Administration Systèmes & Réseaux

Installation et gestion des programmes

May 3, 2015

Houcemeddine HERMASSI

houcemeddine.hermassi@enit.rnu.tn

École Nationale d'Ingénieurs de Carthage ENI-CAR
Université Carthage
Tunisie

Plan de cour

Les bases de la compilation

Le gestionnaire des packets

Les bases de la compilation

Rechercher et installer un package individuel

Installer depuis les sources

Gérer les bibliothèques partagées

Notion de package

Contrairement à d'autres systèmes d'exploitation, il n'est pas courant sur Linux et Unix en général de disposer de logiciels fournis avec un programme l'installation interactif (pas de install.exe). Certains éditeurs proposent des scripts d'installation et bien souvent ceux-ci se contentent de décompresser et de désarchiver quelques fichiers.
Avec Linux, il est très classique de disposer des divers produits, outils, mises à jour, etc. sous forme de paquetages (packages). Un package est un fichier (parfois gros) qui contient le produit à installer et des règles. Ces règles peuvent être multiples :

- Gestion des dépendances : le produit ne pourra être installé que si les produits qu'il utilise lui-même sont déjà présents.
- Pré-installation : des actions sont à prévoir avant de pouvoir installer le produit (changer des droits, créer des répertoires, etc.).
- Post-installation : des actions sont à prévoir après l'installation du produit (paramétrage d'un fichier de configuration, compilation annexe, etc.).

Sur Red Hat, Fedora, SuSE, Mandriva et quelques autres distributions le format de package par défaut est le RPM (Red Hat Package Manager). Sous Debian, Knoppix, Kaella, Ubuntu, c'est le format DPKG (Debian Package). Outre le format, ce sont surtout les outils qui les différencient.
Le fait de disposer des informations de dépendances permet d'obtenir des outils performants qui peuvent seuls les résoudre en cascade. En installant un package, l'outil pourra installer toutes les dépendances nécessaires. On peut parfois spécifier plusieurs emplacements (repositories) pour ces packages, soit locaux (disque dur, CD-Rom, DVD, etc.) soit distants (http, ftp, etc.).

la commande dpkg

La commande **dpkg** est le pendant de rpm pour les distributions Debian et dérivées, dont Ubuntu. Elle fait la même chose, ou presque, que rpm. Les packages Debian portent une extension **.deb** pour les reconnaître et disposent des mêmes informations et moyens qu'un package rpm. La commande **dpkg** est chargée de l'installation, la création, la suppression et la gestion des paquets Debian.
La base de données **dpkg** est généralement placée dans `/var/lib/dpkg"`. Les fichiers qui y sont présents sont au format texte. Cependant n'éditez pas les fichiers à la main. Le fichier `/var/lib/dpkg/status"` contient l'intégralité des packages connus par dpkg avec leur état.

```
$grep Package: /var/lib/dpkg/status | grep glibc"
```

Dpkg dispose d'une interface graphique, GDebi, qui permet d'éviter l'utilisation de la ligne de commande.

Installation

L'option `-i"`, ou `-install"`, installe le ou les packages passés comme argument.

```
# dpkg -i monpaquet.deb "
```

Notez que comme rpm, dpkg ne gère pas seul les dépendances. S'il manque des dépendances, la commande vous en informera. Dans ce cas, vous devez installer les dépendances de la même manière avant d'installer votre package.

Vous pouvez demander l'installation de tous les packages présents au sein d'une arborescence avec le paramètre `-R`, pour récursif. Dans ce cas, indiquez comme argument un nom de répertoires : tous les packages présents dans le répertoire et ses sous-répertoires seront installés.

```
# dpkg -R folder "
```

Mise à jour

La mise à jour s'effectue de la même manière que l'installation, avec le `-i"`. Si vous installez un package déjà présent, **dpkg** en effectue une mise à jour. Ainsi, une installation ou une mise à jour respectent la méthodologie suivante :

- Extraction des fichiers de contrôle du nouveau paquet.
- Quand une ancienne version du même paquet est déjà installée, exécution du script pré-supression de l'ancien paquet.
- Lancement du script de préinstallation s'il est fourni par le paquet.
- Dépaquetage des nouveaux fichiers et sauvegarde des anciens pour pouvoir les restaurer en cas de problème.
- Si une ancienne version du paquet est déjà installée, exécution du script de post-suppression de l'ancien paquet.
- Configuration du paquet.

Si vous souhaitez mettre à jour un package uniquement s'il est déjà installé, vous devez tout d'abord vérifier s'il est installé. Si un package est installé il commence par `ii` dans la liste :

```
# (dpkg -l zip | grep ii >/dev/null) && echo PRESENT || echo ABSENT "
# (dpkg -l slapd | grep ii >/dev/null) && echo PRESENT || echo ABSENT"
```

Pour mettre à jour un paquet seulement s'il est présent utilisez ce genre de ligne de commande, ou un équivalent de votre cru :

```
# (dpkg -l zip | grep ii >/dev/null) && dpkg -l zip.deb "
```

La suppression

La suppression d'un package s'effectue avec le paramètre `-r` (en minuscule). Là encore, c'est à vous de gérer les dépendances.
La suppression d'un paquet effectue les étapes suivantes :

- Exécution du script de pré-suppression.
- Suppression des fichiers installés.
- Exécution du script de post-suppression.

```
# dpkg -r zip "
```

Tout est supprimé sauf les fichiers de configuration et ce, afin d'éviter une reconfiguration de l'outil si vous le réinstallez. Pour tout supprimer, y compris ces fichiers, précisez le paramètre -P (purge).

```
# dpkg -P apache"
```

Si vous remplacez le nom du package par les paramètres `-a"` ou `-pending"` les packages non installés (non dépaquetés) mais présents dans les informations de la base pour être purgés ou supprimés, sont effacés.
L'utilisation des options `-force-all"` et `-purge"` permet de forcer la désinstallation du paquet et de supprimer les fichiers de configuration associés.

```
# dpkg -force-all -purge nom_du_paquet "
```

Lister les paquets

Vous pouvez lister tous les packages Debian connus du système avec le paramètre -l" :

```
# dpkg -l "
```

Vous pouvez indiquer un motif particulier :

```
# dpkg -l "apt*" |grep ii"
```

Une autre méthode consiste à employer l'option –get-selections :

```
# dpkg -get-selections | grep kernel "
```

Trouver un paquet contenant un fichier

Le paramètre -S" suivi du nom d'un fichier (son chemin) permet de retrouver le paquet d'origine.

```
# dpkg -S /usr/bin/basename"
```

Lister le contenu d'un paquet

Le paramètre -L" liste le contenu du ou des paquets indiqués :

```
# dpkg -L coreutils | grep bin "
```

Principe

Que ce soit avec rpm ou dpkg le problème est le même : ces deux outils contrôlent les dépendances des packages pour autoriser ou non leur installation, mais ne les gèrent pas. Autrement dit, si une dépendance sur un package est absente, il ne sera pas installé, sauf si la dépendance est résolue :

- soit en installant auparavant les packages manquants
- soit en indiquant sur la même ligne le chemin de ces mêmes packages.

De même lors d'une mise à niveau il se pose un problème avec les fichiers de configuration. Que faut-il en faire ?
APT permet de résoudre ces problèmes en gérant les dépendances à votre place. APT signifie Advanced Packaging Tool. Au lieu de spécifier un paquet (local ou distant), il prend en charge des dépôts de packages situés sur un CD, un DVD, dans un répertoire local, sur une source distante sur Internet (ftp, http), etc.
Un dépôt contient un ensemble de packages qui dépendent soit les uns des autres, soit d'autres packages en provenance d'autres dépôts. APT peut gérer plusieurs dépôts, à divers endroits. Il se débrouille seul : lorsque vous installez un package, il installe aussi ses dépendances (s'il les trouve).

Configuration

Les dépôts sont indiqués dans le fichier `/etc/apt/sources.list". Le fichier suivant provient d'une installation Debian Etch où les dépôts contrib et non-free ont été rajoutés.`

```
$ cat /etc/apt/sources.list"
# etch"
```

Les dépôts sont préparés côté serveur dans une arborescence de répertoires. La commande **genbasedir** permet de créer un dépôt. La syntaxe d'une ligne du fichier sources.list est la suivante :

```
deb uri distribution composant1 composant2 ... "
```

- `uri" est le chemin vers la racine du ou des dépôts. Ce peut être une URL de type http ou ftp, mais aussi un chemin local (file), un CD-Rom ou DVD-Rom (CDrom), un chemin ssh, etc.`
- La distribution est, comme son nom l'indique, le nom de la distribution Debian. Ici c'est **etch**, mais il est possible de spécifier d'autres versions de la distribution (testing, sarge, etc.) pour pouvoir récupérer des packages d'autres dépôts, plus récents par exemple. L'architecture peut être précisée. Si elle ne l'est pas, APT se débrouille seul pour rajouter le suffixe nécessaire.
- Les composants sont les noms des dépôts pour la distribution donnée.

Configuration

En pratique, l'uri, la distribution et les composants permettent de reconstituer l'url complète d'accès au dépôt.

- Rendez-vous sur http://ftp.fr.debian.org/debian/.
- Cliquez sur le dossier appelé dists. Il contient la liste des distributions Debian.
- Dans dists, cliquez sur etch, le nom de la distribution actuelle.
- Dans etch, vous trouvez des dossiers contrib, main et non-free.

La ligne deb http://ftp.fr.debian.org/debian/ etch main contrib non-free correspond donc aux URLs :

- http://ftp.fr.debian.org/debian/dists/etch/main
- http://ftp.fr.debian.org/debian/dists/etch/contrib
- http://ftp.fr.debian.org/debian/dists/etch/non-free

Si vous continuez, par exemple en rentrant dans **main**, vous trouverez une série de dossiers suffixés en fonction de l'architecture de votre installation Linux. La machine VMWare de test est de type i386. Les packages binaires seront donc cherchés dans **binary-i386**.
Ne soyez pas surpris de ne pas trouver de packages dans ce dernier répertoire, mais des fichiers :

- `Release"` : description du dépôt.
- `Packages.gz"` : index des packages du dépôt, au format gzip.
- `Packages.bz2"` : la même chose au format bzip2.

Mise à jour de la base

Une fois vos dépôts configurés, vous devez mettre à jour la base de données locale de APT avec la commande **apt-get** et l'option **update**.

```
# apt-get update"
```

Mise à jour de la distribution (1/2)

Une fois les dépôts à jour, vous pouvez mettre à jour en une seule commande tous les packages installés sur votre distribution : APT vérifie si des packages plus récents sont disponibles dans les dépôts. Il se base pour cela sur la base de données locale. Si elle n'est pas à jour il est possible que certains des packages trop anciens ne soient plus présents.
Exécutez la commande **apt-get** avec l'option **upgrade**. APT vous informe que huit packages peuvent être mis à jour. Vous pouvez accepter ou refuser. Si vous acceptez, APT télécharge ces packages et leurs éventuelles dépendances, et les installe. Le processus peut être plus ou moins long selon le nombre de mises à jour et le type de support.

```
# apt-get upgrade"
```

Une autre possibilité est de faire une mise à jour profonde (appelée mise à jour distante). APT garde une certaine cohérence dans les packages lors de la mise à jour, notamment concernant la version de la distribution. Vous pouvez spécifier plusieurs distributions Debian dans vos dépôts. Mais même si une distribution est plus récente, un simple upgrade ne va pas transformer la vôtre en la toute dernière. Vous pouvez demander à APT de forcer la mise à jour vers la nouvelle distribution avec un dist-upgrade.

Mise à jour de la distribution (2/2)

Pour les besoins de cet ouvrage, les dépôts de la version de test Debian appelée lenny ont été ajoutés :

```
## lenny "
# security lenny "
```

Effectuez une mise à jour de la base. Notez qu'il a fallu agrandir le cache de APT pour ceci et nettoyer la base :

```
# apt-get update "
# apt-get clean "
# apt-get update "
```

puis

```
$ apt-get dist-upgrade "
```

Rechercher et installer un package individuel

La commande **apt-cache** permet de rechercher un package, par son nom ou son commentaire, au sein de la base de données locale APT.

```
# apt-cache search torrent "
```

La commande **apt-get** install xxx installe le package xxx :

```
# apt-get install vim-gtk "
```

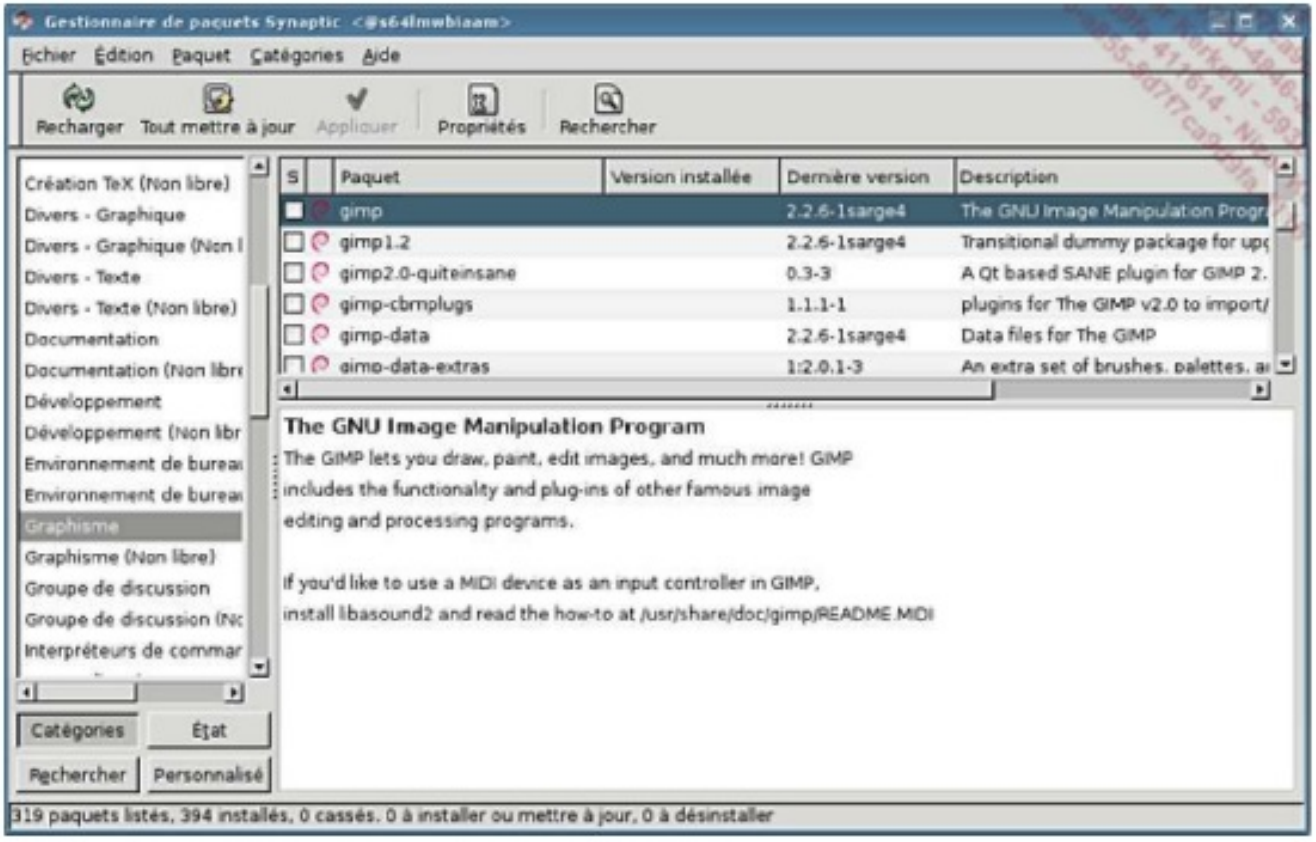
Deux options méritent d'être retenues :

- le **-s** pour la simulation : APT indique ce qu'il devrait faire, mais ne le fait pas.
- le **-f** pour « fix-broken » : APT tente de réparer les problèmes de dépendances comme il le peut (ajout de packages).

Gestionnaire APT
Client graphique

Client graphique

L'outil synaptic est un front-end : une interface graphique qui fait appel aux fonctions de APT. Il permet toutes les opérations proposées par APT tout en étant très convivial.



Installer depuis les sources
Obtenir les sources

Obtenir les sources

Il n'est parfois pas possible d'obtenir un logiciel ou une bibliothèque depuis un package pour sa distribution. Dans ce cas, il reste la solution de compiler et d'installer soi-même le produit depuis les sources.
Cela est possible pour une majorité de produits sous Linux, grâce aux avantages des logiciels libres et de la licence GPL telle que définie au premier chapitre. Tout logiciel libre est fourni avec ses sources. Il est donc possible de reconstruire soi-même le logiciel en le recompilant.
Une archive source est souvent récupérée sur divers sites Internet comme par exemple SourceForge. C'est une archive bien souvent compressée au format tgz (archive tar compressée avec gzip) ou tar.bz2 (archive tar compressée au format bzip2). Elle contient :

- le code source sous forme de fichiers .c, .h, .cpp, etc., selon le langage
- parfois un fichier Makefile permettant d'automatiser la compilation du produit
- souvent un fichier .configure permettant de générer le fichier Makefile en fonction de votre installation et de diverses options.

Installer depuis les sources
Pré-requis et dépendances

Pré-requis et dépendances

Pour compiler votre produit vous devez respecter quelques pré-requis :

- présence de l'outil make
- présence du ou des compilateurs nécessaires, notamment gcc
- présence des dépendances : bibliothèques, interpréteurs, etc.

Ce dernier point est très important. S'il manque une dépendance vous risquez divers problèmes :

- vous n'arriverez pas préparer les sources pour la compilation
- la compilation générera des erreurs
- le produit sera compilé mais avec des possibilités moindres
- le binaire résultant ne se lancera pas.

La commande **./configure** vous fournira les dépendances manquantes et leur version si c'est possible. Dans ce cas vous pouvez soit les installer depuis les packages de votre distribution, soit les installer depuis les sources.
Dans tous les cas, il n'y a pas besoin d'être root pour compiler votre logiciel. Cependant, selon la destination vous devrez passer root pour finaliser l'installation.

Installer depuis les sources
Exemple d'installation

Exemple d'installation

Vous allez compiler et installer le produit PDFedit qui permet d'éditer et de créer des fichiers PDF.

- Téléchargez-le depuis le lien suivant. La version testée est la version 0.4.1 :

http://sourceforge.net/project/showfiles.php?group_id=177354

```
$ ls -l pdfedit-0.4.1.tar.bz2 "
```

- Décompressez le fichier :

```
$ tar xvjf pdfedit-0.4.1.tar.bz2 "
```

- Déplacez-vous dans le dossier `pdfedit-0.4.1"` créé par la décompression :

```
$ cd pdfedit-0.4.1/ "
$ ls -l "
```

- Remarquez la présence du fichier configure qui est exécutable.

```
$ ./configure -help "
```

Une option importante de configure est **–prefix**. Elle définit l'emplacement de l'installation une fois le produit compilé. Par défaut le logiciel s'installe dans `/usr/local/"`.

- Exécutez **./configure** seul. Il vous informera des dépendances manquantes le cas échéant.

```
$ ./configure"
```

Installer depuis les sources
Exemple d'installation

Exemple d'installation

- Notez le bloc en gras : une bibliothèque est manquante. Vous avez deux possibilités :
  - installer la bibliothèque manquante (et son package de développement) soit depuis les sources, soit depuis le gestionnaire de packages de votre distribution
  - ne pas l'installer : elle n'est pas vitale. Cependant il n'y aura pas de support des polices Type1, ce qui est embêtant pour les PDF.
- Après avoir résolu, si vous le voulez, les dépendances, vous devez relancer la commande ./configure. La commande configure crée le fichier Makefile correspondant. Ce fichier contient l'ensemble des règles, chemins et options pour compiler le logiciel. La commande make y fait appel.
- Lancez la compilation avec la commande make. Il se peut que des avertissements apparaissent (lignes warning). Cela ne signifie pas forcément que le programme ne compilera pas ou ne marchera pas par la suite. De toute façon si la compilation produit des erreurs, elle s'arrêtera toute seule avec un message d'erreur du compilateur pouvant parfois (mais pas toujours) vous mettre sur la voie d'une solution.
- La compilation peut être plus ou moins longue selon le produit compilé. Pour PDFEdit, une machine à 1800MHz a mis environ 20 minutes.

```
$ make "
```

Installer depuis les sources
Exemple d'installation

Exemple d'installation

- La compilation s'étant terminée sans erreur, finissez en installant le produit avec make install. Attention, le produit va s'installer dans /usr/local/ ce qui nécessite les droits de l'utilisateur root.

```
$ su -c "make install" "
```

- Lancez le produit :

```
$ pdfedit"
```

Installer depuis les sources
Désinstallation

Désinstallation

La plupart des Makefile, en tout cas ceux générés par configure, permettent la désinstallation. Elle s'effectue par la commande make uninstall.

```
$ su -c "make uninstall"
```

Gérer les bibliothèques partagées
Principe

Principe

Une bibliothèque partagée est un fichier particulier qui contient une liste de fonctions, ou API, accessible à tout programme en ayant besoin sans avoir à les réécrire. À l'opposé de la bibliothèque statique, le programme accède dynamiquement aux fonctions qui sont placées dans un fichier à part. N programmes différents peuvent accéder aux fonctions proposées par la bibliothèque. Les bibliothèques regroupent des fonctions propres à un domaine ou un ensemble de domaines cohérents : traitement d'images, du son, de l'accès à une base de données, etc.
Un ensemble de fonctions proposées par une ou plusieurs bibliothèques partagées forme une **API**, Application Programming Interface, et sont parfois regroupées au sein d'un framework offrant une solution complète pour un domaine donné.
Un lien est établi entre le programme et une bibliothèque partagée lors de l'étape de l'édition des liens par l'éditeur de liens **ld**, lui-même appelé par le compilateur **gcc** avec l'option **-l<lib>**.
Une autre possibilité pour un programme est d'utiliser la fonction C **dlopen** qui ouvre une bibliothèque dynamique comme un fichier et qui accède aux fonctions qui y sont contenues avec des pointeurs de fonctions.
Si un programme dépend d'une bibliothèque partagée et que celle-ci est absente, le programme ne pourra plus fonctionner.
Sous Linux (et Unix en général) les bibliothèques partagées sont appelées des **Shared Objects** (so) dans le sens où il s'agit de fichiers objets sans bloc d'instruction **main**. Ils portent le suffixe **.so**.
Une bibliothèque peut disposer de plusieurs versions, pouvant être ou non compatibles, et la version peut être précisée lors de l'édition des liens, avec une version par défaut possible.

Lieu de stockage

Les bibliothèques partagées sont par convention placées dans des répertoires appelés lib :

- `/lib"` : bibliothèques systèmes de base, vitales
- `/usr/lib"` : bibliothèques utilisateur de base, non nécessaires au boot
- `/usr/local/lib"` : bibliothèques locales aux produits pour la machine
- `/usr/X11R6/lib"` : bibliothèques de l'environnement X Window
- `/opt/kde3/lib"` : bibliothèques de KDE

```
$ ls -l /lib "
```

La bibliothèque la plus importante du système est la bibliothèque C. Tous les programmes compilés sont liés à libc. Il suffit de supprimer ce fichier (une erreur de débutant) pour faire tomber tout le système.

```
$ ls -l libc.so.6 "
```

Les répertoires des bibliothèques contiennent beaucoup de liens symboliques. Ces liens sont là, entre autres, pour gérer les versions et la compatibilité entre les versions. Par exemple quand deux versions cohabitent :

```
$ cd /usr/lib"
$ ls -l libXm.* "
```

Quelles bibliothèques liées ?

La commande **ldd** permet de déterminer quelles sont les bibliothèques liées à un programme, et aussi si celles-ci sont présentes ou non.

```
$ ls -l /lib "
```

La bibliothèque la plus importante du système est la bibliothèque C. Tous les programmes compilés sont liés à libc. Il suffit de supprimer ce fichier (une erreur de débutant) pour faire tomber tout le système.

```
$ ldd pdfedit"
```

Prenez maintenant un cas où une bibliothèque est manquante (elle a été volontairement déplacée pour les besoins de la démonstration) :

```
$ ldd /usr/bin/esd "
```

La bibliothèque libesd.so.0 est manquante. Il est impossible de lancer le programme :

```
$ ./esd "
```

Configurer le cache de l'éditeur de liens

L'édition des liens avec une bibliothèque partagée est dynamique et se fait au moment de l'exécution du programme par le système à l'aide de la bibliothèque **ld.so**. Le binaire fournit le nom des bibliothèques à lier à l'exécution, mais pas le chemin. Les fonctions de ld.so déterminent en fonction de son nom la bibliothèque à utiliser parmi les chemins qu'elles connaissent.
Tout programme est lié à la bibliothèque ld.so ou plutôt **ld-linux.so** (ld-linux.so.2).
Le chargeur de liens ld.so recherche les bibliothèques dans plusieurs endroits dont, et dans cet ordre :

- les chemins précisés dans la variable d'environnement **LD_LIBRARY_PATH**. Les chemins sont séparés, comme pour PATH, par des ":"
- le contenu du fichier `/etc/ld.so.cache"` qui contient une liste compilée (format binaire) des bibliothèques trouvées dans les chemins prédéfinis
- les répertoires `/lib"` et `/usr/lib"`.

La recherche dans `/lib"` et `/usr/lib"` est implicite. De même, le fait de remplir la variable **LD_LIBRARY_PATH** n'empêche en rien la recherche des bibliothèques aux autres endroits si elle n'est pas dans un des chemins de la liste.
Pour éviter la mise en place d'une variable dont le contenu peut être difficile à manipuler, ld.so propose un cache que vous pouvez modifier vous-même. Le cache est construit depuis le contenu du fichier `/etc/ld.so.conf"` et de la commande **ldconfig**.
Ce fichier contient la liste des répertoires contenant les bibliothèques partagées :

```
# cat /etc/ld.so.conf"
```

Plutôt que de modifier ce fichier, un package ou vous-même pouvez décider de rajouter un fichier dans `/etc/ld.so.conf.d"` contenant le ou les chemins de vos nouvelles bibliothèques.
Il ne suffit pas de rajouter le chemin : vous devez regénérer le cache avec la commande **ldconfig**.

```
# ldconfig"
```

Configurer le cache de l'éditeur de liens

La commande **ldconfig** :

- met à jour le cache pour les chemins définis dans /etc/ld.so.conf et associés, ainsi que pour /usr/lib et /lib
- met à jour les liens symboliques sur les bibliothèques
- permet aussi de lister les bibliothèques connues dans le cache.

Option	Rôle
-v	Mode bavard : indique ce que ldconfig effectue
Ne reconstruit pas le cache	
-X	Ne met pas à jour les liens
-p	Liste le contenu du cache

Pour lister les bibliothèques connues de l'éditeur de liens :

```
# ldconfig -p"
```

Pour voir ce que ferait ldconfig mais sans rien mettre à jour :

```
# ldconfig -N -X -v "
```

Enfin pour mettre à jour et voir le résultat :

```
# ldconfig -v "
```

Merci pour votre attention!