SLACKWARE PACKAGES UNLEASHED

Creazione avanzata e canoni convenzionali v. 01 no © CAT – 2003

1. INTRODUZIONE E LICENZA

Questo DRAFT vuole essere una guida alla creazione di pacchetti Slackware, si tratta in gran parte di conoscenze provenienti dall'utilizzo "Massiccio" di Slackware Linux. Come per tutto ciò che ho scritto e pubblicato in precedenza questo documento NON è coperto da Licenza di alcun genere, questa volta mio malgrado. Se siete interessati a conoscerne i motivi potete trovare una risposta al paragrafo 9.3. Questo documento rappresenta un "tutorial", potete prenderne spunto, copiarlo parzialmente o interamente, fatene quello che volete; se citate il mio nome in qualche altro documento che ha preso idee o spunti da questo mi fà ovviamente piacere. Chiunque voglia segnalare errori o correzioni è il benvenuto.

- MARCHI

2. STILE

Slackware ha uno stile tutto suo. Basata su un "mix" tra SYSV e BSD Style; si è guadagnata in Anni la fama di "distribuzione pulita". Senza entrare nello specifico del Mito (sfatato o meno) andiamo ad analizzare la parte di stile che ci interessa in modo da poter creare dei pacchetti Perfetti in puro "Slackware Style".

2.1 Linux for SubGenius

Slackware nasce nell'aprile 1993 dalla geniale mente di Patrick J. Volkerding, è basata su SLS, distribuzione ormai deceduta a cui lo stesso PJV lavorava. E' la più antica distribuzione Linux tra quelle ancora in vita.
Seppur si sia evoluta molto negli anni, la sua struttura si consolida dove può su "shell scripting", ecco uno dei motivi per la quale viene considerata "pulita" (gran parte di ciò che fà uno script possiamo farlo a "mano" da shell).
Come quasi tutti i sistemi operativi sfrutta dei pacchetti precompilati che sostengono la sua intelaiatura e ci permettono di sfruttare appieno le potenzialità di questo *nix.
Le differenze che rendono unica Slackware sono molteplici, con questa semplice spiegazione abbiamo solo toccato la superfice, ma ciò che viene trattato in questo DRAFT sono i pacchetti, e ci basta sapere questo riguardo lo Slackware Style per poter procedere.

2.2 Struttura delle directory

La struttura delle directory sotto Slackware è molto simile a quella di un *nix BSD:

```
drwxr-xr-x    root    bin     bin/
drwxr-xr-x    root    root    boot/
drwxr-xr-x    root    root    dev/
drwxr-xr-x    root    root    etc/
drwxr-xr-x    root    root    home/
drwxr-xr-x    root    root    lib/
drwxr-xr-x    root    root    mnt/
drwxr-xr-x    root    root    opt/
dr-xr-xr-x    root    root    proc/
drwx--x---    root    root    root/
drwxr-xr-x    root    bin     sbin/
drwxrwxrwt    root    root    tmp/
drwxr-xr-x    root    root    usr/
drwxr-xr-x    root    root    var/
```

Come possiamo notare quasi tutte le directory nella root principale sono di proprietà di root.root fatta eccezione per bin/ e sbin/ che sono di proprietà root.bin (vale per quasi tutte le directory bin/ e sbin/ presenti nel sistema, quindi anche /usr/bin/, /usr/sbin/ etc...).

Queste directory usualmente contengono gli eseguibili, le directory lib/ contengono le librerie, le directory man/ contengono le pagine man (e.g. /usr/man di proprietà root.root).
Una nota particolore per la directory /usr/doc/ (anch'essa di propietà root.root) che contiene tutti i documenti ed affini dei software installati.

2.3 Cosa si intende per "pacchetto"?

Per pacchetto intendiamo un software precompilato che automaticamente vada a posizionarsi in alcune directory prestabilite e che funzioni senza la necessità di essere compilato. Un pacchetto slackware altro non è che un software precompilato, archiviato con l'utility tar (man tar per informazioni) comprensivo delle directory prestabilite e compresso con l'utility gzip (man gzip per informazioni).

2.4 Tools predefiniti per la gestione dei pacchetti

Per gestire i pacchetti ci vengono messi a disposizione diversi scripts:

pkgtool	(per la completa gestione dei pacchetti – interfaccia ncurses)
makepkg	(creazione automatizzata di un pacchetto partendo da un precompilato)
explodepkg	(estrae il pacchetto nella directory corrente)
installpkg	(installa il pacchetto)
removepkg	(rimuove il pacchetto)
upgradepkg	(disinstalla il vecchio pacchetto ed installa quello nuovo)

il man anche qui può essere esauriente.

2.5Percorsi Utili

Slackware mantiene un database dei pacchetti e degli scripts aggiunti e rimossi, le directory relative sono:

/var/log/packages
/var/log/removed_packages
/var/log/scripts
/var/log/removed_scripts

2.6Il quadro chiaro

Riepiloghiamo e vediamo di avere un quadro chiaro sullo stile relativo alla creazione di un pacchetto slackware:

Abbiamo detto che:

Un pacchetto è un software precompilato, motivo per il quale dovrebbe contenere almeno un eseguibile, che seguendo lo stile slackware con molta probabilità andrà a posizionarsi sotto /bin/ o /sbin/ o molto più probabilmente sotto /usr/bin/ o /usr/sbin/
per cui il proprietario di questo eseguibile nonche della directory dovrà essere root.bin in questo caso.
Se il pacchetto contiene delle librerie o files che devo inserire in altri PATH con molta probabilità il proprietario dovrà essere root.root.
La documentazione (i README, INSTALL, NEWS, FAQ, CHANGELOG etc...) devo inserirla sotto / usr/doc/nomesoftware-versione (esempio pratico: xchat 1.1.0 avrà la sua documentazione sotto / usr/doc/xchat-1.1.0), questa directory ed il suo contenuto sono anch'essi di proprietà root.root.

3. Canoni e "standard"

Nel paragrafo precedente abbiamo visto alcuni "standard" utilizzati da slackware:

le directory contenenti i binari sono quasi tutte di proprietà root.bin
la directory predefinita per la documentazione è /usr/doc/
le directory che tengono traccia di pacchetti e scripts sono sotto /var/log/

Questi sono alcuni degli "standard" PATH.

Ora esaminiamo il PATH standard che viene "preferito" dai "SubGenius":

Nella maggior parte dei casi sotto Linux i software vengono installati sotto la directory /usr/local (quindi /usr/local/bin o /usr/local/sbin per i binari /usr/local/lib per le librerie /usr/local/etc per i files di configurazione/sistema e così via).

Slackware preferisce /usr/ al posto di /usr/local per i binari.
/etc/ è la directory per i files di configurazione/sistema.
/var/log/ per i logs.

3.1Come posso cambiare il PATH predefinito dal sorgente?

Qui serve una leggera conoscenza concernente la compilazione.

Se il sorgente è dotato di configure (man autoconf) la flag --prefix=/usr può salvarci (e solitamente è così per quanto riguarda i binari), se necessitiamo di files in /etc/ utilizziamo la flag --sysconfdir=/etc. Esistone altre flags per altre directory di sistema, un ./configure --help potrà aiutarci a capire.

Se il sorgente non è dotato del configure possiamo tentare con un make –prefix=/usr, anche qui un make –help ci può venire in aiuto; oppure possiamo editare il Makefile e sostituire i PATH a mano (e.g. /usr/local lo sostituiamo con /usr), attenti a quello che fate se non avete almeno una minima esperienza con la compilazione ed i comandi *nix.

Se il sorgente non è dotato neanche di Makefile e va compilato a mano con il compilatore i casi sono 2:

1)Ne sapete abbastanza quindi sapete già cosa fare ed è già tanto che leggete questo tutorial.
2)Non ne sapete quasi nulla, quindi rileggetevi un pò le basi poi tornate su questo punto e compilatevi il programma che evidentemente non avrà bisogno di un PATH di installazione.

Bene, abbiamo appurato la convenzione dei PATH sotto slackware.

3.2Flags di ottimizzazione del compilatore

Le flags di ottimizzazione del compilatore (CFLAGS) sono degli accorgimenti del gcc (il compilatore appunto) che migliorano le prestazioni del binario in base alle capacità del calcolatore ed alla sua architettura. Esse rendono più "snello" l'eseguibile, cambiando alcuni parametri e facendo meno controlli (cosa che torna utile anche in ambito sicurezza). A dire il vero esistono una "valanga" di flags, e non starò qui a spiegare le varie differenze ed i motivi per cui siano utili anche per la sicurezza, ciò che interessa a noi è lo "standard" utilizzato da slackware.

le CFLAGS che PJV ci propone sono usualmente:

-O2 -march=i386 -mcpu=i686 (fino a release uguali o minori a slackware 9.0)

-O2 -march=i486 -mcpu=i686 (per release maggiori di slackware 9.0)

Per utilizzare queste flags dal sorgente è semplice:

Se il sorgente è dotato di configure:

CFLAGS="-O2 -march=i486 -mcpu=i686" ./configure --prefix=/usr

Se il sorgente è dotato di Makefile lo editiamo e inseriamo/cambiamo le flags (non è fondamentale se non riuscite a cambiarle)

e, dal momento che slackware è la release dei "SubGenius" nessuno ci vieta di ottimizzare ulteriormente.

3.3Strip dei binari e di librerie

Convenzionalmente, per risparmiare spazio e "snellire" il programma sotto slackware vengono "strippati" gli eseguibili (man strip). Strippare un eseguibile significa sostanzialmente togliere i simboli di debug dai files oggetto.

Per i binari "normali" usualmente il comando è il seguente:

#strip nomeprogramma

Per le librerie invece:

#strip --strip-unneeded nomelibreria (togliamo solo alcuni simboli)

3.4Rispamiare spazio su disco

Di default, oltre che ottimizzare e strippare, per risparmiare spazio su disco comprimiamo le pagine man (se presenti) con gzip. Il man decomprime e legge "on-the-fly" le pagine compresse.

3.5Sintassi di un pacchetto

Dalla versione 8.1 di Slackware si è cambiato il metodo di "nominare" i pacchetti (per adeguarsi, e soprattuto per una migliore gestione dei pacchetti anche da parte dei tools di gestione)

Un pacchetto viene nominato così:
nomesorgente-versionesorgente-architettura-build.tgz
dove:
nomesorgente = nome del software sorgente
versionesorgente = versione del software sorgente
architettura = opzioni del compilatore con cui si compila il sorgente (CFLAGS)
build = versione del pacchetto

esempio pratico:

il sorgente è lopster 1.2.0
lo compiliamo con le CFLAGS convenzionali:
-O2 -march=i486(flags per l'architettura) -mcpu=i686
questa è la prima volta che viene fatto questo pacchetto.

quindi il nome del pacchetto sarà.....

nomesorgente = lopster
versionesorgente = 1.2.0
architettura = i486
build = 1

ed eccolo qui:

lopster-1.2.0-i486-1.tgz

Riepilogo:

Dopo aver compilato/ottimizzato dal sorgente e avergli dato il prefisso giusto per i PATH di installazione, strippiamo i binari e le librerie (se ce ne sono) e "gzippiamo" le pagine se ce ne sono.

Questo è lo stile slackware di default a grandi linee.

Ora conosciamo i PATH, e le ottimizzazioni necessarie, non ci resta che fare una prova con un sorgente vero e proprio.

4. Dal Sorgente

Facciamo un esempio pratico di installazione completa (senza creare nessun pacchetto) di un software a caso. Prendiamo come esempio lopster.

Lo scariamo ed eccoci pronti:

```
#tar zxvf lopster-1.2.0.tar.gz
#cd  lopster-1.2.0
#CFLAGS=”-O2 -march=i486 -mcpu=i686” ./configure --prefix=/usr
#make
#strip lopster
#make install
#chown root.bin /usr/bin/lopster
#mkdir /usr/doc/lopster-1.2.0
#cp -a AUTHORS BUGS COPYING ChangeLog INSTALL NEWS README README.pings\
  /usr/doc/lopster-1.2.0
#chown -R root.root /usr/doc/lopster-1.2.0
```

PERFETTO. Abbiamo installato lopster in puro Stile Slackware. Non resta che crearci uno script che faccia automaticamente tutto quello che abbiamo fatto qui e che magari ci metta tutti i files di cui abbiamo bisogno in una directory temporanea per poi creare il pacchetto senza problemi con il tool makepkg.

Per convenzione questo Script porterà il nome di: SlackBuild.

5. SlackBuild

semplificando la spiegazione diciamo che: per poter creare un pacchetto slackware l'utility makepkg ha bisogno di trovarsi tutte le directory ed i files necessari sotto una directory temporanea (simile al concetto di chroot) che per convenzione si chiamerà package-nomesorgente (e.g. Package-lopster) e, sempre per convenzione la posizioneremo nella directory temporanea predefinita (/tmp).

Procediamo quindi con la creazione del nostro SlackBuild (una minima conoscenza di bash scripting è richiesta)

5.1directory temporanea (file SlackBuild)

```
#!/bin/sh
# directory temporanea e variabile directory corrente:
CWD=`pwd`
if [ "$TMP" = "" ]; then
  TMP=/tmp
fi

PKG=$TMP/package-lopster

#Settiamo qualche altra variabile per velocizzare il lavoro:

NAME=lopster
VERSION=1.2.0
ARCH=i486
BUILD=1

if [ ! -d $TMP ]; then
  mkdir -p $TMP # posizione dove creiamo il sorgente
fi
if [ ! -d $PKG ]; then
  mkdir -p $PKG # posizione dove creiamo il package
fi
```

Nel caso la compilazione del sorgente non crei le directory che ci interessano dobbiamo farlo noi stessi, e qui dipende da quali directory servono al sorgente. (nel caso di lopster la compilazione crea tali directory, ma tanto per fare un esempio le creeremo nello SlackBuild così:

```
mkdir -p $PKG/usr # creiamo quella directory che sarà la nostra /usr
mkdir -p $PKG/usr/bin # creiamo quella directory che sarà la nostra /usr/bin
mkdir -p $PKG/usr/share # creiamo /usr/share (richiesta da lopster)
```

per ora chiudiamo lo SlackBuild che riprenderemo tra poco.

5.2in caso di patch

Spesso capita di dover "patchare" un software, per motivi di funzionalità e/o sicurezza. Nel caso di lopster ad esempio un patch per la versione 1.2.0 c'è... scarichiamola e se non è già compressa "gzippiamola" noi stessi, la faremo leggere allo SlackBuild tramite lo zcat (man zcat). La patch compressa è: lopster-1.2.0-1.patch.gz

Riapriamo il nostro SlackBuild ed aggiungiamo la "scompattazione del sorgente in /tmp e la parte sulla patch:

```
echo "+=================+"
echo "| $NAME-$VERSION |"
echo "+=================+"
cd $TMP
tar zxvf $CWD/$NAME-$VERSION.tar.gz # scompattiamo il sorgente
cd $NAME-$VERSION # ci portiamo all'interno della directory appena scompattata
zcat $CWD/lopster-1.2.0-1.patch.gz | patch -p0 #patchamo
```

5.3CFLAGS (file SlackBuild)

Come abbiamo spiegato nel paragrafo 3.2 lanciamo il configure con le flag di ottimizzazione del gcc:

```
CFLAGS="-O2 -march=i486 -mcpu=i686" ./configure --prefix=/usr
make #Compiliamo il software
```

5.4make DESTDIR o prefix

Ora che abbiamo otimizzato e compilato il sorgente abbiamo bisogno di installarlo nella nostra directory temporanea di installazione, come fare?

Se il sorgente è dotato del Makefile ci sono 2 opzioni che ci vengono in aiuto:

make DESTDIR=Directory install

oppure:

make prefix=Directory install

Questo funziona nel 95% dei casi. Nel caso trovassimo vuota la nostra directory temporanea di installazione (o non è presente il Makefile), dovremmo copiarci tutto a mano (eseguibili e quant'altro). Il software che usiamo come esempio supporta il make prefix ...quindi procediamo come di seguito:

```
make prefix=$PKG/usr install
```

5.5inserire i File di documentazione dai sorgenti (file SlackBuild)

```
mkdir -p $PKG/usr/doc/$NAME-$VERSION # creiamo la directory per i docs
cp -a \
  AUTHORS BUGS COPYING ChangeLog INSTALL NEWS README README.pings \
  $PKG/usr/doc/$NAME-$VERSION

 #copiamo i docs in quella che sarà /usr/doc/lopster-1.2.0
```

5.6ottimizzazione (file SlackBuild)

```
strip $PKG/usr/bin/* #strippiamo tutti gli eseguibili presenti nella nostra bin/
```

nel caso di lopster non abbiamo pagine man, se incontrassimo dei software che installano le pagine man dovremmo gzipparle, per esempio:

```
gzip $PKG/usr/man/man(x)/*
```

5.7permessi e proprietari

Settiamo ora i giusti permessi per i files (in caso ci sia sfuggito qualcosa) e diamogli gli esatti proprietari:

```
chown -R root.bin $PKG/usr/bin # usr/bin/ compresa deve essere di proprietà root.bin
chmod 644 $PKG/usr/doc/$NAME-$VERSION/*
chown -R root.root $PKG/usr/doc/$NAME-$VERSION
chown -R root.root $PKG/usr/share
```

5.8la directory install/

Per gestire i pacchetti e per costruirli slackware ci permette di utilizzare una directory speciale da inserire nella directory temporanea di installazione: La directory install/
(e.g. /tmp/package-lopster/install/) Dentro questa directory possiamo inserire 2 files:

slack-desc e doinst.sh

6. slack-desk

Il file slack-desc (da inserire nella directory install/) ci permette di fornire una breve descrizione che verrà visualizzata al momento dell'installazione del pacchetto con pkgtool o con installpkg. Per convenzione il file slack-desc deve avere il seguente formato:

```
# COME EDITARE QUESTO FILE:
# Le "handy ruler" qui sotto semplificano la modifica della descrizione del
# pacchetto. Il primo carattere '|' nella prima linea deve avere subito sotto
# i duepunti ':' che sono preceduti dalla base del nome del pacchetto, e
# il carattere '|' finale a destra è il delimitatore per l'ultimo carattere
# che è possibile inserire. Dovete creare ESATTAMENTE 11 linee perchè la
# formattazione sia corretta. E' anche uso lasciare uno spazio dopo i duepunti.

       |-----handy-ruler------------------------------------------------------|
nome: descrizione
nome:
nome: descrizione.....................................
nome: descrizione.....................................
nome: descrizione.....................................
nome: descrizione.....................................
nome:
nome: descrizione.....................................
nome:
nome:
nome:
```

per lopster ad esempio:

```
# HOW TO EDIT THIS FILE:
# The "handy ruler" below makes it easier to edit a package description.  Line
# up the first '|' above the ':' following the base package name, and the '|' on
# the right side marks the last column you can put a character in.  You must make
# exactly 11 lines for the formatting to be correct.  It's also customary to
# leave one space after the ':'.

        |-----handy-ruler------------------------------------------------------|
lopster: lopster - A GTK napster client
lopster:
lopster: Lopster is a Napster Client developed in C with GTK interface.
lopster:
lopster: source Web Site: http://lopster.sourceforge.net
lopster:
lopster:
lopster:
lopster:
lopster:
lopster:
```

Ricordiamoci che il nome del pacchetto dovrà essere nome-versione-arch-build.tgz ed il file slack-desc deve contenere prima dei duepunti il nome come riportato sul pacchetto, altrimenti la descrizione nella maggior parte dei casi NON si visualizzerà.

7. doinst.sh

Il file doinst.sh (da inserire sotto la directory install/) ci permette di eseguire dei comandi (shell scripting) in caso un pacchetto necessiti di alcuni accorgimenti o modifiche a files esterni. Esso viene anche automaticamente creato/aggiornato in caso di presenza di symlink (che vengono cancellati dal pacchetto per essere successivamente ricreati durante l'installazione di quest'ultimo).

Esempio pratico:

supponiamo di voler aggiungere un utente lopster con cui eseguire il progamma.

Il nostro doinst.sh sarà così:

```
(userdel lopster ; useradd lopster -g users -u 1500 -d /home/lopster)
```

N.B.
Questo è solo un esempio di come utilizzare il file doinst.sh. L'operazione di cui sopra non và eseguita per l'installazione di lopster.

Ovviamente il file accetta qualsiasi tipo di oprazione "fattibile" in shell.

8. stesura finale

Dopo aver creato i files slack-desc e doinst.sh siamo pronti a finire il nostro SlackBuild e a creare il package.

Quindi riepilogando nella directory corrente avremo i seguenti files:

```
slack-desc # file della descrizione
doinst.sh # file dei comandi supplementari
lopster-1.2.0.tar.gz #sorgente compresso di lopster
lopster-1.2.0.patch.gz #patch gzippata per lopster
```

finiamo la stesura del nostro SlackBuild.

```
mkdir -p $PKG/install #creiamo la directory install/
cat $CWD/slack-desc > $PKG/install/slack-desc #copiamo slack-desc in install/
cat $CWD/doinst.sh > $PKG/install/doinst.sh #copiamo doinst.sh in install/

# Creiamo il pacchetto con makepkg.
cd $PKG
makepkg -l y -c n $TMP/$NAME-$VERSION-$ARCH-$BUILD.tgz
```

Le due opzioni dopo il makepkg significano:

> -l, --linkadd y|n (inseriamo i symlinks nel file doinst.sh: raccomandato)
> -c, --chown y|n (settiamo tutti i permessi aroot:root 755: non raccomandato)

quindi -l YES e -c NO

in questo modo eseguendo lo script SlackBuild installaremo il pacchetto e setteremo il tutto in /tmp/package-lopster
ed avremo il pacchetto fatto in /tmp/lopster-1.2.0-i486-1.tgz
in /tmp rimarrà anche il sorgente di lopster decompresso.

Per deletare in automatico le cose superflue (se siamo sicuri che il pacchetto sia PERFETTO) aggiungiumo un accortezza alla fine del nostro SlackBuild:

```
# Deletiamo il materiale extra:
if [ "$1" = "--cleanup" ]; then
  rm -rf $TMP/$NAME-$VERSION
  rm -rf $PKG
fi
```

In quetsto modo se lanciamo lo script con l'opzione --cleanup verranno autamaticamente cancellate le directory del pacchetto sorgente e la directory package-nomepacchetto.

Andiamo a vedere come si presenta il nostro SlackBuild Completo:

```
-------------------------------------------------CUT HERE-------------------------------------------------
#SLACKBUILD SAMPLE

#!/bin/sh
# directory temporanea e variabile directory corrente:
CWD=`pwd`
if [ "$TMP" = "" ]; then
  TMP=/tmp
fi
PKG=$TMP/package-lopster
#Settiamo qualche altra variabile per velocizzare il lavoro:
NAME=lopster
VERSION=1.2.0
ARCH=i486
BUILD=1
if [ ! -d $TMP ]; then
  mkdir -p $TMP # posizione dove creiamo il sorgente
fi
if [ ! -d $PKG ]; then
  mkdir -p $PKG # posizione dove creiamo il package
fi
echo "+=================+"
echo "| $NAME-$VERSION |"
echo "+=================+"
cd $TMP
tar zxvf $CWD/$NAME-$VERSION.tar.gz # scompattiamo il sorgente
cd $NAME-$VERSION # ci portiamo all'interno della directory appena scompattata
zcat $CWD/lopster-1.2.0-1.patch.gz | patch -p0 #patchamo
CFLAGS=”-O2 -march=i486 -mcpu=i686” ./configure --prefix=/usr
make #Compiliamo il software
make prefix=$PKG/usr install
mkdir -p $PKG/usr/doc/$NAME-$VERSION # creiamo la directory per i docs
cp -a \
  AUTHORS BUGS COPYING ChangeLog INSTALL NEWS README README.pings \
  $PKG/usr/doc/$NAME-$VERSION
strip $PKG/usr/bin/* #strippiamo tutti gli eseguibili presenti nella nostra bin/
chown -R root.bin $PKG/usr/bin # usr/bin/ compresa deve essere di proprietà root.bin
chmod 644 $PKG/usr/doc/$NAME-$VERSION/*
chown -R root.root $PKG/usr/doc/$NAME-$VERSION
chown -R root.root $PKG/usr/share
mkdir -p $PKG/install #creiamo la directory install/
cat $CWD/slack-desc > $PKG/install/slack-desc #copiamo slack-desc in install/
cat $CWD/doinst.sh > $PKG/install/doinst.sh #copiamo doinst.sh in install/
# Creiamo il pacchetto con makepkg.
cd $PKG
makepkg -l y -c n $TMP/$NAME-$VERSION-$ARCH-$BUILD.tgz
# Deletiamo il materiale extra:
if [ "$1" = "--cleanup" ]; then
  rm -rf $TMP/$NAME-$VERSION
  rm -rf $PKG
fi
-------------------------------------------------CUT HERE-------------------------------------------------
```

A questo punto nella directory corrente dovremmo avere:

slack-desc # file della descrizione
doinst.sh # file dei comandi supplementari
lopster-1.2.0.tar.gz #sorgente compresso di lopster
lopster-1.2.0.patch.gz #patch gzippata per lopster
SlackBuild # script per la creazione del pacchetto

diamo un chmod +x SlackBuild

e, come utente root lanciamolo:

#./SlackBuild

nel giro di una quantità "ridicola" di giffies abbiamo il nostro lopster-1.2.0-i486-1.tgz in /tmp

se volessimo cancellare le directory superflue:

#./SlackBuild --cleanup

9. Conclusione

9.1Risorse

I Source packages di Slackware, ed una discreta conoscenza di shell scripting.

9.2Note

Esistono ovviamente diversi modi per creare un pacchetto Slackware, in molti si sono prodigati per facilitare/facilitarsi l'utilizzo dei vari SlackBuild, Ed anche gli esempi proposti da questo DRAFT non bastano (guardatevi i Sources dei pacchetti Originali e/o di chi li crea seguendo i canoni originali), ma questo è il Modo giusto per creare i pacchetti in PERFETTO stile Slackware.

Mi sento di aggiungere che Slackware, da buon Linux si basa sulle glibc che sono il "core" del suo funzionamento, Molti software per Linux si basano su queste librerie, per cui si può dedurre che diverse versioni di glibc implicano il mancato funzionamento di un pacchetto precompilato, non dimentichiamo che anche la versione del compilatore è importante.

9.3Ringraziamenti e Note Personali

Sinceramente, per una serie di motivi NON avevo intenzione di scrivere questo DRAFT. Ciò che mi ha spinto maggiormente è stata l'arroganza di tutti quelli che hanno scritto tutorial e/o appunti ridicoli o parzialmente "errati" su questo argomento e che hanno avuto il coraggio di mettere sotto GPL l'insieme di poche conoscenze che dovrebbero essere libere e basta e non vincolate da nomi e indirizzi di mail. Dedico questo tutorial a queste persone comprese quelle che hanno avuto il coraggio di lamentarsi perchè qualcuno gli ha copiato delle parti delle "loro" (in parte inesatte) spiegazioni, magari facendo esempi con software di cui il maintainer ufficiale dei pacchetti sono io. Per onor di cronaca mi ci sono volute poco più di 4 ore per scrivere questo DRAFT, metterlo sotto un qualche "vincolo" mi sembra un insulto alla Libertà di espressione e conoscenza che è rimasta in RETE. Mi scuso fin da ora per il tono, ma sono realmente seccato dal comportamento di alcuni. Infine un particolare ringraziamento a Lorys che, con le sue domande, mi ha dato l'energia di scrivere.