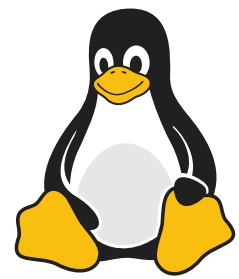
GNU/Linux

Compression et Installation de logiciels

Nicolas Delanoue

Université d'Angers - Polytech Angers

POLYTECH°
ANGERS



Définitions

- Une *archive* est un fichier qui contient d'autres fichiers.
- Une *archive tar* est un fichier qui met « bout-à-bout » tous les fichiers ; et conserve la structure des répertoires.

Remarque

Il est possible de compresser une archive `tar`, on parle alors d'*archive tar compressée*.

Type de compression libre - (algorithme & utilitaires)

- `gzip` (GNUzip, extension `.gz`)
- `bzip2` (de J. Seward, extension `.bz2`)

Mécanisme de compression en deux étapes :

1. on crée une archive tar,
2. on compresse cette archive.

Exemple

En général, on fait les deux d'un seul coup :

```
tar -czvf dossier.tar.gz dossier/
```

Explications

- c créer archive,
- z compression gzip,
- j compression bz2,
- v mode bavard,
- f utiliser le fichier qui suit.

Décompression

On se place dans le répertoire où on veut extraire et on utilise à nouveau l'outil `tar`.

```
tar -xjvf fichier.tar.bz2
```

Explications

- x extraire archive,
- z compression gzip,
- j compression bz2,
- v mode bavard,
- f utiliser le fichier dont le nom suit

Compatibilité des formats de fichiers Windows & Linux

On pourra aussi utiliser le format `zip` avec les commandes du type :

Compression : `zip -r dossier.zip dossier/`

Décompression : `unzip dossier.zip`

Qui fournit des logiciels ?

Les logiciels sont fournis par :

- l'éditeur de la distribution,
- des organisations à but non lucratif,
- des individus isolés,
- mais aussi des entreprises.

Ils sont fournis sous la forme d'archives installables, appelées paquets (package).

Comment sont-il classiquement distribués ?

Les logiciels sont fournis dans une ou plusieurs paquets. Leur format :

- .rpm (standard RedHat)
- .deb (standard Debian => Ubuntu)
- .tar.gz ou .tar.bz2
- ...

C'est l'équivalent des .exe, .msi, .zip de Windows.

Définition

Un *paquet* est une archive contenant :

- des binaires (exécutables ou librairies),
- ou bien des sources (à compiler),
- des fichiers de configuration,
- de la documentation,
- une liste de dépendances,
- des instructions d'installation, sous forme de scripts.

Définition

L'outil `dpkg` (debian package) est le nom du programme de gestion des paquets.

Utilisation de dpkg

Exemple de commandes	Signification	Explication
`dpkg -i lepaquet.deb`	install	installer un paquet
`dpkg -r lepaquet`	remove	supprimer un paquet
`dpkg -l`	list	afficher la liste des paquets
`dpkg -L lepaquet`	List files	afficher la liste des fichiers du paquet
`dpkg -S lefichier`	Search	afficher le paquet qui fournit le fichier passé en argument.

Remarques

- Les paquets portent l'extension `.deb`,
- L'outil `dpkg` n'utilise pas les dépôts ; il faut avoir le fichier `.deb`,

Définition

Une *dépendance* est une chose qui est requis par le paquet.

Définition

Une *dépendance* est une chose qui est requis par le paquet.

Remarques

- Avant d'installer un paquet, il faut donc installer les paquets dépendants avant.
- A l'inverse, quand on désinstalle un paquet, on peut désinstaller tous les paquets qui dépendent de lui.

Définition

Une *dépendance* est une chose qui est requis par le paquet.

Remarques

- Avant d'installer un paquet, il faut donc installer les paquets dépendants avant.
- A l'inverse, quand on désinstalle un paquet, on peut désinstaller tous les paquets qui dépendent de lui.

- Les dépendances sont un casse-tête.
- Les distributions proposent des dépôts :
  - des emplacements donnat accès à une collection de paquets (avec toutes leurs dépendances),
  - soit sur DVD,
  - soit sur des sites web.

Principe de fonctionnement de l'outil `apt`

La commande `apt` télécharge les paquets `.deb`, vérifie les dépendances puis, s'appuie sur `dpkg` pour les installer.

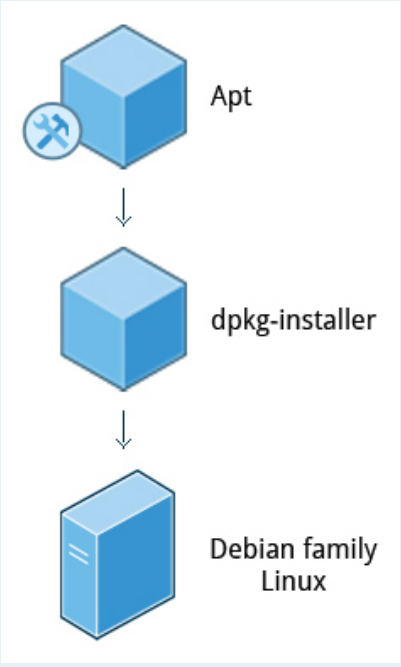


FIGURE – apt s'appuie sur dpkg.

Utilisation de `apt-get`

- `apt-get` permet, entre autres :
  - d'installer : `$ apt-get install lepaquet`
  - de désintaller : `$ apt-get remove lepaquet`
- `apt-get` utilise pour cela les dépôts configurés dans Synaptic et localisés dans le fichier `/etc/apt/sources.list`
- Pour mettre à jour l'état des paquets : `apt-get update`
- Pour mettre à jour les paquets : `apt-get upgrade`

Les méthodes d'installation

Méthode	**Outil**	**Avantages / inconvénients**
Interface graphique	Logithèque Ubuntu	+ simple + ergonomique – lent – peu de maitrise sur les détails
Interface graphique	Synaptic Ubuntu	+ simple + bonne vue d'ensemble – lourd – lent
Gestion haut niveau	`apt-get`	+ simple – assez lent
Gestion bas niveau	`dpkg`	+ rapide – nécessite le fichier-paquet – problèmes de dépendances

Compilation des sources	`./configure`
	`make`
	`make install`
		+ binaires parfaitement adaptés au système
		– problèmes de dépendances
		– compilation pas évidente

L'outil : `apt-cache`

`apt-cache` permet d'interroger les dépôts au sujet des paquets.

- Trouver les paquets contenant le mot-clé :
  `apt-cache search la_cle`
- Obtenir des infos sur un paquet :
  `apt-cache show lepaquet`
- Obtenir les dépendances de un paquet :
  `apt-cache depends lepaquet`

Définition

La *compilation*, c'est transformer le code source en code binaire.

Pourquoi compiler des sources ?

- pour garantir l'adaptation des binaires aux systèmes,
- pour modifier les options de compilation,
- pour modifier les sources,
- parce qu'on n'a pas toujours le choix (ex : pilote de carte graphique propriétaire).

Démonstration

```
gcc main.c -o prog.exe
```

Etapes de l'installation via compilation

1. récupérer l'archive avec les sources, utiliser `firefox` ou `wget`,
2. l'extraire dans `/usr/local/src` (utiliser `tar ...`)
3. générer un `Makefile` (utiliser `./configure`),
4. compiler (utiliser `make`),
5. installer (utiliser `make install`).

La compilation - L'outil `Makefile`

Dans un projet constitué de nombreux fichiers source, la compilation est ardue.
Un fichier `Makefile` permet :

- de compiler le projet,
- d'une façon simple et standard,
- en évitant les commandes répétitives,
- en limitant les risques de mauvaises manipulations.

make

En fonction de la cible souhaitée, `make` appelle des commandes indiquées dans le fichier `Makefile` du répertoire courant.

Structure d'un fichier `makefile`

```
cible1 : dependance1 dependance2
        commandes pour fabriquer cible1
        à partir de dependance1 et dependance2

cible2 : dependance3
        commande pour fabriquer
        cible2 à partir de dependance3
```

Remarques

Une dépendance peut être une cible.

Quelques cibles courantes

- pas d'argument, c'est la commande par défaut.
  En général, cela correspond à tout compiler.
- avec l'argument `install`, cela correspond à installer les binaires.

Remarques

Dans certains cas, il n'y a pas de `Makefile`.
Dans ce cas, il existe un fichier `configure`. C'est un script qui

- vérifie l'environnement de compilation,
- génére un `Makefile` approprié.

On le lance par : `./configure`, on peut ensuite exécuter `make`.