Il pacchetto `rpm`

mobytrick

18 settembre 2014

Preambolo

La tecnologia `rpm` è stata proposta da RedHat ed adottata anche da altri vendor nell'arena Linux. `rpm` permette la distribuzione del software sia tramite pacchetti (quindi su base binaria) sia tramite source. Nel confronto tra `deb` ed `rpm`, le due tecnologie si equivalgono. Ma per quanto riguarda la creazione, `rpm` appare un po' più farraginosa.

Inside

Più in dettaglio, un file `rpm` è costituito da un'intestazione binaria seguita da un contenitore creato tramite utility `cpio` e compresso tramite `gzip`. Il contenitore comprende sia i file che si devono installare che il file `.spec` la cui importanza è fondamentale. È peculiare la cosiddetta *firma* che garantisce la provenienza del pacchetto. Per quelli scaricati da Internet e sui quali c'è qualche dubbio, si può controllare preventivamente l'autenticità e regolarsi di conseguenza.

Il nome di un pacchetto `rpm` ha la seguente struttura:

```
nome_pacchetto-versione-release.architettura.rpm
```

ove `release` indica quante volte un eseguibile è stato trasformato in pacchetto. Se il pacchetto è congegnato in modo da contenere il source, allora il campo architettura è sostituito da `src`.

Per la costruzione di un file `rpm` si tratta di creare una directory di riferimento, che qui ha il nome `ROOT_RPM`, entro la quale si creano alcune cartelle dal nome predefinito e con gli scopi specificati più avanti

```
# mkdir ~utente/ROOT_RPM
# cd ROOT_RPM
# mkdir BUILD RPMS SOURCES SPEC SRPMS tmp
```

Le directory create hanno il seguente scopo:

1. `BUILD`: directory di lavoro
2. `RPMS`: i binari creati dal processo di costruzione del pacchetto
3. `SOURCES`: i sorgenti, le patch, le icone, etc. etc.
4. `SPEC`: il file .spec
5. `SRPMS`: i sorgenti creati dal processo di costruzione del pacchetto
6. `tmp`: ulteriore cartella di lavoro

Parallelamente, in `${HOME}`, è necessario costruire il file `.rpmmacros`, riportato qui sotto, in cui vengono definite alcune variabili usate nella fase di costruzione del file `rpm`. La prima riga, e solo quella, tiene conto della situazione ambientale.

```
%_topdir /home/utente/ROOT_RPM
%_builddir %{_topdir}/BUILD
%_rpmdir %{_topdir}/RPMS
%_sourcedir %{_topdir}/SOURCES
%_specdir %{_topdir}/SPEC
%_srcrpmdir %{_topdir}/SRPMS
%_tmppath %{_topdir}/tmp
```

Il file `.spec`, collocato nella cartella SPEC, va costruito manualmente. Il suo contenuto governa il flusso dei file e come procedere. Le azioni, ad esempio installazione oppure rimozione, si esplicano nell'esecuzione di script.

In pratica

Consideriamo la costruzione di un ipotetico pacchetto, che chiameremo `esempio` (supremo sforzo di fantasia, eh?), costituito dall'eseguibile `esempio`, collocato in `/usr/local/bin`, e dalla relativa man page `esempio.7.gz`, collocata in `/usr/local/man/man7`. Sulla macchina su cui si costruisce il file `rpm` i due summenzionati file si trovano già dove indicato. Il file `.spec` potrebbe chiamarsi `esempio.spec`. È naturale chiamare il file `.spec` col nome del pacchetto, ma non è un'imposizione. L'estensione invece è obbligatoria. Schematicamente `.spec` contiene un preambolo seguito da *sezioni*, ognuna delle quali individuata da una parola chiave.

```
# preambolo
Name: esempio
Summary: Eseguibile di prova
Version: 1.0
Release: 1
License: GPL
Group: Applications
Packager: Anonymous <anonymous@localhost>
%description Pacchetto applicativo di prova
%files /usr/local/bin/%{name}
%doc /usr/local/man/man7/%{name}.7.gz

# pre-installation section
%pre echo "%{name} installandum est"

# post-installation section
%post echo "%{name} installato"
```

Giova ricordare che il file ha una sua sintassi. Le righe che iniziano col carattere `#` sono commenti.
In `Release`: 1, `Release` è un cosiddetto *tag* ed 1 è il suo valore. Il separatore (il carattere `:`) può essere preceduto/seguito da un numero arbitrario, anche nullo, di caratteri bianchi. Il tag non è case sensitive e può essere riferito tramite `%{nome-tag}`. Vige la doppia regola: 1 tag per riga, 1 riga per tag.
Da quanto detto, il preambolo definisce alcuni tag. L'ordine non è importante. Quelli disponibili sono molti di più di quelli citati. Perlopiù sono autoesplicativi. Quelli assolutamente indispendabili sono: `Name`, `Version` e `Release`. Per qualcuno di quelli citati, qualche cenno supplementare.
La differenza tra `Version` e `Release` consiste in questo: `Version` si riferisce al programma, mentre `Release` si riferisce al pacchetto. In altre parole `Release` è la versione del pacchetto.
`Summary` e `%description`. `Summary` è una descrizione succinta (1 riga) del programma. `%description` introduce una descrizione completa che può estendersi su più righe.
`%file` ha il compito di elencare i file coinvolti.
`%doc` è una sottospecifica di file.
Tra i tag non citati in `.spec` ma che risultano importanti ci sono `Requires` e `Conflicts`. Il primo richiede la presenza dei pacchetti citati, pena la non installazione a meno di non procedere in maniera forzosa. I pacchetti possono essere separati dalla virgola che assume il valore di AND oppure dal carattere spazio. Inoltre i pacchetti possono essere seguiti da un operatore relazionale

e dal numero di versione. Gli operatori ammessi sono: `>`, `>=`, `=`, `<=` e `<`, dal significato ovvio. Esempio:

```
Requires: foo > 2.3
```

Per installare il presente pacchetto deve essere installato il pacchetto `foo` con una versione successiva alla 2.3. Le stesse considerazioni valgono anche per `Conflicts`. L'installazione non avrà luogo se ci sono già presenti i pacchetti citati. In questo caso il separatore ha funzione di OR.
`%pre` e `%post` stanno rispettivamente per pre- e post- installation. Sono specificate delle azioni da eseguirsi prima/dopo che il pacchetto `rpm` è stato installato. I comandi vengono eseguiti in ambiente di shell a mo' di script, con tutto ciò che ne consegue: sostituzioni, costrutti, etc.
Le sezioni che sottendono uno script sono:

- `%build` costruzione del pacchetto
- `%clean` usato per ripartire da zero
- `%install` installazione
- `%post` post installazione
- `%postun` post disinstallazione
- `%prep` operazioni propedeutiche
- `%pre` pre installazione
- `%preun` pre disinstallazione

Le sezioni vanno specificate sono quando devono essere indicate delle operazioni in aggiunta a quelle eseguite per default. Nel caso in questione, il pacchetto installa alcuni file. Non ci sarebbe bisogno di alcun extra, ma per ragioni didattiche sono state previste delle operazioni aggiuntive. Si tratta di 2 messaggi di conferma. Il comando per costruire il file `rpm` è il seguente:

```
# cd ~utente/ROOT_RPM/SPEC
# rpmbuild -bb esercizio.spec
```

ove `-bb` sta per **b**uild **b**inary in quanto si parte da file binari già pronti. Questo l'output:

```
Processing files: esempio-1.0-1
Requires(interp): /bin/sh /bin/sh
Requires(rpmlib): rpmlib(CompressedFileNames) <= 3.0.4-1 rpmlib(PayloadFilesHave
Requires(pre): /bin/sh
Requires(post): /bin/sh
Checking for unpackaged file(s): /usr/lib/rpm/check-files %{buildroot}
Wrote: /home/utente/ROOT_RPM/RPMS/i386/esempio-1.0-1.i386.rpm
```

Si nota che il file `rpm` è stato costruito nella cartella `../RPMS/i386`. È richiesta pure l'esecuzione degli script di pre- e post- installazione. Il processo di costruzione ha dedotto che il comando è stato utilizzato su una macchina con architettura hardware Intel 386 e si è regolato di conseguenza.
Il pacchetto è stato costruito. Ora si tratta di verificarlo, per essere sicuri di non aver commesso errori. Quindi:

```
# cd ~utente/ROOT_RPM/RPMS/i386
# rpm -vv -qpl esempio-1.0-1.i386.rpm
D: Expected size: 6746 = lead(96)+sigs(180)+pad(4)+data(6466)
D: Actual size: 6746
D: esempio-1.0-1.i386.rpm: Header SHA1 digest: OK (027c91660a6d0b15408aa0f2f2b1e
-r-xr-xr-x 1 root root 9903 Jan 24 16:07 /usr/local/bin/esempio
-r--r--r-- 1 root root  223 Feb  3 19:39 /usr/local/man/man7/esempio.7.gz
D: May free Score board((nil))
```

Si constata che il pacchetto `rpm` è costituito dai due file, come stabilito da `.spec`. È possibile pure una controprova tramite:

```
# cd ~utente/ROOT_RPM/RPMS/i386
# rpm2cpio esempio-1.0-1.386.rpm | cpio -iv --list
-r-xr-xr-x 1 0 0 9903 Jan 24 16:07 ./usr/local/bin/esempio
-r--r--r-- 1 0 0  223 Feb  3 19:39 ./usr/local/man/man7/esempio.7.gz
```

Come ricordato, il formato di un file `rpm` è alquanto complesso. È costituito da una sezione iniziale che contiene svariate informazioni e da un successivo contenitore di file. `rpm2cpio` non fa altro che saltare la sezione iniziale e decomprimere, internamente, il contenitore. Questo è stato costruito tramite utility `cpio` mettendo assieme i file citati in `%file` che `%doc`. La precedente pipe non fa altro che mostare i costituenti.
Visto che i controlli hanno sancito la regolarità del file `rpm` testé creato, questo può essere utilizzato su un'installazione che prevede l'immissione del software in questa maniera.