Ricomincio da DEB – come creare un Debian Package – pt.2

Robertino: *Ma mammina dice che io ho i complessi nella testa.*
Gaetano: *E fuss''o Ddio! Quali complessi? Tu tieni 'n'orchestra intera 'n'cap, Robbe'.*

E allora, dove eravamo rimasti? Ah, si! Abbiamo **ricominciato da tre** (*anzi, da DEB*), con **un articolo** che invece di descrivere *"come sviluppare"* (nel nostro amato **C**) una applicazione, descriveva *"come distribuire"* la stessa applicazione (nel nostro amato **Linux**). E noi non faremo come il **Robertino** qui sopra, che aveva un po' di confusione in testa su quale via intraprendere, ma faremo come gli consigliava l'impagabile **Gaetano**, e prenderemo una via molto creativa, scriveremo uno *shell script* per creare automaticamente un **Debian Package**!



…ma possibile che non hai mai scritto un installer?…
La prima parte dell'articolo l'avevamo conclusa su questa lista di cose da fare (*con il grado di difficoltà incluso!*):

- costruire l'albero di installazione (*facile!*)
- copiarci dentro eseguibili e servizi (*facilissimo!*)
- scrivere il file control (*si può fare…*)
- scrivere gli eventuali script di pre/post installazione (*spoiler: in questo semplice esempio ci serve solo lo script postinst*)

E avevo anticipato che nella seconda parte (*e cioè qui*) avremmo cercato di fare il tutto con lo *spirito del programmatore*, e cioè avremmo presentato uno *shell script* che esegue i punti precedenti in modo automatico, e che diventerà la base (con piccoli adattamenti) per altri *script* di creazione per qualsiasi applicazione che svilupperemo… un compito troppo ambizioso? In realtà è molto semplice!.

Ok, andiamo per passi: vediamo di descrivere i punti della lista e poi, alla fine, automatizzeremo il tutto. Cominciamo con i primi 2 punti (condensati in 1 solo)!

1) costruire l'albero di installazione + copiarci dentro eseguibili e servizi

Nel nostro caso l'albero di installazione avrà questa forma:

```
├── DEBIAN
│ ├── control
│ └── postinst
├── etc
│ └── systemd
│ └── system
│ |── cansetup.service
│ └── canrecv.service
└── usr └── bin |── cansetup.sh └── canrecv
```

per cui, una volta creato (da qualche parte nella nostra directory di lavoro) un albero della forma descritta, sarà un gioco da ragazzi copiarci dentro i file presentati nella prima parte dell'articolo (a parte i *file control* e *postinst* che non abbiamo ancora visto). Notare che questo primo punto lo faremo, automaticamente, attraverso il nostro *shell script* di creazione del package, quello che vedremo tra poco.

2) scrivere il file control

Il *file control* ha, come visto nel precedente articolo, una struttura minima abbastanza semplice. Una struttura che si può complicare, a piacere, aggiungendo altre linee di dettagli che, in alcuni casi, possono anche essere utili, ma direi che per la nostra applicazione *canrecv* (che è un semplice *CAN Server*) è sufficiente un *file control* come questo:

```
Package: canrecv
Version: 1.0-amd64
Architecture: amd64
Depends: libc6 (>= 2.34)
Maintainer: A.Abate <artcprogramming@gmail.com>
```

Description: canrecv DEB package Installa la applicazione canrecv e i relativi servizi systemd

dove si possono cambiare, se necessario, la versione *Version* (*N.B.: deve cominciare sempre con un numero*) e l'architettura *Architecture* (nel"esempio ho messo quella su cui lavoro io). Le dipendenze *Depends* sono quelle ottenute con il procedimento descritto **nello scorso articolo**, il responsabile *Mantainer* sarà l'autore dell'*installer* stesso e la descrizione corta *Description* sarà quella adatta all'*installer* in oggetto, seguita nella linea successiva dalla descrizione lunga, che può essere anche multi-linea, con linee che cominciano sempre con uno spazio. Facile, no?

3) scrivere gli eventuali script di pre/post installazione.

Per il nostro *package* è necessario solo uno *script* di post-installazione, uno *script* che installa e attiva i nuovi servizi. E vediamolo!

```
#!/bin/bash # set uscita immediata su errore
set -e # abilita e avvia il servizio cansetup
systemctl enable cansetup.service
systemctl start cansetup.service
systemctl daemon-reload # abilita e avvia il servizio canrecv
systemctl enable canrecv.service
systemctl start canrecv.service
systemctl daemon-reload
```

Questo *script* è necessario, in questo caso, perché il nostro *package* contiene dei servizi *systemd,* che devono essere opportunamente installati nel sistema. Il procedimento consiste nell'abilitazione del servizio, seguita dal suo avvio e dal riavvio del *daemon* di *systemd* (tutto questo ripetuto per i due servizi inclusi nel *package).* Ovviamente per altri tipi di applicazione uno script di *postinst* potrebbe non essere necessario o fare ben altre cose. Comunque i tipi di *script* disponibili li trovate elencati nella **prima parte dell'articolo**.

A questo punto abbiamo anche i *file* che ci mancavano da aggiungere al nostro albero creato qui sopra, quindi cosa ci manca? Ma ci manca il nostro (*oramai mitico*) *shell script* per automatizzare il tutto! Vediamolo!

```
#!/bin/bash # check degli argomenti
if [ $# -eq 0 ]; then echo "numero argomenti errato" echo "uso: \$0 nome_release (e.g.: \$0 1.0-amd64)" exit 1
fi # set della directory di lavoro
mydir=canrecv-$1 # compilo l'ultima versione di canrecv.c
cd ..
gcc canrecv.c -o install/base/canrecv
cd install # creo l'albero di directory destinazione
mkdir $mydir
mkdir $mydir/DEBIAN
mkdir $mydir/usr
mkdir $mydir/usr/local
mkdir $mydir/usr/local/bin
mkdir $mydir/etc
mkdir $mydir/etc/systemd
mkdir $mydir/etc/systemd/system # copio i file nella directory destinazione
cp base/control $mydir/DEBIAN
cp base/postinst $mydir/DEBIAN
cp base/cansetup.sh $mydir/usr/local/bin
cp base/canrecv $mydir/usr/local/bin
cp base/cansetup.service $mydir/etc/systemd/system
cp base/canrecv.service $mydir/etc/systemd/system # costruisco il DEB package
dpkg-deb --build --root-owner-group $mydir/
```

Avrete notato che è uno script molto compatto che esegue solo le operazioni indispensabili (**opportunamente commentate**):

- check degli argomenti
- set della directory di lavoro
- compilo l'ultima versione di *canrecv.c* (*nel package deve entrare sempre l'ultima versione, no?*)
- creo l'albero di directory destinazione (*usando delle semplici istruzioni mkdir*)
- copio i file nella directory destinazione (*usando delle*

semplici istruzioni cp)
- costruisco il **DEB package** (*usando il classico comando **Linux** dpkg-deb*)

Semplice, no? E come si può usare lo *script*? Io suggerisco suggerisco di creare un ambiente di creazione di questo tipo:

```
├── base
│   ├── canrecv
│   ├── canrecv.service
│   ├── cansetup.service
│   ├── cansetup.sh
│   ├── control
│   └── postinst
└── buildpkg.sh
```

con una directory *"base"* dove risiede la base di installazione (che è, anche, la destinazione della istruzione di compilazione presente nello *script*, non so se avete notato). Entrando in questo ambiente di creazione ed eseguendo il comando:

```
aldo@Linux $ ./buildpkg.sh 1.0-amd64
dpkg-deb: generazione del pacchetto "canrecv" in "canrecv-1.0-amd64.deb".
aldo@Linux $
```

otterremo che il nostro albero di creazione si espanderà così:

```
├── base
│   ├── canrecv
│   ├── canrecv.service
│   ├── cansetup.service
│   ├── cansetup.sh
│   ├── control
│   └── postinst
├── buildpkg.sh
├── canrecv-1.0-amd64
│   ├── DEBIAN
│   │   ├── control
│   │   └── postinst
│   ├── etc
```

```
│   │   └── systemd
│   │   └── system
│   │   ├── canrecv.service
│   │   └── cansetup.service
│   └── usr
│   └── local
│   └── bin
│   ├── canrecv
│   └── cansetup.sh
└── canrecv-1.0-amd64.deb
```

ovvero, troveremo il nostro **Debian Package** in forma espansa (la *directory* **canrecv-1.0-amd64**) e compressa (il *file* **canrecv-1.0-amd64.deb**), e quest'ultima ci permetterà di distribuire il nostro installer a chi lo vorrà!

E qui direi che l'articolo si può dire concluso. Abbiamo **ricominciato da tre** (anzi, **da DEB**), e per una volta abbiamo parlato di *distribuzione* invece che di *programmazione.* Quasi quasi nel prossimo articolo, per cavalcare l'onda, potrei parlare si una argomento toccato di striscio **nella prima parte**, e cioè la compilazione abbinata alla distribuzione (**Autotool**). Beh, vedremo, e in ogni caso sarà una sorpresa! (*spero lieta…*).

Ciao, e al prossimo post!

https://italiancoders.it/ricomincio-da-deb-come-creare-un-debian-package-pt-2/