CPE Lyon - 4ETI, 3IRC & 3ICS - Année 2022/2023

Administration Système

TP 4 - Gestion des paquets

Ce quatrième TP a pour but de vous faire prendre en main `dpkg` et le gestionnaire de paquets `apt`. Pour chaque question, donnez une commande permettant d'obtenir le plus précisément possible le résultat demandé (i.e. en filtrant toutes les informations superflues).

Exercice 1. Commandes de base

1. Commencez par mettre à jour votre système avec les commandes vues dans le cours.
2. Créez un *alias* "**`maj`**" de la ou des commande(s) de la question précédente. Où faut-il enregistrer cet alias pour qu'il ne soit pas perdu au prochain redémarrage ?
3. Utilisez le fichier `/var/log/dpkg.log` pour obtenir les 5 derniers paquets **`installés`** sur votre machine.
4. Listez les derniers paquets qui ont été installés explicitement avec la commande **`apt install`**
5. Utilisez les commandes **`dpkg`** et **`apt`** pour compter de deux manières différentes le nombre de total de paquets installés sur la machine (ne pas hésiter à consulter le manuel !). Comment explique-t-on la (petite) différence de comptage ? Pourquoi ne peut-on pas utiliser directement le fichier `dpkg.log` ?
6. Combien de paquets sont disponibles en téléchargement sur les dépôts Ubuntu ?
7. A quoi servent les paquets **`glances`**, **`tldr`** et **`hollywood`** ? Installez-les et testez-les.
8. Quels paquets proposent de jouer au sudoku ?
   ⚠N'installez **pas** le paquet `gnome-sudoku` ou `ksudoku` sous peine de devoir probablement réinstaller votre VM

Exercice 2.

A partir de quel paquet est installée la commande **`ls`** ? Comment obtenir cette information **en une seule commande**, pour n'importe quel programme ? Utilisez la réponse à cette question pour écrire un script appelé **`origine-commande`** (sans l'extension **`.sh`**) prenant en argument le nom d'une commande, et indiquant quel paquet l'a installée.

Exercice 3.

Ecrire une commande qui affiche "INSTALLÉ" ou "NON INSTALLÉ" selon le nom et le statut du package spécifié dans cette commande.

Exercice 4.

Lister les programmes livrés avec **`coreutils`**. En particulier, on remarque que l'un deux se nomme **`[`**. De quoi s'agit-il ?

Exercice 5. aptitude

Installez les paquets **`emacs`** et **`lynx`** à l'aide de la version **graphique** d'**`aptitude`** (et prenez deux minutes pour vous renseigner et tester ces paquets).

Exercice 6. Installation d'un paquet par PPA

Certains logiciels ne figurent pas dans les dépôts officiels. C'est le cas par exemple de la version "officielle" de Java depuis qu'elle est développée par Oracle. Dans ces cas, on peut parfois se tourner vers un "dépôt personnel" ou `PPA`.

1. Installer la version Oracle de Java (avec l'ajout des PPA)

   ```
   sudo add-apt-repository ppa:linuxuprising/java
   sudo apt update
   sudo apt install oracle-java15-installer
   ```

2. Vérifiez qu'un nouveau fichier a été créé dans **`/etc/apt/sources.list.d`**. Que contient-il ?

Exercice 7. Installation d'un logiciel à partir du code source

Lorsqu'un logiciel n'est disponible ni dans les dépôts officiels, ni dans un PPA, ou encore parce qu'on souhaite n'installer qu'une partie de ses fonctionnalités, on peut se tourner vers la compilation du code source. C'est ce que nous allons faire ici, avec le programme `cbonsai` (https://gitlab.com/jallbrit/cbonsai)

1. Commencez par cloner le dépôt git suivant :

   ```
   git clone https://gitlab.com/jallbrit/cbonsai
   ```

   Ceci permet de récupérer en local le code source du logiciel `cbonsai`.

2. Rendez vous dans le dossier `cbonsai`. Un fichier `README.md`) est livré avec les sources, et vous explique comment compiler le programme (vous pouvez installer un lecteur de Markdown pour Bash, comme **`mdless`** pour vous faciliter la lecture de ce type de fichiers).

   Un fichier **`Makefile`** est également présent. Un `Makefile` est un fichier utilisé par l'outil **`make`**, et contient toutes les directives de compilation d'un logiciel. Un Makefile définit un certain nombre de *règles* permettant de construire des *cibles*. Les cibles les plus communes étant `install` (pour la compilation et l'installation du logiciel) et `clean` (pour sa suppression).

   En suivant les consignes du fichier `README.md`, et en installant les éventuels paquets manquants, compilez ce programme et installez le **en local**.

3. Malheureusement, cette installation "à la main" fait qu'on ne dispose pas des bénéfices de la gestion de paquets apportés par `dpkg` ou `apt`. Heureusement, il est possible de transformer un logiciel installé "à la main" en un paquet, et de le gérer ensuite avec `apt` ; c'est ce que permet par exemple l'outil **`checkinstall`**.
4. Recommencez la compilation à l'aide de `checkinstall` :

   ```
   sudo checkinstall
   ```

Un paquet a été créé (fichier **`xxx.deb`**), et le logiciel est à présent installé (tapez **`cbonsai`** depuis n'importe quel dossier pour vous en assurer) ; on peut vérifier par exemple avec **`aptitude`** qu'il provient bien du paquet qu'on a créé avec **`checkinstall`**.

Vous pouvez à présent profiter d'un instant de zenitude avant de passer au dernier exercice.

Exercice 8. Création de dépôt personnalisé

Dans cet exercice, vous allez créer vos propres paquets et dépôts, ce qui vous permettra de gérer les programmes que vous écrivez comme s'ils provenaient de dépôts officiels.

Création d'un paquet Debian avec `dpkg-deb`

1. Dans le dossier **scripts** créé lors du TP 2, créez un sous-dossier **origine-commande** où vous créerez un sous-dossier **DEBIAN**, ainsi que l'arborescence **usr/local/bin** où vous placerez le script écrit à l'exercice 2
2. Dans le dossier DEBIAN, créez un fichier **control** avec les champs suivants :

```
Package: origine-commande    #nom du paquet
Version: 0.1                 #numéro de version
Maintainer: Foo Bar          #votre nom
Architecture: all            #les architectures cibles de notre paquet (i386, amd64...)
Description: Cherche l'origine d'une commande
Section: utils               #notre programme est un utilitaire
Priority: optional           #ce n'est pas un paquet indispendable
```

3. Revenez dans le dossier parent de origine-commande (normalement, c'est votre $HOME) et tapez la commande suivante pour construire le paquet :

```
dpkg-deb --build origine-commande
```

Félicitations ! Vous avez créé votre propre paquet !

Création du dépôt personnel avec reprepro

1. Dans votre dossier personnel, commencez par créer un dossier **repo-cpe**. Ce sera la racine de votre dépôt
2. Ajoutez-y deux sous-dossiers : **conf** (qui contiendra la configuration du dépôt) et **packages** (qui contiendra nos paquets)
3. Dans conf, créez le fichier **distributions** suivant :

```
Origin: Un nom, une URL, ou tout texte expliquant la provenance du dépôt
Label: Nom du dépôt
// Suite: stable
Codename: focal              #!! A MODIFIER selon la distribution cible !!
Architectures: i386 amd64    #(architectures cibles)
Components: universe         #(correspond à notre cas)
Description: Une description du dépôt
```

4. Dans le dossier repo-cpe, générez l'arborescence du dépôt avec la commande

```
reprepro -b . export
```

5. Copiez le paquet origine-commande.deb créé précédemment dans le dossier packages du dépôt, puis, à la racine du dépôt, exécutez la commande

```
reprepro -b . includedeb cosmic origine-commande.deb
```

   afin que votre paquet soit inscrit dans le dépôt.
6. Il faut à présent indiquer à apt qu'il existe un nouveau dépôt dans lequel il peut trouver des logiciels. Pour cela, créez (**avec sudo**) dans le dossier **/etc/apt/sources.list.d** le fichier **repo-cpe.list** contenant :

```
deb file:/home/VOTRE_NOM/repo-cpe cosmic multiverse
```

   (cette ligne reprend la configuration du dépôt, elle est à adapter au besoin)
7. Lancez la commande **sudo apt update**.
   Féliciations ! Votre dépôt est désormais pris en compte ! ... Enfin, pas tout à fait... Si vous regardez la sortie d'**apt update**, il est précidé que le dépôt ne peut être pris en compte car il n'est pas *signé*. La signature permet de vérifier qu'un paquet provient bien du bon dépôt. On doit donc signer notre dépôt.

Signature du dépôt avec GPG

`GPG` est la version `GNU` du protocole `PGP` (Pretty Good Privacy), qui permet d'échanger des données de manière sécurisée. Ce système repose sur la notion de *clés de chiffrement asymétriques* (une clé *publique* et une clé *privée*)

1. Commencez par créer une nouvelle paire de clés avec la commande

   ```
   gpg --gen-key
   ```

   Attention ! N'oubliez pas votre passphrase !!!
2. Ajoutez à la configuration du dépôt (fichier `distributions` la ligne suivante :

   ```
   SignWith: yes
   ```

3. Ajoutez la clé à votre dépôt :

   ```
   reprepro --ask-passphrase -b . export
   ```

   Attention ! Cette méthode n'est pas sécurisée et obsolète ; dans un contexte professionnel, on utiliserait plutot un `gpg-agent`.
4. Ajoutez votre clé publique à votre dépôt avec la commande

   ```
   gpg --export -a "auteur" > public.key
   ```

5. Enfin, ajoutez cette clé à la liste des clés fiables connues de `apt` :

   ```
   sudo apt-key add public.key
   ```

Félicitations ! La configuration est (enfin) terminée ! Vérifiez que vous pouvez installer votre paquet comme n'importe quel autre paquet.