Séance n° 3 : Gestion des paquets sous Debian et Ubuntu

1 Sudo

Passer *root* avec la commande `su` est parfois contraignant, d'autant plus que cela oblige à entrer son mot de passe à chaque fois. Pourtant, il est dangereux de rester root en permanence, puisque cela augmente les conséquences des erreurs éventuelles. `sudo` est un petit outil permettant de passer root pour une exécuter une commande.

Q1. Installez `sudo` sur votre système : `apt-get install sudo`

Q2. Consultez le contenu du fichier de configuration `/etc/sudoers`

Q3. Dans `/etc/sudoers`, un groupe est prévu pour contenir les utilisateurs ayant l'autorisation d'exécuter n'importe quelle commande dans ce groupe. Mais l'utilisation de ce groupe est parfois désactivée par défaut. Éditez la configuration de `sudo` et activez ce groupe s'il est désactivé. Pour éditer la configuration de *sudo*, il est préférable de passer par `visudo`, ce qui permet de vérifier la syntaxe du fichier avant d'enregistrer les modifications. Par défaut, `visudo` utilise `vi` pour éditer la configuration, mais vous pouvez positionner la variable d'environnement `EDITOR` pour utiliser un autre éditeur : `EDITOR=nano visudo`.

Q4. Ajoutez votre utilisateur au groupe permettant d'exécuter n'importe quelle commande. Déconnectez-vous et reconnectez-vous (c'est nécessaire pour que l'ajout d'un groupe soit pris en compte). Vérifiez avec `id` que le groupe a bien été ajouté.

Q5. Avec votre compte utilisateur normal, essayez d'exécuter une commande avec sudo, par exemple `sudo id`, et vérifiez que vous êtes bien devenu l'utilisateur *root* pendant l'exécution de la commande.

Après que le mot de passe ait été demandé par sudo, il n'est plus demandé pendant 15 minutes.

Il est également possible d'utiliser sudo pour permettre à un utilisateur d'exécuter une commande particulière en tant que root. Cela peut être utilisé pour permettre à un utilisateur de recharger une configuration, par exemple.

2 Chroots

Il est parfois pratique de pouvoir disposer de plusieurs versions d'une même distribution, ou de distributions différentes, sur la même machine. Cela est possible en utilisant :

— des outils de virtualisation, qui émulent une machine complète en proposant diverses optimisations : *Xen*, *KVM*, *VirtualBox*, ou, dans le monde propriétaire, *VMWare* ;

— des outils permettant de gérer des *Conteneurs* : il s'agit alors d'utiliser des fonctionnalités particulières du noyau Linux pour permettre l'utilisation en parallèle de différents *contextes*. La solution la plus couramment utilisée est *Linux Containers (LXC)*.

Il existe aussi une technique bien plus simple, le *chroot*. Il s'agit de créer, à l'intérieur de son système de fichier, une arborescence contenant un système complet, puis d'utiliser la commande `chroot` (qui utilise elle-même l'appel système `chroot(2)`) pour **changer le répertoire racine du processus courant**.

Nous allons maintenant créer un *chroot* contenant une distribution Ubuntu.

Q1. Installez le paquet `debootstrap`.

Q2. Positionnez-vous dans un répertoire avec quelques Go d'espace disque disponible (par exemple `/global1`), et créez une arborescence pour *Ubuntu 18.04 Bionic* en utilisant le mirroir Ubuntu local de la salle ASRALL : `http://asrall-gw:9999/ubuntu/`. En tant que *root* :
`debootstrap --variant=minbase bionic ubuntu http://asrall-gw:9999/ubuntu`
Il peut être nécessaire d'ajouter l'option `--no-check-gpg`.

Q3. Observez le contenu de l'arborescence que vous venez de créer dans `/global1/ubuntu`.

Q4. Avec la commande `dpkg -l`, regardez les versions d'`apt` et de `bash` en dehors du chroot. Puis utilisez la commande `chroot` pour vous placer dans votre *chroot* Ubuntu :
`chroot /global1/ubuntu /bin/bash`. Regardez à nouveau les versions d'`apt` et de `bash`. Que constatez-vous ? Confirmez que vous avez bien des versions différentes, comme avec une machine virtuelle.

Q5. Regardez maintenant les processus s'exécutant dans le *chroot*. Que constatez-vous ?

Q6. Pour pouvoir fonctionner, `ps` a besoin d'accéder au répertoire `proc`, qui contient des informations sur les processus en cours d'exécution. Il est possible de monter certains répertoires du système hôte dans le chroot, en utilisant `mount --bind`. Depuis l'extérieur du chroot, montez `/proc` dans le chroot :
`mount --bind /proc /global1/ubuntu/proc`. Vérifiez que cela vous permet de regarder les processus s'exécutant depuis l'intérieur du chroot (avec `ps afx`). Pour `/proc`, une autre possibilité est de le remonter avec
`mount -t proc nimportequoi /global1/ubuntu/proc`. Essayez, après avoir démonté `/proc` dans le chroot.

Q7. Réalisez les opérations précédentes avec une machine virtuelle Ubuntu obtenue avec Vagrant. Que pouvez-vous conclure du niveau d'isolation d'un chroot ?

Q8. Vos données personnelles (`/home/login`) ne sont pas accessibles dans le *chroot*. Pourquoi ne peut-on pas les rendre accessibles avec un lien symbolique ? Pourquoi ne peut-on pas les rendre accessibles avec un lien dur (*hard link*) ? Rendez-les accessibles avec un *bind mount*.

Q9. Démontez votre `/home` du *chroot* pour éviter toute erreur de manipulation dans la suite du TP.

Les *chroots* donnent une fausse impression de sécurité à l'administrateur système. Contrairement aux machines virtuelles ou aux *conteneurs*, qui tentent réellement d'empêcher l'utilisateur de sortir des limites qu'on lui a fixées, il est très facile de sortir des limites d'un *chroot* à

partir du moment où on dispose des droits *root* dans le *chroot*. Toutefois, les *chroots* sont très pratiques pour tester rapidement une opération quelconque dans un environnement de travail « jetable ».

3 Gestion des paquets sous Debian/Ubuntu

3.1 Utilisation basique

Nous allons maintenant utiliser notre *chroot* pour réaliser diverses manipulations sur les paquets installés, sans risquer d'endommager notre système de travail.

Q1. Placez-vous dans le *chroot*, et installez le paquet `wget` : `apt-get install wget`

Q2. Avec `wget`, récupérez le fichier `http://asrall-gw:9999/ubuntu/pool/main/r/ruby-defaults/ruby_2.3.0+1_all.deb`. Installez ce paquet manuellement avec `dpkg -i`. Que constatez-vous ? Chaque paquet *deb* contient des méta-données (visibles avec `apt-cache show`) indiquant, entre autres, les dépendances entre ce paquet et d'autres paquets. Mais en installant un paquet avec `dpkg -i`, on contourne ce mécanisme.

Q3. Corrigez le problème avec la commande `apt-get install -f`.

Q4. Supprimez maintenant le paquet `ruby` que vous venez d'installer manuellement. Le paquet `ruby2.3` est-il supprimé ?

Q5. Consultez le fichier de configuration `/etc/apt/sources.list`. Rajoutez-y des lignes pour les dépots des *distributions* `bionic-updates` et `bionic-security`, qui contiennent respectivement des mises à jour classiques et des mises à jour de sécurité.

Q6. Rafraichissez les méta-informations sur les dépots avec `apt-get update`, et mettez à jour votre *chroot* avec `apt-get upgrade`. Combien de paquets sont mis à jour ?

Q7. Les paquets Debian sont répartis en 3 *sections* :
- *main* : logiciels libres
- *non-free* : logiciels non-libres, mais toutefois distribuables par Debian
- *contrib* : logiciels libres, mais nécessitant des logiciels non-libres pour fonctionner

Ceux d'Ubuntu sont répartis en 4 *sections* :

	logiciels libres	logiciels non-libres
officiellement supportés	*main*	*restricted*
supportés uniquement par la communauté	*universe*	*multiverse*

Si les logiciels les plus utilisés sont dans *main*, il arrive fréquemment que des logiciels utiles doivent être récupérés dans *universe* ou *multiverse*.
Essayez d'installer le paquet `links`. Que constatez-vous ?

Q8. Rajoutez les dépots `universe`, et installez `links`.

3.2 Utilisation de paquets plus récents

Il arrive fréquemment que les utilisateurs aient besoin d'une version plus récente d'un logiciel. Toutefois, l'administrateur, pour diverses raisons, souhaite souvent éviter de mettre à jour toute la distribution vers une version plus récente. Il devient alors nécessaire de pouvoir mettre à jour de manière sélective certains paquets. Plusieurs solutions sont possibles pour cela :

— Installation manuelle d'un fichier `.deb`. Mais cela pose des problèmes en cas de dépendances sur d'autres paquets qui ne pourraient pas être satisfaites.

— Utilisation de dépots *backports*. Ces dépots contiennent des paquets provenant de versions plus récentes de la distribution, recompilés pour utiliser les dépendances présentes dans la version utilisée de la distribution. Il en existe pour Debian (`http://backports.debian.org/`) et Ubuntu (dépots avec *distribution* `-backports`).

— Mise à jour sélective via *APT pinning*.

3.2.1 Utilisation des `backports`

Q1. À quoi sert le paquet `iproute2` ? Installez-le. Quelle version est installée ?

Q2. Ajoutez les dépots `bionic-backports`, puis installez le paquet `iproute2` depuis `bionic-backports` avec : `apt-get install -t bionic-backports iproute2`
Vérifiez qu'une version plus récente est bien utilisée.

3.2.2 Compilation et installation manuelle de logiciels

Il arrive parfois que les logiciels nécessaires ne sont pas disponibles sous forme de paquets, ou pas disponibles dans une version assez récente. Il faut alors les installer manuellement. La procédure peut varier fortement d'un logiciel à un autre, mais pour les logiciels écrits en C/C++, il faut généralement exécuter d'abord le script `configure` (avec `./configure`), puis lancer la compilation avec `make`, éventuellement lancer des tests avec `make test`, et finalement installer le logiciel avec `make install`.

Q1. À quoi sert CVS ?

Q2. Téléchargez la version 1.12.13 de CVS, disponible dans le répertoire *feature* sur `http://ftp.gnu.org/non-gnu/cvs/`. Faites bien attention à récupérer le fichier en HTTP, et non en FTP : sinon, il faut configurer le proxy FTP.

Q3. Décompressez l'archive, et compilez CVS, éventuellement en installant les paquets nécessaires à sa compilation (vous pouvez utiliser `apt-file` pour les trouver). A priori, il faudra au moins installer le paquet `build-essential`, qui permet d'installer tout l'environnement de compilation pour le C (compilateur, etc.)

3.3 Installation de versions plus anciennes

Il est parfois nécessaire d'installer une version plus ancienne d'un logiciel : les versions plus récentes peuvent être buggées, ou ne plus être compatibles avec un autre logiciel utilisé. Ou plus simplement, il peut être intéressant de ne pas dérouter l'utilisateur avec un changement de version qui ne serait pas utile.

Q1. À l'aide de http://packages.ubuntu.com/, récupérez le fichier .deb de la version du paquet iproute2 contenue dans *Ubuntu 16.04 Xenial* (il s'agit du fichier iproute2_4.3.0-1ubuntu3_amd64.deb). Installez ce paquet (vérifiez bien que la version que vous avez installée correspond).

Q2. Mettez à jour votre système. Que constatez-vous ? Installez à nouveau l'ancienne version de iproute2.

Q3. Il est donc nécessaire de bloquer (maintenir, *hold* en anglais) la version installée de iproute2. Cela se fait en manipulant directement la base de données de dpkg :

```
echo nompackage hold | dpkg --set-selections
```

L'opération inverse (permettant au paquet d'être à nouveau mis à jour automatiquement) est :

```
echo nompackage install | dpkg --set-selections
```

Dans les commandes ci-dessus, nompackage représente le nom du paquet, pas le nom du fichier *.deb* contenant le paquet.

Q4. Bloquez la version de iproute2 actuellement installée. Vérifiez qu'une mise à jour du système ne met pas à jour iproute2.

Q5. Débloquez la version de iproute2 installée, et mettez à jour votre système.

3.4 Aptitude, Synaptic

Il existe plusieurs gestionnaires de paquets pour Debian, en plus d'APT (utisable avec apt-get, apt-cache, et plus récemment apt), on peut mentionner *Synaptic*, qui est un gestionnaire de paquets graphique (et donc inutilisable sur un serveur sans interface graphique), et aptitude. *Aptitude* propose une interface graphique en mode console, et fournit quelques fonctionnalités intéressantes, notamment la possibilité de définir de manière très fine les versions des paquets à installer lors d'une mise à jour.

aptitude est également utilisable directement en ligne de commande. Le nom des commandes est souvent identiques à celles d'APT (aptitude install, aptitude search, etc).

Q1. Installez et lancez aptitude, et installez un paquet.

3.5 APT pinning

L'*APT pinning* permet de sélectionner, paquet par paquet, la version à installer. Il est fréquemment utilisé avec Debian *testing* et Debian *unstable*. Cela permet par exemple d'utiliser la version de Debian *testing* pour tous les paquets, sauf pour votre éditeur de texte, pour lequel vous voulez la version de Debian *unstable*.

Pour cette partie du TP, nous allons utiliser une machine virtuelle Vagrant avec la version stable de Debian. Nous allons la mettre à jour vers Debian *testing*, puis activer le *pinning* avec Debian *unstable*.

Debian *testing* est la version de Debian en cours de développement. Elle n'est pas aussi stable que la version stable de Debian, mais contient des logiciels plus récents. Du coup, Debian *testing* est parfois un bon compromis sur les portables ou les stations de travail où des versions plus récentes sont préférées à la stabilité de Debian *stable*. Sauf cas très particuliers, Debian *testing* n'est jamais utilisé sur un serveur.

Q1. Créez un environnement Vagrant avec la box `debian/contrib-buster64`. Activez le mode GUI (graphique). Le reste du TP est à faire dans cet environnement.

Q2. Dans `/etc/apt/sources.list`, remplacez *stable* ou *buster* (l'actuelle version stable) par *testing*.

Q3. Mettez à jour votre système (c'est long). Rebootez. Vous utilisez maintenant Debian *testing*.

Q4. Modifiez ou créez les fichiers de configuration suivants :

— */etc/apt/sources.list* :

```
# dépots pour Debian testing
deb http://asrall-gw:9999/debian/ testing main contrib non-free
deb-src http://asrall-gw:9999/debian/ testing main contrib non-free
deb http://security.debian.org/ buster/updates main contrib non-free
deb-src http://security.debian.org/ buster/updates main contrib non-free

# dépots pour Debian sid (unstable)
deb http://asrall-gw:9999/debian/ sid main contrib non-free
deb-src http://asrall-gw:9999/debian/ sid main contrib non-free
```

— */etc/apt/preferences* :

```
Package: *
Pin: release a=testing
Pin-Priority: 800

Package: *
Pin: release a=unstable
Pin-Priority: 700
```

Q5. Vérifiez qu'une mise à jour de votre système n'installe pas de paquets.

Q6. Vérifiez que vous pouvez installer manuellement un paquet présent dans Debian *unstable*, mais pas encore dans Debian *testing* (ou avec une version plus ancienne dans Debian *testing*). Un bon exemple est le paquet `gcc-snapshot`, qui contient la version de développement du compilateur GCC.

3.6 Conclusion sur la gestion de paquets sous Debian/Ubuntu

Q1. À l'aide des pages de manuel de `apt-get`, `apt-cache` et `dpkg`, complétez maintenant la colonne *Debian/Ubuntu* de la fiche de référence à la fin de ce TP.

Q2. Depuis APT 1.0 (disponible dans Debian 8 'jessie'), il existe maintenant une commande `apt`. Dans la fiche de référence, indiquez également comment réaliser chaque opération avec `apt`.

Action	Debian / Ubuntu (dpkg + APT)	Red Hat / Centos / Fedora (RPM + YUM)
Rechercher des paquets et des informations sur les paquets		
Rechercher un paquet par mots-clés		
Rechercher quel paquet (non-installé) contient un fichier		
Afficher des informations sur un paquet		
Lister les paquets installés sur le système		
Lister les fichiers d'un paquet installé		
Rechercher de quel paquet installé provient un fichier		
Gérer les paquets installés		
Installer un paquet et ses dépendances		
Installer manuellement un paquet, sans gérer les dépendances		
Supprimer un paquet installé précédemment		
Mettre à jour les méta-informations sur les paquets		
Mettre à jour tous les paquets installés		
Bloquer les mises-à-jour d'un paquet		
Annuler le blocage des mises-à-jour		
Fichiers de configuration du gestionnaire des paquets		
Reconfigurer (avec debconf) un paquet installé		non disponible