Packaging d'applications pour Debianoïdes

Sous linux il est souvent fastidieux d'installer une application directement à partir des sources à cause des dépendances et autres manipulations pas forcément très évidentes. Il est donc parfois très pratique de pouvoir packager ses sources pour que notre application puisse être installée le plus facilement possible sur les postes de nos utilisateurs. Il existe plusieurs formats de packages pour linux, les deux plus connues sont bien sûr les RPMs pour RedHat, Fedora et CentOS, ainsi que les DEBs pour tous les systèmes debianoïdes (ubuntu, mint...). C'est donc ce dernier format qui va nous intéresser.

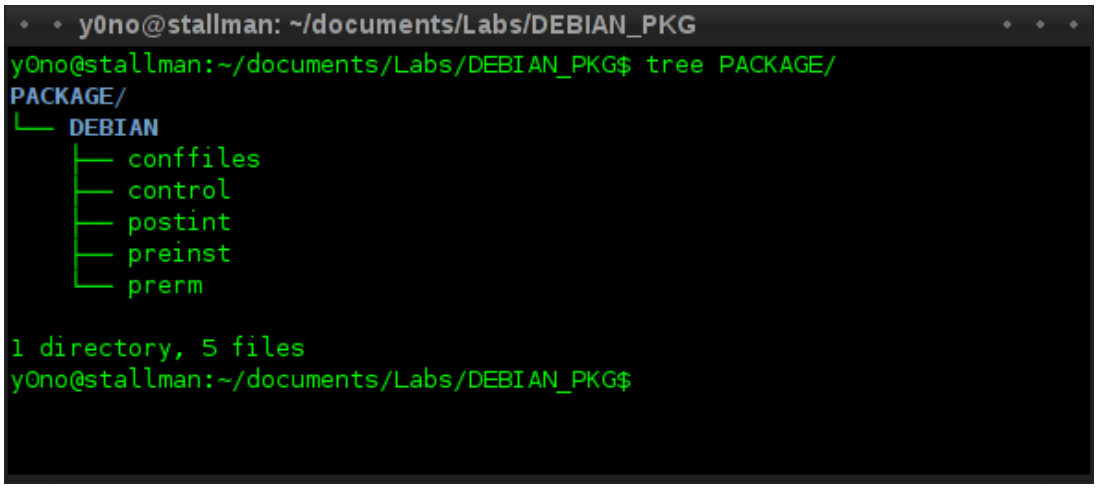
Packager une application est souvent plus fastidieux, cela demande une organisation claire du projet mais permet une flexibilité inégalable pour l'installation, désinstallation,et même la distribution !
Ne perdons pas de temps est étudions tout de suite la structure d'un paquet debian.

I L'arborescence

Dans un premier temps il faut créer un dossier où l'on mettra tout notre bazar, appelons le PACKAGE histoire de faire simple. A l'intérieur de celui ci nous allons créer un dossier DEBIAN, ce dernier contiendra tous les fichiers de configuration de notre application. Nous en retiendrons 6 importants à la mise en place de notre package :

- **control :** Ce fichier contient toutes les informations du package, son nom, son auteur, sa decription, etc...
- **preinst :** Ce fichier bash permet d'exécuter un script juste avant l'installation du package
- **postinst :** Ce fichier bash permet d'exécuter un script juste après l'installation du package
- **prerm :** Comme vous l'aurez deviné celui ci permet d'exécuter un script juste avant la désinstallation du package
- **postrm :** Allez celui là je vous laisse deviner...
- **conffiles :** The last but not least, ce fichier permet d'inscrire le chemin de tous les fichiers de configuration de l'application, nous reviendrons sur son fonctionnement plus tard.

Je reste volontairement très flou sur le rôle de ces fichiers car j'y reviendrais plus en détails dans la suite de ce HowTo.
Maintenant que l'on sait à quoi sert le dossier DEBIAN, on peut donc passer à la suite. Revenons dans notre dossier PACKAGE, il doit acutellement ressembler à ça :



Maintenant nous allons passer à l'arborescence de notre projet. La première question que l'on pourrait se poser est :

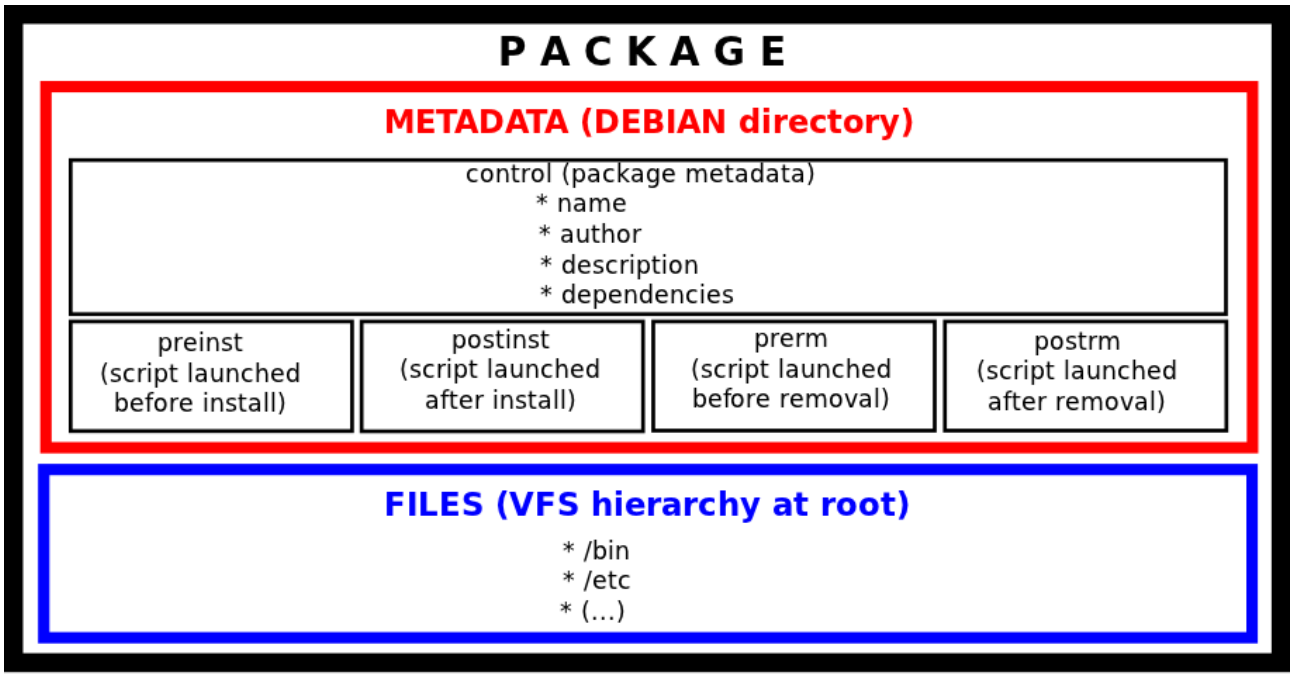
« Mais comment vais-je dire à mon packager d'installer tel fichier à tel endroit ? »

Celà peut sembler complexe mais la réponse est pourtant simple, il suffit de créer notre arborescence dans le dossier PACKAGE ! Comme un exemple vaut mieux que tout les mots voici une capture d'un projet avec l'arboresence :

```
y0no@stallman: ~/documents/Labs/DEBIAN_PKG
y0no@stallman:~/documents/Labs/DEBIAN_PKG$ tree PACKAGE/
PACKAGE/
├── DEBIAN
│   ├── conffiles
│   ├── control
│   ├── postint
│   ├── preinst
│   └── prerm
├── opt
│   └── monApp
│       └── truc.py
└── usr
    └── LauncherTruc

4 directories, 7 files
y0no@stallman:~/documents/Labs/DEBIAN_PKG$
```

Et puis c'est tout, on en a terminé avec notre dossier package, tout est organisé comme il faut. Pour résumer voici notre structure :



La suite va donc maintenant se passer dans les fichiers du dossier DEBIAN que nous avons créé précèdément, je vous avez dis qu'on y reviendrait =)

II Configurations et Scripts

fichier 'control' :

Comme je le disais précédement le fichier contrôle est ici pour décrire notre paquet. Nous y retrouverons donc son nom, son auteur, sa catégorie, sa taille, ses dépendances, etc... Voyons ensemble un exemple de fichier :

```
Package: PACKAGE
Priority: optional
Section: devel
Maintainer: y0no <yoann.onoditbiot@gmail.com>
Installed-size: 5000
Architecture: all
Version: 1.0
Depends: python2.6
Description: Test pour packaging debian.
```

Les intitulés sont plutôt clairs mais nous allons quand même essayer d'en apprendre un peu plus.

- **Package** est le nom de votre application, celui avec lequel on la retrouvera dans synaptic par exemple;)
- **Priority** est son classement dans les dépots, il y a plusieurs niveaux : critical, important, standard, optionnal et extra à vous de voir selon vos besoins mais il s'agit souvent d'optional.
- **Section** est la branche dans laquelle se classe notre application, vous pourrez retrouver la liste ici : http://packages.debian.org/stable/
- **Maintainer** est bien sûr votre nom, pseudo, enfin ce que vous voulez à propos de vous =)
- **Installed-size** est la taille que votre application prend sur le système une fois déployé, c'est bien sûr qu'une indication.
- **Architecture** est l'architecture sur laquelle notre application sera disponible (!!!!!!!)
- **Version** est bien sûr la version de votre application pour les mises à jours toussa...
- **Depends** représent les dépendances qu'à notre application vis à vis du système, il est possible de définir une version précise par exemple *'python (>= 2.6)'*
- **Description** est donc la description de votre application.

fichiers 'pre/postinst' :

Ces deux fichiers permettent l'exécution de script bash avant et après l'installation du package, on peut donc facilement imaginer par exemple la compilation d'une dépendance non disponible dans les dépots en préinstallation, et l'exécution de différentes routines en post installation par exemple pour checker la bonne mise en œuvre de notre application.

Exemple preinst :

```
#!/bin/bash

wget http://truc.com/bidul.tar.gz
tar xvf bidul.tar.gz
cd bidul/
./configure
make
```

Exemple de postinst :

```
#!/bin/bash

echo -n "Set 'truc' dir permissions "
chmod 0777 -R /opt/monApp/truc 2>> bugreport
if [[ $? -ne 0 ]]
then
        echo "[FAILED]"
else
        echo "[OK]"
fi
exit 0
```

fichier 'pre/postrm' :

Bon pas de surprise en vue, comme vous l'auriez deviné, ces deux fichiers font la même chose que les deux précédents mais au moment de la suppression de notre paquet. Pas besoin d'exemples on reste sur le même type de script que les précédents.

fichier 'conffiles' :

Ce fichier est un peu plus particulier quant à lui, en effet il ne s'exécute pas lors d'une simple désinstallation, et encore moins à l'installation. Pour faire plus concret peut être avez vous remarqué que lors d'une désinstallation de paquet par exemple via '*dpkg -r'* ou '*apt-get remove',* les fichiers de configurations restaient présents et qu'il fallait forcer leurs désinstallations par exemple avec *'apt-get remove –purge ...'* ? Et bien nous y sommes, c'est pour avoir ce fonctionnement que 'conffiles' est présent. Tout les chemins de fichiers que nous noterons dans ce fichier ne seront pas supprimés lors d'une simple suppression de notre paquet, ce genre de comportement peut donc être intéressant dans certains cas de figures.

Exemple de conffiles :

```
/etc/monApp/app.conf
/etc/monApp/advanced/app.conf
```

fichier 'md5sums' :

Ce fichier contient la somme md5 de chaque fichier présents dans notre package lors de l'extraction, c'est bien sûr un fichier optionnel, mais vu qu'il y a une commande permettant de le générer facilement pourquoi s'en priver ?

```
find . -type f ! -regex '.*\.hg.*' ! -regex '.*?debian-binary.*' ! -regex '.*?DEBIAN.*' -printf '%P ' | xargs md5sum > DEBIAN/md5sums
```

Ce qui donnera :

```
d41d8cd98f00b204e9800998ecf8427e  opt/monApp/truc.py
d41d8cd98f00b204e9800998ecf8427e  usr/LauncherTruc
```

III Packaging

Nous y voilà, vous en avez rêvé, voire même plus, le packaging de votre application ! Mais d'abord quelques vérifications sont nécessaires.

Clearing :

Tout d'abord nous devons nous assurer d'une chose essentielle, nous devons fournir un paquet propre, c'est à dire enlever tous les fichiers temporaires et non nécessaires au fonctionnement de notre superbe programme. Dans le cas PACKAGE qui est codé en python , on peut très bien imaginer supprimer tous les fichiers .pyc créés lors de la première exécution d'un script python ainsi que tous les fichiers terminant par ~ générer par certains éditeurs de texte. Pour ça on peut éxecuter les instructions suivantes à la racine de notre projet :

```
find . -iname "*.pyc" -exec rm {} \;
find . -iname "*~" -exec rm {} \;
```

Permissions :

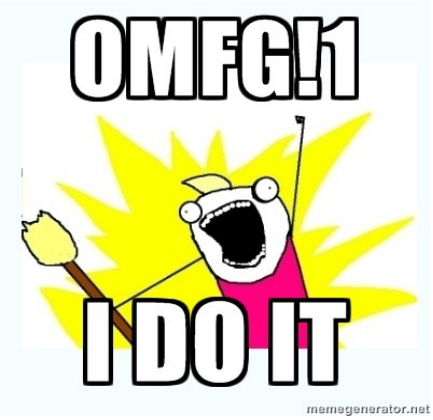
Bien, nous y sommes presque, il ne nous reste plus qu'une étape avant le graal. Les scripts 'post/pre' que nous avons créé précèdemment ne sont pour le moment que de simples fichiers textes, vous l'aurez donc compris il va falloir leur attribuer les permissions nécessaires pour qu'ils s'éxécutent au moment voulu :

```
chmod 0755 {pre,post}*
```

FUSIONNNNNN !

Prenez une grande inspiration et préparez vous à être ébobi par ce qu'il va se passer, maintenant que tout est prêt voici l'instruction ultime qui va faire passer notre application de bricolage du dimanche à celle d'incontournable de la galaxie inuxienne, voir même plus loin encore !

```
fakeroot dpkg -b PACKAGE monApp.deb
```



Bon d'accord, j'ai peut être un peu abuser sur le suspense pour une aussi petite commande, mais il n'empêche que nous avons notre application packagée : *monApp.deb* . Il ne vous reste plus qu'à l'essayer et à la partager !

IV Conclusion

Même si cela peut paraître difficile à première vue, packager une application reste plutôt enfantin. Même si aujourd'hui nous n'avons vu que les rudiments dans ce domaine, nous remarquons qu'avec un peu d'organisation, on peut rapidement déployer son application dans un format plus digeste pour les utilisateurs pas forcément habiles avec un terminal ou même les personnes préfèrant garder un système stable et propre.

V Bonus

Juste comme ça avant de vous abandonner à votre nouveau hobbie, voici un script permettant d'aller encore un peu plus vite :

```
#!/bin/bash

ROOT=PACKAGE
PKG_NAME=MonApp

find . -iname "*.pyc" -exec rm {} \;
find . -iname "*~" -exec rm {} \;
chmod 0755 ${ROOT}/DEBIAN/{pre,post}*
fakeroot dpkg -b ${ROOT} $PKG_NAME.deb
```

Sur ce, je vous laisse vous amuser !
Stay Free