Installazione dei programmi in LINUX : l'approccio RPM

3 novembre 2002

Indice

1 Introduzione

Linux , come altri sistemi operativi (Open Source e commerciali) , offre la possibilità , tramite i pacchetti , di gestire in maniera ordinata e intelligente l'installazione e disinstallazione dei programmi . I pacchetti non sono altro che degli archivi compressi dove sono immagazzinati gli eseguibili , le librerie condivise ,la documentazione , gli header per lo sviluppo, etc.. , necessari per quel determinato programma . Diversamente da altri SO Linux offre una serie di soluzioni software per la gestione dei pacchetti. In questo breve documento ci proponiamo di illustrare il funzionamento di *rpm* , un programma per la gestione dei pacchetti usato in distibuzioni diffuse come Red Hat , Mandrake e Suse , ma anche da tante altre distribuzioni.

1.1 Perché il software a "pacchetti" ?

Anche quando facciamo una installazione minima di una qualsiasi distribuzione ci possiamo rendere conto che il numero di programmi che abbiamo a disposizione è ingente ; stesso discorso sulle librerie condivise . In ambienti senza la gestione dei pacchetti per ogni singolo programma avremo dovuto eseguire una certa procedura di intallazione , eseguendo probabilmente azioni ripetitive e monotone come scegliere le directory o cose simili . Stesso discorso per la disinstallazione , con conseguente mal di testa per l'utente di fronte all'ennesimo bottone da premere. E' innegabile infatti che installare programmi in piattaforme come WinXXXX sia semplice e intuitivo , ma di certo non rapido : dopo la n-esima formattazione ci si perdono intere serate a installare i programmi preferiti (utili fino alla successiva formattazione) cercando di organizzare le cartelle in maniera intelligente per non rendere ancora piú disordinato il nostro *filesystem* .

Se invece la gestione dell' installazione fosse demandata a un programma che automaticamente sceglie dove installare , cosa installare , dovremmo aspettare solo il tempo necessario a trasferire su dico i

programmi , librerie etc.. . Ma non è solo una questione di velocità dell'installazione l'elemento che fa ritenere la gestione a pacchetti la soluzione più intelligente : l'asso nella manica è costituito dall'architettura dei pacchetti. Il singolo pacchetto in se non possiede in generale tutto ciò che serve al suo funzionamento (questo vale in generale per qualsiasi tipo di pacchetto stiamo trattando) ma parte delle librerie o dei programmi risiedono su altri pacchetti che quindi devono essere installati sulla macchina per il funzionamento del nostro pacchetto. Si ha insomma il riuso del software anche a livello binario , e questo è possibile grazie alla natura Open dei programmi per linux che consente ad uno sviluppatore di avvalersi di librerie e programmi già pronti e di studiare di questi ultimi i sorgenti e di produrre software senza dover ogni volta inventare l'acqua calda .

Il sistema viene a essere caratterizzato quindi da dipendenze fra i vari pacchetti . I prodotti commerciali invece devono includere per il loro funzionamento tutte le librerie di cui necessitano , senza dare (ma anche avere) la possibilità di riusarle a livello binario .In quest'ottica si può capire perché i programmi commerciali abbiano in genere un programma per installarli e uno per disinstallarli e perché si può scegliere dove installarli .

La gestione dei pacchetti comporta :

- riuso binario del software
- gestione pianificata del *filesystem*
- velocità di installazione e disinstallazione
- possibilità di automatizzare gli update

Questo discorso vale per qualsiasi tipo di pacchettizzazione : *rpm , deb , tgz* hanno la stessa filosofia di fondo , cioè quella di dare uno strumento automatico per la gestione dei programmi.

1.2 Pacchetti sorgenti

Come tutto il software libero anche quello a pacchetti può presentarsi sia in forma binaria che in forma di sorgente. I gestori di pacchetti si occupano in maniera ordinata anche dei sorgenti fornendo strumenti per la ricompilazione automatica . Viene sostanzialmente lanciati la triade *configure , make , make install.*

La maggior parte delle volte vengono automaticamente "patchati" e allo script configure vengono passati argomenti speciali. La comodità risiede nel fatto che possiamo ottenere un pacchetto per la nostra architettura ma che possiamo esportare anche in altre macchine (con la medesima architettura :-P). Possiamo cosí creare quasi "from scratch" la nostra distribuzione .

Non sono tuttavia tutte rose e fiori infatti perché la compilazione vada a buon fine devono essere installati tutti ma proprio tutti gli header , le librerie condivise . Non è detto che configure faccia tutti i controlli necessari , per cui può capitare errori in compilazione perché qualche include non esiste o li linker non trova una libreria ...

Sono dell'avviso che ottimizzazione che si può avere tramite la ricompilazione dei singoli pacchetti sia un ottimo modo di sfruttare la nostra architettura , e deve essere affrontata soprattutto nei pacchetti nevralgici del nostro sistema .

2 Il sistema RPM

Rpm è il sistema usato da Red Hat , Mandrake e Suse è un tool che consente la creazione , l'intallazione , l'update , la rimozione , le query ,la verifica (crittografica) dell'integrità del pacchetto stesso. Il pacchetto consiste in un archivio compresso di file e *meta-data* . I meta-data sono una serie di script di utilità , descrizione sul pacchetto , la *signature* crittografica etc.

La descrizione del pacchetto oltre a darci indicazioni sulla sua funzione , ci informa anche quali librerie o programmi fornisce e/o necessi-

ta. Come ho già detto , conoscere le dipendenze di un pacchetto significa sapere quali altri pacchetti sono necessari per effettuare Installazione. Tuttavia la dipendenza che viene richiesta non è relativa ad un pacchetto ma ad un file che sta all'interno di un altro pacchetto (magari con un nome molto diverso :-( ) e in questi casi gioca un ruolo essenziale l'esperienza , l'intuito e non ultima la pazienza.

Infatti rpm non è adatto alle persone con poca pazienza : se dovete installare un pacchetto con rpm e questo dipende da molti altri , siate certi che occorrerà installare una buona parte dei pacchetti richiesti ! Quando poi tutte le dipendenze saranno state soddisfatte il pacchetto verrà installato e configurato attraverso degli script che eseguono in modo trasparente per l'utente delle azioni che richiederebbero più tempo con una installazione manuale. Rpm si basa su un *database* in cui vengono memorizzati i pacchetti installati e le librerie o programmi forniti dai pacchetti installati. Il check delle dipendenze non viene effettuato sul filesystem ma su questo database , quindi se cancellassimo fisicamente una libreria rpm se ne potrebbe accorgere solo in fase di esecuzione degli script di configurazione del pacchetto e non durante la fase del check delle dipendenze . Rpm offre una serie di strumenti per gestire questo db.

2.1 Opzioni fondamentali di rpm

Come la maggior parte dei programmi per Linux , rpm è a linea di comando , e questa caratteristica è secondo me il suo punto di forza : è sempre possibile inserirlo all'interno di nostri script , si può usare in remoto con una console , si possono installare 10.000.000 di pacchetti con un paio di caratteri . Vediamo come si usa .....

2.1.1 Installazione e update di pacchetti

Per installare i pacchetti la linea di comando è molto semplice :

rpm -ivh Nome_del_pk_numero_1.rpm ....

Nome_del_pk_numero_n.rpm

Ora se tutte le dipendenze sono soddisfatte (speriamo !) rpm inizia a lavorare e come output dovremmo ottenere una cosa simile

1: Nome_del_pk_numero_1.rpm ##############[20%]

....

n: Nome_del_pk_numero_n.rpm ##############[100%]

questo output si ottiene con le opzioni v (*verbose)* e h (*hash)* che consiglio di mettere perché rpm normalmente non riporta le azioni che sta eseguendo .

Per fare l'upgrade di un pacchetto la linea di comando è

rpm -Uvh Nome_del_pk_numero_1.rpm ....

Nome_del_pk_numero_n.rpm

La U è in maiuscolo , fate attenzione! Io uso sempre questa linea di comando perché l'opzione i non è in grado di fare l'upgrade mentre U può fare anche il lavoro d i. E' molto conveniente usare una sola linea di comando per installare più pacchetti e nel caso aggiungere quelli che servono per soddisfare le dipendenze . Provate ad esempio a fare una installazione minima di RH e poi installate con rpm l'interfaccia grafica , vedrete che la linea di comando diventerà di 5 o 6 righe ! Ma si può esser certi che quello che abbiamo installato è lo stretto indispensabile per far andare l'interfaccia grafica .

Posto che le dipendenze siano soddisfatte è possibile che avvengano situazioni che facciano risentire rpm , le più importanti sono:

- i/o error (mai fatto una installazione su disco danneggiato ?)
- il controllo crittografico fallito (rpm corrotto)

2.1.2 Opzioni speciali

Delle volte capita di cancellare una libreria , quindi noi diligentemente prendiamo il pkg corrispondente e cerchiamo di installarlo : sorpresa ! rpm si rifiuta perché è già stato installato . Niente paura basta usare

l'opzione ***–force*** alla fine della nostra linea di comando . Questa opzione rimuove il pacchetto e lo reinstalla . E' comodo in certe situazioni tipo

rpm -Uvh *.rpm –force

dove magari abbiamo già installato un po di pacchetti (ma non ci ricordiamo quali) , con l'opzione –force re-installiamo anche quelli (male non fa).

Può accadere anche di installare un programma che richiede una versione vecchia di qualche libreria . In questo caso è conveniente creare dei link simbolici alla nostra libreria con il nome della vecchia , e usare l'opzione ***–nodeps*** che ci consente di installare senza controllare le dipendenze. Questo trucco funziona 99 volte su 100 , ma se non funziona occorre procurarsi la versione sorgente del programma e provare a ricompilare. Oppure , extrema ratio, procurarsi la libreria che richiede e installarla (deprecated).

2.1.3 Opzione pericolose (sigh !)

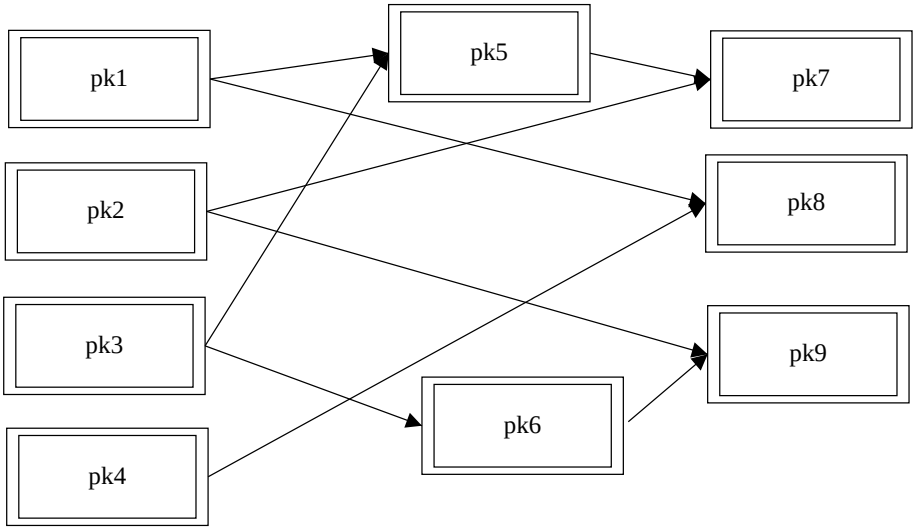
Ci sono alcune opzioni che un utente normale che ci tenga alla propria macchina non dovrebbe mai usare , facciamo una piccola panoramica :

- ***–ignoresize*** non controlla lo spazio disponibile , del tipo binari a metà
- ***–ignoreos*** non controlla se il sistema operativo va bene per quel pacchetto
- ***–ignorearch*** non controlla l'architettura , provate poi ad eseguire un binario per ppc su un x86
- ***–nomd5*** non controlla se il pacchetto è corrotto , cosi se ci va bene otteniamo dei seg. fault bizzarri

E' ovvio che queste opzioni vanno usate in caso in situazioni particolari . Nell'uso quotidiano su i pc desktop non servono.

2.1.4 Rimozione di pacchetti

La rimozione dei pacchetti se non presenta difficoltà per la linea di comando, offre qualche spunto di riflessione sulla struttura ramificata delle dipendenze . Consideriamo il seguente grafico della dipendenze:



Vediamo che se volessimo disinstallare *pk3* dovremo disinstallare anche *pk6* e *pk5* , ma anche *pk9* dipendente da *pk6* e *pk7* dipendente da *pk5* : quando disintalliamo un pacchetto dobbiamo eliminare tutta la sottocatena di dipendenze. Se per esempio volessimo disinstallare XFree86-libs , probabilmente dovremo disinstallare più del 60% dei pacchetti. E' ovvio che un pacchetto cosi fondamentale come XFree-libs è richiesto da una quantità enorme di pacchetti . Discorso diverso è ad esempio per i front-end di qualche programma : Xcdroast , gcombus etc.. sono solo delle interfacce grafiche per cdrecord , e una loro eventuale rimozione non richiede altre disinstallazionivediamo in .vediamo in pratica la linea di comando :

rpm -e pk_n_1 ... pk_n_m

Ricordatevi di riportare oltre al nome del pacchetto anche il suo numero di versione . E' importante ,altrimenti non trova il pacchetto nel db . Non si possono usare le opzioni –force o –nodeps (sarebbe troppo pericoloso) e ci vuole un po di pazienza per disinstallare pacchetti molto rinomati (provate a eliminare le *zlib*).

2.2 Le query

Come in tutti i database che si rispettino anche quello di rpm consente le *query* ovvero delle interrogazioni sui dati contenuti all'interno. Il db di rpm fornisce informazioni sul contenuto dei pacchetti , sul loro funzionamento , sulle librerie e programmi che offrono o richiedono , su gli eventuali conflitti con pacchetti simili.

2.2.1 Opzioni principali sulle query

Per effettuare una query occorre usare l'opzione ***-q*** seguita da le altre opzioni per specificare il tipo di query. Queste sono

- ***a*** fornisce le informazioni su tutti i pacchetti installati
- ***i*** fornisce le informazioni sul funzionamento del pacchetto
- ***l*** fornisce le informazioni sui file che compongono il pacchetto
- opzione lunga ***–requires*** mostra le librerie e i programmi richiesti (utile per le disinstallazione)
- opzione lunga ***–provides*** mostra le librerie e i programmi forniti (utile per le installazioni)

Forniamo qualche esempio :

1. ***rpm -qi Nome_pk_1*** questo comando ci da informazione sul pacchetto richiesto
2. ***rpm -qil Nome_pk_2*** questo comando ci da informazione sul pacchetto richiesto e sul suo contenuto
3. ***rpm -qail | grep -B 50 file_ricercato | more*** questo comando fornisce per ogni pacchetto l'informazioni sul contenuto e sulla sua funzione; grep filtra l'output lasciando passare "solo" le 50 righe prima a quella dove compare la stringa file_ricercato ; more ci fa vedere le cose pian piano (utile quando si vuole scoprire in che pacchetto sta un certo file...)

4. ***rpm -q –provides Nome_pk_3*** ci da i programmi e/o librerie esportate

3 Rpmbuild e i pacchetti sorgenti

Come avevamo già accennato, i sistemi di gestione dei pacchetti prevede una gestione intelligente e ordinata dei sorgenti ; anche rpm presenta una soluzione per questo problema : rpmbuild è lo script che permette di effettuare la compilazione in maniera semplice dando la possibilità di costruire un pacchetto pronto ad essere installato o eventualmente ridistribuito su altre macchine.

I pacchetti sorgenti sono riconoscibili attraverso l'estensione .srpm o .src.rpm ; prima di ricompilare i pacchetti occorre installarli come un pacchetto qualsiasi , successivamente si deve andare nella cartella dove risiedono i sorgenti installati (questo dipende dalla distribuzione usata , in Red Hat sono nella cartella /usr/src/redhat/) e in particolare nella sottocartella SPECS dove sono contenuti i *file spec* che sono per rpmbuild l'equivalente dei makefile per make. Ricordiamo che rpmbuild non è un compilatore ma uno script che esegue i comandi contenuti nel file spec , e quindi per poterlo usare dobbiamo avere installati il gcc il make etc.. .

La sintassi è molto semplice :
rpmbuild -bb file.spec per creare solo il pacchetto
rpmbuild -bi file.spec per creare il pacchetto e installarlo

Consiglio di usare l' opzione lunga ***–clean*** che cancella i file temporanei dentro la cartella BUILD parallela a SPECS

E' anche possibile ottimizzare per la nostra architettura con l'opzione ***–target=my_arch-linux-my_dist*** che genererà a termine compilazione un file con estensione per esempio .586.rpm (se usiamo questa ottimizzazione) . I Pacchetti binari Vengono messi nella directory RPMS/*my_arch* ; se non specifichiamo il tipo di architettura viene

compilato per 386 , come sono ad esempio i pacchetti della Red Hat. Consiglio ,

per chi volesse effettuare una installazione quasi from scratch per ottimizzare i pacchetti rpm per la sua architettura di seguire l'ordine dei pacchetti suggerito nel legendario HOW TO *"linux from scratch"*. per le altre persone consiglio di ricompilare le Glibc ,le Binutils , bash , gcc , make , zlib , Xfree e i pacchetti di Gnome o KDE (a seconda delle preferenze) .

4 Conclusioni

Usando rpm si ha la sensazione a volte che all'aumentare dei pacchetti le dipendenze non soddisfatte anzichè diminuire aumentino ... Rpm è sicuramente un buon sistema per la gestione dei pacchetti ma non il migliore . Occorrerebbe avere la possibilità di fare quello che succede in fase di installazione , cioè gli script si accorgono delle dipendenze non soddisfatte e installano i pacchetti necessari . Sarebbe bello , a linea di comando mettere su un sistema senza X scrivere *rpm -Uvh xbill.xxx.i386.rpm* e il sistema installa X i font etc...

Esistono interfacce grafiche che fanno tutto ciò ma sarebbe bello avere uno strumento a linea di comando che faccia tutto questo. Se abbiamo solo una connessione remota alla nostra macchina e per giunta via modem è improponibile voler usare *xhost* per poter visualizzare la finestra dell'applicazione grafica. Altri sistemi , come i ports di FreeBSD , il tool ap-get etc.. di Debian si preoccupano di questo . In parte Mandrake con *urpmi* ha corretto la rotta , ma siamo ancora lontani. Per la ricompilazione dei pacchetti siamo ancora più distanti dagli esempi riportati prima , e non si capisce perche non sia possibile come FreeBSD effettuare un *make world* per sfruttare fino all'osso il nostro hardware . Il fatto che rpm non sia automatico può essere un bene perché ci resposabilizza e ci fa conoscere la funzione di tanti pacchetti di cui spesso ignoriamo l'esistenza , ma credo che un compromesso sia

possibile . Esistono delle risorse per l'utente che non sa dove andare a pescare un rpm per risolvere una dipendenza : http://rpmfind.net [1] è obbligatorio per chi non voglia impazzire per qualche installazione manuale . Esistono software come [2]Red Carpet o il servizio Up2date di Red Hat che aiutano molto l'utente non troppo paziente .

Riferimenti bibliografici

[1] Rpmfind è un motore di ricerca per rpm , si possono impostare numerose chiavi di ricerca che garantiscono sempre di trovare quello di cui si ha davvero bisogno

[2] Red Carpet è un programma per gestire in maniera "visuale" update e installazioni . Si trova sul sito www.ximian.com , come sempre con licenza GPL , è disponibile per diverse distribuzioni rpm-based