Mon premier rpm

http://alexandre-mesle.com

7 juin 2010

Table des matières

1 Avant de commencer

1.1 RPM

Les `.rpm` sont des fichiers permettant d'installer des packages. Par package, on entend n'importe quel ensemble de fichiers (exécutables ou non) qui peuvent se greffer au système. Il peut s'agir de fichiers de configuration, d'une documentation, du noyau d'un système, ou de logiciels.

Le logiciel `rpm` est un gestionnaire de packages qui permet de consulter, supprimer, rechercher, etc. des packages. Lors de l'installation, il prend en argument un fichier `.rpm` et on peut donc effectuer l'installation comme suit :

```
[klaus@localhost rpmbuild]$ su
Password:
[root@localhost bin]# rpm -ivh domino-1-1.noarch.rpm
Préparation...              ########################################### [100%]
   1:domino                 ########################################### [100%]
```

Vous remarquerez qu'il est nécessaire d'être le root pour installer un package, les options `-ivh` servent à préciser qu'il s'agit d'une installation, et `domino-1-1.noarch.rpm` est le nom du fichier `.rpm` contenant le package.

1.2 URPMI

Le logiciel `urpmi` est une extension de `rpm` allant chercher sur Internet le package et l'installant automatiquement. `urpmi` est très pratique dans le sens où il gère les **dépendances**. Par exemple, si un logiciel A ne peut fonctionner que si le logiciel B est installé, et que vous exécutez `urpmi A`, `urpmi` va vous installer les deux packages A et B.

Ce mécanisme représente le niveau de confort le plus élevé que peut souhaiter un utilisateur : fini les `./configure`, `make`, `make install`, et les messages d'erreur incompréhensibles qui nuisent au déploiement de vos logiciels. Il vous suffit de faire des rpms et l'utilisateur n'aura presque rien à faire pour installer vos packages.

1.3 RPMBUILD

`rpmbuild` est un logiciel permettant de générer des `rpms`. Les divers tutoriaux décrivant son utilisation n'étant pas simples, j'ai eu les plus grandes difficultés à le prendre en main, et j'ai perdu un temps considérable à chercher des éléments noyés dans des centaines de pages de documentation.

Afin d'éviter à d'autres de perdre autant de temps que j'en ai perdu, je me suis proposé de faire le tutoriel le plus simple possible, en expliquant les bases pas à pas à l'aide d'un exemple fort connu en programmation.

Nous allons ensemble créer un `.rpm` pour le programme `helloWorld`, qui sera écrit en C. Je suppose que vous êtes un minimum familier avec la programmation et que vous n'avez jamais créé de `rpms`.

2 Préparatifs

2.1 `helloWorld`

Le programme pour lequel nous allons créer un installateur est le suivant :

```
#include<stdio.h>

int main(int argc, char** argv)
{
  printf("Hello world!\n");
  return 0;
}
```

Il se compile comme indiqué ci-dessous

```
[klaus@localhost rpmbuild]$ gcc -o helloWorld -Wall helloWorld.c
```

Et il s'exécute avec

```
[klaus@localhost rpmbuild]$ ./helloWorld
Hello world!
```

2.2 Binaires et Sources

Nous devons faire la la distinction entre deux types de packages `.rpm` :
- les **binaires** contiennent les fichiers déjà compilés, ils dépendent de la plate-forme sur laquelle vous souhaitez installer votre package. Ils portent l'extension `.XXX.rpm` où `XXX` est la plate-forme (i586, noarch, etc).
- less **sources** contiennent les sources et les instructions nécessaires pour que la compilation puisse se faire automatiquement sur la plateforme de l'utilisateur. On a cette fois-ci l'extension `.src.rpm`.

2.3 Installation de `rpmbuild`

Pour installer `rpmbuild`, nous allons tout simplement utiliser `urpmi`. Connectez vous en root, et suivez les instructions, comme indiqué ci-dessous.

```
[klaus@localhost rpmbuild]$ su
Password:
[root@localhost rpmbuild]# urpmi rpm-build


    http://api.mandriva.com/mirrors/basic.2010.0.i586.list: media/main/release/rpm-build-4.6.0-6mnb2.i586.
installation de rpm-build-4.6.0-6mnb2.i586.rpm depuis /var/cache/urpmi/rpms
Préparation ...                  ##############################################
      1/1: rpm-build             ##############################################
[root@localhost rpmbuild]# exit
exit
```

2.4 Principe de `rpmbuild`

Nous allons dans la partie suivante générer un package binaire. Dans ce cas `rpmbuild` exécute toutes les instructions nécessaires à la compilation et à l'installation de votre programme. Une fois que votre programme est compilé et installé, `rpmbuild` prend une "photographie" de vos répertoires. Lors de l'installation, cette photographie est recopiée dans l'arborescence du système de l'utilisateur. Par photographie, on entend une liste de fichiers ainsi que les répertoires dans lesquels ils devront être recopiés.

3 Répertoires

La première étape va être d'effectuer une installation, mais bien entendu, cette installation doit être fictive. Au lieu de l'installer dans un répertoire dont le chemin partira de la racine `/` de votre arborescence, vous allez créer un répertoire `$HOME/rpmbuild` (vous pouvez l'appeler `$HOME/RPM`, `$HOME/toto` ou `$HOME/trouduc` si ça vous chante) qui servira de "laboratoire" pour la compilation et l'installation. Pour que `rpmbuild`, fonctionne correctement, il convient de lui indiquer le chemin du laboratoire. Editez le fichier `$HOME/.rpmmacros` et placez-y les données suivantes :

```
%_topdir       ~/rpmbuild
%_tmppath      ~/rpmbuild/tmp
```

Le répertoire `$HOME/rpmbuild/tmp` sera utilisé par `rpmbuild` pour placer des fichiers qui seront détruits après la création du package. Vous devez ensuite créer un ensemble de répertoires :

```
mkdir -p ~/rpmbuild/SOURCES ~/rpmbuild/SPECS ~/rpmbuild/RPMS ~/rpmbuild/SRPMS  ~/rpmbuild/tmp ~/rpmbuild/B
```

Passons en revue ces répertoires :
- `$HOME/rpmbuild/SOURCES` : contient les fichiers sources de l'application.
- `$HOME/rpmbuild/RPMS` : contiendra le rpm binaire à la fin de l'exécution de `rpmbuild`.
- `$HOME/rpmbuild/SRPMS` : contiendra le rpm source à la fin de l'exécution de `rpmbuild`.

– `$HOME/rpmbuild/SPECS` : contient un fichier de configuration qui sera utilisé par `rpmbuild` pour déterminer comment compiler les sources, les installer, quelles sont les dépendances, le nom de l'application, etc.
– `$HOME/rpmbuild/BUILDROOT` : répertoire qui servira de laboratoire, les binaires y seront placés pour la photo :-)

4 Fichier de spécification

La phase délicate est l'élaboration du fichier de spécification, appelé généralement `spec`. Ce fichier est en deux parties, que j'appellerai l'**entête** et les **commandes**.

4.1 Entête

Le fichier de spécification contient les informations suivantes dans son entête :
– `BuildRoot` : chemin vers le laboratoire
– `Summary` : description du package en une ligne
– `Licence` : type de licence (GNU, GPL, ...)
– `Name` : nom du package
– `Version` : version du package
– `Release` : release du package
– `Group` : catégorie du package (Education, Developpment, ...)
– `BuildArchitecture` : architecture de la machine de l'utilisateur (i586, noarch, ...)
– `%description` : description détaillée du package

Ces informations, techniquement, n'indiquent pas à `rpmbuild` comment construire le package. Il s'agit simplement de données qui aideront les utilisateurs à s'y retrouver.

4.2 Commandes

Ensuite, on crée des sections contenant les commandes qui vont être exécutées aux différentes étapes de la construction du `rpm`.
– `%build` : commandes de compilation des sources
– `%install` : commandes d'installation des binaires
– `%clean` : commandes pour faire le ménage sur votre machine, une fois le package construit
– `%files` : liste des fichiers (avec le chemin complet) tels qu'ils seront recopiés sur la machine de l'utilisateur.

5 Création et installation de `hello-world`

5.1 Construction du package

Voici le fichier de spécification que j'ai conçu pour le package `hello-world`.

```
# This is a sample spec file for helloWorld

BuildRoot: %{_topdir}/BUILDROOT/
Summary:  Prints "Hello world!" on stdout
License:  GNU
Name:  hello-world
Version:  1
Release:  1
Group:  Education
BuildArchitectures:  i586

%description
Prints "Hello world!" on stdout, that's all !

%build
```

```
cd %{_sourcedir};
gcc -Wall -o helloWorld helloWorld.c

%install
rm -fR $RPM_BUILD_ROOT;
mkdir -p $RPM_BUILD_ROOT/usr/bin;
cd %{_sourcedir}
mv helloWorld $RPM_BUILD_ROOT/usr/bin/;

%clean
rm -rf $RPM_BUILD_ROOT/

%files
%attr(755,root,root)
/usr/bin/helloWorld
```

Avant de le passer en revue nous allons l'exécuter. Pour ce faire, il convient de placer correctement dans l'arborescence tous les fichiers intervenant dans la création du package.

```
[klaus@localhost rpmbuild]$ cp -f helloWorld.c ~/rpmbuild/SOURCES/;
[klaus@localhost rpmbuild]$ cp -f spec ~/rpmbuild/SPECS/;
```

On génère ensuite le package contenant les binaires en exécutant la commande `rpmbuild -v -bb $HOME/rpmbuild/SPECS/spe` L'option `-v` sert à rendre `rpmbuild` "verbeux", `-bb` précise que l'on souhaite créer un package binaire, `$HOME/rpmbuild/SPECS/spe` est le chemin du fichier de spécification.

```
[klaus@localhost rpmbuild]$ rpmbuild -v -bb ~/rpmbuild/SPECS/spec;
Exécution_de(%build): /bin/sh -e /home/klaus/rpmbuild/tmp/rpm-tmp.9T9bcW
+ umask 022
+ cd /home/klaus/rpmbuild/BUILD
+ '[' 1 -eq 1 ']'
+ '[' 1 -eq 1 ']'
+ cd /home/klaus/rpmbuild/SOURCES
+ gcc -Wall -o helloWorld helloWorld.c
+ exit 0
Exécution_de(%install): /bin/sh -e /home/klaus/rpmbuild/tmp/rpm-tmp.7lD22f
+ umask 022
+ cd /home/klaus/rpmbuild/BUILD
+ '[' 1 -eq 1 ']'
+ rm -fR /home/klaus/rpmbuild/BUILDROOT/hello-world-1-1.i386
+ mkdir -p /home/klaus/rpmbuild/BUILDROOT/hello-world-1-1.i386/usr/bin
+ cd /home/klaus/rpmbuild/SOURCES
+ mv helloWorld /home/klaus/rpmbuild/BUILDROOT/hello-world-1-1.i386/usr/bin/
+ '[' -n '' ']'
+ /usr/share/spec-helper/clean_files
+ '[' -n '' ']'
+ /usr/share/spec-helper/compress_files .lzma
+ '[' -n '' ']'
+ /usr/share/spec-helper/relink_symlinks
+ '[' -n '' ']'
+ /usr/share/spec-helper/clean_perl
+ '[' -n '' ']'
+ /usr/share/spec-helper/lib_symlinks
+ '[' -n '' ']'
+ /usr/share/spec-helper/gprintify
+ '[' -n '' ']'
```

```
+ /usr/share/spec-helper/fix_mo
+ '[' -n '' ']'
+ /usr/share/spec-helper/translate_menu
+ '[' -n '' ']'
+ /usr/share/spec-helper/fix_pamd
+ '[' -n '' ']'
+ /usr/share/spec-helper/remove_info_dir
+ '[' -n '' ']'
+ /usr/share/spec-helper/fix_eol
+ DONT_STRIP=
+ /usr/share/spec-helper/strip_and_check_elf_files
Traitement des fichiers: hello-world-1-1
Finding  Provides: /usr/lib/rpm/mandriva/filter.sh ' ' ' ' '/home/klaus/rpmbuild/BUILDROOT/hello-world-1-1
Finding  Requires: /usr/lib/rpm/mandriva/filter.sh ' ' ' ' '/home/klaus/rpmbuild/BUILDROOT/hello-world-1-1
Requires(rpmlib): rpmlib(PayloadFilesHavePrefix) <= 4.0-1 rpmlib(CompressedFileNames) <= 3.0.4-1
Requires: libc.so.6 libc.so.6(GLIBC_2.0) rtld(GNU_HASH)
Vérification des fichiers non empaquetés: /usr/lib/rpm/check-files /home/klaus/rpmbuild/BUILDROOT/hello-wo
Ecrit: /home/klaus/rpmbuild/RPMS/i586/hello-world-1-1.i586.rpm
Exécution_de(%clean): /bin/sh -e /home/klaus/rpmbuild/tmp/rpm-tmp.n60m98
+ umask 022
+ cd /home/klaus/rpmbuild/BUILD
+ rm -rf /home/klaus/rpmbuild/BUILDROOT/hello-world-1-1.i386/
+ exit 0
```

5.2 Installation du package

Une fois cela fait vous n'avez plus qu'à installer le `rpm` que vous trouverez dans `$HOME/rpmbuild/RPMS/i586/hello-world-1-1`

```
[klaus@localhost rpmbuild]$ su
Password:
[root@localhost rpmbuild]# rpm -ivh /home/klaus/rpmbuild/RPMS/i586/hello-world-1-1.i586.rpm
Préparation...              ########################################### [100%]
   1:hello-world            ########################################### [100%]
[root@localhost rpmbuild]# exit
exit
```

Le package est installé ! Cela se vérifie ainsi :

```
[klaus@localhost rpmbuild]$ helloWorld
Hello world!
```

Vous pouvez vous simplifier la tâche en écrivant les commandes dans un fichier (je l'ai appelé `rpmbuild.sh`, en 755) :

```
#!/bin/sh
mkdir -p ~/rpmbuild/SOURCES ~/rpmbuild/SPECS ~/rpmbuild/RPMS ~/rpmbuild/SRPMS  ~/rpmbuild/tmp ~/rpmbuild/B
cp -f helloWorld.c ~/rpmbuild/SOURCES/;
cp -f spec ~/rpmbuild/SPECS/;
rpmbuild -v -bb ~/rpmbuild/SPECS/spec;
cp -f ~/rpmbuild/RPMS/*/*rpm .;
```

Vous n'aurez ainsi qu'une commande à saisir pour créer votre package :

```
[klaus@localhost rpmbuild]$ ./rpmbuild.sh
...
[klaus@localhost rpmbuild]$ ls
hello-world-1-1.i586.rpm  ...
```

6 Analyse du fichier de spécification de `hello-world`

6.1 Entête

Commençons par examiner l'entête du fichier de spécification :

```
BuildRoot: %{_topdir}/BUILDROOT/
Summary:  Prints "Hello world!" on stdout
License:  GNU
Name:  hello-world
Version:  1
Release:  1
Group:  Education
BuildArchitectures:  i586

%description
Prints "Hello world!" on stdout, that's all !
```

Seul le `BuildRoot` mérite des explications approfondies. Il s'agit de la racine du répertoire qui va servir de laboratoire. Lors de la construction du package, le fichier exécutable est placé dans `$HOME/rpmbuild/BUILDROOT/usr/bin/`. Lors de l'installation sur la machine de l'utilisateur, le `BuildRoot` est supprimé du chemin, l'exécutable est donc placé dans `/usr/bin/`. Le `%{_topdir}` est lu dans le fichier `$HOME/.rpmmacros`.

6.2 Commandes

Détaillons maintenant les sections contenant les commandes. Commençons par la section `%build` :

```
%build
cd %{_sourcedir};
gcc -Wall -o helloWorld helloWorld.c
```

`%{_sourcedir}` est le répertoire contenant les sources, il se construit de la façon suivante : `%{_topdir}/SOURCES`. La première commande positionne le shell dans le répertoire contenant les sources et la deuxième commande compile la source. La deuxième section, `%install`, contient les instructions nécessaires pour installer les exécutables à la bonne position dans le "laboratoire" :

```
%install
rm -fR $RPM_BUILD_ROOT;
mkdir -p $RPM_BUILD_ROOT/usr/bin;
cd %{_sourcedir}
mv helloWorld $RPM_BUILD_ROOT/usr/bin/;
```

`$RPM_BUILD_ROOT` est une variable d'environnement du shell positionnée automatiquement sur le `BuildRoot` précisé dans l'entête. La première instruction créée dans le laboratoire le répertoire qui va contenir le binaire, se place dans le `%{_sourcedir}` et recopie l'exécutable au bon endroit. C'est à ce moment que se fait la "photo" et que le package est construit. Les fichiers générés ne servent plus à rien et doivent être détruits une fois la construction terminée. On fait le ménage en exécutant les commandes de la section `%clean`.

```
%clean
rm -rf $RPM_BUILD_ROOT/
```

Le seul fichier généré se trouvant dans `BuildRoot`, on fait le nettoyage de façon quelque peu expéditive en vidant le répertoire `BuildRoot`. Lorsque la photo se fait, la liste des fichiers trouvés est comparée à celle se trouvant dans la section `%file`.

```
%files
%attr(755,root,root)
/usr/bin/helloWorld
```

Cette section permet de récupérer seulement les fichiers qui feront partie du package, et de préciser entre autres les droits de ces fichiers. Ils doivent apparaître dans cette section tels qu'on les trouvera sur la machine de l'utilisateur, donc sans le `BuildRoot`. L'instruction `%attr(755,root,root)` signifie que le fichier sera en droits `755`, appartient à l'utilisateur `root` qui est dans le groupe `root`.

On vérifie la façon dont les sections sont exécutées en observant la trace d'exécution de `rpmbuild` :

- Exécution de la section `%build` :

```
Exécution_de(%build): /bin/sh -e /home/klaus/rpmbuild/tmp/rpm-tmp.mYkjvt
+ umask 022
+ cd /home/klaus/rpmbuild/BUILD
+ '[' 1 -eq 1 ']'
+ '[' 1 -eq 1 ']'
+ cd /home/klaus/rpmbuild/SOURCES
+ gcc -Wall -o helloWorld helloWorld.c
+ exit 0
```

- Exécution de la section `%install` :

```
Exécution_de(%install): /bin/sh -e /home/klaus/rpmbuild/tmp/rpm-tmp.mlKdL2
+ umask 022
+ cd /home/klaus/rpmbuild/BUILD
+ '[' 1 -eq 1 ']'
+ rm -fR /home/klaus/rpmbuild/BUILDROOT/hello-world-1-1.i386
+ mkdir -p /home/klaus/rpmbuild/BUILDROOT/hello-world-1-1.i386/usr/bin
+ cd /home/klaus/rpmbuild/SOURCES
+ mv helloWorld /home/klaus/rpmbuild/BUILDROOT/hello-world-1-1.i386/usr/bin/
```

- Exécution de la section `%clean` :

```
Exécution_de(%clean): /bin/sh -e /home/klaus/rpmbuild/tmp/rpm-tmp.swL3yp
+ umask 022
+ cd /home/klaus/rpmbuild/BUILD
+ rm -rf /home/klaus/rpmbuild/BUILDROOT/hello-world-1-1.i386/
+ exit 0
```

7 Pour finir

Ce n'était qu'un exemple détaillé ayant pour but de vous initier à `rpmbuild`. Il vous reste encore à voir comment gérer les dépendances (en utilisant `Requires`), de sorte que `urpmi` aille tout seul chercher les packages nécessaires au bon fonctionnement du votre.

Il vous reste aussi à étudier les packages sources, qui diffèrent peu des packages binaires au niveau de la spécification. Dans un package source, la "photo" est prise avant la compilation, et les dernières étapes sont effectuées sur la machine de l'utilisateur.

Je vous souhaite de bien continuer votre chemin dans les monde des `rpms` ! Si vous avez des améliorations à me suggérer, n'hésitez pas !