

Gestione del software

- **Ciclo di vita**
 - installazione
 - aggiornamento
 - disinstallazione
- **Problematiche**
 - prerequisiti hardware/s.o.
 - dipendenze da/di altri componenti software
 - configurazione

1

Installazione manuale

- **Da binari**
 - semplice copia nei "posti giusti"
 - verifica manuale della compatibilità con l'architettura
 - verifica manuale del soddisfacimento delle dipendenze
- **Da sorgente**
 - necessità di compilazione
 - indipendenza dall'architettura
 - possibile maggior flessibilità nel soddisfacimento delle dipendenze

2

Installazione manuale

- Dipendenze del componente software da altri
 - Nel caso di un'installazione da binari, probabile necessità di disporre non solo dei software indicati come prerequisiti, ma anche che essi siano di una versione specifica
 - Nel caso di installazione da sorgente, qualche grado di flessibilità (possibilità che i sorgenti dispongano di diverse interfacce per adeguarsi a cosa si trova sul sistema)
 - Necessità di disporre non solo dei componenti runtime relativi ai software richiesti, ma anche delle librerie di sviluppo (prototipi, interfacce, librerie per collegamento statico, ...)
 - In un sistema “ideale” ho tutti i sorgenti per cui dispongo sempre di tutti questi elementi
 - Nelle distribuzioni, per flessibilità, ogni pacchetto software ha un corrispondente pacchetto -dev o -devel (vedi prossime slide)
-

3

Installazione manuale tipica in Linux

- Il caso più comune è quello di software
 - distribuito per mezzo di un archivio tar.gz
 - scritto in C
 - predisposto alla compilazione tramite autoconf
 - verifica se sono soddisfatti tutti i prerequisiti
 - rileva le versioni ed le collocazioni dei pacchetti sul sistema
 - accetta dall'utente la specifica di varianti (attivazione/disattivazione di funzionalità, preferenze architetturali, ...)
 - genera i Makefile sulla base delle specificità del sistema e delle scelte operate dall'utente
-

4

Installazione manuale tipica in Linux

■ I passi tipici quindi sono:

- reperimento del software
- estrazione del pacchetto
- esame delle scelte disponibili
- configurazione dei sorgenti
- compilazione
- installazione

- **NOTA:** solo quest'ultima operazione può richiedere i diritti di superutente, e quindi si deve evitare di compiere le precedenti come *root*. Sono noti casi di malware che sfruttano proprio la cattiva abitudine di eseguire una o più delle operazioni preliminari con diritti eccessivi.

5

Installazione manuale tipica in Linux

■ estrazione del pacchetto

- solitamente si presenta come archivio tar compresso
- è buona prassi determinare una collocazione sensata per i sorgenti ed estrarre in tale directory l'archivio
 - nel caso si stia per affrontare un upgrade sostanziale del sistema, che coinvolga numerose applicazioni, può essere utile raccogliere in modo più chiaro tutti i pacchetti che verranno installati unitariamente
- è prudente testare l'archivio prima dell'estrazione per verificare la gerarchia di directory che genera

- **Es:**

```
cd /usr/local/src
tar tvzf net-snmp-5.4.tar.gz
tar xvzf net-snmp-5.4.tar.gz
```

6

Installazione manuale tipica in Linux

- esame delle scelte disponibili
 - si entra nella directory generata dall'estrazione e si esamina il contenuto
 - è bene leggere i file README ed INSTALL che di solito accompagnano il software
 - se esiste un eseguibile di nome `configure` lo si lancia con il parametro `--help` per ottenere la lista dei parametri di configurazione disponibili
 - scelte comuni riguardano la collocazione del software, l'attivazione o la disattivazione di sottocomponenti, la predisposizione dei componenti attivati come moduli dinamicamente caricabili piuttosto che la loro integrazione statica nel codice, ...
-

7

Installazione manuale tipica in Linux

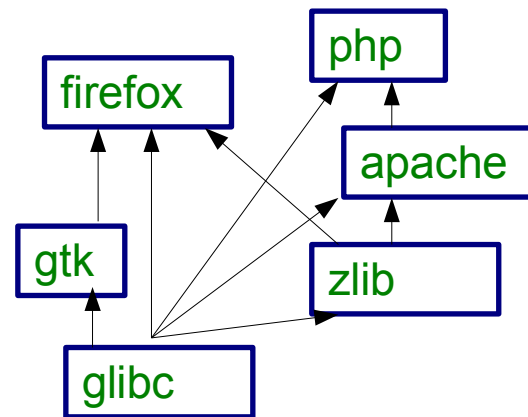
- configurazione dei sorgenti
 - si lancia nuovamente `configure` con i parametri scelti
 - si risolvono i problemi evidenziati da `configure` (tipicamente assenza di pacchetti necessari come prerequisiti)
 - `configure` non è a prova d'errore,
 - compilazione
 - si lancia `make` o si seguono le indicazioni presenti nell'output generato dal passo precedente
 - installazione
 - si lancia `sudo make install`
-

8

Installazione assistita

- Comunemente effettuata per mezzo di software ausiliari
 - package manager specifico della distribuzione Linux (rpm/yum, dpkg/apt, ...)
 - installer per Windows
- Un tool di installazione
 - può farsi carico delle verifiche relative alle dipendenze
 - non può configurare ogni dettaglio del sistema in modo specifico
 - può generare dinamicamente dati specifici

Esempio di grafo delle dipendenze:



A → B significa che A “serve” per B; “serve” può essere una dipendenza tra funzionalità logiche (non ha senso avere un linguaggio di generazione pagine web senza un web server) o fisiche (un binario linkato dinamicamente non gira senza tutte le librerie di cui importa i simboli)

9

Pacchetti

- Le *distribuzioni* di Linux organizzano il software in *pacchetti* e dispongono di un *package manager* per la loro gestione
- Un pacchetto si presenta sotto forma di singolo file che contiene in forma compatta l'insieme di
 - software precompilato
 - criteri per la verifica della compatibilità e dei prerequisiti
 - procedure di pre/post-installazione
- La garanzia della compatibilità con un determinato sistema può essere data solo a patto di vincolare con precisione alcuni parametri:
 - architettura
 - versione della distribuzione
 - versione del software contenuto nel pacchetto

10

Debian e Red Hat

- Due distribuzioni capostipite da cui sono state derivate quasi tutte le varianti più diffuse
<http://upload.wikimedia.org/wikipedia/commons/9/9a/Gldt1009.svg>
- Due sistemi di gestione dei pacchetti con molte somiglianze
 - Tool di basso livello per la gestione dei singoli pacchetti
 - Tool intermedi per la gestione coordinata di pacchetti e dipendenze
 - Tool per il reperimento automatico da *repository* dei pacchetti necessari

11

Pacchetti

- I pacchetti per le distribuzioni Debian e derivate (es. Ubuntu) sono in formato *.deb*

– **aptitude**-0.2.15.9-2_**i386**.deb



- I pacchetti per le distribuzioni RedHat e derivate (es. CentOS, Fedora) sono in formato *.rpm*

– **httpd**-2.4.6-45.el7.centos.**x86_64**.rpm

12

Gestione dei pacchetti .deb

database location: /var/lib/dpkg, /var/lib/apt
sources file: /etc/apt/sources.list
update sources: apt-get update
key management: apt-key
search: apt-cache search keywords
install: dpkg -i filename.deb
apt-get install packagenames
upgrade: apt-get upgrade [packagenames]
remove: dpkg -r packagename
apt-get remove packagenames

13

Gestione dei pacchetti .rpm

database location: /var/lib/rpm
sources file: /etc/yum.conf
update sources: yum update
key management: rpm --import keyfile
search: yum search keywords
install: rpm -i filename.rpm
yum install packagenames
upgrade: yum upgrade [packagenames]
verify integrity: rpm -V [packagenames[a]
remove: rpm -e packagenames
yum remove packagenames

14

deb e rpm

■ Link per deb

<http://www.debian.org/doc/manuals/debian-reference/ch02.en.html>
http://guide.debianizzati.org/index.php/Introduzione_all'_Apt_System

■ Link per rpm

<http://yum.baseurl.org/wiki/YumCommands>
<http://yum.baseurl.org/wiki/RpmCommands>

15

Esempio di pacchetti base e development

■ zlib1g

- /usr/lib/libz.so.1.2.3.3

per ogni funzione, es. *compress*:

codice oggetto in formato adatto per il linking dinamico

■ zlib1g-dev

- /usr/lib/libz.a

codice oggetto in formato adatto per il linking statico

- /usr/include/zconf.h

- /usr/include/zlib.h

prototipo per il compilatore

- /usr/include/zlibdefs.h

Con questa suddivisione si risparmia (molto) spazio sui sistemi che non sono usati per *sviluppare* codice basato su questa libreria, nei quali il primo pacchetto fornisce da solo il necessario per *usare* codice già pronto in forma binaria che referencia le funzioni della libreria

■ Su sistemi *deb* → pacchetti “-dev”

■ Su sistemi *rpm* → pacchetti “-devel”

16

Esempio di verifica delle dipendenze dinamiche

■ Idd /usr/sbin/sshd

```
linux-gate.so.1 => (0xffffe000)
libwrap.so.0 => /lib/libwrap.so.0 (0xb7ef7000)
libpam.so.0 => /lib/libpam.so.0 (0xb7eed000)
libdl.so.2 => /lib/tls/i686/cmov/libdl.so.2 (0xb7ee8000)
libseldlinux.so.1 => /lib/libseLinux.so.1 (0xb7ed2000)
libresolv.so.2 => /lib/tls/i686/cmov/libresolv.so.2 (0xb7ebf000)
libcrypto.so.0.9.8 => /usr/lib/i686/cmov/libcrypto.so.0.9.8 (0xb7d7c000)
libutil.so.1 => /lib/tls/i686/cmov/libutil.so.1 (0xb7d78000)
libz.so.1 => /usr/lib/libz.so.1 (0xb7d63000)
libnsl.so.1 => /lib/tls/i686/cmov/libnsl.so.1 (0xb7d4a000)
libcrypt.so.1 => /lib/tls/i686/cmov/libcrypt.so.1 (0xb7d1c000)
libgssapi_krb5.so.2 => /usr/lib/libgssapi_krb5.so.2 (0xb7cf3000)
libkrb5.so.3 => /usr/lib/libkrb5.so.3 (0xb7c6b000)
libk5crypto.so.3 => /usr/lib/libk5crypto.so.3 (0xb7c46000)
libcom_err.so.2 => /lib/libcom_err.so.2 (0xb7c43000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb7af8000)
/lib/ld-linux.so.2 (0xb7f11000)
libsepol.so.1 => /lib/libsepol.so.1 (0xb7ab7000)
libkrb5support.so.0 => /usr/lib/libkrb5support.so.0 (0xb7aaf000)
libkeyutils.so.1 => /lib/libkeyutils.so.1 (0xb7aad000)
```

17

Problematiche di aggiornamento

- Quando si aggiorna un pacchetto software già in uso sul sistema, si deve tener conto di potenziali problemi derivanti da:
 - **prerequisiti**
 - pacchetti che devono esistere perchè il candidato funzioni bene
 - come nel caso dell'installazione
 - potrebbe non essere facile aggiornare i pacchetti-prerequisiti senza causare problemi ad altri software che li utilizzano
 - **configurazione**
 - eventuali modifiche incompatibili apportate al formato delle direttive di configurazione già messe a punto per la versione funzionante
-

18

Problematiche di aggiornamento

■ (continua)

- dipendenze di altri software e test di non regressione
 - modifiche apportate alle interfacce o alle funzionalità del software potrebbero influire sul funzionamento di altri software
 - *configurazione del PATH* per impostare l'ordine di ricerca degli eseguibili nelle directory
 - predisposizione di configurazioni di test per far coesistere le due versioni durante le fasi di verifica
 - es. binding a socket, porte, IP diversi --> problemi di trasparenza per l'utente, licenze, configurazione delle controparti se il software da testare interagisce attraverso interfacce standard

19

Problematiche di aggiornamento

- (continua dipendenze e test)
 - *configurazione del loader* per far convivere differenti versioni di librerie dinamiche, si veda la man page `ld(1)`, specialmente le sezioni sui parametri `-rpath` e `-rpath-link`
 - modifica dei settaggi di default in `/etc/ld.so.conf` (da applicare con `ldconfig`)
 - uso delle variabili `LD_LIBRARY_PATH` in fase di loading e `LD_RUN_PATH` in fase di linking

Esempio:

```
# ldd /usr/sbin/sshd
...
libz.so.1 => /usr/lib/libz.so.1 (0xb7e0c000)
...
# export LD_LIBRARY_PATH=/usr/local/lib
# ldd /usr/sbin/sshd
...
libz.so.1 => /usr/local/lib/libz.so.1 (0xb7dab000)
```

20

Disinstallazione

- Presenta gli stessi problemi dell'aggiornamento in termini di eventuale dipendenza di altri software da quello che si sta per rimuovere
 - in entrambi i casi può essere molto difficile prevedere gli effetti sul sistema se la gestione è manuale
 - il *grafo delle dipendenze* è quindi il valore aggiunto più significativo dei sistemi a pacchetti
 - può essere molto utile sfruttare la possibilità offerta dai package manager di creare i propri pacchetti, per gestire il software installato manualmente tramite il sistema automatico di verifica delle dipendenze (ma ciò significa censirle con precisione all'installazione)

21

Distribuzioni: criteri per la scelta

Architetture supportate

- tutte le distribuzioni supportano i processori Intel 32bit, la maggior parte quelli a 64bit, alcune sono disponibili per tutte le varietà di processori su cui è stato portato il kernel
- È bene ricordare che i pacchetti di terze parti potrebbero non essere disponibili per tutte le architetture supportate

Stabilità vs. Aggiornamento

- il processo di rilascio frequente e continuo del software nel mondo GNU/Linux ha come conseguenza inevitabile che le versioni più aggiornate possano essere meno stabili
- vi sono distribuzioni che hanno come filosofia l'inclusione dei pacchetti più recenti (e quindi con funzionalità maggiori) anche a costo di una minor robustezza, ed altre che garantiscono l'inclusione solo di software ben collaudato

22

Distribuzioni: criteri per la scelta

Supporto e durata

- La disponibilità di supporto garantito è tipica delle distribuzioni commerciali, ma anche con le distribuzioni gratuite più diffuse, in virtù della dimensione della relativa comunità di utenti, è semplice risolvere eventuali problemi
- Per installazioni di tipo server esistono varianti denominate LTS (Long Term Support): per 5/7 anni chi cura la distro garantisce che gli aggiornamenti non modifichino le API (tipicamente viene garantito solo il backporting dei security fix, non quello di tutti i bug fix)

Ampiezza del set di pacchetti

- Si va dai 1500 delle distro minimali ai 26000 di Debian
- Una scelta intelligente mette tutto l'essenziale in 1 CD

23

Lavorare coi repository

- Un'esigenza molto comune è quella di installare software ben supportato ma non incluso per qualsiasi motivo nei canali ufficiali della distribuzione
- Si aggiunge semplicemente il repository all'elenco
 - Apt (deb):

```
/etc/apt/sources.list.d/virtualbox.list :  
deb http://download.virtualbox.org/virtualbox/debian xenial contrib
```
 - Yum (rpm):

```
/etc/yum.repos.d/epel.repo :  
[epel]  
name=Epel Linux -  
baseurl=http://mirror.example.com/repo/epel5_x86_64  
enabled=1  
gpgcheck=0
```

24

Gestire la provenienza dei pacchetti

- Si può generare confusione se un pacchetto con lo stesso nome è presente in versioni diverse in repository differenti
- I package manager, di default, scelgono sempre la versione più avanzata
- In alcuni casi anche aggiornamenti nello stesso repo sono indesiderabili
- La situazione va controllata e gestita
 - Controllo della provenienza di un pacchetto
 - Yum: `repoquery -i [package name]`
 - Apt: `apt-cache showpkg [package name]`
 - Elenco dei pacchetti provenienti da un repo
 - Yum: `yum list installed | grep [repo name]`
 - Apt: vari comandi per estrarre manualmente info dai file della cache

25

Limitare le modifiche automatiche

- Per evitare a priori problemi in sistemi con dipendenze complesse (ad esempio mix di pacchetti installati manualmente e via package manager)
 - Version locking/pinning
 - Apt
editare `/etc/apt/preferences.d/*`
<https://wiki.debian.org/AptPreferences>
 - Yum
`yum install yum-plugin-versionlock`
poi
`yum versionlock [package name]`
o editare a mano
`/etc/yum/pluginconf.d/versionlock.list`

26