



## Créer un environnement de test 'la(p|m)p' avec Chroot

Le 3 octobre 2008 à 12:31.

Lorsque l'on a besoin de créer un environnement de test il est nécessaire que sa composition soit strictement contrôlé (paramétrages, applications et services disponibles et lancées, etc). Il n'est du coup généralement pas conseillé d'utiliser sa machine de travail, sauf si elle ne sert qu'à cela, sous peine d'en détériorer le fonctionnement en cas de test malheureux ou de modifier sans le vouloir le comportement de ce que l'on cherche à tester. Nous sommes alors contraints d'avoir soit une machine virtuelle dédiée aux tests, soit une machine physique supplémentaire.

Ce serait sans compter sur la commande `chroot` qui permet dans certaines conditions d'obtenir un environnement de test ou d'intégration identique à celui en production, sans machine physique supplémentaire et sans que la machine principale soit ralentie ou compromise.

### Qu'est-ce qu'un chroot ?

Comme vous le savez sûrement déjà, le système fichier d'un \*NIX est construit autour d'une racine (le /) sur laquelle les partitions sont ensuite "montées" formant ainsi l'espace de fichier accessible. Ce que l'on sait moins c'est que cette racine est un paramètre du processus. Tout processus lancé peut avoir pour racine un chemin arbitraire issu de celle de son processus parent. Si ce paramètre n'est pas défini, ce qui est généralement le cas, c'est '/' qui est utilisé par défaut, en quelque sorte la racine de la racine du parent. La commande `chroot` permet de lancer une commande et dans la foulée de définir ce paramètre pour le processus engendré.

La commande `chroot` peut donc avoir deux paramètres principaux : un chemin dans l'arborescence et une commande. Ainsi, si en tant qu'utilisateur `root` nous lançons `chroot /lapp /bin/bash`, voilà ce qui se passe :

1. La commande `chroot` engendre un processus qui a pour racine la même que celle de son processus parent, généralement le / (mais rien n'empêche de faire des poupées russes...).
2. En interne, `chroot` appelle la fonction kernel `chroot("/lapp")`. Le kernel va donc modifier la valeur de la racine pour ce processus et lui associer la valeur `/lapp`.
3. Le processus de `chroot`, exécute la commande passée en 2nd paramètre, `/bin/bash`. Comme la racine de `chroot` a été changée, c'est en quelque sorte bien le `/bin/bash` à partir de la nouvelle racine `/lapp` qui va être exécuté.
4. Cette exécution débouche sur la création d'un processus fils de celui du `chroot` qui hérite donc de cette nouvelle racine.
5. Tout ce que `/bin/bash` lancera par la suite héritera de cette nouvelle racine jusqu'à ce que l'on tape `exit` qui mettra fin au processus `/bin/bash`, et par domino à celui du `chroot` qui a permis son lancement.

Le premier constat que nous pouvons tirer de cela est qu'il faut que notre dossier `/lapp` contienne un `bin/bash`, mais aussi toutes les bibliothèques dont a besoin `bash` pour fonctionner, ainsi que tous les dossiers systèmes qui lui sont nécessaire et qu'il s'attend à trouver comme sous une "vraie" racine. Ainsi la seule problématique posée sera le peuplement de notre nouvelle racine.

Le second constat est qu'un `chroot` **n'est pas une virtualisation**. Une virtualisation définit des conteneurs bien hermétiques avec leur mémoire propre, leur espace disque propre, leurs périphériques propres, etc. Le `chroot` ne fait rien de tout cela. La mémoire, le CPU, les périphériques (réseau, disque, écran, etc) sont partagées par l'environnement racine et `chrooté`. C'est d'ailleurs pour cela qu'un `chroot` est aussi rapide que si tout était lancé sur la racine principale.

`Chroot` ne fait donc que "faire croire" à un processus qu'il travaille sous la "vraie racine" alors qu'il n'est que dans un sous-dossier. Ainsi la seule garantie que fournit le `chroot` est l'**étanchéité des fichiers** dans le sens enfant-parent. C'est pour cet aspect que le `chroot` est d'ailleurs généralement utilisé, pour "mettre en prison" des applications critiques (serveur web, serveur mail, etc.) de sorte à ce que si une vulnérabilité permet à un cracker de casser le serveur, il ne puisse atteindre les fichiers se trouvant hors du `chroot` et ainsi limiter les dégâts. A noter que ce billet **ne traite pas** de cette utilisation critique du `chroot` et que d'un point de vue général il existe nombre d'exploits permettant de s'échapper de cette prison.

Malgré ses limitations, le `chroot` est juste parfait pour une utilisation : la création d'un environnement de test et/ou d'intégration pour des applications WEB. Il permet en effet très facilement de créer une configuration LA(P|M)P (Linux-Apache-Postgres/MySQL-PHP), facile à maintenir, facile à recréer, testable de la machine principale sans modification particulière comme si tout était installé "localement".

### Mise en oeuvre

Notre but est donc ici de créer un environnement `chrooté` comprenant un Linux de base, Apache, Postgresql et PHP. La première étape est donc de créer notre future racine. Je le fait en `/lapp` mais cela pourrait être virtuellement n'importe où, y compris sur le dossier d'une clef mémoire ou sur un espace volatile type `tmpfs`.

```
|| sudo [1] mkdir [2] /lapp
```

Maintenant vient le moment de peupler notre racine. Là, il y a trois approches possibles.

Nous avons l'approche dite "bourrine" consistant à dupliquer sauvagement son environnement principal :

```
| cp [3] -a /usr /bin /lib /sbin /var /lapp
```

L'inconvénient est que c'est souvent très volumineux et peu adapté à un environnement de test contrôlé dans la mesure où des tas de choses inutiles sont injectées dans la nouvelle racine.

Ensuite il y a l'approche "minimaliste" qui cherche à ne copier que ce qui est nécessaire. Cela peut se faire à la main avec la commande `ldd` ou beaucoup plus simplement, comme me l'a indiqué Guyou, avec les outils [makejail](#) [4] ou [chrootbin](#) [5]. Mais cette technique est plus adaptée au lancement d'un serveur dans une prison chroot qu'à notre besoin.

Enfin nous avons l'approche "raisonnable" consistant à utiliser les outils fournis par les debian's et autres mandriva's qui permettant d'installer tout simplement sur une racine arbitraire une distribution complète. C'est clairement dans notre cas cette approche qui s'impose.

Sous mandriva (et distributions dérivées), l'installation d'un linux de base sur une racine se fait simplement par la commande suivante :

```
| sudo [1] urpmi basesystem urpmi locales-fr --root /lapp
```

Simple et diablement efficace. Comme vous l'aurez noté, ce sont trois paquets qui sont ici installés: le linux de base, la langue française et le système de gestion de paquets. Ce dernier point nous offrira beaucoup de souplesse car au sein même du chroot il sera du coup possible d'installer, désinstaller ou même mettre à jour la distribution.

A noter que ce n'est pas du "one shot", il est toujours possible d'ajouter des paquets à la racine "de l'extérieur" par :

```
| sudo [1] urpmi un_paquer --root /lapp
```

Les distributions basées sur debian disposent elle aussi d'une méthode très efficace pour créer une racine minimale en utilisant cette fois la commande `debootstrap` .

```
| sudo [1] debootstrap --arch i386 --include=locales-all sid /lapp http://ftp  
[6].fr.debian.org/debian
```

Vous pouvez remplacer `sid` par toute distribution debian disponible (etch, sarge, etc..). Pour le reste cela fonctionne comme la méthode précédente à la différence que la gestion de paquet n'est pas optionnelle et vous aurez donc automatiquement accès à `apt-get` dans le chroot.

Un détail intéressant est que la commande `debootstrap` est aussi disponible sous Mandriva et vous permet sans aucun problème, d'installer une racine de debian dans une mandriva.

Notre racine étant maintenant peuplée et les utilitaires que nous avons utilisé ont normalement pris soin de créer à la "racine" les dossier vitaux `tmp` , `var` , `dev` , `sys` et `proc` . Si ce n'est pas le cas, faites le vous-même.

Maintenant comme vous le savez, certains dossiers de Linux ne sont pas de "vrais" dossier mais une vue sur des variables du système. C'est le cas de `/proc` , `/sys` et `/dev` . Si nous lançons maintenant notre nouvelle racine, ces dossiers seraient donc vides provoquant des erreurs sur certaines fonctions. Il nous faut donc avant monter ces dossiers. Pour ce faire, nous allons simplement les "relier" (`mount --bind`) aux mêmes dossiers de la vraie racine.

```
| sudo [1] mount [7] -bind /dev /lapp/dev  
| sudo [1] mount [7] -bind /proc /lapp/proc  
| sudo [1] mount [7] -bind /sys /lapp/sys
```

Concernant le réseau dans un chroot, la configuration sera la même que celle de l'environnement parent. Tout fonctionnera donc sans avoir à faire quoi que ce soit. En revanche il est important de vérifier que la résolution de noms se fait correctement. Pour cela nous allons simplement recopier le fichier `/etc/resolv` :

```
| cp [3] /etc/resolv.conf /lapp/etc/resolv.conf
```

Maintenant tout est en place, c'est le moment du lancement.

```
| sudo [1] chroot [8] /lapp /bin/bash
```

Voilà, une fois ceci exécuté, nous sommes "prisonnier" de la nouvelle racine et toutes les commandes qui suivent sont contrainte dans cet espace. Pour en sortir il suffit de faire un "exit". Comme vous le voyez tout fonctionne, vous pouvez utiliser le réseau, installer des paquets, etc. C'est ce que nous allons faire pour mettre en place notre environnement de test :

```
| urpmi apache-mod_php postgresql-server
```

Ceci fait, vous pouvez constater que votre système fonctionne en lançant les services :

```
| /etc/init.d/httpd start  
| /etc/init.d/postgresql start
```

Maintenant, en allant, sur votre environnement principal aidé d'un navigateur web sur l'adresse `http://localhost`, vous devriez avoir accès à la page de test du serveur Apache. Pour le reste, à vous de jouer...

## Automatisation

Comme vous l'aurez constaté, le chroot n'effectue pas, et heureusement, un démarrage réel du linux installé à la nouvelle racine. Il se contente d'exécuter `/bin/bash` ce qui vous oblige de votre côté à lancer les services à la main.

Pour rendre cela beaucoup plus simple à utiliser sur une base quotidienne, le plus propre reste de créer un service dédié au lancement de notre racine qui va créer le montages et lancer apache, postgres, etc... Son arrêt fera l'inverse :

```
#!/bin/sh

# chkconfig: 345 56 50
# description: This startup script launches Chroot Starter

### BEGIN INIT INFO
# Provides: lappd
# Required-Start:
# Required-Stop:
# Default-Start: 345
# Short-Description: Chroot Starter
# Description: This startup script launches Chroot Starter
### END INIT INFO

PATH=/usr/sbin:/usr/bin:/sbin:/bin
NAME=lapp
DESC="Chroot Starter for $NAME"
ROOT=$NAME

# Source function library.
. /etc/init.d/functions

case "$1" in
  start)
    if [ ! -z "$(mount | grep $ROOT)" ] ; then
      gprintf "$NAME is already started\n"
      exit 3
    fi

    gprintf "Starting chroot for %s" "$NAME"
    mount [7] --bind /proc $ROOT/proc
    mount [7] --bind /dev $ROOT/dev
    mount [7] --bind /sys $ROOT/sys
    chroot [8] $ROOT /init.sh [9] start
    echo_success
    echo
    ;;

  stop)
    if [ -z "$(mount | grep $ROOT)" ] ; then
      gprintf "$NAME is already stopped\n"
      exit 3
    fi

    gprintf "Stopping chroot for %s" "$NAME"
    chroot [8] $ROOT /init.sh [9] stop
    umount [10] $ROOT/proc
    if [ $? -ne 0 ] ; then
      echo_failure
      echo
      exit 3;
    fi
    umount [10] $ROOT/dev
    if [ $? -ne 0 ] ; then
      echo_failure
```

```

    echo
    exit 3;
fi
umount [10] $ROOT/sys
if [ $? -ne 0 ] ; then
    echo_failure
    echo
    exit 3;
fi
echo_success
echo
exit 0
;;

restart|force-reload)
    $0 stop
sleep [11] 1
    $0 start
;;
esac
exit 0

```

*/etc/init.d/lapp*

Ensuite dans notre racine il faut rajouter, et rendre exécutable, un fichier `/init.sh` contenant le démarrage/arrêt des services `httpd` et `postgresql`, et qui prendra en paramètre `start` ou `stop` :

```

#! /bin/sh

/etc/init.d/httpd $1
/etc/init.d/postgresql $1

```

*/init.sh*

Après, vous pouvez rendre le lancement de ce service automatique (avec `chkconfig` ou l'outil dédié à cela pour votre distribution), ou faire cela à la main :

```

# lancement du service
gaston$ sudo /etc/init.d/lapp start
Starting chroot for lapp [ OK ]

# ouverture d'une session sur la racine
gaston$ sudo chroot /lapp /bin/sh
# ...
# exit

# extinction du service
gaston$ sudo /etc/init.d/lapp stop
Starting chroot for lapp [ OK ]
gaston$

```

*exemple d'utilisation*

Personnellement j'ai mis ce service en démarrage manuel et ajouté un profile dans `gnome-terminal` qui me lance directement la commande `chroot /lapp /bin/sh`.

## Conclusion

La commande `chroot` ne manque donc pas d'intérêt en restant très simple à mettre en oeuvre si on en saisi bien les fondamentaux. C'est une bonne alternative à la "lourdeur" de la virtualisation et permet de créer à une vitesse record des environnements de développement rapides, efficaces et non-polluant pour le reste du système.

<http://artisan.karma-lab.net/node/1055>  
(C) artisan numerique - CC BY-SA

### Liens:

- [1] <http://pwet.fr/man/linux/commandes/sudo>
- [2] <http://pwet.fr/man/linux/commandes/mkdir>
- [3] <http://pwet.fr/man/linux/commandes/cp>
- [4] <http://www.floc.net/makejail/>
- [5] <http://freshmeat.net/projects/chrootbin/>
- [6] <http://pwet.fr/man/linux/commandes/ftp>

- [7] <http://pwet.fr/man/linux/commandes/mount>
- [8] <http://pwet.fr/man/linux/commandes/chroot>
- [9] <http://pwet.fr/man/linux/commandes/sh>
- [10] <http://pwet.fr/man/linux/commandes/umount>
- [11] <http://pwet.fr/man/linux/commandes/sleep>