# .rpm, .deb, tout sur les paquets sous Linux !

S. Aicardi

Journées Mathrice, Clermont-Ferrand, 5-7 Octobre 2010

# Comment sont distribués les logiciels libres ?

Les développeurs mettent à disposition les sources d'une ou de plusieurs versions du logiciel sur un site web (soit spécifique, soit communautaire comme SourceForge).

Ces sources sont téléchargeables sous la forme d'une archive `.tar.gz` ou `.tar.bz2`.

Elles sont prévues pour être compilées sous un très grand nombre d'architectures (Linux, BSD, MacOS, Cygwin, AIX, HPUX, etc.)

# Exemple : HelloWorld

La société HelloWorld inc développe un service réseau qui répond “Hello World!” sur le port 12345.

Les sources sont disponible sur le site `http://www.helloworld.com`.

La derniere version disponible est la 2.7.1.8, téléchargeable à l'adresse `http://www.helloworld.com/downloads/helloworld-2.7.1.8.tar.gz`.

# Comment installer un logiciel libre ?

Un fichier `README` décrit sommairement le logiciel et un fichier `INSTALL` qui explique la démarche à suivre pour l'installation.

Le plus souvent, l'installation se fait en trois étapes :

```
./configure
make
sudo make install
```

# HelloWorld : installation manuelle

```
$ tar xzf helloworld-2.7.1.8.tar.bz2
$ cd helloworld
$ ./configure
$ make
$ sudo make install
```

Après ces opérations, on obtient deux fichiers `/opt/helloworld/helloworldd` et `/opt/helloworld/helloworldd.8`.

# Les limites de l'installation manuelle

- L'installation se fait dans des répertoires divers (`/opt`, `/usr/local`, etc.)
- La configuration et l'intégration au système reste à faire.
- La compilation et l'exécution dépendent de versions spécifiques d'autres logiciels.
- La désinstallation doit se faire à la main.
- Le suivi de version doit se faire par ses propres moyens.

# Rôle des distributions

Les distributions Linux dispensent l'utilisateur de ces tâches. Elles :

- fournissent les logiciels déjà compilés,
- installent les logiciels, les librairies, les fichiers de configuration de manière uniforme,
- fournissent une configuration opérationnelle,
- gèrent les dépendances et le suivi des versions.

# Rôle des distributions

Les distributions Linux dispensent l'utilisateur de ces tâches. Elles :

- fournissent les logiciels déjà compilés,
- installent les logiciels, les librairies, les fichiers de configuration de manière uniforme,
- fournissent une configuration opérationnelle,
- gèrent les dépendances et le suivi des versions.

grâce à des paquets.

# Formats de paquets

Les deux plus courants sont :

- RPM utilisé par notamment par RedHat et ses déclinaisons/clones, Suse et Mandriva. C'est le format de paquet recommandé par la Linux Standard Base.
- DEB utilisé par Debian et les distributions dérivées dont Ubuntu.

D'autres formats sont utilisés : .tgz par Slackware, .tar.xz par Arch Linux, ebuild par Gentoo, etc.

# Contenu d'un paquet

Quel que soit son format, un paquet contient :

- une archive contenant les fichiers à installer,
- des métadonnées : nom, numéro de version, informations, dépendances, sommes de contrôle, signature, etc.
- des scripts complémentaires d'installation ou de désinstallation.

# Anatomie d'un fichier .deb

Nom du fichier : `paquet_version-révision_arch.deb`

C'est une archive `ar` contenant trois fichiers :

- `debian-binary` : la version du format .deb, actuellement 2.0.
- `control.tar.gz` : les métadonnées de contrôle
- `data.tar` souvent compressée en `.gz`, `.bz2` ou `.lzma`

# Anatomie d'un fichier .deb

L'archive `control.tar.gz` contient au-moins :

- `md5sums`,
- `control`.

Elle peut également contenir :

- `preinst`, `postinst`, `prerm`, `postrm` (scripts pré/post-(dés)instal- lation),
- `conffiles` (liste des fichiers de configuration modifiables par l'utilisateur),
- `config`, `templates` (pour gérer la configuration du paquet avec `debconf`).

# Anatomie d'un fichier .deb : exemple de fichier control

```
Package: helloworld
Version: 2.7.1.8-1
Architecture: i386
Maintainer: Stephane Aicardi <aicardi@math.polytechnique.fr>
Installed-Size: 62
Depends: libc6 (>= 2.4)
Section: net
Priority: extra
Homepage: http://www.helloworld.com/
Description: The Helloworld network daemon
 The helloworldd program is an operating system daemon that
 answers "Hello World!" to any connection attempt.
```

# Anatomie d'un fichier .deb : vrai exemple de fichier `control`

```
Package: openssh-server
Source: openssh
Version: 1:5.1p1-5
Architecture: amd64
Maintainer: Debian OpenSSH Maintainers <debian-ssh@lists.debian.org>
Installed-Size: 812
Depends: libc6 (>= 2.7-1), libcomerr2 (>= 1.01), libkrb53 (>= 1.6.dfsg.2),
  libpam0g (>= 0.99.7.1), libselinux1 (>= 2.0.59), libssl0.9.8 (>= 0.9.8g
 -9), libwrap0 (>= 7.6-4~), zlib1g (>= 1:1.1.4), debconf (>= 1.2.0) |
 debconf-2.0, libpam-runtime (>= 0.76-14), libpam-modules (>= 0.72-9),
 adduser (>= 3.9), dpkg (>= 1.9.0), openssh-client (= 1:5.1p1-5), lsb-base
  (>= 3.2-13), openssh-blacklist, procps
Recommends: xauth, openssh-blacklist-extra
Suggests: ssh-askpass, rssh, molly-guard
Conflicts: rsh-client (<< 0.16.1-1), sftp, ssh (<< 1:3.8.1p1-9), ssh-krb5
 (<< 1:4.3p2-7), ssh-nonfree (<< 2), ssh-socks, ssh2
Replaces: openssh-client (<< 1:3.8.1p1-11), ssh, ssh-krb5
Provides: ssh-server
Section: net
Priority: optional
Description: secure shell server, an rshd replacement
 This is the portable version of OpenSSH, a free implementation of
 the Secure Shell protocol as specified by the IETF secsh working
 group.
 .
 Ssh (Secure Shell) is a program for logging into a remote machine
```

# Anatomie d'un fichier .rpm

Nom du fichier : `paquet-version-révision.arch.rpm`

C'est un fichier binaire contenant quatre parties :

- une amorce (*lead*) sur 96 octets (destinée à `file`)
- des signatures (taille, signature PGP, condensats)
- les metadonnées (*header*) : nom du paquet, version, révision, taille, description, license, url, changelog, scripts, format de l'archive, type de compression de l'archive, etc.
- l'archive au format cpio compressé éventuellement en gzip, bzip2 ou xz. (NB : accessible par la commande `rpm2cpio`.)

L'extraction des données est très fastidieuse sans la librairie rpm ou les commandes en ligne.

Pour plus de détail sur le format des fichiers rpm, voir le site de rpm.org

# Base de données des paquets

Pour gérer le suivi des fichiers installés, les désinstallations, les mises à jour, le système doit maintienir une base de données des paquets installés contenant :

- le numéro de version,
- la liste des fichiers installés,
- les fichiers de configuration,
- les scripts de pré/post-(dés-)installation,
- les dépendances,
- etc.

# Base de données des paquets .deb

Pour les systèmes utilisant les .deb, la base de données est le répertoire `/var/lib/dpkg` qui contient :

- un fichier texte `status` qui contient la liste des paquets avec état, numéro de version, description, dépendances, etc.
- un répertoire `info` qui stocke pour chaque paquet des fichiers `paquet.list`, `paquet.md5sums`, `paquet.preinst`, etc.

# Base de données des paquets .rpm

Pour les systèmes utilisant les .rpm, la base de données est le répertoire `/var/lib/rpm` qui contient des fichiers au format Berkeley DB.

Le fichier `Packages` contient toutes les informations sur les paquets installés. Les autres fichiers sont des index permettant d'accélerer les requètes à la base de données. En cas de corruption, la commande `rpm --rebuilddb` permet de reconstruire les index à partir du fichier Packages.

Là encore, on ne peut y accéder que par les commandes `rpm` ou avec la `librpm`.

# Comment manipuler les paquets à bas niveau ?

| | RPM | DEB |
|---|---|---|
| Installation | `rpm -i pkg.rpm` | `dpkg -i pkg.deb` |
| Désinstallation | `rpm -e pkg` | `dpkg -r pkg` |
| Liste des paquets | `rpm -qa` | `dpkg -l` |
| Contenu d'un paquet installé | `rpm -ql pkg` | `dpkg -L pkg` |
| Contenu d'un fichier paquet | `rpm -qlp pkg.rpm` | `dpkg -c pkg.deb` |
| Provenance d'un fichier | `rpm -qf file` | `dpkg -S file` |
| Informations sur un paquet | `rpm -qi pkg` | `dpkg -p pkg` |

# Conversion de paquets

`alien` est l'outil de conversion de format de paquet.

Exemple d'utilisation :

```
alien --to-deb package.rpm
alien --to-rpm package.deb
```

# Dépôts

Un **dépôt de logiciels** est un espace de stockage de paquets disponible sur internet et maintenu à jour par une distribution.

L'accès aux dépôts de logiciels se fait avec un **gestionnaire de paquets**, tel que APT pour les distributions basées sur Debian et YUM pour les distributions basées sur RedHat.

Le gestionnaire de paquets simplifie l'installation d'un paquet avec toutes ses dépendances et le suivi des mises à jours.

# Gestion des depôts avec APT

Les dépôts sont définis dans le fichier `/etc/apt/sources.list`, soit dans le répertoire `/etc/apt/sources.list.d`.

Chaque dépôt se présente sous la forme d'une ligne :

```
deb     http://security.debian.org/ lenny/updates  main
deb-src http://security.debian.org/ lenny/updates  main
```

# Gestion des dépôts avec YUM

Les dépôts sont définis dans le fichier `/etc/yum.conf` ou dans le répertoire `/etc/yum.repos.d`.
Chaque dépôt se présente sous la forme suivante :

```
[epel]
name=Extra Packages for Enterprise Linux 5 - $basearch
#baseurl=http://download.fedoraproject.org/pub/epel/5/$basearch
mirrorlist=http://mirrors.fedoraproject.org/mirrorlist?repo=epel-5&arch=$basearch
failovermethod=priority
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL
```

# Format d'un dépôt APT

Le dépôt défini par la ligne :
`deb http://mondepot.org/ lenny main`
contient à l'adresse `http://mondepot.org/dists/lenny/main/`
l'arborescence suivante :

```
binary-amd64
binary-i386
sources
```

En outre, il contient à l'adresse
`http://mondepot.org/dists/lenny` un fichier `Release`
contenant les sommes de contrôles de tous les éléments de l'arborescence et un fichier `Release.gpg` qui est la signature du précédent.

# Format d'un dépôt APT

Chaque répertoire `binary-`*`arch`* contient au-moins un fichier `Packages.gz`. C'est un fichier texte gzippé de la forme suivante :

```
Package: xpdf
Priority: optional
Section: text
Installed-Size: 36
Maintainer: Hamish Moffatt <hamish@debian.org>
Architecture: all
Version: 3.02-1.4+lenny2
Replaces: xpdf-i (<= 0.90-8)
Depends: xpdf-reader, xpdf-utils, xpdf-common
Conflicts: xpdf-i (<= 0.90-8)
Filename: pool/updates/main/x/xpdf/xpdf_3.02-1.4+lenny2_all.deb
Size: 1270
MD5sum: 6a4da9738ca93522b57cafadb598ca65
SHA1: 412b9ac40836deab02e1de28a5601417bc0c7415
SHA256: e21ab043f15ce40b35d48ea8dd3152db735277b0c50953d6edefe35113c61a08
Description: Portable Document Format (PDF) suite
Tag: interface::x11, role::dummy, special::obsolete, use::viewing, works-with::text, wor

Package: xpdf-common
...
```

# Format d'un dépôt RPM

Un dépot RPM contient les répertoires `repodata` et `headers`.

Le répertoire `repodata` contient :

- primary.xml.gz : la liste des paquets avec leur description, somme de contrôle, URL et dépendances,
- filelists.xml.gz : la liste des fichiers installés,
- other.xml.gz : les informations autres dont le changelog,
- repodata.xml : les sommes de contrôle des fichiers précédents.

Le répertoire `headers` contient des fichiers `paquet-version-révision.arch.hdr` qui sont des versions gzippées de la partie header du fichier rpm correspondant.

# Comment manipuler les paquets grâce aux dépôts ?

| | YUM | APT |
|---|---|---|
| Installation | `yum install pkg` | `apt-get install pkg` |
| Désinstallation | `yum remove pkg` | `apt-get remove pkg` |
| Mise à jour base | | `apt-get update` |
| Mise à jour des paquets | `yum update` | `apt-get upgrade` |
| Mise à jour distrib. | `yum upgrade` | `apt-get dist-upgrade` |

# Pourquoi fabriquer des paquets ?

- diffuser un logiciel développé localement,
- faciliter le déploiement d'un logiciel non empaqueté sur un parc ou un cluster,
- faciliter le déploiement d'une version différente sans changer de distribution,
- permettre des installations automatiques.

# Structure d'un paquet source

À l'exception des paquets spécifiques à une distribution, un paquet source aura la structure suivante :

- une copie de l'archive compressée fournie par les développeurs du logiciel,
- des correctifs (*patchs*) apportés par la distribution,
- des fichiers de contrôle décrivant le logiciel, ses dépendances, sa compilation et son intégration au système

# Structure d'un paquet source .deb

Un paquet source debian sera constitué de trois fichiers :

- `paquet_version.orig.tar.gz` : l'archive d'origine
- `paquet_version-révision.dsc` : une fichier de description du paquet source (version, sommes de contrôles, signature)
- `paquet_version-revision.diff.gz` : la différence compressée entre l'archive d'origine et l'archive prête à être compilée

# Structure d'un paquet source .deb

En général, le fichier de différences se contente de créer dans l'archive d'origine un répertoire `debian` qui contient :

- `changelog`
- `control`
- `copyright`
- `rules`
- éventuellement un répertoire `patches`, des fichiers complémentaires comme `preinst`, `prerm`...

# Structure d'un paquet source .rpm

Un paquet source rpm est un fichier d'extension `.src.rpm`. La partie archive cpio contient l'archive d'origine du logiciel, des patchs éventuels et un fichier `.spec` qui contient les instructions de compilation et d'installation, la description et les dépendances.

Il s'installe avec la commande

```
rpm -i paquet-version-revision.src.rpm
```

comme un fichier `.rpm` standard, mais n'est pas ajouté à la base de données des paquets installés.

# Fabrication d'un paquet .rpm : environnement

Les variables utilisées par les outils de fabrication peuvent être définies dans le fichier `/.rpmmacros`.

Répertoire de base `%_topdir` (par défaut `/usr/src/redhat`) :

- `BUILD`
- `RPMS`
- `SOURCES`
- `SPECS`
- `SRPMS`

Quand on installe un fichier `.src.rpm`, le fichier `.spec` va dans `SPECS`, le reste va dans `SOURCES`.

# Fabrication d'un paquet .rpm : étapes

La fabrication d'un paquet .rpm à partir des sources passe par les étapes suivantes :

- réunir ou écrire fichier `.spec`, archive et patchs,
- `%prep` : préparer (`-bp`) le répertoire de compilation,
- `%build` : compiler (`-bc`) le programme,
- `%install` : installer (`-bi`) le programme dans un répertoire temporaire,
- fabriquer le paquet binaire (`-bb`) et le paquet source (`-bs`) à partir du répertoire temporaire.

Pour faire les quatre dernières étapes :

```
rpmbuild -ba fichier.spec
```

# Fabrication d'un paquet .rpm : le fichier .spec

```
Name:           helloworld
Version:        2.7.1.8
Release:        1
License:        GPL
URL:            http://www.helloworld.com/
Summary:        Hello World TCP daemon
Group:          Applications/Internet
Source:         helloworld-%{version}.tar.gz
Source1:        helloworldd.init
Patch0:         helloworld-2.7.1-DESTDIR.patch
BuildRoot:      %{_tmppath}/%{name}-%{version}-build

%description
The helloworldd program is an operating system daemon that answers "Hello
World!" to any connection attempt.

%prep
%setup -q -n helloworld-%{version}
%patch0 -p1 -b .DESTDIR

%build
make
```

# Fabrication d'un paquet .rpm : le fichier .spec

```
%install
make DESTDIR=${RPM_BUILD_ROOT} install
install -m 755 $RPM_SOURCE_DIR/helloworldd.init ${RPM_BUILD_ROOT}/etc/rc.d/init.d
 /helloworldd
echo "HELLOWORLD_PORT=12345" > ${RPM_BUILD_ROOT}/etc/sysconfig/helloworld

%post
service helloworldd start
/sbin/chkconfig --add helloworldd

%preun
service helloworldd stop
/sbin/chkconfig --del helloworldd

%files
%defattr(-,root,root)
%doc ChangeLog README INSTALL COPYING
%doc %{_mandir}/man8/helloworldd.8*
%{_sbindir}/helloworldd
%config /etc/rc.d/init.d/helloworldd
%config /etc/sysconfig/helloworld

%changelog
* Thu Sep 30 2010 aicardi@math.polytechnique.fr
- package created
```

# Fabrication d'un paquet .rpm : intégration à la distribution

Dans l'exemple du paquet `helloworld.rpm`, il a fallu :

- modifier le `Makefile` pour qu'il prenne en compte la variable `DESTDIR`,
- fournir un fichier de démarrage `/etc/rc.d/init.d/helloworldd`,
- prévoir un fichier de configuration dans le répertoire standard `/etc/sysconfig`,
- ajouter le démarrage automatique à l'installation et l'extinction à la désinstallation.

# Fabrication d'un paquet .rpm : si on est pressé

Si on dispose déjà d'un paquet source rpm et si on ne veut faire aucune modification sur ce paquet, on peut utiliser la commande :

```
rpmbuild --rebuild paquet.src.rpm
```

# Fabrication d'un paquet .deb : environnement

Pas de répertoires spécifiques à créer dans ce cas. Il suffit de décompresser l'archive initiale et de se placer dans le répertoire ainsi obtenu.

Il suffit alors de créer dans ce répertoire source un répertoire `debian` contenant au minimum les fichiers `copyright`, `changelog`, `control` et `rules`.

Si on part d'un paquet source, il suffit de le décompresser avec la commande `dpkg-source -x paquet.dsc`.

# Fabrication d'un paquet .deb : environnement

La commande `dh_make` facilite la création du répertoire `debian` en préparant des fichiers valides, ainsi que des canevas de fichiers de démarrage.

```
dh_make --copyright gpl -f ../paquet.tar.gz
```

Il suffit alors de parcourir et modifier le contenu du répertoire `debian`. Les fichiers `.ex` sont des exemples qui peuvent être supprimés ou renommés.

# Fabrication d'un paquet .deb

On peut alors fabriquer le paquet binaire et source avec la commande `dpkg-buildpackage`. Les paquets sont créés dans le répertoire parent du répertoire source.

La commande `lintian` permet de vérifier la conformité aux règles de packaging de debian du paquet ainsi créé.

Si on ne veut pas signer le paquet et vérifier la conformité aux règles de packaging de debian la commande `debuild -us -uc` est préférable.

On peut alors tester le paquet en l'installant `sudo debi`.

# Fabrication d'un paquet .deb : si on est pressé

Pour recompiler à partir des sources un paquet disponible dans apt

```
apt-get source paquet
sudo apt-get build-dep paquet
cd paquet-version
debuild -us -uc
sudo debi
```

# Fabrication d'un paquet .deb : pbuilder

Pour garantir une compilation et installation correcte sur toute plateforme, l'outil `pbuilder` permet de créer un environnement minimal et d'y compiler le paquet.

```
sudo pbuilder create
sudo pbuilder build paquet*.dsc
```

Par défaut, le répertoire de construction est `/var/cache/pbuilder`. Il est possible de spécifier un autre répertoire dans le fichier `.pbuilderrc` ou en ligne de commande avec les options `--basetgz`, `--buildplace`, `--buildresult`.

On peut ainsi produire des paquets pour différentes distributions, architectures.

# Fabrication d'un dépôt

Pour faire un dépot YUM, il suffit de rassembler les fichiers `.rpm` dans un répertoire et y exécuter la commande : `createrepo .`

Pour faire un dépot APT, il suffit de créer l'arborescence :

```
mkdir -p dists/lenny/main/binary-i386
```

d'y placer les fichiers .deb, et d'y exécuter la commande :

```
dpkg-scanpackages . /dev/null dists/lenny/main/ |
gzip -9c > Packages.gz
```

# Fabrication d'un dépôt APT : signature GPG

Si on s'en tient là, apt va protester que les paquets qu'on cherche à installer ne sont pas authentifiés. On peut passer outre avec l'option `--allow-unauthenticated`, ou signer notre dépôt apt avec une clé GPG. Pour cela :

- on crée un fichier résumé des données de notre dépot :

```
apt-ftparchive release . > Release
```

- on signe ce fichier avec une clé GPG :

```
gpg -abs -o Release.gpg Release
```

- on ajoute notre clé GPG aux clés de confiance du système :

```
gpg --export --armor > gpg_key
apt-key add gpg_key
```

# Métapaquets

Un métapaquet est un paquet qui n'installe aucun fichier, mais qui a des dépendances. C'est un moyen simple d'installer en une commande un grand nombre de paquets.

Exemples d'utilisation :

- faciliter des mises à jour futures : `linux-generic`
- installer un environnement complet : `ubuntu-desktop`, `kubuntu-desktop`

Un métapaquet se contruit comme un paquet standard. Sous Debian, le paquet `equivs` en facilite la création.

# Bibliographie

Sur DEB/APT :

- Doc Debian sur le format .deb
- Doc Debian pour les mainteneurs de paquets
- Doc ubuntu sur pbuilder

Sur RPM/YUM :

- Le site officiel rpm.org
- Le site concurrent rpm5.org
- Le site officiel de yum
- Documentation Fedora sur la création de rpms
- Maximum RPM