Traduit le 3 juillet 1999

RPM HOWTO (RPM at Idle)

Donnie Barnes, *djb@redhat.com* mailto:djb@redhat.com v2.0, 8 Avril 1997

Contents

1 Introduction

RPM est le gestionnaire de paquetages Redhat (RedHat package manager). Malgré le fait qu'il contienne RedHat dans le nom, il se veut totalement un système de paquetages ouvert disponible pour tous. Il permet aux utilisateurs de prendre le code source pour des nouveaux logiciels et de l'"empaqueter" sous forme de source ou de binaire pour que les binaires puissent être simplement installés et suivis et les sources recompilées simplement. Il maintient aussi une base de donées de tous les paquetages et de leurs fichiers qui peut être utilisée pour vérifer les paquetages et chercher des informations a propos des fichiers et/ou des paquetages.

RedHat Software encourage les autres vendeurs de distributions à prendre le temps de s'intéresser à RPM et à l'utiliser pour leur propre distribution. RPM est complètement flexible et simple d'utilisation, bien qu'il fournisse la base d'un système très puissant. Il est aussi complètement ouvert et disponible, bien que nous appréciions les rapports de bugs et les correctifs. La permission d'utiliser et distribuer RPM gratuitement est admise conformément à la GPL.

Une documentation plus complète est disponible sur RPM dans le livre d'Ed Bailey, Maximum RPM. Ce livre est disponible pour le téléchargement ou l'achat sur www.redhat.com *http://www.redhat.com/* `http://www.redhat.com/` .

2 Overview

Premièrement, laissez-moi décrire la philosophie de RPM. Un but de l'étude était de permettre l'utilisation des sources "de base". Avec RPP (notre ancien système de paquetages duquel rien de RPM n'est dérivé), nos paquetages sources étaient des sources "bidouillées" à partir desquelles nous compilions. Théoriquement, quelqu'un peut installer un RPP source puis le compiler sans prblèmes. Mais les sources n'étaient pas les originales, et il n'y avait pas de référence comme quels changements avsions nous fait pour que les sources compilent. Il devait télécharger les sources de base séparément. Avec RPM, vous avez les sources de base ainsi qu'un patch que nous avons utilisé pour compiler. Nous y voyons un grand avantage. Pourquoi ? Il y a plusieurs raisons. Tout d'abord, si une nouvelle version d'un programme sort, vous ne devez pas nécessairement repartir de rien pour obtenir la compilation par les RedHat Labs. Vous pouvez regarder le patch pour voir ce que vous avez besoin de faire. Toutes les valeurs par défaut de compilation sont facilement visibles par ce moyen.

RPM est aussi conçu pour avoir de puissantes options de requête. Vous pouvez chercher à travers la base de données entière des paquetages ou seulement certains fichiers. Vous pouvez aussi simplement trouver à quel paquetage un fichier appartient, et d'où il vient. Les fichiers RPM eux-mêmes sont des archives compressées, mais vous pouvez interroger des paquetages individuels simplement et rapidement grâce à un en-tête binaire spécial ajouté au paquetage avec tout ce dont vous pouvez avoir besoin de savoir sur le contenu sous forme non-compressée. Cela permet des requêtes plus rapides.

Une autre fonctionnalité puissante est la capacité de vérifier des paquetages. Si vous avez peur d'avoir effacé un fichier important pour un paquetage, vérifiez-le simplement. Vous serez avertis des anomalies. A ce stade, vous pouvez réinstaller le paquetage so nécessaire. Les fichiers de configuration que vous aviez sont bien sûr préservés.

Nous aimerions remercier les gens de la distribution BOGUS pour beaucoup de leurs idées et concepts qui sont inclus dans RPM. Quoique RPM ait été complètement écrit par RedHat Software, ses fonctions sont basées sur le code écrit par BOGUS (PM et PMS).

3 Information générale

3.1 Se procurer RPM

Le meilleur moyen de se procurer RPM est d'installer RedHat Linux. Si vous ne le voulez pas, vous pouvez tout de même obtenir et utiliser RPM. Vous pouvez vous le procurer sur ftp.redhat.com *ftp://ftp.redhat.com//pub/redhat/code/rpm* `ftp://ftp.redhat.com//pub/redhat/code/rpm` .

3.2 Ce que RPM requiert

Ce qui est principalement requis pour faire tourner RPM est cpio 2.4.2 ou supérieur. Quoique ce système soit conçu pour être utilisé avec Linux, il peut très bien être porté sur d'autres systèmes Unix. Il a, en fait, été compilé sur SunOS, Solaris, AIX, Irix, AmigaOS, et d'autres. Faites attention, les paquetages binaires générés sur un système Unix de type différent ne seront pas compatibles.

Ce sont les exigences minimales pour installer des RPMs. Pour construire des RPMs à partir de sources, vous avez aussi besoin de ce qui est normalement requis pour compiler un paquetage, comme gcc, make, etc.

4 Utiliser RPM

Dans sa forme la plus simple, RPM peut être utilisé pour installer des paquetages:

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

La commande suivant la plus simple est la désinstallation d'un paquetage:

```
rpm -e foobar
```

Une des plus complexes mais très utile des commandes vous permet d'installer des paquetages via FTP. Si vous êtes connectés à internet et voulez installer un nouveau paquetage, tout ce que vous avez besoin de faire est de spécifier le fichier avec une URL valide, comme dans:

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

Notez que RPM va maintenant interroger et/ou installer via FTP.

Bien que ce soient des commandes simples, RPM peut être utilisé d'une multitude de façons comme le montre le message Usage:

```
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}
```

Vous pouvez trouver plus de détails sur ce que font ces options dans la page de man de RPM.

5 Que puis-je vraiment faire avec RPM ?

Rpm est un utilitaire très utile (!), comme vous pouvez le voir, avec de nombreuses options. Le meilleur moyen de de leur donner un sens est de regarder des exemples. J'ai abordé la simple installation/désinstallation plus haut, alors voici plus d'exemples :

- Imaginez que vous ayez effacé des fichiers par accident, mais que vous ne soyez pas sûr que vous les avez effacé. Si vous ne voulez pas vérifier votre système complet et voir ce qui manque, vous ferez :

  ```
  rpm -Va
  ```

- Imaginez que parcouriez un fichier que vous ne reconnaissez pas. Pour trouvez à quel paquetage il appartient, vous ferez :

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

  La sortie sera :

  ```
  xjewel-1.6-1
  ```

- Vous avez trouvé un nouveau RPM de koules, mais vous ne savez pas ce que c'est. Pour avoir des informations à son propos, faites :

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

  La sortie sera :

```
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: (none)                      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.
```

- Maintenant vous voulez voir quels fichers le RPM de koules va installer. Vous ferez :

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

  La sortie est :

```
/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
```

Ce sont juste quelques exemples. De plus créatifs peuvent être proches de ce que vous pouvez vraiment faire en étant familier de RPM.

6 Compiler des RPMs

Compiler ses RPMs est très simple, spécialement si vous pouvez obtenir du logiciel que vous essayez qu'il se compile tout seul.

La procédure de base pour compiler un RPM est la suivante :

- Assurez-vous que votre fichier /etc/rpmrc est paramétré pour votre système.
- Récupérez les sources donc vous compilez le RPM pour la compilation sur votre système.
- Faites un patch des changements que vous devez faire aux sources pour qu'elles compilent correctement.
- Faites un fichier spec pour le paquetage.
- Assurez-vous que chaque chose est à sa place.

En utilisation normale, RPM construit aussi bien des paquetages sources que des binaires.

6.1 Le fichier rpmrc

Maintenant, la seule configuration de RPM is disponible via le fichier /etc/rpmrc. Un exemple de celui-ci ressemble à :

```
require_vendor: 1
distribution: I roll my own!
require_distribution: 1
topdir: /usr/src/me
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
```

La ligne require_vendor fait que RPM trouve une ligne vendor. Elle peut provenir du fichier /etc/rpmrc ou de l'en-tête du fichier spec lui-même. Pour désactiver ceci, mettez le nombre à 0. Cela reste vrai pour les lignes require_distribution et require_group.

La ligne suivante est la ligne distribution. Pour pouvez définir cela ici ou plus tard, dans l'en-tête du fichier spec. Quand vous compilez pour une distribution particulière, il est conseillé de s'assurer que cette ligne est correcte, bien que ça ne soit pas requis. La ligne vendor fonctionne selon le même principe, mais peut être n'importe quoi (ex: Joe's Software and Rock Music Emporium).

RPM supporte aussi maintenant la création de paquetages sur des architectures multiples. Le fichier rpmrc peut conserver une variable "optflags" pour compiler ce qui requiert des options spécifiques à l'architecture durant la compilation. Voir plus loin les paragraphes concernant l'utilisation de cette option.

En supplément des macros ci-dessus, il y en a beaucoup plus. Vous pouvez utiliser :

```
rpm --showrc
```

pour savoir comment vos options sont définies et quels sont les options disponibles.

6.2 Le fichier Spec

Nous avons commencé à parler du fichier spec. Les fichiers spec sont requis pour construire un paquetage. Le fichier spec est une description du logiciel accompagnée des instructions concernant sa compilation, ainsi qu'une liste des fichiers pour tous les binaires qui seront installés.

Il est recommandé nommer votre fichier spec conformément à une convention standard, c'est à dire nom_du_paquetage-numéro_de_version-numéro de release.spec.

Voici un petit fichier spec (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
```

6.3 L'en-tête

L'en-tête comporte des champs standard que vous devez remplir. Il y a quelques restrictions bien sûr. Les champs doivent être remplis comme suit :

- Summary: C'est la description du paquetage en une ligne.
- Name: Cela doit être la partie "nom" du fichier rpm que vous projetez d'utiliser.
- Version: Cela doit être la partie "version" du fichier rpm que vous projetez d'utiliser.
- Release: C'est le numéro de release pour un paquetage d'une même version (par exemple si vous construisez un paquetage et que vous le trouvez qu'il est légèrement râté et que vous souhaitez le reconstruire, le paquetage suivant aura le numéro de release 2).
- Icon: c'est le nom du fichier icône pour utilisation par un autre utilitaire d'installation de "haut niveau" (comme glint ou gnorpm). Ce doit être un gif et doit être placé dans le répertoire SOURCES.
- Source: Cette ligne pointe sur l'emplacement d'origine des sources de base. Il est utilisé si vous voulez réobtenir les sources ou regarder si il existe une version plus récente. Restriction: le nom du fichier dans cette ligne doit concorder avec le nom du fichier que vous avez sur votre propre système (c'est à dire ne pas télécharger les sources et changer le nom du fichier). Vous pouvez aussi spécifier plus d'un fichier source en utilisation des lignes comme :

```
Source0: blah-0.tar.gz
Source1: blah-1.tar.gz
Source2: fooblah.tar.gz
```

  Ces fichiers iront dans le répertoire SOURCES (la structure des répertoires est abordée dans un autre paragraphe, "L'arborescence du répertoire des sources".)

- Patch: C'est l'emplacement où vous pouvez trouvez le patch si vous voulez le retélécharger. Restriction: le nom du fichier dpot concorder avec celui que vous utilisez qaudn vous faites VOTRE patch. Notez que vous pouvez avoir plusieurs fichiers patch de la même façon que vous pouvez avoir plusieurs sources. Vous auriez ainsi quelque chose comme :

```
Patch0: blah-0.patch
Patch1: blah-1.patch
Patch2: fooblah.patch
```

Ces fichiers vont dans le répertoire SOURCES.

- Copyright: Cette ligne indique la façon dont le package est protégé légalement. Vous pouvez utiliser quelque chose comme GPL, BSD, MIT, public domain, Distributable, ou commercial.
- Buildroot: Cette ligne vous permet de spécifier un répertoire comme "root" pour la compilation et l'installation du nouveau paquetage. Vous pouvez l'utiliser pour faciliter les tests de votre paquetage avant de l'installer sur votre machine.
- Group: Cette ligne est utilisée pour donner aux programmes d'installation de haut niveau l'emplacement de ce paquetage dans leur structure hiérarchique. La arborescence des groupes ressemble actuellement à :

```
Applications
        Communications
        Editeurs
                Emacs
        Ingéniérie
        Tableurs
        Bases de données
        Graphiques
        Réseau
        Mail
        News
        Publication
                TeX
Base
        Noyau
Utilitaires
        Archive
        Console
        Fichiers
        Système
        Terminal
        Texte
Démons
Documentation
X11
        XFree86
                Serveurs
        Applications
                Graphiques
                Réseau
        Jeux
                Stratégie
                Vidéo
        Amusements
        Utilitaires
        Librairies
        Gestionnaires de fenêtres
Librairies
```

```
Réseaux
        Admin
        Démons
        News
        Utilitaires
Développement
        Débuggeurs
        Librairies
                Libc
        Langages
                Fortran
                Tcl
        Construction
        Contrôle de version
        Utilitaires
Shells
Jeux
```

- %description Ce n'est pas pas vraiment un champ de l'en-tête, mais doit être décrit avec le reste de celui-ci. Vous avez besoin d'un champ description par paquatage/sous-paquetage. C'est un champ multilgine qui est utilisé pour donner une description claire du paquetage.

6.4 Prep

C'est la seconde section du fichier spec. Il est utilisé pour préparer les sources à la compilation. Vous mettez ce que vous avez besoin de faire pour patcher les sources et paramétrer, comme ce que vous mettriez pour compiler les sources.

Une chose importante: chacune de ces sections est simplement un emplacement pour exécuter des scripts shell. Vous pourriez simplement faire un script sh et le mettre après le tag %prep pour décompresser et patcher vos sources. Nous avons conçu des macros pour aider à cela, toutefois.

La première de ces macros est %setup. Dans sa forme la plus simple (pas d'options de ligne de commande), elle décompresse simplement les sources et se rend dans le répertoire des sources. Elle prend aussi les options suivantes :

- -n nom va définir le nom du répertoire de compilation. La valeur par défaut est $NAME-$VERSION. D'autres possibilités, parmi lesquelles $NAME, ${NAME}${VERSION}, ou n'importe quoi utilisé par le fichier tar principal. (Notez que ces variables "$" ne sont pas des variables réelles disponibles à l'intérieur du fichier spec . En réalité, elles sont juste utilisées ici à la place du nom de l'exemple. Il est nécessaire d'utiliser les vrais noms et versions dans votre paquetage, et non une variable.)
- -c va créer et se rendre dans le répertoire donné avant de détarrer.
- -b # va détarrer Source# avant de se rendre dans le répertoire (et cela n'a aucun sens avec -c donc ne les associez pas). C'est utile seulement avec de multiples fichiers source.
- -a # va détarrer Source# après s'être rendu dans le répertoire.
- -T Cette option supplante l'action par défaut de détarrer le Source et requiert un -b 0 ou -a 0 pour détarrer le fichier source principal. Vous en aurez besoin quand il y a des sources secondaires.
- -D N'efface pas le répertoire avant la décompression. C'est seulement utilse où vous avez plus d'un macro setup Cela doit être utilisé uniquement dans les macros setup après la première (mais jamais dans celle-ci).

La macro suivante est la macro %patch. Cette macro aide à automatiser le processus d'application des patches aux sources. Il comporte plusieurs options, listées ici :

- # va appliquer Patch# comme fichier patch
- -p # spécifie le nombre de répertoires à éliminer pour la commande patch(1) (NdT: option -p).
- -P L'action par défaut est d'appliquer Patch (ou Patch0). Ce paramètre inhibe ce comportement par défaut et requierrera un 0 pour détarrer le fichier. Cette option est très utile en seconde (ou plus) macro %patch qui requiert un numéro différent de la première macro.
- Vous pouvez aussi faire %patch# au lieu de faire la commande réelle: %patch # -P

Ce sont toutes les macros dont vous avez besoin. Après que vous les ayez faites, vous pouvez également faire un autre réglage dont vous avez besoin via un script sh. Tout ce que vous incluez jusqu'à de la macro %build (évoquée dans la section suivante) est exécuté par sh. Regardez l'exemple plus haut afin de vous donner une idée du genre de choses que vous pouvez faire ici.

6.5 Compiler

Ils n'y a pas de vraies macros pour cette section. Vous devez juste mettre ici les commandes dont vous avez besoin pour compiler le logiciel lorsque vous avez détarré les sources, patchées celles-ci, et vous être rendu dans le répertoire. C'est juste un autre ensemble de commandes passées à sh, donc n'importe quelle commande acceptée par sh peut être placée ici (y compris des commentaires). Votre répertoire de travail courant est rétabli dans ces sections au répertoire racine des cources, gardez cela en mémoire. Vous devez changer de répertoire pour atteindre les sous-répertoires si nécessaire.

6.6 Installation

De même, il n'y a pas ici non plus de vraies macros. Vous mettrez ici simplement les commandes donc vous avez besoin pour installer. Si vous avez un "make install" disponible dans le paquetage que vous compilez, mettez-le ici. Sinon, vous pouvez patcher le Makefile pour un "make install" et faire juste un make install ici, ou l'installer à la main ici avec des commandes sh. Considérez que votre répertoire courant est le répertoire racine de vos sources.

6.7 Scripts d'installation/désinstallation optionnels

Vous pouvez mettre des scripts qui seront exécutés avant et après l'installation et la désinstallation de paquetages binaires. Une des principales raison pour ça est la nécessité de lancer /sbin/ldconfig après l'installation ou la suppression de paquetages contenant des librairies partagées. Les macros pour chacun de ces scripts sont les suivantes:

- %pre est la macro qui fait les scripts de pré-installation.
- %post est la macro qui fait les scripts de post-installation.
- %preun est la macro qui fait les scripts de pré-désinstallation.
- %postrun est la macro qui fait les scripts de post-désinstallation.

Le contenu de ces sections doit ressembler à un script sh, sauf que vous n'avez pas besoin de #!/bin/sh.

6.8 Fichiers

C'est la section où vous devez lister les fichiers pour le paquetage binaire. RPM n'a aucun moyen de connaitre quels fichiers sont installés par le "make install". Il n'y a PAS de moyen de le savoir. Certains ont suggéré de faire un "find" avant et après l'installation. Avec un système multiutilisateur, c'est inacceptable car d'autres fichiers qui n'ont rien à voir peuvent être crées pendant le processus d'installation.

Il y a plusieurs macros disponibles pour faire des choses spéciales. Elles sont listées et décrites ici:

- %doc est utiliser pour marquer la documentation dans le paquetage source que vous voulez installer dans un paquetage binaire. Les documents seront installés dans /usr/doc/$NAME-$VERSION-$RELEASE. Vous pouvez lister plusieurs documents sur la ligne de commande avec cette macro, ou les lister séparément en utilisant une macro pour chaque.
- %config est utilisé pour marquer les fichiers de configuration dans un paquetage. Cela inclut des fichiers comme sendmail.cf, passwd, etc. Si vous déinstallez par la suite un paquetage contenant des fichiers de configuration, les fichiers non modifiés seront supprimés et les fichiers modifiés seront conservés avec l'extension .rpmsave. Vous pouvez bien sûr mettre plusieurs fichiers avec cette macro.
- %files -f `nomfichier` va vous permettre de lister vos fichiers dans un fichier arbitraire à l'intérieur du répertoire de compilation des sources. C'est pratique dans le cas où vous avez un paquetage qui ne peut pas construire sa propre liste de fichiers. Vous incluez alors simplement cette liste de fichiers ici, et vous ne devez pas lister les fichiers spécifiquement.

Le plus gros inconvénient dans le liste de fichier est la liste des répertoire. Si vous listez /usr/bin par accident, votre paquetage va contenir tous les fichiers de /usr/bin sur votre système.

6.9 Le compiler

6.9.1 L'arborescence du répertoire des sources

La première chose dont vous avez besoin est une arborescence de compilation bien configurée. C'est configurable dans /etc/rpmrc. La plupart des gens utiliseront simplement /usr/src.

Vous aurez probablement besoin de créer les répertoires suivants pour construire l'arborescence de compilation:

- BUILD est le répertoire où toutes les compilations par RPM ont lieu. Vous ne devez pas faire vos tests de compilation quelquepart en particulier, mais c'est là où RPM va faire sa compilation.
- SOURCES est le répertoire où vous mettrez le fichier source tarré original et vos patches. C'est là que RPM regarde par défaut.
- SPECS est le répertoire où tous les fichiers spec vont.
- RPMS is le répertoire où RPM va mettre tous ses binaires RPMs compilés.

6.9.2 Test de la compilation

La premireè chose que vous voudrez probablement faire est de compiler proprement la source sans utiliser RPM. Pour faire cela, décompressez les sources, et changez le nom du répertoire en $NAME.orig. Ensuite re-décompressez les sources. Utilisez ces sources pour compiler. Allez à l'intérieur de celui-ci et suivez les instructions de compilation pour le compiler. Si vous devez éditer des choses, vous aurez besoin d'un patch.

Dès que vous réuississez à le compiler, nettoyez le répertoire des sources. Effacez les fichiers qui ont été obtenus par le script configure. Ensuite, remontez au répertoire parent. Vous ferez ensuite quelque chose comme :

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

Cela créera un patch pour vous que vous pourrez utilisez dans votre fichier spec. Notez que le "linux" que vous voyez dans le nom du patch est juste un identificateur. Vous voudrez sûrement utiliser quelque chose de plus descriptif comme "config" ou "bugs" pour décrire pourquoi vous avez dû faire un patch. De plus il est recommandé de vérifier dans le fichier patch que vous avez créé que vous n'avez pas inclus de binaires par accident avant de l'utiliser.

6.9.3 Générer la liste des fichiers

Maintenant que vous avez des souces qui vont compiler et que vous savez comment le faire, compilez-les et installez-les. Regardez la sortie de la séquence d'installation et construisez la liste de fichiers que vous utiliserez dans le fichier spec à partir de celle-ci. On construit habituellement le fichier spec en parallèle avec toutes ces étapes. Vous pouvez créer celui de base et remplir ses parties les plus simples, et ensuite remplir les autre étapes au fur et à mesure.

6.9.4 Compiler le package avec RPM

Dès que vous avez un fichier spec, vous êtes prêt à essayer et à compiler votre paquetage. Le voie la plus utilisée pour faire cela est avec une commande qui ressemble à la suivante :

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

Il y a d'autres options utiles avec le paramètre -b :

- p signifie d'éxécuter simplement la section prep du fichier spec.
- l est un vérificateur de liste qui fait des contrôles sur %files.
- c fait le prep et compile. C'est utile quand vous n'êtes pas sûr du tout de la source que vous compilez. Cela semble peu utile parce que vous travaillerez avec les sources elles-mêmes jusqu'à ce qu'elles compilent et commencerez seulement à travailler avec RPM, mais dès que vous serez accoutumé à l'utilisation de RPM vous trouverez des cas où vous les utiliserez.
- i fait un prep, compile, et installe.
- a fait tout (paquetages source et binaire).

Il y a plusieurs modificateurs à l'option -b. Ce sont les suivants:

- –short-circuit va sauter à une étape (peut être utilisé uniquement avec c et i).
- –clean efface l'arborescence de compilation quand le travail est terminé.
- –keep-temps va conserver tous les fichiers temporaires et les scripts fais dans /tmp. Vous pouvez actuellement voir quels fichiers ont été crées dans /tmp en utilisant l'option -v.
- –test n'exécute aucune des étapes réelles, mais conserve les fichiers temporaires.

6.10 Le tester

Dès que vous avez des rpms source et binaire pour votre paquetage, vous devez le tester. La voie la plus simple et la meilleure est d'utiliser pour les tests une machine totalement différente de celle sur laquelle vous avez construit le paquetage. Après tout, vous avec juste fait un ensemble de "make install" sur votre propre machine, alors il pourrait aussi bien être installé.

Vous pouvez faire un rpm -u nom_du_paquetage sur le paquetage pour tester, mais vous pouvez être déçu parce durant la construction du paquetage, vous avez fait un make install. Si vous avez laissé quelque chose en dehors de votre liste des fichiers, il ne sera pas désinstallé. Vous réinstallerez alors le paquetage binaire et votre système sera de nouveau complet, mais votre rpm toujours pas. Gardez à l'esprit que vous faites un rpm -ba paquetage, la plupart des peersonnes installeront simplement celui-ci avec rpm -i paquetage. Assurez-vous de ne rien faire dans la section build ou install qui aura besoin d'être fait quand les binaires seront installés par eux-mêmes.

6.11 Que faire avec vos nouveaux RPMs

Dès que vous avez construit votre propre rpm de quelque chose (si il n'a pas déjà été "RPMisé"), vous pouvez faire profier les autres de votre travail (si votre rpm est un logiciel librement redistribuable). Pour cela, vous l'uploaderez sur *ftp://contrib.redhat.com/* `ftp://contrib.redhat.com/`

6.12 Que faire maintenant ?

Regardez les sections précédentes Tests et Que faire ... Nous voulons tous les RPMs que nous pouvons obtenir, et nous voulons que ce soient tous les bons RPMs. Prenez le temps de les tester correctement, et ensuite prenez le temps de les uploader afin que chacun en bénéficie. De même, assurez-vous que vous uploadez uniquement des logiciels librement redistribuables. Les logiciels commerciaux et les sharewares ne doivent pas être uploadés à moins que le copyright le permette explicitement. Cela inclut Netscape, ssh, pgp, etc.

7 Construire des RPM pour plusieurs architectures

RPM peut maintenant être utilisé pour construire des paquetages pour intel 386, le Digital Alpha faisant tourner linux, et le Sparc. Il a été signalé qu'il fonctionnait aussi bien sur des stations de travail SGI et HP. De nombreuses options permettent de construire des paquetages sur toutes les plateformes facilement. La première de celles-ci est la directive "optflags" dans /etc/rpmrc. Elle peut être utilisée pour positionner des options utilisés durant la compilation concernant des valeurs spécifiques à l'architecture. Elles peuvent être utilisées pour faire différentes choses qui dépend de l'architecture sur laquelle vous compilez. Une fonctionnalité est la directive "Exclude" dans le header.

7.1 Exemple de fichier spec

La partie qui suit est extraite du fichier spec pour le paquetage fileutils. Il est paramétré pour compiler aussi bien sur Alpha que sur Intel.

```
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*
```

7.2 Optflags

Dans cet exemple, vous pouvez voir comment la directive "optflags" est utilisée dans le /etc/rpmrc. Selon l'architecture sur laquelle vous compilez, la valeur est donnée à RPM_OPT_FLAGS. Vous devez patcher le Makefile pour votre paquetage pour utiliser cette variable à la place des directives normales que vous utilisez probablement (comme -m486 et -O2). Vous pouvez obtenir un meilleur aspect de ce dont vous avez à faire par l'installation du paquetage source, la décompression de celui-ci et l'examen du Makefile. Ensuite regardez au patch pour le Makefile et voyez les changements à faire.

7.3 Macros

la macro %ifarch est très important pour tout cela. LA plupart du temps vous autre besoin de faire un patch ou deux qui sera spécifique à une architecture seulement. Dans ce cas, RPM va vous permettre d'appliquer ce patch uniquement sur cette architecture.

Dans l'exemple plus haut, fileutils a un patch pour les machines 64 bits. Manifestement, cela doit uniquement être appliqué sur Alpha à ce jour. Donc, on ajoute une macro %ifarch pour le patch 64 bits comme suit:

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

Cela garantira que le patch ne sera pas appliqué sur une autre architecture que Alpha.

7.4 Exclure des architectures des paquetages

Comme vous pouvez maintenir les RPMs sources dans un répertoire pour toutes les plateformes, nous avons implémenté la capacité d'exclure des paquetages d'être compilées sur certaines architectures. C'est ce que vous pouvez faire avec quelque chose comme

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

et vous obtenez les vrais paquetages compilés. Si vous n'avez pas encore porté une application sur une certaine platefome, tout ce que vous devez faire est une ligne comme:

```
ExcludeArch: axp
```

à l'en-tête du fichier spec des paquetages source. Ensuite recompilez les paquetages sur les plateformes sur lesquelles il compile. Vous aurez alors un paquetage source qui compile sur Intel et peut facilement être sauté sur Alpha.

7.5 Pour finir

Utilisez RPM pour construire des paquetages multi-architectures est habituellement plus simple à faire que d'obtenir du paquetage lui-même qu'il compile sur des architectures différentes. Aussi plus les paquetages compilent difficilement plus vous obtiendrez de facilité (Ndt: ?). Comme toujours, la meilleure aide quand la construction d'un RPM vous pose problème est de regarder un paquetage source similaire.

8 Note de Copyright