Using RPM-SuperVisor

Klaus Franken (rpmsv@klaus.franken.de), v1.11, 06.11.2001

RPM-SuperVisor: dieses Tool bestimmt, was der RedHat Package Manager zu tun hat.

Contents

1 Idee

Es kann mühsam und langweilig sein, die Auswahl der benötigten Pakete für einen einzelnen Rechner zu verwalten. Irgendein Paket fehlt dann doch immer wieder oder ist nicht aktuell genug. Auch der umgekehrte Fall ist schädlich: eine Unmenge von Paketen ist installiert, die nicht gebraucht werden, und niemand traut sich etwas zu löschen.

RPMSV versucht die zu installierenden Pakete zu verwalten, also festzustellen, welche Pakete

1. installiert
2. upgedatet
3. gelöscht

werden sollen und kann dies direkt ausführen.

Verwaltet werden können Pakete im Format des **RedHat Package Managers** (RPM), der von verschiedenen Linux-Distributionen wie **RedHat** oder **SuSE** benutzt wird, aber auch für andere Unice verfügbar ist. Siehe *http://www.rpm.org* <http://www.rpm.org>. RPMSV greift dabei direkt auf den RPM zu (ruft ihn jeweils mit entsprechenden Kommandozeilen auf). Es handelt sich also um einen Manager für einen Manager und wird daher *Supervisor* genannt.

Besonderes Augenmerk wurde darauf gelegt, daß möglichst viel der Konfiguration auf dem Server an zentraler Stelle geschieht und damit ein hohes Maß an Standardisierung erreicht wird. Zentraler Bestandteil ist die Definition von (Paket-) **Gruppen**, die die Aufgabe eines Systems beschreibt. Siehe nächstes Kapitel.

1.1 Auswahl der Pakete (Gruppen)

Besondere Anforderung an RPMS ist es, viele Rechner zu verwalten, aber dennoch sollen oder können nicht alle Rechner gleich installiert werden. Ein Rechner soll vielleicht ein Datenbankserver sein, ein anderer eine Workstation. Ein dritter Rechner soll für Testzwecke beides sein.

In allen Fällen soll aber die Basisauswahl an Paketen identisch sein, pro Aufgabe werden nur zusätzliche Pakete benötigt.

Die Pakete, die für eine Aufgabe benötigt werden, werden in **Gruppen** zusammengefaßt. Aus Sicht eines Rechners wird nun entschieden, zu welchen Gruppen er gehören will, sozusagen *abonniert* der Rechner diese Gruppen.

Aus der Summe aller Pakete aller abonnierten Gruppen kann nun genau entschieden werden, welche Pakete installiert sein sollen - und welche nicht. Das Ermitteln der Pakete geschieht immer online, d.h. es wird ermittelt, zu welchen Gruppen ein System gehört und welche Pakete aktuell installiert sind, daraus wird ermittelt, welche Pakete neu installiert werden müssen und welche gelöscht werden müssen (update: s.u.). Die Gruppenzugehörigkeit kann jederzeit geändert werden.

Die einzelnen Gruppen dürfen sich dabei durchaus überschneiden, es wird immer die Gesamtmenge ermittelt. In der Praxis könnte es aber sinnvoller sein, Basisgruppen zu definieren.

Im Beispiel oben würde man drei Gruppen definieren:

- `base` - Basissystem
- `database` - Alle Pakete, die für die DB benötigt werden
- `workstation` - Alle X-, KDE- etc. Pakete.

Rechner 1 würde zu `base` und `database` gehören, Rechner 2 zu `base` und `workstation`. Soll auf Rechner 2 ein Datenbanktest gemacht werden, wird `database` abonniert. Nach dem Test wird das Abonnement von `database` aufgehoben und alle Pakete wieder gelöscht (sofern sie nicht auch in `base` oder `workstation` sind).

Die Paketbeschreibungen können nachträglich geändert werden, soll z.B. auf jedem Datenbankrechner ein Diagnosetool installiert sein, wird dieses in die Beschreibung der Gruppe `database` eingetragen und alle Abonnenten dieser Gruppe werden das Diagnosetool beim nächsten Updatelauf nachinstallieren.

1.2 Paketquellen und Updates

Die Pakete liegen möglichst immer zentral auf einem **Server**. Der zu administrierende Rechner, hier der **Client**, greift über das Filesystem darauf zu, also über NFS oder ähnliches. Eine Erweiterung auf das FTP-Protokoll wäre denkbar und relativ einfach, erscheint aber momentan nicht notwendig.

Auf dem Server können die RPM's in verschiedenen Verzeichnissen liegen, man kann also z.B. auseinanderhalten, welche Pakete von der originalen Distribution kommen, welche Update-Pakete man sich besorgt hat und welche Pakete man sich selbst erstellt hat. Dem Client ist der genaue Ort vergleichsweise egal, dort werden die Pakete nur über den Namen (ohne Versionsnummer etc.) angesprochen.

Die Verzeichnisse mit dem RPM-Paketen müssen nicht modifiziert werden und können auch read-only im Zugriff sein. Man kann also die Original-CDROM der Distribution verwenden.

Aus Performancegründen werden aber nicht immer online alle vorhandenen Pakete untersucht, sondern es werden **lists** über alle vorhandenen Pakete gepflegt, die in einem gesonderten Verzeichnis abgelegt werden. Diese lists werden durch `rpmsv` erstellt und beinhalten Name der Datei, Name des Pakets und Versionsnummer.

Wie oben beschrieben, kann ein Programmpaket in verschiedenen Versionen vorhanden sein, z.B. das Paket von der Originaldistribution und ein Updatepaket. Man kann folgendermaßen damit umgehen:

- direkt: in der Gruppenbeschreibung wird direkt auf eine bestimmte Versionsnummer verwiesen, die installiert werden soll. Diese Methode ist am wenigsten elegant und nur für den *Notfall* gedacht.
- Installation: Wird ein Paket zur Installation ausgewählt, wird automatisch die neueste vorhandene Version benutzt.
- Update Version 1: beim *normalen* Update (Schalter `-u`) wird geprüft, ob die installierte Version noch auf dem Server vorhanden ist, wenn ja wird nichts unternommen. Wenn nicht, wird auf die neueste vorhandene Version upgedatet.

  Damit soll ein *sanfter* Updateweg geschaffen werden: nicht die gesamte Rechnerlandschaft soll gezwungen werden sofort upzudaten, denn eventuell ist ein Update mit Nacharbeiten verbunden und die neue Version ist vielleicht nur bei neuen Installationen sinnvoll (*never touch a running system*).

  Will man dennoch den Update erzwingen, löscht man das alte, nicht mehr zu verwendene Paket vom Server (das ist bei einer CDROM etwas schwierig, hier fehlt noch die Möglichkeit einzelne Paketdateien auszuschließen...)
- Update Version 2: Will man auf jeden Fall das neueste Paket installieren, gibt man den Schalter `-n` anstatt `-u` an.

  Dieser Weg ist nicht nur Versionsjunkies, sondern vor allem für den Test der neuen Pakete gedacht.

2 Installation

2.1 Vorausetzungen:

RPMSV ist ein Perl-Script. Entwickelt und getestet mit Version 5.005_03. Andere Verionen sollten auch funktionieren.

Der Server könnte ein beliebiges System sein, das NFS-Server (oder ein anderes Network Filesystem) beherrscht, sein. Lediglich zum Erstellen der `lists` wird dort `rpmsv` gestartet, welches Perl voraussetzt und einen RPM-Query Aufruf durchführt; dieses könnte man auch auf einem NFS-Client ausführen, was aber aus Performancegründen nicht empfehlenswert ist.

Der Client sollte ein RPM-basiertes Linux-System sein (z.B. SuSE oder RedHat) und muß Perl installiert haben. Entwickelt und getestet wurde `rpmsv` mit RedHat (6.0, 6.1, 6.1EV).

2.2 Download und Homepage

Siehe: *http://www.franken.de/users/klaus/rpmsv*

2.3 RPMSV installieren.

Da `rpmsv` nur sinnvoll mit einem RPM-basierten System zu benutzen ist, wird es auch ausschließlich als RPM-Paket verteilt. Im Folgenden wird davon ausgegangen, daß sowohl Server wie Client Linux-Systeme sind.

Sowohl auf dem Server wie auf dem Client wird das rpmsv-Paket installiert, z.B. mit:

```
rpm -ihv rpmsv-1.6-1.i386.rpm
```

Wünschenswert wäre, das `rpmsv` gleich bei der Grundinstalltion installiert wird... Zumindest aber sollte man `rpmsv` in die Basisgruppe aufnehmen, damit es durch sich selbst upgedatet werden kann - außerdem würde es sich sonst selbst löschen ;-(

2.4 Directory-Struktur

Folgende Files und Directories werden durch das rpmsv-Paket installiert:

- `/usr/bin/rpmsv` Der *RPM SuperVisor*
- `/etc/rpmsv/` Konfigurationdirectory für `rpmsv`
- `/etc/rpmsv/rc.local` lokale Konfiguration für `rpmsv`
- `/etc/rpmsv/groups/` Abonnierte Gruppen für das lokale System
- `/usr/lib/rpmsv/` Directory für einige Zusätze
- `/usr/lib/rpmsv/rc.global.example` Beispiel für Konfiguration des Servers
- `/usr/lib/rpmsv/scripts/` Scripte für `rpmsv`
- `/usr/doc/rpmsv-?.?/` Dokumentation

2.5 Server-Konfiguration

Der Aufbau eines RPMSV-Server erfordert (im Gegensatz zum Client) ein wenig Handarbeit.

Im folgenden gehe ich von folgenden Pfaden aus:

`/rzlinux`

Das Basisverzeichnis für RPMSV. Dieses Verzeichnis wird (read-only) an alle Clients exportiert und unter diesem Namen gemountet.

`/rzlinux/rh61`

Eventuell werden verschiedene Distributionen gepflegt, z.B. RedHat 6.0 und RedHat 6.1. Hier werden die Daten für RedHat 6.1 in `/rzlinux/rh61` gehalten.

`/rzlinux/rh61/rc.global`

Die globale Konfiguration für diese Distribution. Diese Datei wird von den Clients aus der lokalen Konfiguration heraus eingelesen.

`/rzlinux/rh61/groups`

Die Gruppenbeschreibungen. Pro Gruppe wird eine Datei mit dem Namen der Gruppe als Dateiname angelegt, in der die Pakete aufgezählt werden. Leerzeilen und Kommentare (beginnend mit #) sind erlaubt.

`/rzlinux/rh61/lists`

Hier werden für jedes definierte Directory mit RPM-Paketen Listen gehalten, die mit `rpmsv -lists` automatisch generiert werden.

`/rzlinux/rh61/rpms`

Directory für die RPM-Pakete. Hier werden weitere Subdirectories für RPM-Pakete verschiedener Herkunft angelegt. Jedes Subdirectory wird in der Konfigurationsdatei eingetragen, normalerweise also in `/rzlinux/rh61/rc.global`.

`/rzlinux/rh61/rpms/orig`

Verzeichnis für die Originalen RPM-Pakete. Könnte z.B. ein Link auf `/mnt/cdrom/RedHat/RPMS/` sein.

`/rzlinux/rh61/rpms/update`

Update-Pakete, die man sich z.B. per FTP besorgt hat.

`/rzlinux/rh61/rpms/myrpms`

Selbst erstellte RPM-Pakete.

Zunächst ist die Verzeichnisstruktur anzulegen, z.B.:

```
mkdir -p /rzlinux/rh61/groups
mkdir -p /rzlinux/rh61/lists
mkdir -p /rzlinux/rh61/rpms/orig
mkdir -p /rzlinux/rh61/rpms/update
mkdir -p /rzlinux/rh61/rpms/myrpms
```

Jetzt wird die globale Konfiguration angepasst in `/rzlinux/rh61/rc.global` (als Vorlage kann `/usr/lib/rpmsv/rc.global.example` genommen werden. Für obiges Beispiel:

```
@RPMSVRPMS=(
    "$RPMSVBASE/rpms/orig",
    "$RPMSVBASE/rpms/update",
    "$RPMSVBASE/rpms/myrpms",
);

1;
```

Die Datei hat Perl-Syntax, sie sollte also immer syntaktisch korrekt sein und mit dem Kommando `1;` enden. Die Variable `RPMSVBASE` wird dabei von den Clients in `/etc/rpmsv/rc.local` gesetzt.

Mindestens eines der Verzeichnisse unter `/rzlinux/rh61/rpms` füllt man mit Paketen, z.B. mit

```
cp -av /mnt/cdrom/RedHat/RPMS/*rpm /rzlinux/rh61/rpms/orig
```

Nach jeder Änderung an den Paketen, müssen die Index-Files unter `/rzlinux/rh61/lists` angepasst werden:

```
rpmsv --lists
```

Als letztes müssen die Gruppenbeschreibungen gepflegt werden. Unter `/usr/doc/rpmsv-1.6/example_groups` sind (mehr oder weniger gut gepflegte) Beispiele, die nach `/rzlinux/rh61/lists` kopiert werden können. Siehe aber auch 4.2 (Pflege von Gruppen).

Nicht vergessen, das Directory `/rzlinux` für alle potentiellen Clients zu exportieren - ein Read-Only-Zugriff reicht aus.

2.6 Client-Konfiguration

Anpassen der `/etc/rpmsv/rc.local`: hier wird festgelegt, über welchen Pfad der rpmsv-Server anzusprechen ist, wo die globale Konfiguration ist und ggf. welche Dinge lokal überschrieben werden sollen.

Typischerweise wird nur angepasst, welcher welcher Pfad (für die ev. distributionsabhängigen Pakete) benutzt werden soll in der Variablen `RPMSVBASE`.

Der Rest der Konfiguration sollte auf den Server verlagert werden, daher sollte dort eine `rc.global` gepflegt werde, die durch die Funktion `readConfig` eingelesen werden kann.

Eine einfache Konfiguration sieht folgendermaßen aus:

```
$RPMSVBASE='/rzlinux/rh61';
readConfig("$RPMSVBASE/rc.global");
1;
```

Auch diese Datei hat wie die `rc.global` Perl-Syntax. Es können ansonsten alle Werte, die sonst in der `rc.global` stehen gesetzt, überschrieben oder erweitert werden - warum auch immer.

Der rpmsv-Pfad muß gemountet sein und als nächstes werden die Gruppen abonniert, siehe unten.

3 Using RPMSV

3.1 Auswahl der Gruppen

Auf dem Client wird bestimmt zu welchen Gruppen er gehört (welche Gruppen *abonniert* sind) mit dem Schalter `-ga` bzw `-groupadd`. Um z.B. die Gruppen `base` und `workstation` zu abonnieren:

```
rpmsv -ga base -ga workstation
```

Die verfügbaren und abonnierten Gruppen kann man sich ermitteln mit: `rpmsv -gl` bzw `rpmsv -grouplist`.

3.1.1 Alternative

Die Zugehörigkeit zu einer bestimmten Gruppe wird (momentan) lediglich durch eine (leere) Dateien mit dem Namen der Gruppe unter `/etc/rpmsv/groups` signalisiert, obiges Bsp. kann also auch folgendermassen erreicht werden:

```
touch /etc/rpmsv/groups/base
touch /etc/rpmsv/groups/workstation
```

3.2 Aufruf von `rpmsv`

Mit `rpmsv -help` wird die vollständige Sysntax und die Versionsnummer angezeigt.

Beim (normalen) Aufruf muß man sich über zwei Dinge Gedanken machen:

1. **Welche** Pakete sollen bearbeitet werden? (**Auswahl**)
2. **Was** soll damit geschehen? (**Aktion**)

Bei der **Auswahl** kann man wählen (auch mehrfach) zwischen:

- `-i` **install**: alle Pakete, die in einer der abonnierten Gruppen aufgeführt sind, aber nicht installiert.
- `-g` **remove**: alle Pakete, die installiert sind, aber in keiner der abonnierten Gruppen aufgeführt sind.
- `-u` **update**: Paket soll upgedatet werden (siehe auch 1.2 (Paket-Update))
- `-n` **newest**: Paket soll auf jeden Fall upgedatet werden

Bei der **Aktion** kann man wählen zwischen:

- `-s` Short: kurze Anzeige der Pakete
- `-l` Long: lange Anzeige der Pakete mit Versionnummer etc.
- `-script` Ausführen: `rpm` wird entsprechend aufgerufen
- `-script -t` Test: `rpm`-Script wird erstellt, aber nur angezeigt, nicht ausgeführt
- *ohne Angabe*: Liste der Pakete, ein Paket pro Zeile

3.3 Installation von Paketen

Typischer Ablauf: abonnieren einer einen Gruppe mit anschliessendem Prüfen was zu installieren ist und die Ausführung:

```
rpmsv --groupadd workstation -i -l
rpmsv -i --script
```

3.4 Update von Paketen

Wie im Kapitel 1.2 (Paket-Update) bechrieben hat man die Wahl zwischen zwei Modi. Auf produktiven Systemen sollte die normale Variante ausgeführt werden, hier z.B. mit vorheriger Anzeige was passieren wird:

```
rpmsv -u -l
rpmsv -u --script
```

Auf einem Testrechner, der immer uptodate sein soll, führt man aus:

```
rpmsv -n --script
```

3.5 Löschen von Paketen

Alle Pakete, die in keiner der abonnierten Gruppen aufgelistet sind, ermittelt man mit:

```
rpmsv -g -l
```

Um diese auch tatsächlich zu löschen, führt man aus:

```
rpmsv -g --script
```

3.6 Automatisierter Abgleich

Folgende Kommandozeile könnte man z.B. als täglichen Cronjob und/oder durch ein Initscript beim Booten ablaufen lassen, um alles auf einmal abgleichen zu lassen:

```
rpmsv -i -u -g --script
```

4 Tipps und Tricks

4.1 Brauch ich mindestens zwei Rechner?

Nein, natürlich nicht. RPMSV-Client und -Server können identisch sein. Ein Server kann sich mit den Paketen und der Konfiguration, die er selbst vorhält, abgleichen. Es ist nichts besonderes dabei zu beachten, nur daß (wie sonst) auch, die Directory-Struktur passen muß.

4.2 Pflege von Gruppen

Das Erstellen der Gruppendefinitionen ist eine mühsame Aufgabe, RPMSV wurde aber deshalb entwickelt, damit man dies für die Systemlandschaft nur einmalig tun muß. RPMSV kan den Administrator dabei helfen, die jeweiligen Dateien zu pflegen, in dem es die Differenz zwischen den ermittelten Pakete mit denen der abonnierten Gruppen ermitteln kann.

Beispiel: Gegeben sind die drei schon oben genannten Gruppen `base`, `database` und `workstation`. Als nächstes soll es eine Gruppe `samba` geben, die einen Samba-Server beschreibt.

Einen Rechner, der z.B. den Gruppen `base` und `workstation` angehört (also die Arbeitstation des Administrators ;-), wird so hergerichtet, daß alle notwendigen Pakete für einen Samba-Server vorhanden sind. Es werden also die Pakete `samba-common`, `samba` und `samba-client` nachinstalliert. (Das Paket `samba-client` wird für einen Samba-Server eigentlich nicht benötigt, sondern sollte auf jedem System oder auf jeder Workstation installiert werden - es ist die Entscheidung des Administrators dieses Paket in einer entsprechenden Gruppe einzutragen.)

Ruft man nur `rpmsv -g` auf, werden alle Pakete angezeigt, die in keiner der abonnierten Gruppen enthalten sind. Dieses Ergebnis kann man direkt für die neue Gruppendefinition benutzen, z.B. durch:

```
rpmsv -g > /rzlinux/rh6.1/groups/samba
```

Eventuell ist eine Gruppe nur dann installierbar, wenn die Pakete einer anderen Gruppe auch installiert sind/werden (RPM-Paketabhängigkeiten). Dazu gibt es in der Gruppenbeschreibung die Möglichkeit andere Gruppen einzubinden, sozusagen ein `#include`. Soll z.B. eine Gruppe `second` alle Pakete der Gruppe `first` ebenfalls mit auswählen, so kann man in der Datei `/rzlinux/rh61/groups/second` die folgende Zeile aufnehmen:

```
{first}
```

4.3 Pflege von lokalen Gruppen

Eventuell sollen auf einem System Pakete installiert sein, die nicht auf dem Server in einer Gruppe definiert sind, ist es sinnvoll diese Pakete explizit zu definieren - insbesondere um zu verhindern, daß diese Pakete durch `rpmsv -g` ungewollt gelöscht werden.

Dazu können auch Gruppendefinitionen lokal auf einem System definiert werden, indem sie unter `/etc/rpmsv/groups/local` in einer Datei aufgelistet werden.

Siehe auch 4.2 (Pflege von Gruppen).

4.4 Arbeiten mit verschiedenen Distributionen auf einem Server

Der Server kann mehrere Distributionen vorhalten, z.B. in den Directories `/rzlinux/rh61` und `/rzlinux/rh62`. Der Client steuert in seiner Konfigurationsdatei `/etc/rpmsv/rc.local` über die Variable `$RPMSVBASE` (bzw. welche Konfiguration vom Server über die Funktion `readConfig` eingelesen wird), welche Distribution er benutzt.

Hierbei kann es sinnvoll sein, explizit die lokale Konfigurationsdatei anzugeben: `-config` *`file`*. Diese Option ist insbesondere auf dem Server wichtig, um die entsprechenden Listen zu erstellen (`-list`).

4.5 Chroot-Installationen

Für verschiedene Zwecke kann es sinnvoll sein, RPM-Pakete in einer chroot-Umgebung zu installieren, z.B. für:

- Test: Zum Test der Paketdefinitionen oder zum Testen einzelner RPM-Pakete
- RPM-Pakete erstellen: Wenn man selbst RPM-Pakete erstellen will, so stellt man fest, daß das Compilieren, Installieren und anschliessende Einpacken der Pakete das laufende System verändert (in dem Dateien installiert werden, die nicht in der RPM-Datenbank aufgeführt sind) - zumindest wenn nicht die `inibiuld`-Option von RPM benutzt wird bzw. werden kann.

  Solange man kein dedizierten Entwicklungsystem hat, kann es sinnvoll sein, sich eine chroot-Umgebung zu installieren, in der die RPM-Pakete gebaut werden. Dadurch wird das laufende (Produktiv-) System nicht verunstaltet und somit für jede Administration unbrauchbar.

Es gibt den Schalter `-chroot`. Damit wird bei jedem Aufruf von `rpm` der Schalter `-root` vorangestellt, das gilt sowohl beim Abfragen der installierten Pakete (`rpm -qa`) als auch beim Installieren, Updaten oder Löschen von Paketen. Aus Sicht von RPMSV muß diese Umgebung initialisiert werden mit dem Schalter `initchroot` (genaugenommen wird dabei eine minimale Directory-Struktur) eingerichtet)

Ein Beispiel dafür:

```
rpmsv --chroot /usr/local/chroot --initchroot
rpmsv --chroot /usr/local/chroot -ga base -ga devel -i --script
chroot /usr/local/chroot bash
```

4.6 Remote-Installationen

Motivation: Der zu verwaltene Client steht hinter einer Firewall und soll aus sicherheitsgründen kein NFS-Zugang zum Fileserver mit den aktuellen Paketen bekommen. Es wäre also bisher nicht möglich auch diesen Rechner abzugleichen.

Voraussetzung: Von einem anderen Rechner (mit Zugang zu den aktuellen Paketen) besteht eine SSH-Verbindung (`ssh` und `scp`) zum Zielrechner für die Kennung `root` ohne Passwortabfrage.

Lösung: Auf dem *inneren* Rechner wird `rpmsv` mit dem Schalter `-remote` ***`host`*** gestartet. Hierbei wird rpmsv selbst auf dem Server gestartet, alle RPM-Aufrufe werden aber mit `ssh` auf den Remote-Rechner durchgeführt, ebenso werden alle benötigten Pakete mit `scp` zunächst übertragen.

Beispiel: Rechner `web` soll upgedated werden, hat aber kein Zugang zum RPMSV-Server `server`, es besteht aber ein SSH-Zugang von `server` zu `web` ohne Passwortabfrage.

Anstatt auf Rechner `web` zu starten:

```
rpmsv -i -u --script
```

wird auf Rechner `server` gestartet:

```
rpmsv --remote web -i -u --script
```

4.7 Wo ist der Plattenplatz hin ? (`-listrpmsizes`, `-listfiles`, `-listfilesnorpm`)

Motivation: manchmal fragt man sich, warum die Platte so voll ist. Folgende Kommandos stehen zur Verfügung, um besseren Überblick zu behalten:

`-listrpmsizes`

Listet die Summe aller Dateien für jedes installierte Paket aufsteigend sortiert (Größte Paket zuletzt) und die Summe aller installierten Pakete (Paketname `All packages`). Es wird sich darauf verlassen, daß die RPM-Datenbank richtige Werte liefert.

Beispiel:

```
$ rpmsv --unit MB --listrpmsizes
...
      18 MB  netscape-common-4.77-0.6.2
      20 MB  kdelibs-devel-2.1.1-0.6x.5
      20 MB  kde-i18n-German-2.1.1-0.6x.2
      22 MB  qt-devel-2.3.0-0.6x.3
      28 MB  tetex-1.0.6-7
      28 MB  kdebase-2.1.1-0.6x.8
      31 MB  glibc-profile-2.1.3-22
      31 MB  glibc-2.1.3-22
      33 MB  glibc-devel-2.1.3-22
     642 MB  All packages
```

`-listfiles`

Listet den belegten Platz für jede Dateien unterhalb der gegebenen Directories und - durch eine leere Zeile getrennt - die Summe für alle Directories. Es wird aufsteigend sortiert (die größten Dateien bzw. Directories zuletzt).

Es werden die Dateigrößen im Filesystem ermittelt. Symbolische Links werden nicht aufgelöst (es wird also die Größe des symbolischen Links, nicht die Größe auf die der Link zeigt berechnet). Das Filesystem wird nicht verlassen (`find -xdev`), jeder Mountpunkt muss also einzeln angegeben werden!

Beispiel:

```
$ rpmsv --unit KB --listfiles /etc/skel/Desktop,/var/tmp/test
       0 KB  /var/tmp/test/A
       0 KB  /etc/skel/Desktop/Linux Documentation
       0 KB  /etc/skel/Desktop/www.redhat.com
       0 KB  /etc/skel/Desktop/kontrol-panel
       0 KB  /etc/skel/Desktop/Printer
       0 KB  /etc/skel/Desktop/.directory
      27 KB  /var/tmp/test/rpmsv

       1 KB  /etc/skel/Desktop/
       1 KB  /etc/skel/
       1 KB  /etc/
      27 KB  /var/tmp/test/
      27 KB  /var/tmp/
      27 KB  /var/
      28 KB  /
```

`-listfilesnorpm`

Hilft zu ermitteln, wo Dateien sind, die nicht zu einem RPM-Paket gehören. Dieser Schalter arbeitet fast genauso wie `-listfiles`, nur wird zusätzlich geprüft, ob jede Datei zu einem Paket gehört, was natürtlich etwas änger dauert.

Beispiel:

```
$ rpmsv --unit B --listfilesnorpm /etc/skel
      97 B  /etc/skel/.kde/share/config/kcmdisplayrc.rpmsave
     217 B  /etc/skel/.kde/share/config/desktop0rc.rpmsave

     314 B  /etc/skel/.kde/share/config/
     314 B  /etc/skel/.kde/share/
     314 B  /etc/skel/.kde/
     314 B  /etc/skel/
     314 B  /etc/
     314 B  /
```

Auch hierbei wird das Filesystem nicht verlassen, wer alle nicht-RPM-Dateien finden will und mehrere Dateisysteme hat (z.B. `/`, `/usr` und `/var`, muß angeben: `rpmsv -listfilesnorpm /,/usr,/var`

`-unit`

Dient für obige Funktionen zur Angabe, ob Dateigrößen in Bytes, Kilobytes (Bytes / 1024) oder Megabytes (Kilobytes / 1024) erfolgen soll.

Standard ist Bytes, Parameter sind `B`, `KB` bzw. `MB`.

Achtung: der Schalter `-unit` muß **vor** den anderen Schaltern angegeben werden!