Il formato `deb`

mobytrick

17 settembre 2014

`Preambolo`

Le distribuzioni Linux forniscono il software sotto forma di *pacchetti*. Il concetto di pacchetto si basa sulla raccolta di tutti i file che servono per la fruizione di un eseguibile. Quindi, non solo l'eseguibile stesso, ma anche la man page, eventuali file di configurazione, librerie specifiche, e via discorrendo. Ma non basta. Sono presenti pure gli script per l'installazione e lo smantellamento ed i controlli delle dipendenze. I pacchetti sono congegnati in modo da essere auto sufficienti. L'unico intervento è limitato alla decisione del sistemista: potrebbe darsi il caso che non se ne faccia nulla. Come da pessima tradizione nel mondo open source, non c'è stata uniformità nella metodologia di costruzione. Inevitabile il pateracchio: i pacchetti si presentano in formati differenti ed ovviamente incompatibili tra loro. Quelli più diffusi sono:

- `deb`. Tipico delle distribuzioni Debian (donde il nome) e di quelle derivate
- `rpm`. Tipico di RedHat (`rpm` = **R**edHat **P**ackage **M**anager) e di quelle derivate

Nell'arsenale di Debian esiste il prodotto `alien` il cui scopo dovrebbe essere quello di tramutare un pacchetto costruito con tecnologia `rpm` nel corrispondente `deb`, e viceversa. Il consiglio è quello di costruire un pacchetto in uno dei due formati, evitando la trasformazione.

`Anatomia di deb`

Un pacchetto in formato `deb` consta di 3 parti:

1. `debian-binary` è un file che informa circa la versione `deb` utilizzata. Ebbene, sì. Anche rimanendo fedeli ad una tecnologia, non si può prescindere dalla sua evoluzione. Alla data di questo scritto la versione corrente di `deb` è la 2.0

2. `data.tar.gz`. È la raccolta, fatta tramite utility tar e poi compressa via gzip, di tutti i file che costituiscono il pacchetto. Nel file `data.tar.gz` i file non sono messi dentro alla rinfusa. Sono collocati nel medesimo schema in cui appariranno dopo l'inserzione

3. `control.tar.gz`. È la raccolta, fatta tramite utility tar e poi compressa via gzip, degli script per l'installazione e lo smantellamento ed altro ancora

Più in dettaglio, il *tarball* `control.tar.gz` a sua volta è costituito da 4 file:

1. `control` obbligatorio

2. `md5sums` opzionale. Contiene l'`md5sum` (controllo di integrità) per tutti i file contenuti in `data.tar.gz`

3. `postinst` opzionale. Script che, se presente, verrà eseguito dopo che il pacchetto sarà stato installato

4. `prerm` opzionale. Script che, se presente, verrà eseguito prima della rimozione del pacchetto

A parte `control` e `md5sums`, sempre presenti, gli altri file possono mancare oppure assumere nomi diversi. Il file `control` è di testo. Ha una struttura ed una sintassi. Ciascuna riga inizia con una parola chiave seguita dai caratteri `:` (colon + spazio). Quelle precedute dall'asterisco sono obbligatorie.

- *Package:
- *Version:
- Section:
- Priority:
- Architecture:
- Depends:
- Suggests:

- Conflicts:
- Replaces:
- Installed-Size:
- *Maintainer:
- *Description:

`Package` è il nome del pacchetto. Si possono usare le lettere minuscole, quelle maiuscole, le cifre, i caratteri `+` e `-`. Attenzione a non usare un nome di pacchetto già in uso.

`Version` è il numero di versione. Composto da major e minor release, separati da un punto. Opzionalmente può essere aggiunta anche la revisione, preceduta da un trattino

`Maintainer` ha più che altro funzione amministrativa. Comprende i dati anagrafici e l'indirizzo di posta

`Description` autoesplicativo. Nella stessa riga c'è una descrizione breve (max 60 caratteri). Nelle righe successive ci si può dilungare ma il primo carattere deve essere uno spazio. Per inserire una riga bianca, questa deve contenere un unico carattere, il `.` (punto)

`Section` in Debian tutti i pacchetti sono categorizzati, ovvero raggruppati in maniera omogenea. Ad ogni gruppo viene assegnato un nome. Ad esempio: mail, X11, net, admin e così via

`Priority` indica l'importanza del pacchetto. Valori accettabili: extra, required, optional, etc.

`Architecture` indica se il pacchetto è specifico per un determinato hardware, oppure no. Valori accettabili: all, i386, sparc, i686, etc.

`Depends` elenca i pacchetti che devono essere presenti. Il pacchetto viene comunque installato.

`Pre-Depends` elenca i pacchetti che devono già essere presenti obbligatoriamente. Se mancano, l'installazione non ha luogo

`Conflicts` elenca i pacchetti con i quali c'è incompatibilità software

`Recommends` elenca i pacchetti che dovrebbero normalmente accompagnare l'installazione

`Suggests` elenca i pacchetti che potrebbero utilmente corredare il package che si sta installando

In `Depends`, `Pre-Depends`, `Conflicts`, `Recommends` e `Suggests` i pacchetti sono separati da un separatore. Se è il carattere `,` (virgola), esso ha valore di AND. Se invece è `|` (barra verticale) ha valore di OR. Il nome del pacchetto

può essere seguito opzionalmente da un numero di versione, preceduto da un operatore relazionale. Il tutto racchiuso tra parentesi tonde. Ad esempio:

```
Pre-Depends: libc6 (>> 2.2.4-4)
```

significa che il pacchetto, per essere installato, necessita che la versione di libc6 sia maggiore a quanto indicato. Gli altri operatori relazionali sono: `<<`, `<=`, `=` e `>=`, dall'ovvio significato. Il file `control` va costruito a mano. Gli script di pre-, rispettivamente post- installazione possono essere anche dei *placeholders*.

Creazione di un file deb

Un file `deb` può essere creato tramite l'utility `dh-make`. Qui verrà spiegata la sua costruzione *a mano*.

Facciamo l'esempio di un pacchetto che consiste dell'eseguibile e della man page. Trattandosi di un pacchetto non ufficiale, è bene non inquinare il sistema operativo. L'eseguibile va posto in `/usr/local/bin`, la man page in `/usr/local/man/man7` (la sezione 7 delle man page è dedicata a tutti quei programmi di difficile catalogazione. Una miscellanea, insomma). Conviene creare una directory apposita, supponiamo `~utente/ROOT_DEB`, ove si procederà alla costruzione del pacchetto. Quindi:

1. creazione della struttura della directory:

   ```
   mkdir -p ~utente/ROOT_DEB/usr/local/bin
   mkdir ~utente/ROOT_DEB/usr/local/man/man7
   ```

   Le cartelle hanno come maschera di protezione 755

2. la struttura va popolata. L'eseguibile va posto nella sottocartella `bin` con maschera 755 mentre la man page va posta nella sottocartella `man/man7` con maschera 644

3. creazione dei file `data.tar.gz` e `md5sums` tramite:

   ```
   tar -C ROOT_DEB -zvpcf data.tar.gz .
   cd ROOT_DEB
   find . -type f -exec md5sum {} \; |
   grep -v md5sums > md5sums
   mv ../data.tar.gz .
   ```

4. creazione del file debian-binary tramite:

```
cd ~utente/ROOT_DEB
echo "2.0" > debian-binary
chmod 644 debian-binary
```

5. creazione dei file postinst e prerm

```
cat >> postinst << EOF
#!/bin/bash
exit 0
EOF
cp postinst prerm
chmod 755 postinst prerm
```

6. creazione del file `control`, con maschera 644. Da quanto detto il file, quasi ridotto al minimo indispensabile è:

```
cat >> control << EOF
Package: esempio
Version: 1.0
Section: 7
Architecture: all
Maintainer: anonymous <anonymous@localhost>
Description: greetings
In base all'ora stampa un messaggio appropriato.
EOF
chmod 644 control
```

Per costruire il file `control.tar.gz` si usa il comando:

```
tar zvcpf control.tar.gz control md5sums postinst prerm
```

Ora tutto l'occorrente per la crezione del pacchetto `deb` è pronto. Supposto che il pacchetto si chiami `esempio` e la versione sia `1.0` non resta che crearlo tramite:

```
ar -vr esempio_1.0.deb debian-binary control.tar.gz data.tar
```

Per il nome vige la seguente regola: `nome-del-pacchetto_versione`. Ma possono essere incluse altre informazioni riguardanti ad esempio la lingua, l'architettura, il nome della distribuzione, etc etc. L'estensione è ovviamente `.deb`.

Ad ogni buon conto è fatta. Prima il doveroso controllo, tramite:

```
# dpkg --contents esempio_1.0.deb
drwx------ utente/utenza    0 2011-02-09 19:02 ./
drwxr-xr-x utente/utenza    0 2011-01-25 19:14 ./usr/
drwxr-xr-x utente/utenza    0 2011-01-25 19:14 ./usr/local/
drwxr-xr-x utente/utenza    0 2011-01-25 19:15 ./usr/local/bin/
-r-xr-xr-x utente/utenza 9903 2011-01-24 16:07 ./usr/local/bin/esempio
drwxr-xr-x utente/utenza    0 2011-01-25 19:14 ./usr/local/man/
drwxr-xr-x utente/utenza    0 2011-02-03 19:49 ./usr/local/man/man7/
-r--r--r-- utente/utenza  223 2011-02-03 19:39 ./usr/local/man/man7/esempio.7.gz
```

Si ottiene l'elenco dei file che fannno parte del pacchetto. Poiché l'output sembra corretto si può pensare all'installazione vera e propria, ammesso di poter operare con i privilegi di superuser.

```
# sudo dpkg -i ~utente/ROOT_DEB/esempio_1.0.deb
(Reading database ... 89008 files and directories currently installed.)
Unpacking esempio (from .../ROOT_DEB/esempio_1.0.deb) ...
Setting up esempio (1.0) ...
Processing triggers for man-db ...
```

Che dire? Il file `deb` testé creato dovrebbe essere a circolazione limitata. Per fare le cose in maniera professionale ci si attiene a quanto prescritto dalle policy di Debian, cui si rimanda.