GNU/Linux

Compression and Software Installation

Nicolas Delanoue

Université d'Angers - Polytech Angers

Definitions

- An *archive* is a file that contains other files.
- An *tar archive* is a file that puts « end-to-end » all files ; and keep the structure directories.

Remark

One can compress an archive `tar`, in this case, we call it a *compressed tarball*.

Free (libre) compression - (algorithm & software)

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

Two-step compression mechanism :

1. one creates the tar archive,
2. one compresses this archive.

Example

Usually, we do both at once :

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

Explanations

c	create the archive,
z	compression gzip,
j	compression bz2,
v	verbose mode,
f	use the following given file.

Decompression

We place ourselves in the directory where we want to extract and we use again the tool `tar`.

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

Explanations

x	extract the archive,
z	compression gzip,
j	compression bz2,
v	verbose mode,
f	use the following given file.

Windows & Linux file format compatibility

One can also use the `zip` format with commands like :

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

Decompression : `unzip dossier.zip`

Who provides software ?

The software is provided by :

- the editor of the distribution,
- non-profit organizations,
- of isolated individuals,
- but also companies.

They are provided in the form of installable archives, called packages (package).

How are they classically distributed ?

The software is provided in one or more packages. Their format :

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

It is the equivalent of Windows .exe, .msi, .zip.

Definition

An *package* is an archive containing :

- binaries (executables or libraries),
- or sources (to compile),
- from the configuration files,
- from the documentation,
- a list of dependencies,
- of the installation instructions, in the form of scripts.

Definition

The `dpkg` (debian package) tool is the name of the package management.

Using dpkg

Example de commandes	Signification	Explication
`dpkg -i lepaquet.deb`	install	instal a package
`dpkg -r lepaquet`	remove	remove a package
`dpkg -l`	list	show the package list
`dpkg -L lepaquet`	List files	show the file list provided by a package
`dpkg -S lefichier`	Search	show the package which provides the file passed in argument.

Remarks

- Packages have the extension `.deb`,
- The `dpkg` tool does not use repositories ; you have to have the file `.deb`,,

Definition

An *dependency* is something that is required by the package.

Definition

An *dependency* is something that is required by the package.

Remarks

- Before installing a package, you must therefore install the dependent packages before.
- Conversely, when we uninstall a package, we can uninstall all packages that depend on it.

Definition

An *dependency* is something that is required by the package.

Remarks

- Before installing a package, you must therefore install the dependent packages before.
- Conversely, when we uninstall a package, we can uninstall all packages that depend on it.

- Dependencies between software are a puzzle.
- Distributions offer deposits : i.e. locations giving access to a collection of packages (with all their dependencies),

Working principle of the apt tool

The `apt` command downloads the `.deb` packages, check the dependencies then, rely on `dpkg` to install them.

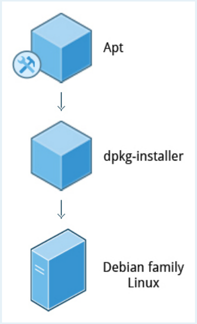

Figure – apt uses dpkg.

Using `apt`

- `apt` can, in particular :
  - install : `$ apt install lepaquet`
  - remove : `$ apt remove lepaquet`
- `apt` uses deposits which are choosen via `/etc/apt/sources.list`
- To update the package states : `apt update`
- To upgrade all the packages : `apt upgrade`

Installation methods

Méthode	**Outil**	**Avantages / inconvénients**
Interface graphique	Logithèque Ubuntu	+ simple + ergonomi - slow - little control over the details
Interface graphique	Synaptic Ubuntu	+ simple + good overview - heavy - slow
High level management	`apt`	+ simple - quite slow
Low level management	`dpkg`	+ fast - need to have the package file - dependency must be manualy resolved

Sources compilation	`./configure`
	`make`
	`make install`
		+ binairies that perfectly matched to the system
		- dependency problems
		- compilation not so easy for newbies

The tool apt-cache

The tool `apt-cache` allows you to query the repositories about packages.

- Find packages containing the keyword :
  `apt-cache search key`
- Get info about a package :
  `apt-cache show lepaquet`
- Getting the dependencies of a package :
  `apt-cache depends lepaquet`

Definition

To compile is to transform the source code into binary code.

Why compile sources ?

- to guarantee the adaptation of binaries to systems,
- to modify the compilation options,
- to modify the sources,
- because we don't always have the choice (ex : proprietary graphic card driver).

Demonstration

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

Installation steps via compilation

1. get the archive with the sources, (use `firefox` or `wget`),
2. extract it in `/usr/local/src` (use `tar` ...)
3. generate a makefile (use `./configure`),
4. compile (use `make`),
5. install (use `make install`).

The compilation - The tool `Makefile`

In a project made up of many source files, compilation is source files, compilation is difficult.
A `Makefile` file allows :

- to compile the project,
- in a simple and standard way,
- by avoiding repetitive commands,
- by limiting the risks of bad manipulations.

`make`

Depending on the desired target, `make` calls commands commands indicated in the `Makefile` file of the current directory.

Structure of a `makefile` file

```
target1 : dependency1 dependency2
          commands to make target1
          from dependency1 and dependency2

target2 : dependance3
          commands to make target3
          from dependency3
```

Remarks

A dependency can be a target.

Some common targets

- no argument, this is the default command.
  In generally, this corresponds to compiling everything
- with the argument `install`, this corresponds to install the binaries.

Remarks

In some cases, there is no `Makefile`.
In this case, there is a `configure` file. It is a script that

- checks the compilation environment,
- generates an appropriate makefile.

It is launched by : `./configure`, one can then execute `make`.