem Quellcode für eine neue Software Pakete mit Quell- und Binärcode zu erstellen, mit denen es möglich ist, Binaries einfach zu installieren und zu verwalten sowie Quellcode einfach zu übersetzen. Er besitzt auch eine eigene Datenbank mit Informationen über alle Pakete und deren zugehörige Dateien, die zum Testen von Paketen und zum Abfragen von Informationen bezüglich Dateien und/oder Paketen benutzt werden kann.

Red Hat Software erlaubt es anderen Herstellern von Distributionen ausdrücklich, den Paketmanager auszutesten und ihn für den Aufbau der eigenen Distributionen einzusetzen. RPM ist sehr flexibel und einfach zu benutzen, obwohl es die Grundlage für ein komplexes und großes System darstellt. Es ist darüber hinaus vollständig offen und für jeden verfügbar, wir würden uns jedoch über Berichte bezüglich Fehler und deren Beseitigung freuen. Die Benutzung und Verbreitung des RPM wird lizenzfrei unter der GPL gewährt.

Eine ausführlichere Dokumentation bezüglich des RPM ist mit dem Buch *Maximum RPM* von Ed Bailey verfügbar. Dieses Buch steht zum Herunterladen oder zum Kauf auf `http://www.redhat.com` zur Verfügung.

3 Übersicht

Zunächst einige Worte über die Philosophie, die hinter dem RPM steht. Eines der Ziele des RPM war die Möglichkeit der Verwendung der *ursprünglichen* Quellen. Für den RPP (unser altes Paketsystem, aus dem sich *kein* Teil des RPM herleitet) waren die Quellpakete die gepatchten Quellen, aus denen wir die fertigen Pakete herstellten. Im Prinzip konnte man ein Quell-RPP installieren und es anschließend ohne weitere Probleme mit `make` übersetzen. Aber die Quellen waren nicht mehr die Originale, und es gab keinen Hinweis darauf, was wir für Änderungen hatten vornehmen müssen, um es zum Laufen zu bringen. Die Originalquellen mußten separat heruntergeladen werden. Mit RPM hat man sowohl die Originalquellen als auch den Patch, von dem kompiliert wurde. Unserer Meinung nach ist das aus verschiedenen Gründen ein großer Vorteil. Wenn z.B. eine neue Version eines Programmes herauskommt, ist es nicht mehr nötig, noch einmal komplett von vorne anzufangen, um es unter RHL (Red Head Linux) zu kompilieren. Man kann sich den Patch anschauen, um zu sehen, was *eventuell* zu ändern ist. Alle einzukompilierenden Defaults usw. sind so einfach herauszufinden.

RPM hat ebenfalls einen leistungsfähigen Abfragemechanismus. Man kann die gesamte Paketdatenbank nach bestimmten Paketen oder einzelnen Dateien durchsuchen und so einfach feststellen, zu welchem Paket eine Datei gehört und wo sie herkommt. Die RPM-Dateien selbst sind komprimierte Archive, die einzelnen Pakete können jedoch einfach und *schnell* nach bestimmten Informationen durchsucht werden, da ein spezieller Header im Binärformat an jedes Paket angefügt wird. Dieser Header ist unkomprimiert und enthält alle wichtigen Informationen. Das ermöglicht das *schnelle* Durchsuchen von Paketen.

Eine andere nützliche Funktion ist die Möglichkeit, Pakete zu überprüfen. Falls man vermutet, daß man ein für ein Paket wichtiges File gelöscht haben könnte, kann man dieses damit einfach kontrollieren. Man wird dann über Probleme mit der gegenwärtigen Konfiguration informiert und kann falls notwendig das Paket neu installieren. Konfigurationsdateien werden, soweit vorhanden, nicht überschrieben.

Wir möchten den Leuten von der Bogusdistribution für viele ihrer Ideen und Konzepte danken, die im RPM enthalten sind. Obwohl der RPM vollständig von Red Hat Software geschrieben wurde, basiert er doch auf von BOGUS geschriebenem Code (PM and PMS).

4 Allgemeine Informationen

4.1 Quellen für den RPM

Die beste Möglichkeit, den RPM zu installieren, ist natürlich, Red Hat Linux zu installieren. Wenn man das nicht will, kann man sich den RPM separat besorgen. Er ist von `ftp.redhat.com` zu bekommen.

4.2 Voraussetzungen für den RPM

Die Hauptvoraussetzung, um den RPM benutzen zu können, ist cpio 2.4.2 oder größer. Obwohl das System primär für den Einsatz unter Linux gedacht ist, ist es wahrscheinlich auch auf die meisten anderen UNIX-Systeme portierbar. Bisher wurde es schon auf SunOS, Solaris, AIX, Irix, AmigaOS und anderen Systemen übersetzt. Die Binärpakete sind jedoch zwischen den verschiedenen Unixtypen nicht kompatibel!

Das sind die minimalen Voraussetzungen für die Installation des RPM. Um den RPM aus den Quellen selbst zu kompilieren, braucht man auch noch die üblichen Dinge, die zum Übersetzen notwendig sind, d.h. `gcc`, `make`, usw.

5 Benutzung des RPM

Die einfachste Möglichkeit der Anwendung ist die Installation eines Paketes:

```
rpm -i foobar-1.0-1.i386.rpm
```

Ebenso einfach ist die Deinstallation eines Paketes:

```
rpm -e foobar
```

Ein etwas komplizierteres, aber *sehr* nützliches Kommando erlaubt die Installation von Paketen via FTP. Wenn man eine Verbindung ins Internet hat und ein neues Paket installieren will, muß man die Dateien nur zusätzlich mit einem gültigen URL versehen, etwa so:

```
rpm -i ftp://ftp.pht.com/pub/linux/redhat/rh-2.0-beta/RPMS/foobar-1.0-1.i386.rpm
```

Man beachte, daß der RPM jetzt via FTP arbeitet und installiert.

Außer für diese simplen Kommandos kann rpm auch noch für viele andere Aufgaben eingesetzt werden, wie dies die `Usage` Message andeutet:

```
RPM version 2.3.9
Copyright (C) 1997 - Red Hat Software
This may be freely redistributed under the terms of the GNU Public License

usage: rpm {--help}
       rpm {--version}
       rpm {--initdb}   [--dbpath <dir>]
       rpm {--install -i} [-v] [--hash -h] [--percent] [--force] [--test]
                        [--replacepkgs] [--replacefiles] [--root <dir>]
                        [--excludedocs] [--includedocs] [--noscripts]
                        [--rcfile <file>] [--ignorearch] [--dbpath <dir>]
                        [--prefix <dir>] [--ignoreos] [--nodeps]
                        [--ftpproxy <host>] [--ftpport <port>]
                        file1.rpm ... fileN.rpm
       rpm {--upgrade -U} [-v] [--hash -h] [--percent] [--force] [--test]
                        [--oldpackage] [--root <dir>] [--noscripts]
                        [--excludedocs] [--includedocs] [--rcfile <file>]
                        [--ignorearch]  [--dbpath <dir>] [--prefix <dir>]
                        [--ftpproxy <host>] [--ftpport <port>]
                        [--ignoreos] [--nodeps] file1.rpm ... fileN.rpm
       rpm {--query -q} [-afpg] [-i] [-l] [-s] [-d] [-c] [-v] [-R]
                        [--scripts] [--root <dir>] [--rcfile <file>]
                        [--whatprovides] [--whatrequires] [--requires]
                        [--ftpuseport] [--ftpproxy <host>] [--ftpport <port>]
                        [--provides] [--dump] [--dbpath <dir>] [targets]
       rpm {--verify -V -y} [-afpg] [--root <dir>] [--rcfile <file>]
                        [--dbpath <dir>] [--nodeps] [--nofiles] [--noscripts]
                        [--nomd5] [targets]
       rpm {--setperms} [-afpg] [target]
       rpm {--setugids} [-afpg] [target]
       rpm {--erase -e} [--root <dir>] [--noscripts] [--rcfile <file>]
                        [--dbpath <dir>] [--nodeps] [--allmatches]
                        package1 ... packageN
       rpm {-b|t}[plciba] [-v] [--short-circuit] [--clean] [--rcfile  <file>]
                        [--sign] [--test] [--timecheck <s>] specfile
       rpm {--rebuild} [--rcfile <file>] [-v] source1.rpm ... sourceN.rpm
       rpm {--recompile} [--rcfile <file>] [-v] source1.rpm ... sourceN.rpm
       rpm {--resign} [--rcfile <file>] package1 package2 ... packageN
```

```
rpm {--addsign} [--rcfile <file>] package1 package2 ... packageN
rpm {--checksig -K} [--nopgp] [--nomd5] [--rcfile <file>]
                    package1 ... packageN
rpm {--rebuilddb} [--rcfile <file>] [--dbpath <dir>]
rpm {--querytags}
```

Mehr dazu, was die verschiedenen Optionen bewirken, findet man in der Manpage des RPM.

6 Wozu der RPM *wirklich* gut ist

Der RPM ist ein sehr nützliches Tool und hat wie gesehen verschiedene Optionen. Der beste Weg, sie zu verstehen, ist, sich ein paar Beispiele anzuschauen. Einfaches Installieren/Deinstallieren habe ich oben behandelt, also folgen hier noch ein paar andere Beispiele:

- Sagen wir, man hat aus Versehen ein paar wichtige Dateien gelöscht. Um sein gesamtes System zu überprüfen und festzustellen, was fehlt, gibt man ein:

  ```
  rpm -Va
  ```

- Falls man eine Datei findet, von der man nicht weiß, was für eine es ist, kann man herausfinden, zu welchem Paket sie gehört, indem man eingibt:

  ```
  rpm -qf /usr/X11R6/bin/xjewel
  ```

  Die Ausgabe würde wie folgt aussehen:

  ```
  xjewel-1.6-1
  ```

- Man findet ein neues Koules rpm, hat aber keine Ahnung, was das eigentlich ist. Um ein paar Informationen dazu zu bekommen, gibt man ein:

  ```
  rpm -qpi koules-1.2-2.i386.rpm
  ```

  Die Ausgabe würde wie folgt aussehen:

  ```
  Name        : koules                 Distribution: Red Hat Linux Colgate
  Version     : 1.2                          Vendor: Red Hat Software
  Release     : 2                        Build Date: Mon Sep 02 11:59:12 1996
  Install date: (no                      Build Host: porky.redhat.com
  Group       : Games                    Source RPM: koules-1.2-2.src.rpm
  Size        : 614939
  Summary     : SVGAlib action game with multiplayer, network, and sound support
  Description :
  This arcade-style game is novel in conception and excellent in execution.
  No shooting, no blood, no guts, no gore.  The play is simple, but you
  still must develop skill to play.  This version uses SVGAlib to
  run on a graphics console.
  ```

- Wenn man sehen will, welche Dateien Koules installieren würde, hilft ein:

  ```
  rpm -qpl koules-1.2-2.i386.rpm
  ```

  Die Ausgabe ist in diesem Falle:

```
/usr/doc/koules
/usr/doc/koules/ANNOUNCE
/usr/doc/koules/BUGS
/usr/doc/koules/COMPILE.OS2
/usr/doc/koules/COPYING
/usr/doc/koules/Card
/usr/doc/koules/ChangeLog
/usr/doc/koules/INSTALLATION
/usr/doc/koules/Icon.xpm
/usr/doc/koules/Icon2.xpm
/usr/doc/koules/Koules.FAQ
/usr/doc/koules/Koules.xpm
/usr/doc/koules/README
/usr/doc/koules/TODO
/usr/games/koules
/usr/games/koules.svga
/usr/games/koules.tcl
/usr/man/man6/koules.svga.6
```

Das sind nur einige Beispiele. Andere und komplexere kann man schnell zusammenstellen, wenn man erst einmal mit dem RPM vertraut ist.

7 RPM-Pakete erstellen

Ein eigenes RPM-Paket zusammenzustellen ist recht einfach, insbesondere wenn man die Software, die man in ein Paket zusammenpacken möchte, dazu bringen kann, ohne Eingriff eines Nutzers automatisch zu kompilieren.

Der grundlegende Ablauf beim Erstellen eines RPM-Paketes ist wie folgt:

- Sicherstellen, daß `/etc/rpmrc` für das System konfiguriert ist.
- Der Sourcecode, der in das Paket soll, muß auf dem eigenem System korrekt kompilieren.
- Einen Patch für alle Veränderungen erstellen, die notwendig waren, damit die Quellen korrekt übersetzt wurden.
- Erstellen einer Spec-Datei für das Paket.
- Sicherstellen, daß alles da ist, wo es hingehört.
- Paket erstellen mit dem RPM.

Standardmäßig erzeugt der RPM sowohl Quell- als auch Binärcode.

7.1 Die Datei rpmrc

Im Moment wird die Konfiguration des RPM ausschließlich über die Datei `/etc/rpmrc` erledigt. Eine Beispieldatei könnte so aussehen:

```
require_vendor: 1
distribution: Marke Eigenbau!
require_distribution: 1
topdir: /usr/src/meins
vendor: Mickiesoft
packager:  Mickeysoft Packaging Account <packages@mickiesoft.com>
```

```
optflags: i386 -O2 -m486 -fno-strength-reduce
optflags: alpha -O2
optflags: sparc -O2

signature: pgp
pgp_name: Mickeysoft Packaging Account
pgp_path: /home/packages/.pgp

tmppath: /usr/tmp
```

Die Zeile `require_vendor` veranlaßt RPM dazu, eine Vendorzeile zu verlangen. Diese kann dann sowohl in `/etc/rpmrc` als auch im Header der Spec-Datei selbst festgelegt werden. Um dieses Feature abzuschalten, wird es auf `0` gesetzt. Gleiches gilt für die Zeilen `require_distribution` und `require_group`.

Die nächste Zeile ist die `distribution` Zeile. Diese kann entweder hier oder erst später im Header der Spec-Datei definiert werden. Wenn man das Archiv für eine bestimmte Distribution zusammenstellt, ist es sinnvoll, die Zeile entsprechend anzugeben, es ist jedoch nicht unbedingt erforderlich. Die `vendor` Zeile ist ähnlich, kann jedoch alles mögliche enthalten (z.B. Joe's Software and Rock Music Emporium).

RPM unterstützt jetzt auch das Packen von Archiven für verschiedene Architekturen. Die Datei `rpmrc` kann "optflags" Variablen beinhalten, die für einzelne Architekturen spezifische Flags für das Kompilieren usw. enthalten. Die Verwendung dieser Variablen wird in einem späteren Abschnitt behandelt.

Zusätzlich zu den obigen Makros gibt es noch einige weitere. Man kann

```
rpm --showrc
```

benutzen, um herauszufinden, wie die Tags im Moment gesetzt sind und was die verfügbaren Flags sind.

7.2 Die Spec-Datei

Kommen wir jetzt zu den Spec-Dateien. Sie sind erforderlich, um ein Paket zu erstellen. Sie enthalten eine Beschreibung der Software, Hinweise zum Übersetzen der selbigen und eine Dateiliste aller Binaries, die installiert werden.

Man sollte die Spec-Datei nach der Standardkonvention benennen. Diese lautet Paketname-Strich-Versionsnummer-Strich-Releasenummer-Punkt-spec.

Hier ein Beispiel für eine Spec-Datei (vim-3.0-1.spec):

```
Summary: ejects ejectable media and controls auto ejection
Name: eject
Version: 1.4
Release: 3
Copyright: GPL
Group: Utilities/System
Source: sunsite.unc.edu:/pub/Linux/utils/disk-management/eject-1.4.tar.gz
Patch: eject-1.4-make.patch
Patch1: eject-1.4-jaz.patch
%description
This program allows the user to eject media that is autoejecting like
CD-ROMs, Jaz and Zip drives, and floppy drives on SPARC machines.

%prep
%setup
%patch -p1
```

```
%patch1 -p1

%build
make RPM_OPT_FLAGS="$RPM_OPT_FLAGS"

%install
install -s -m 755 -o 0 -g 0 eject /usr/bin/eject
install -m 644 -o 0 -g 0 eject.1 /usr/man/man1

%files
%doc README COPYING ChangeLog

/usr/bin/eject
/usr/man/man1/eject.1
```

Die erläuternden Texte wie `%description` oder `Summary` sollten sinnvollerweise in Englisch verfaßt werden, es sei denn, es handelt sich um ein Paket, das nur für deutschsprachige Anwender gedacht ist.

7.3 Der Header

Der Header hat einige Standardfelder, die angegeben werden müssen. Es gibt auch ein paar kleine Problemzonen", die man kennen sollte. Die Felder haben folgenden Inhalt:

- `Summary:` Kurze Beschreibung des Paketes (eine Zeile)
- `Name:` Namensstring aus dem Dateinamen, den das RPM-Paket erhalten soll
- `Version:` Versionsstring aus dem Dateinamen, den das RPM-Paket erhalten soll
- `Release:` Releasenummer für Pakete gleicher Version (d.h. wenn man nach dem Zusammenpacken und Veröffenlichen seines Paketes feststellt, daß es doch noch den einen oder anderen kleineren Bug enthält, bekommt die neue Ausgabe nur eine neue, um eins höhere Releasenummer)
- `Icon:` Name der Icondatei, die von anderen Highlevel-Installationstools verwendet wird (wie z.B. Red Hat's "glint"). Es muß ein Gif sein, und sich im SOURCES-Verzeichnis befinden.
- `Source:` Diese Zeile verweist auf die Stelle, von der die originalen Quellen stammen. Diese kann man dann verwenden, wenn man einmal das ursprünglichen Paket haben will, oder nach neueren Versionen sucht. Nachteil: Der Dateiname in dieser Zeile muß dem Dateinamen auf dem eigenen System entsprechen, d.h. man sollte das heruntergeladene Archiv nicht umbenennen. Man kann auch mehrere Quelldateien mit mehreren Zeilen wie hier angeben:

  ```
  Source0: ftp://sunsite.unc.edu/pub/foo/bar/blah-0.tar.gz
  Source1: ftp://sunsite.unc.edu/pub/foo/bar/blah-1.tar.gz
  Source2: ftp://sunsite.unc.edu/pub/foo/bar1/fooblah.tar.gz
  ```

  Diese Dateien würden im `SOURCES` Verzeichnis gespeichert werden. Die Verzeichnisstruktur wird in einem späteren Abschnitt diskutiert: "Die Struktur der Quellenverzeichnisse".

- `Patch:` Die Stelle, von der man den Patch wenn nötig herunterladen kann. Bedingung: Der Dateiname muß hier dem Namen entsprechen, der benutzt wird, wenn man den Patch SELBST anwendet. Auch hier kann man wieder mehrere Patchdateien angeben:

  ```
  Patch0: ftp://sunsite.unc.edu/pub/foo/bar/blah-0.patch
  Patch1: ftp://sunsite.unc.edu/pub/foo/bar/blah-1.patch
  Patch2: ftp://sunsite.unc.edu/pub/foo/bar1/fooblah.patch
  ```

Diese Dateien werden auch im `SOURCES` Verzeichnis gespeichert.

- `Copyright:` Das Copyright des Paketes. Hier sollte etwas von der Art GPL, BSD, MIT, public domain, distributable, oder commercial stehen.
- `BuildRoot:` Hier kann ein Verzeichnis festgelegt werden, welches als "root"-Verzeichnis für die Übersetzung und Installation des Paketes dient. Man kann diese Möglichkeit einsetzen, um ein Paket erst zu testen, bevor man es endgültig auf dem Rechner installiert.
- `Group:` Diese Zeile teilt einem Highlevel-Installationsprogram ,wie z.B. Red Hat's "glint", mit, wo in der Pakethierarchie es einzuordnen ist. Der Groups-Baum sieht zur Zeit etwa so aus:

```
Applications
    Communications
    Editors
        Emacs
    Engineering
    Spreadsheets
    Databases
    Graphics
    Networking
    Mail
    Math
    News
    Publishing
        TeX
Base
    Kernel
Utilities
    Archiving
    Console
    File
    System
    Terminal
    Text
Daemons
Documentation
X11
    XFree86
        Servers
    Applications
        Graphics
        Networking
    Games
        Strategy
        Video
    Amusements
    Utilities
    Libraries
    Window Managers
Libraries
Networking
    Admin
    Daemons
    News
    Utilities
```

```
Development
    Debuggers
    Libraries
        Libc
    Languages
        Fortran
        Tcl
    Building
    Version Control
    Tools
Shells
Games
```

- `%description` Dies ist kein direkter Teil des Headers, wird aber trotzdem hier kurz mit erwähnt. Dies ist ein mehrzeiliges Feld, und sollte eine kurze Beschreibung des Paketes geben. Man benötigt einen Punkt Beschreibung pro Paket und/oder Unterpaket.

7.4 Prep

Das ist der zweite Abschnitt in der Spec-Datei. Hiermit werden die Quellen fürs Übersetzen angepaßt. Man sollte hier alle Dinge veranlassen, die nötig sind, um die Quellen zu patchen und für ein `make` vorzubereiten.

Anmerkung: Jeder dieser Abschnitte ist eigentlich nur ein Platz, an dem Shellskripte ausgeführt werden. Man könnte einfach ein Shellskript schreiben, das nach dem `%prep` Tag eingetragen wird und die Quellen entpackt und patcht. Wir haben jedoch Makros geschrieben, die das ein wenig vereinfachen sollen.

Das erste dieser Makros ist das `%setup` Makro. Im einfachsten Fall (keine Parameter auf der Kommandozeile) entpackt es nur die Quellen und macht ein `cd` ins Quellverzeichnis. Es versteht die folgenden Optionen:

- `-n name` setzt den Namen des Übersetzungsverzeichnisses auf den angegebenen Namen `name`. Der Defaultwert ist `$NAME-$VERSION`. Andere Möglichkeiten sind `$NAME`, `${NAME}${VERSION}` oder was immer das Haupttararchiv enthält. (Man beachte, daß die hier genannten "$" Variablen *keine* echten Variablen, die im Specfile zur Verfügung stehen, sind, sondern nur als Platzhalter anstelle von Beispielnamen stehen. Man muß die echten Namen und Versionen der Pakete einsetzen, keine Variablen.)
- `-c` erstellt und wechselt in das angegebene Verzeichnis *vor* dem Entpacken.
- `-b #` entpackt Source# *vor* dem Wechseln in das Verzeichnis (nicht in Verbindung mit `-c` verwenden). Diese Option ist nur sinnvoll für mehrere Quelldateien.
- `-a #` entpackt Source# *nach* dem Wechseln in das Verzeichnis.
- `-T` Diese Option verhindert die Default-Aktion des Entpackens der Quellen und erfordert ein `-b 0` oder `-a 0` um die Hauptquelldatei zu entpacken. Das wird benötigt, wenn es noch andere Quellen gibt.
- `-D` Das Verzeichnis *nicht* vor dem Entpacken löschen. Das ist nur sinnvoll, wenn man mehr als ein Setup-Makro hat. Die Option sollte *nur* in Setup-Makros *nach* dem ersten Makro benutzt werden, aber nie im ersten.

Das nächste verfügbare Makro ist das `%patch` Makro. Dieses Makro erleichtert das Anwenden von Patches auf die Quellen. Es kennt die folgenden Optionen

- # verwendet Patch# als Patchfile.
- `-p #` gibt die Anzahl der Verzeichnisse an, die für das patch(1) Kommando vom Pfad (siehe auch Man-Page von `patch`)entfernt werden.

- `-pN` Führt ein `patch -pN < $RPM_SOURCE_DIR/patchname/` aus, wobei patchname der Name des Patches aus der `Patch`-Zeile im Header ist. Diese Option ist nützlich für die Anwendung weiterer Patches, wenn für diese eine andere Anzahl von Verzeichnissen vom Pfad entfernt werden muß als für das erste Makro.
- `%patch#` ist an Stelle des richtigen Kommandos `%patch # -pN` auch möglich

Das sollten alle Makros sein, die man benötigt. Wenn man diese richtig verstanden hat, kann man auch andere Arten von Setup's via `sh`-Skripten erstellen. Alles was bis zum `%build` Makro, näher erläutert im nächsten Abschnitt, eingefügt wird, wird von der `sh` abgearbeitet. Siehe das Beispiel oben für Aktionen, die hier ausgeführt werden können.

7.5 Build

Für diesen Abschnitt gibt es keine eigenen Makros. Hier werden alle Kommandos eingetragen, die notwendig sind, um ein Programm zu übersetzen, nachdem die Quellen entpackt und gepacht wurden und man ins entsprechende Verzeichnis gewechselt hat. Auch dies hier ist wieder nur eine Folge von Anweisungen, die der `sh`zur Ausführung übergeben werden, d.h. alle gültigen `sh`-Kommandos können hier benutzt werden, inklusive Kommentare. **Das aktuelle Verzeichnis wird nach jedem dieser Abschnitte wieder auf das Toplevel-Verzeichnis der Quellen gesetzt**, man sollte das immer beachten. Man kann, wenn notwendig, in ein Unterverzeichnis wechseln.

7.6 Install

Auch für diesen Abschnitt gibt es keine eigenen Makros. Es werden hier ebenfalls alle Kommandos aufgelistet, die notwendig sind, um das fertige Programm zu installieren. Falls das Paket ein `make install` hat, kann man es hier einsetzen. Wenn nicht, kann das Makefile entsprechend gepatcht werden, oder die Installation von Hand mit Shellkommandos bewerkstelligt werden. Das aktuelle Verzeichnis ist hier wieder das Toplevel-Verzeichnis der Quellen.

7.7 Optionale Pre- und Postinstall/deinstall Skripten

Hier können Skripten angegeben werden, die vor und nach der Installation und der Deinstallation von fertig übersetzten Paketen abgearbeitet werden. Einer der Hauptgründe dafür ist das Ausführen von speziellen Programmen, wie z.B. `ldconfig` nach dem Installieren oder Entfernen von Paketen mit Shared Libraries. Die Makros für die jeweiligen Skripte sind:

- `%pre` ist das Makro zum Ausführen von Pre-Install Skripten.
- `%post` ist das Makro zum Ausführen von Post-Install Skripten.
- `%preun` ist das Makro zum Ausführen von Pre-Deinstall Skripten.
- `%postun` ist das Makro zum Ausführen von Post-Deinstall Skripten.

Der Inhalt dieser Abschnitte ist ein beliebiges `sh`-Skript, jedoch *ohne* die `#!/bin/sh` Zeile.

7.8 Files

In diesem Abschnitt *müssen* die Dateien des fertig übersetzten Paketes angegeben werden. RPM hat keine Möglichkeit, herauszufinden, welche Dateien als Ergebnis eines `make install` im System installiert werden. Hierfür gibt es *KEINE* Möglichkeit. Es gab Vorschläge, dieses mit einem `find` vor und nach dem Installieren zu bewerkstelligen. In einem Multiusersystem, bei dem zur gleichen Zeit von anderen Prozessen Dateien geschaffen werden können, die mit dem Paket nichts zu tun haben, ist dies jedoch nicht sinnvoll.

Hier gibt es ebenfalls einige Makros für spezielle Zwecke. Folgende Makros stehen zur Verfügung:

- `%doc` bezeichnet Dokumentation aus dem Quellpaket, die zusammen mit den übersetzten Binärpaketen installiert werden soll. Sie wird in `/usr/doc/$NAME-$VERSION-$RELEASE` installiert. Man kann mehrere Dokumente auf einer Kommandozeile auflisten oder je eins pro separatem Makro.
- `%config` bezeichnet Konfigurationsdateien aus dem Paket. Dies beinhaltet z.B. Dateien wie sendmail.cf, passwd usw. Wenn man später das Paket deinstalliert, werden unveränderte Kofigurationsdateien mit dem Paket gelöscht und veränderte mit einem angehängten `.rpmsave` gesichert. Auch hier können einem Makro mehrere Dateien übergeben werden.
- `%dir` bezeichnet ein Verzeichnis in einer Dateiliste, welches komplett mit zum Paket gehört. Normalerweise wird ein Verzeichnis, das *OHNE* ein `%dir` Makro aufgeführt wird, zusammen mit seinem *KOMPLETTEN* Inhalt in die Dateiliste aufgenommen und später mit dem Paket zusammen installiert.
- `%files -f <filename>` ermöglicht das Auflisten der zugehörigen Dateien in einer beliebigen anderen Datei im Übersetzungsverzeichnis der Quellen. Das ist sinnvoll, wenn man ein Paket hat, das seine eigene Dateiliste zusammenstellen kann. Man kann diese Liste dann hier angeben, und muß sie dann nicht extra hier einfügen.

Das größte Problem der Dateilisten ist das Auflisten von Verzeichnissen. Wenn man aus Versehen `/usr/bin` mit auflistet, werden *alle* Dateien in `/usr/bin` des Systems als zum Paket gehörig betrachtet. .

7.9 Übersetzen

7.9.1 Die Struktur der Quellenverzeichnisse

Zunächst braucht man natürlich einen passenden Verzeichnisbaum für die Quelldateien. Man kann diesen in der Datei `/etc/rpmrc` festlegen, üblicherweise wird hier einfach `/usr/src` verwendet.

Eventuell müssen die folgenden Verzeichnisse angelegt werden, um einen Verzeichnisbaum für das Übersetzen zu erstellen:

- `BUILD` ist das Verzeichnis, in dem die Übersetzung durch den RPM stattfindet. Man muß seine Testübersetzungen nicht in diesem speziellen Verzeichnis durchführen, aber hier ist es, wo der RPM *seine* Übersetzungen durchführt.
- `SOURCES` hier sollten alle tar-Archive der originalen Quellen und die Patches stehen. Hier sucht der RPM per Default nach den Quelldateien.
- `SPECS` hier gehören die Spec-Dateien hin.
- `RPMS` das Verzeichnis für die fertig übersetzten rpm's.
- `SRPMS` das Verzeichnis für die Quell-rpm's.

7.9.2 Testübersetzung

Zunächst sollte man dafür sorgen, daß die Quellen korrekt ohne den RPM übersetzt werden. Dazu entpackt man die Quellen und benennt das Verzeichnis um nach $NAME.orig. Danach werden die Quellen ein zweites Mal entpackt. Diese zweite Version wird dann zum Übersetzen benutzt. Man wechselt ins Quellverzeichnis und folgt den Anweisungen zum Übersetzen des Paketes. Wenn man, um die Quellen zu übersetzen, Änderungen an ihnen vornehmen mußte, braucht man ein Patch. Um einen solchen zu erstellen, entfernt man alle Dateien, die von einem `configure` Skript o.ä. angelegt wurden, aus dem Verzeichnis, das die jetzt vollständig übersetzbaren Quellen enthält. Danach wechselt man in das übergeordnete Verzeichnis des obersten Quellverzeichnisses. Danach wird der Patch erstellt mit

```
diff -uNr verzname.orig verzname > ../SOURCES/dirname-linux.patch
```

o.ä., je nach konkretem Fall. Das erzeugt einen Patch, der in der Spec-Datei verwendet werden kann. Der "linux" Teil des Patchnamens ist hier nur als Beispiel, man kann hier einen Namen verwenden, der etwas mehr über den *Grund* für diesen Patch aussagt, z.B. "config" oder "bugs". Man sollte außerdem noch einen Blick in die fertige Patchdatei werfen, um sicherzustellen, daß nicht aus Versehen Binaries o.ä. mit eingetragen wurden.

7.9.3 Erstellen der Dateiliste

Nachdem man funktionierende Quellen hat, werden diese übersetzt und installiert, so wie es beim Endbenutzer geschehen würde. Aus dem Resultat dieser Installation wird dann die Dateiliste erstellt. Wir erstellen die Spec-Datei normalerweise gleichzeitig mit den beschriebenen Schritten. Man erstellt eine Startversion der Datei und setzt die einfachen Teile ein, und ergänzt dann nach und nach die anderen Teile im weiteren Verlauf.

7.9.4 Erstellen des Paketes mit dem RPM

Wenn die Spec-Datei fertig ist, kann man versuchen, ein RPM-Paket zu erzeugen. Dies geschieht meistens mit einem

```
rpm -ba foobar-1.0.spec
```

Es gibt weitere Optionen, die mit dem `-b` Switch zusammen nützlich sind:

- `p` Ausführen des `prep` Abschnittes der Spec-Datei.
- `l` ist ein Listencheck, der einige Tests mit `%files` macht.
- `c` Ausführen von prep und kompilieren. Kann verwendet werden, um zu testen, ob die Quellen auch wirklich durchkompilieren. Das sieht auf den ersten Blick etwas sinnlos aus, da man seine Quellen üblicherweise erst zum Übersetzen bringt und dann erst den RPM anwirft, aber wenn man erst mit dem RPM vertraut ist, gibt es Gelegenheiten, in denen das sinnvoll ist.
- `i` Ausführen von prep, kompilieren, installieren.
- `b` Ausführen von prep, kompilieren, installieren, erstellen nur des Binärpaketes.
- `a` alles erstellen (Quell- und Binärpakete).

Es gibt noch einige Optionen zum Modifizieren des Ablaufs des `-b`-Switch:

- `-short-circuit` beginnt direkt mit dem angegebenen Abschnitt (nur zusammen mit c und i).
- `-clean` entfernt nach Beendigung die Übersetzungsverzeichnisse.
- `-keep-temps` behält alle temporären Dateien und Skripte, die in /tmp angelegt wurden. Mit der Option `-v` kann man sehen, was in /tmp erzeugt wurde.
- `-test` führt keine wirklichen Aktionen aus, aber bewahrt temp-Dateien.

7.10 Austesten

Wenn man ein fertiges Paket für die Quellen und die Binaries hat, sollte man es testen. Das Beste und Einfachste ist es, das Paket auf einem völlig anderen Rechner zu testen als dem, auf dem man es erstellt hat. Schließlich hat man das Paket bereits mehrfach auf diesem Rechner mit `make install` o.ä. installiert, so das es nun bereits recht gut installiert sein sollte.

Man kann das Paket mit `rpm -u packagename` zwar wieder deinstallieren, wenn man aber in der Liste der Dateien eine oder mehrere vergessenen hat, die aber durch das `make install` installiert wurden, werden diese nicht wieder deinstalliert. Wenn man danach das Paket wieder installiert, ist es wieder vollständig auf dem Rechner, auch wenn das rpm selbst unvollständig ist. Darüber hinaus sollte man beachten, daß man selbst zwar eine Testinstallation mit `rpm -ba package` macht, die meisten Leute jedoch nur ein `rpm -i package` machen werden. Man muß daher darauf achten, daß in den `build` oder `install` Abschnitten nichts getan wird, was auch für eine reine Installation der Binaries notwendig ist.

7.11 Wenn das neue rpm fertig ist

Wenn man ein neues rpm von irgend etwas fertiggestellt hat, kann man es - unter der Voraussetzung, daß es noch nicht als rpm existiert - anderen zur Verfügung stellen. Hierbei muß das, was man zusammengepackt hat, natürlich ein entsprechendes Copyright haben. Man kann es dann auf den FTP-Server `ftp.redhat.com` stellen.

7.12 Was weiter?

Es sei hier noch einmal besonders auf die vorhergehenden Abschnitte zum Testen und der Verwendung der fertigen rpm's hingewiesen. Wir wollen so viele rpm's wie möglich in guter Qualität zur Verfügung stellen. Bitte die rpm's gründlich austesten und dann zum Nutzen aller ins Netz stellen. *Unbedingt* sollte man jedoch vorher sicherstellen, daß das Copyright dieses auch *zuläßt*. Kommerzielle Software und Shareware sollte nicht öffentlich zur Verfügung gestellt werden, es sei denn, das Copyright der entsprechenden Software läßt das ausdrücklich zu. Das gilt z.B. für Programme wie Netscape, ssh, pgp, usw.

8 RPM's für verschiedene Architekturen

RPM unterstüzt jetzt das Erstellen von Paketen für Intel i386, Digital Alpha mit Linux und Sparc. Berichte besagen, daß es auch auf SGI's und HP's funktionieren soll. Eine Reihe von Eigenschaften vereinfachen das Erstellen von Paketen auf allen Plattformen. Die erste ist die "optflags" Direktive in `/etc/rpmrc`. Sie kann dazu benutzt werden, Flags, die beim Übersetzen von Paketen verwendet werden, auf architekturspezifische Werte zu setzen. Ein anderes Beispiel sind die "arch" Makros in der Spec-Datei. Sie können dazu benutzt werden, unterschiedliche Aktionen auszulösen, je nachdem, auf welcher Architektur übersetzt wird. Eine andere Möglichkeit ist die "Exclude" Direktive im Header.

8.1 Beispiel für eine Spec-Datei

Der folgende Ausschnitt ist Teil des "fileutils"-Paketes. Es ist für das Übersetzen auf der Intel- und Alphaplattform konfiguriert.

```
Summary: GNU File Utilities
Name: fileutils
Version: 3.16
Release: 1
```

```
Copyright: GPL
Group: Utilities/File
Source0: prep.ai.mit.edu:/pub/gnu/fileutils-3.16.tar.gz
Source1: DIR_COLORS
Patch: fileutils-3.16-mktime.patch

%description
These are the GNU file management utilities.  It includes programs
to copy, move, list, etc, files.

The ls program in this package now incorporates color ls!

%prep
%setup

%ifarch alpha
%patch -p1
autoconf
%endif
%build
configure --prefix=/usr --exec-prefix=/
make CFLAGS="$RPM_OPT_FLAGS" LDFLAGS=-s

%install
rm -f /usr/info/fileutils*
make install
gzip -9nf /usr/info/fileutils*

.
.
.
```

8.2 Optflags

In diesem Beispiel wird gezeigt, wie man die "optflags"-Direktive aus `/etc/rpmrc` einsetzt. In Abhängigkeit von der jeweiligen Architektur wird die Variable `RPM_OPT_FLAGS` auf den entsprechenden Wert gesetzt. Das Makefile muß angepaßt werden, damit hier dann diese Variable anstelle der üblichen Direktiven wie `-m486` oder `-O2` verwendet wird. Man kann sich ein Gefühl dafür verschaffen, was gemacht werden muß, indem man sich erwähntes Quellpaket installiert und das Makefile ansieht. Dazu kann man sich dann im Vergleich den Patch anschauen, um zu sehen, welche Änderungen notwendig sind.

8.3 Makros

Sehr wichtig ist das Makro `%ifarch`. Meistens muß man den einen oder anderen Patch machen, der nur für eine bestimmte Architektur angewandt werden soll. In diesem Fall sorgt der RPM mit Hilfe dieses Makros für die korrekte Verwendung der Patches. Im obigen Beispiel hat das fileutils-Paket einen Patch für 64-bit Architekturen. Dieser Patch soll natürlich im Moment nur für Alpha-Rechner angewandt werden. Deshalb wird ein `%ifarch` Makro um den 64-bit Patch herumgelegt:

```
%ifarch axp
%patch1 -p1
%endif
```

Das stellt sicher, daß der Patch ausschließlich für Alphas verwendet wird.

8.4 Ausschließen von Architekturen in Paketen

Damit man die Quell-rpm's für alle Plattformen weiterhin in einem Verzeichnis aufbewahren kann, haben wir die Möglichkeit geschaffen, Pakete von der Übersetzung auf bestimmten Architekturen auszuschließen. Das ermöglicht weiterhin Dinge wie eine Übersetzung ganzer Gruppen von Paketen mit einem Mal, z.B. mit einem

```
rpm --rebuild /usr/src/SRPMS/*.rpm
```

,wobei man trotzdem dann nur die passenden übersetzten Pakete erhält. Falls ein Programm noch nicht auf eine bestimmte Plattform portiert wurde, reicht ein Einfügen einer Zeile wie:

```
ExcludeArch: axp
```

in den Header der Spec-Datei des Quellpaketes. Danach erstellt man das rpm auf der Architektur, auf der es läuft, und erhält so ein Paket, das z.B. auf Intel übersetzt wird, aber auf Alphas einfach übergangen wird.

8.5 Zusammenfassung

Oft ist es einfacher, den RPM zum Erstellen von Paketen für mehrere Architekturen einzusetzen als das Paket selber dazu zu bringen. Wie üblich ist der beste Weg, wenn man mit seinem rpm nicht mehr weiter weiß, einen Blick auf ähnliche andere rpm's zu werfen.