CENTOS: Contruction de paquets RPM bases

Table of Contents

- CENTOS: Contruction de paquets RPM bases
- Le fichier .rpmmacros
- Le fichier SPEC
  - En tête
  - Dépendances
    - BuildRequires ou BR
    - Requires
  - Macros
  - Section %prep
  - Section %build
  - Section %install
  - Section %files
  - Section Changelog
  - Fichiers de langues (locales)
  - Fichiers .desktop (applications graphiques)
- Construction du paquet
  - Réglage des patches, BR, ...
  - Compilation
  - Empaquetage, contrôles des %files
  - Construction des fichiers RPM

RPM (**R**edHat **P**ackage **M**anager) est un système de gestion de paquets destiné aux systèmes RHEL.

Le fichier .rpmmacros

Le fichier .rpmmacros permet de définir les macros du système, pour les dadpter à un environnement particulier. Toutes les macros définies dans ~/.rpmmacros remplacent les macros système définies.

Dans l'exemple suivant :

- **La macro %_smp_mflags** : Ne contient généralement que l'option -j (-jobs) de la commande make, qui définit le nombre de travaux pouvant être exécutés simultanément (16 au maximum). Généralement, cette valeur est définie par le nombre de cœurs de processeur plus un. (l'exemple suivant permet de configurer automatiquement le nombre de travaux simultanés en fonction du nombre de CPUS RPM_BUILD_NCPUS et du nombre de processeur online NPROCESSORS_ONLN)
- **La macro %__arch_install_post** : Contient diverses vérifications qui doivent être exécutées sur les fichiers de la location BUILDROOT en utilisant check-rpaths et check-buildroot. Cette macro est optionnelle.

```
~/.rpmmacros

% packager   VOTRE NOM

% _topdir% (echo $ HOME) / rpmbuild

% _smp_mflags% (\
      [-z "$RPM_BUILD_NCPUS"]\\\
          && RPM_BUILD_NCPUS = "`/usr/bin/nproc2>/dev/null|| \\\
                               /usr/bin/getconf _NPROCESSORS_ONLN`";\\\
     if ["$ RPM_BUILD_NCPUS" -gt 16];  then \\\
         echo "-j16"; \\\
     elif ["$RPM_BUILD_NCPUS" -gt 3];  then \\\
         echo "-j $RPM_BUILD_NCPUS"; \\\
     else\\\
         echo "-j3"; \\\
     Fi)

 %__arch_install_post \
    [ "%{buildarch}" = "noarch" ] || QA_CHECK_RPATHS=1 ; \
    case "${QA_CHECK_RPATHS:-}" in [1yY]*) /usr/lib/rpm/check-rpaths ;; esac
\
    /usr/lib/rpm/check-buildroot
```

Le fichier SPEC

Un fichier SPEC peut être considéré comme la recette utilisée par l'utilitaire rpmbuild pour créer un RPM. Il indique au système de compilation quoi faire en définissant des instructions dans une série de sections.

Pour créer un modèle de fichier SPEC, exécuter la commande suivante :

```
$ rpmdev-newspec nom_du_spec
```

Certains types de paquets ont besoin de fichier SPEC “particuliers” ; comme par exemple les paquets Python, Perl, Ruby… La liste des modèles à cet effet se trouve dans les fichiers /etc/rpmdevtools/spectemplate-*.spec, c'est le fichier spectemplate-minimal.spec qui sera utilisé par défaut. La dernière commande deviendra dans ce cas :

```
$ rpmdev-newspec -t {template} nom_du_spec
```

Pour empaqueter un logiciel écrit en Perl par exemple, la commande sera :

```
$ rpmdev-newspec -t perl nom_du_spec
```

Cette commande créera un squelette dans le fichier nom*du*spec.spec dont le contenu sera :

```
 Name:           nom_du_spec
```

```
Version:
Release:        1%{?dist}
Summary:

Group:
License:
URL:
Source0:
BuildRoot:      %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u}
-n)

BuildRequires:
Requires:

%description

%prep
%setup -q

%build
%configure
make %{?_smp_mflags}

%install
rm -rf $RPM_BUILD_ROOT
make install DESTDIR=$RPM_BUILD_ROOT

%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-,root,root,-)
%doc

%changelog
```

En tête

```
Name:           nom_du_spec
Version:
Release:        1%{?dist}
Summary:

Group:
License:
URL:
Source0:
BuildRoot:      %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u}
-n)
```

Le début du fichier est composé des entrées suivantes :

- **Name :** nom du fichier (la valeur par défaut est le nom du fichier SPEC, et ne devrait pas être modifiée).
- **Version :** version du logiciel empaqueté. Numérique uniquement (les lettres des versions alpha, béta, cvs... sont reportés dans l'entrée Release).
- **Release :** version du paquetage. La valeur par défaut est 1%{?dist}. En théorie, un paquet à destination de extras ne passera en version 1 que lorsqu'il sera proposé. Un paquet en version Alpha, Beta ou RC devrait donc avoir une valeur inférieure, 0.1.rc2%{?dist} par exemple.
- **Summary :** brève description du logiciel empaqueté.
- **Group :** groupe d'applications auquel appartiendra votre paquet. La liste des groupes est disponible dans le fichier /usr/share/doc/rpm-*/GROUPS :

  $ cat /usr/share/doc/rpm-*/GROUPS

- **License :** la licence du programme. Rapellez vous que le Projet Fedora n'accepte que des logiciels entièrement libres. Si une quelconque restriction est appliquée à votre programme, il ne pourra pas être intégré à Extras. Une liste de licences possibles est disponible sur le wiki fedoraproject.
- **URL :** site web du logiciel.
- **Source0 :** URL complète de téléchargement des sources du logiciel. Il est nécessaire ici de renseigner l'url complète vers le fichier source original de l'application. Il ne saurait être question de modifier le paquet source, vous devrez dans ce cas utiliser des patches.
- **BuildRoot :** la valeur par défaut est %{*tmppath}/%{name}-%{version}-%{release}-root-%(%{id*u} -n) et ne devrait pas être modifiée.
- **%description :** description du paquet. Les lignes de texte ici ne doivent pas excéder 79 caractères.

Dépendances

```
BuildRequires:
Requires:
```

Un paquet RPM possède deux sortes de dépendances : les dépendances pour la compilation et celles pour l'utilisation.

Les paquets doivent ici être simplement listés, séparés par un espace. Soyez le plus restrictif possible, si votre paquet dépend de 'a' et 'b', et que 'a' dépend lui même de 'b', il suffit de mettre 'a' en BR ou Requires, yum fera le reste

BuildRequires ou BR

Définit les dépendances de compilation. On rencontre souvent l'abréviation BR pour désigner cette entrée. Les BR seront utilisés lors de la construction du RPM par extras ; le passage à mock produira une erreur si les BR ne sont pas corrects.

La liste des BR correspond généralement à un *-devel, beaucoup de dépendances (bibliothèques, *.so et perl) sont gérées automatiquement.

Note : Installation des dépendances

Pour pouvoir (re)construire un paquet, il faudra installer tous les paquets listés en BuildRequires. Pour ce faire, utiliser la commande suivante :

```
$ su -lc 'yum-builddep nomdupaquet'
```

Requires

Définit les dépendances d'installation. Les Requires sont utilisés lors de l'installation du paquet par l'utilisateur.

Pour connaître les dépendances d'un paquet rpm, utiliser la commande :

```
$ rpm -qp --requires paquet.rpm
```

ou encore :

```
$ rpm -q --requires paquet
```

Macros

On peut trouver les macros existantes dans les fichiers /usr/lib/rpm/macros et /usr/lib/rpm/i386-linux/macros, et aussi /etc/rpm/* qui peuvent contenir des macros ajoutés par d'autres RPM (ex : php, pear, jpackage...)

Section %prep

```
%prep
%setup -q
```

Cette section prépare votre logiciel pour son empaquetage.

C'est ici que les sources seront décompressées, et que les patchs éventuels (que ce soient les vôtres ou cexu fournis par le projet initial) seront appliqués.

Cette étape travaille uniquement dans le dossier BUILD.

Section %build

```
%build
%configure
make %{?_smp_mflags}
```

Cette section compile le logiciel.

C'est ici que seront exécutées les commandes ./configure et make. L'exécution du script de configuration vérifiera (si le logiciel empaqueté les renseigne correctement) les autres programmes requis pour sa compilation. Vous pourrez, au fur et à mesure des erreurs signalées par le script de configuration, renseigner les BuildRequires.

Cette étape travaille uniquement dans le dossier BUILD.

Section %install

```
%install
rm -rf $RPM_BUILD_ROOT
make install DESTDIR=$RPM_BUILD_ROOT

%clean
rm -rf $RPM_BUILD_ROOT
```

La section %install du fichier .spec installe les fichiers binaires, les pages de manuel, les fichiers de configuration et tous les autres composants appartenant au package RPM dans leurs emplacements respectifs dans le répertoire racine de construction virtuelle. Il faut définir le mot clé BuildRoot dans la première partie, c'est le répertoire racine de la construction virtuelle dans lequel seront installés le fichiers.

La macro $RPM_BUILD_ROOT doit être définie dans le répertoire défini dans le mot clé BuildRoot de l'en-tête du fichier SPEC du RPM. En règle générale, on le définit comme suit: %{_tmppath}/%{name}-%{version}}. Ceci est considéré comme le répertoire racine de premier niveau par RPM.

L'exemple suivant présente une installation assez simple qui illustre un des scripts de %install les plus faciles que l'on puisse rencontrer :

```
%install
mkdir -p $ RPM_BUILD_ROOT/usr/bin
install -m755 $RPM_BUILD_DIR/%{name}-%{version}/rpmproc
$RPM_BUILD_ROOT/usr/bin
install -m644 $RPM_BUILD_DIR/%{name}-%{version}/rpmproc.conf
$RPM_BUILD_ROOT/etc
```

1. **La première ligne** identifie que tout ce qui est dans cette section appartient au script %install .
2. **La deuxième ligne** crée le répertoire /usr/bin dans l'environnement BuildRoot.Il est nécessaire d'utiliser la macro $RPM_BUILD_ROOT, sinon, tout sera installé dans le répertoire /usr/bin du système, ce qui est la dernière chose à faire. L'utilisation -p avec la commande mkdir permet de créer tous les répertoires parents s'ils n'existent pas déjà.
3. **La troisième ligne** effectue l’installation du programme dans /usr/bin, en spécifiant les droits 0755, (lecture/écriture/exécution pour le propriétaire et en lecture/exécution pour les autres). Il s’agit généralement du mode standard pour la plupart des exécutables exécutés par l’utilisateur sur n’importe quel système Linux. La macro $RPM_BUILD_DIR est utilisée ici car c’est le répertoire qui stocke le répertoire de niveau supérieur après la désarchivage du fichier

tar source (/usr/src/RPM/BUILD)
4. **La quatrième ligne** effectue l'installation du fichier rpmproc.conf dans /etc, en spécifiant les droits 0644, (lecture/écriture pour le propriétaire et en lecture seule pour les autres)

Section %files

```
%files
%defattr(-,root,root,-)
%doc
```

La section %files du fichier .spec est une section très importante car elle indique à RPM quels fichiers doivent être inclus dans les packages binaires. Dans l'exemple suivant :

```
%files
%defattr(-,root,root)
%doc README COPYING COPYRIGHT
/usr/bin/rpmproc
/etc/rpmproc.conf
```

1. **La première ligne** identifie les lignes qui suivent en tant que liste de fichiers.
2. **La deuxième ligne** fournit les attributs par défaut pour les fichiers de la liste, chaque fichier qui suit appartiendra à l'utilisateur root et au groupe root, en conservant les autorisations qui leur sont attribuées. La syntaxe pour %defattr est %defattr([mode],[utilisateur],[groupe]) (Pour exclure n'importe lequel des attributs, utiliser un caractère - comme nous cela est fait dans l'exemple, ce qui exclut la définition d'un attribut de mode de fichier par défaut).
3. **La troisième ligne** identifie les fichiers faisant partie de la documentation. Cela a un double objectif. La premier consiste à identifier les fichiers de documentation. Le second consiste à inclure ces fichiers dans l'archive RPM du répertoire /usr/doc/%{name}-%{version}. La directive %doc extrait les fichiers du répertoire de construction (dans ce cas, le répertoire /usr/src/RPM/BUILD/rpmproc-1.3/) et les copie automatiquement dans le répertoire /usr/doc/rpmproc-1.3/ de BuildRoot, puis les ajoute au package RPM. (pour ajouter des fichiers depuis un sous-répertoire, il faut simplement préfixr le nom du fichier avec le nom du répertoire, comme ceci %doc somedir/README somedir/COPYING otherdir/otherfile COPYRIGHT)
4. **Les lignes suivantes** listent les fichiers et les répertoires à "installer". Dans l'exemple on ajoute les fichiers /usr/bin/rpmproc et /etc/rpmproc.conf de l'environnement BuildRoot au package. Dans ce cas, il n'est pas nécessaire de spécifier la macro $RPM_BUILD_ROOT avant chaque fichier ou répertoire car RPM sait qu'il s'agit du répertoire racine virtuel à utiliser lors de l'ajout de fichiers au package. Il ajoute en interne cette macro avant chaque fichier (on peut utiliser les caractères génériques pour spécifier de fichiers dans un répertoire particulier par exemple :/usr/bin/* inclura tout ce qui est dans le répertoire /usr/bin/).

Section Changelog

La partie changelog du fichier SPEC indique les modifications apportées au fichier. Chaque entrée est formatée de la façon suivante :

```
* Sun Dec 3 2006 Nom Auteur <adresse@email> version_logiciel-version_paquet
- Modification
```

```
- ...
```

La date au bon format peut être facilement connue via la commande :

```
LC_TIME=en_US date +"%a %b %e %Y"
```

Fichiers de langues (locales)

Si l'application empaquetée contient des fichiers de localisation (généralement, ces fichiers sont installés dans %{_datadir}/locale et se nomment %{name}.mo), il faudra les traiter d'une façon particulière.

Tout d'abord, ajoutez gettext à vos BuildRequires. Ensuite, il faut trouver les fichiers de localisation, vous utiliserez pour cela la macro %find_lang dans la section %install de la façon suivante :

```
%find_lang %{name}
```

Enfin, il faut inclure les fichier de localisation dans la section %files :

```
%files -f %{name}.lang
```

Fichiers .desktop (applications graphiques)

Si l'application possède une interface graphique, il est probable que l'on ai besoin d'un fichier .desktop afin qu'un raccourci apparaissent dans les menus. Il est possible que l'application intègre déjà une fichier .desktop qui sera alors simplement intégré lors de la résolution des %files. Dans le cas contraire, il suffit de créer le fichier ~/rpmbuild/SOURCES/nom_application.desktop selon le modèle suivant :

```
[Desktop Entry]
Encoding=UTF-8
Name=Comical
GenericName=Comic Archive Reader
Comment=Open .cbr & .cbz files
Exec=comical
Icon=comical.png
Terminal=false
Type=Application
Categories=Application;Graphics;
Version=0.9.4
```

Puis ajouter dans lee fichier SPEC :

dans BuildRequires :

```
 desktop-file-utils
```

en source, votre fichier .desktop :

```
 Source10:  %{name}.desktop
```

pour la prise en charge du fichier, dans la section %install :

```
desktop-file-install --vendor=""                                    \
   --dir=%{buildroot}%{_datadir}/applications/   \
   %{buildroot}%{_datadir}/applications/foo.desktop
```

Dans le cas ou le fichier .desktop fait référence à une image (comical.png), il est nécessaire de mettre à jour le cache des icônes. Dans le cas contraire, l'icône ne sera visible qu'après reconnexion de l'utilisateur à sa session graphique.

Dans la section %post du fichier SPEC, mettez à jour le cache pour les icones :

```
touch --no-create %{_datadir}/icons/hicolor
if [ -x %{_bindir}/gtk-update-icon-cache ]; then
  %{_bindir}/gtk-update-icon-cache -q %{_datadir}/icons/hicolor;
fi
update-mime-database %{_datadir}/mime &> /dev/null || :
update-desktop-database &> /dev/null || :
```

Dans la section %postun du fichier SPEC, mettez également à jour le cache pour les icones :

```
touch --no-create %{_datadir}/icons/hicolor
if [ -x %{_bindir}/gtk-update-icon-cache ]; then
  %{_bindir}/gtk-update-icon-cache -q %{_datadir}/icons/hicolor;
fi
update-mime-database %{_datadir}/mime &> /dev/null || :
update-desktop-database &> /dev/null || :

Ressources :

   Spécification
   Catégories
```

Construction du paquet

Afin de construire le paquet, les sources du programme doivent être dans une archive .tar.gz dans le répertoire ~/rpmbuild/SOURCES/ (Le nom du paquer .tar.gz doit correspondre à la valeur renseignées dans **Source0** dans le fichier SPEC).

Une fois cette étape complétée, suivre simplement les étapes suivantes :

Réglage des patches, BR, ...

```
rpmbuild -bp fichier.spec
```

Execute la section %prep

Compilation

```
rpmbuild -bc --short-circuit fichier.spec
```

Execute les sections %prep, %build

L'option –short-circuit permet de forcer la compilation à redémarrer à la dernière étape valide (-bc, -bi uniquement)

Empaquetage, contrôles des %files

```
rpmbuild -bi --short-circuit fichier.spec
```

Execute les sections %prep, %build, %install, %check

Note: L'option **–short-circuit** permet de forcer la compilation à redémarrer à la dernière étape valide (-bc, -bi uniquement)

Construction des fichiers RPM

(Cette dernière commande va exécuter de nouveau les procédures précédentes)

```
rpmbuild -ba fichier.spec
```

Execute les sections %prep, %build, %install, %check, package (bin, src)