CENTOS: Contruction de paquets RPM avancé

Table of Contents

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

La commande rpmbuild

Syntaxe de la commande

```
rpmbuild -b<stage> options file1.spec … fileN.spec
```

On peut indiquer un ou plusieurs fichiers .spec

Etapes <stage>

<stage>	Libellé
p	rpmbuild -bp — Execute la section %prep
c	rpmbuild -bc — Execute les sections %prep, %build
i	rpmbuild -bi — Execute les sections %prep, %build, %install, %check
b	rpmbuild -bb — Execute les sections %prep, %build, %install, %check, package (bin)
a	rpmbuild -ba — Execute les sections %prep, %build, %install, %check, package (bin, src)
l	rpmbuild -bl — Vérifie les listes de la section %files

Options

Options	Libellé
–short-circuit	Forcer la compilation à démarrer à une étape particulière (-bc, -bi uniquement)
–test	Créer, enregistrer des scripts de construction pour révision
–clean	Nettoyer après la construction
–sign	Ajoute une signature numérique
–buildroot <root>	Exécute %install en utilisant <root> en tant que racine
–buildroot <path>	Exécute %install en utilisant <path> en tant que racine
–buildarch <arch>	Effectue la construction pour l'architecture <arch>
–buildos <os>	Effectue la construction du système d'exploitation <os>
–timecheck <secs>	Affiche un avertissement si les fichiers sont anciens (si l'âge des fichiers est supérieur à <secondess>
-vv	Affiche les informations de débogage
–quiet	Produire le plus petit résultat possible
–rcfile <rcfile>	Définit le fichier rpmrc alternatif sur <rcfile>

L'option --short-circuit

L'option **–short-circuit** indique à la commande **rpmbuild** de démarrer à un emplacement particulier de la construction. Plutôt que de suivre toutes les étapes jusqu'à l'étape de construction demandée,

Cela fonctionne uniquement avec les options **-bc** et **-bi**, ainsi que les options **-tc** et **-ti**.

Par exemple, si on exécute la commande **rpmbuild -bc** pour arrêter après la section **%build**, on peut utiliser l'option **–short-circuit** pour redémarrer la construction dans la section **%build**. Si on rencontre un problème dans la section **%build** et qu'on vous le corrige, on peut redémarrer la génération dans la section %build plutôt que d'extraire à nouveau toutes les sources.

Cette option est particulièrement utile lorsqu'on debogue une compilation d'un package. Sans l'option **–short-circuit**, on passerait probablement beaucoup de temps à recompiler le code déjà compilé.

Lors du développement normal d'un package RPM, exécuter chaque section de construction, corriger les erreurs et redémarrer là où ou on a détecté le problème. Ce cycle sera déroulé plusieurs fois avant que le RPM ne fonctionne enfin correctement.

Ne jamais distribuer de paquet créer après un nombre de tours conséquent avec l'option **–short-circuit**. Au lieu de cela, une fois que tout fonctionne, repartir de zéro et reconstruire le RPM. Ceci permet d'éviter tout problème avec un RPM partiellement créé.

Traitement des dépendances

Les dépendances permettent à un constructeur de packages d'exiger que d'autres packages ou fonctionnalités soient installés avant ou en même temps. RPM s'assure que les dépendances sont satisfaites chaque fois que les packages sont installés, effacés ou mis à niveau.

Macros

Les macros disponibles:

1. __find_provides
2. __find_prereq
3. __find_requires
4. __find_conflicts
5. __find_obsoletes

si elles existent, sont développées sous la forme de programmes exécutables. Pour chaque paquet, le programme reçoit le fichier manifeste glob sur stdin et renvoie des dépendances sur stdout.

La configuration rpm par défaut ne contient que __find_provides /usr/lib/rpm/find-provides __find_requires /usr/lib/rpm/find-requires qui peut être remplacé (ou même indéfini) dans un fichier de spécification.

Exemple

Désactiver la recherche de dépendances

```
%define __find_provides %{nil}
%define __find_requires %{nil}
%define _use_internal_dependency_generator 0
```

Vérification post-installation

La vérification post-installation des fichiers installés est déterminée par trois variables:

- __arch_install_post
- __os_install_post
- __spec_install_post

Chacune pouvant être examinée avec rpm –eval %var, par exemple :

```
rpm --eval %__os_install_post
```

```
/usr/lib/rpm/brp-compress
/usr/lib/rpm/brp-strip /usr/bin/strip
/usr/lib/rpm/brp-strip-static-archive /usr/bin/strip
/usr/lib/rpm/brp-strip-comment-note /usr/bin/strip /usr/bin/objdump
```

check-rpaths

Permet de détecter les erreurs RPATH (chemins utilisés au moment de l' exécution, codés en dur dans un fichier exécutable ou une bibliothèque) et provoque un échec **rpmbuild**. Pour ignorer ces erreurs, on peut utiliser la variable d'environnement '$QA_RPATHS' qui est un masque permettant les valeurs suivantes:

- **0x0000** valeur par défaut.
- **0x0001** RPATH standard (par exemple, /usr/lib); problèmes mineurs mais introduisant des chemins de recherche redondants sans fournir un avantage. Ils peuvent également causer des erreurs dans environnements multilib.
- **0x0002** RPATH invalides; ce sont des RPATH qui ne sont ni absolus ni les noms de fichiers relatifs et peuvent donc être un risque de sécurité
- **0x0004** RPATH non sécurisés; ce sont des RPATH relatifs qui sont un risque de sécurité
- **0x0008** RPATH spéciaux '$ORIGIN' apparaissent après d'autres RPATHs; ceci est juste un problème mineur mais généralement indésirable
- **0x0010** un RPATH est vide; il n'y a aucune raison pour ces RPATH et provoquent un travail inutile lors du chargement des bibliothèques
- **0x0020** un RPATH fait référence à '..' d'un chemin absolu; ça va casser la fonctionnalité quand le chemin avant '..' est un lien symbolique

Exemple

Pour ignorer les RPATH standard et vides, exécutez **rpmbuild** comme ceci

```
$QA_RPATHS=$[0x0001|0x0010] rpmbuild my-package.src.rpm
```

Pour vérifier les fichiers existants, définir **$RPM_BUILD_ROOT** et exécuter **check-rpaths** comme ceci

```
$RPM_BUILD_ROOT=<répertoire root>/usr/lib/rpm/check-rpaths
```