Gestione del software

Installazione, aggiornamento, e controllo dei servizi

Marco Prandini

Gestione del software

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

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

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)

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

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.

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
```


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, ...

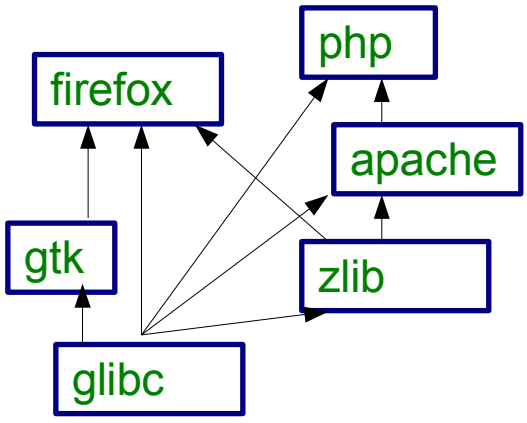
Installazione manuale tipica in Linux

- 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, può servire un'indagine manuale a volte complessa
- compilazione
  - si lancia `make` o si seguono le indicazioni presenti nell'output generato dal passo precedente
- installazione
  - si lancia `sudo make install`

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)

Pacchetti

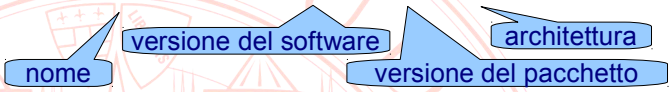
- Le *distribuzioni* di Linux organizzano il software in *pacchetti* e dispongono di un *package manager* per la loro gestione
- Un pacchetto si presenta sotto forma di singolo file che contiene in forma compatta l'insieme di
  - software precompilato
  - criteri per la verifica della compatibilità e 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

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/GIdt1009.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

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

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`

(i suffisssi `-get` e `-cache` possono essere omessi nelle distribuzioni più recenti, in cui il comando `apt` regge tutti i sotto-comandi come `search`, `update`, `install`, ...)

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`

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

Esempio di pacchetti base e development

- zlib1g
  - /usr/lib/libz.so.1.2.3.3

- zlib1g-dev
  - /usr/lib/libz.a
  - /usr/include/zconf.h
  - /usr/include/zlib.h
  - /usr/include/zlibdefs.h

per ogni funzione, es. *compress*:

codice oggetto in formato adatto per il linking dinamico

codice oggetto in formato adatto per il linking statico

prototipo per il compilatore

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 referenzia le funzioni della libreria

- Su sistemi *deb* → pacchetti "-dev"
- Su sistemi *rpm* → pacchetti "-devel"

Esempio di verifica delle dipendenze dinamiche

- `ldd /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)
libselinux.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)
```

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

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

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
    - Es:

```
# 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)
```


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)

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

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

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
    ```


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

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`

Build your own repo (rpm)

- I package manager sono molto utili per "tenere in ordine"
- È sconsigliabile mischiare installazioni manuali con pacchetti
- Non è difficile pacchettizzare le proprie applicazioni!
- Nel mondo RPM
  - si configura un ambiente di build per un utente (non root!)
  - si preparano i sorgenti e tutti i file che devono essere inclusi in un pacchetto
  - si effettua il build del pacchetto
  - lo si carica su di un server web
  - si generano gli indici che rendono riconoscibile il sito come repository

Build your own repo (rpm)

- Per il build, si configura un ambiente per un utente (non root!)
  - file `~/.rpmmacros`

```
%packager       Marco Prandini <marco.prandini@unibo.it>
%vendor         DISI
%_topdir        /home/prandini/rpmbuild
%_signature     gpg
%_gpg_name      Marco Prandini (Unibo) <marco.prandini@unibo.it>
```

  - devono essere presenti alcune cartelle sotto `_topdir`

`SPECS` contiene i file di specifica dei pacchetti
`SOURCE` contiene i sorgenti da compilare/includere
`BUILD` contenitore per il pacchetto "aperto"
`SRPMS` destinazione dei pacchetti sorgente
`RPMS` destinazione dei pacchetti binari

Build your own repo (rpm)

- Un file di specifiche contiene tra le altre cose
  - elenco dei sorgenti `Souces` o `Sources0`, `Sources1`...
    - devono essere in SOURCES, in formato .tar.gz, e contenere una directory di primo livello con lo stesso nome del file
  - `Requires`
    - indica le dipendenze esplicite, rpm fa da solo l'elenco delle dipendenze implicite (esamina i binari ed include tutte le librerie necessarie a lanciarli)
  - `BuildRequires`
    - indica i pacchetti che servono per fare il build del pacchetto
  - `BuildRoot` è dove il build viene eseguito (possibile fonte di rischio visto che è parametrizzato)
- Si usa `rpmbuild -ba --rmsource --sign test.spec`
  - genera `RPMS/architettura/test-1.0-1.noarch.rpm` e `SRPMS/...`
  - rimuove i sorgenti originali
  - per ripristinare i sorgenti basta reinstallarli con `rpm -Uh SRPMS/test…`
- Si copia il file rpm sul server web
  - ad esempio in `/var/www/repos/myrepo/RPMS`
  - il repo sarà visibile dai client con `baseurl=http://server/myrepo`

e vi si lancia
  - `createrepo /var/www/repos/myrepo/RPMS`

Sistemi modulari

- Da approfondire
- Ubuntu snap
- Container → Docker

Gestione dei servizi in Linux

Gestione dei processi

- Dopo un'installazione "minimale"...

```
milk:~# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1948   468 ?        Ss   May15   0:02 init [2]
[... kernel processes ...]
root      1753  0.0  0.0   2704   392 ?        S<s  May15   0:00 udevd --daemon
daemon    2953  0.0  0.0   1688   408 ?        Ss   May15   0:00 /sbin/portmap
root      3231  0.0  0.0   1624   568 ?        Ss   May15   0:26 /sbin/syslogd
root      3237  0.0  0.0   1576   340 ?        Ss   May15   0:00 /sbin/klogd -x
bind      3251  0.0  0.1  39732  1964 ?        Ssl  May15   0:00 /usr/sbin/named
root      3266  0.0  0.0  39500   944 ?        Ssl  May15   0:00 /usr/sbin/lwres
root      3339  0.0  0.0   1572   444 ?        Ss   May15   0:00 /usr/sbin/acpid
103       3344  0.0  0.0   2376   760 ?        Ss   May15   0:00 /usr/bin/dbus-d
106       3352  0.0  0.1   6116  1972 ?        Ss   May15   0:03 /usr/sbin/hald
root      3353  0.0  0.0   2896   716 ?        S    May15   0:00 hald-runner
106       3359  0.0  0.0   2016   472 ?        S    May15   0:00 hald-addon-acpi
106       3367  0.0  0.0   2020   480 ?        S    May15   0:00 hald-addon-keyb
root      3387  0.0  0.0   1808   360 ?        S    May15  14:15 hald-addon-stor
root      3414  0.0  0.0   1864   396 ?        Ss   May15   0:00 /usr/sbin/dhcdb
root      3421  0.0  0.1   3984  1164 ?        Ss   May15   0:00 /usr/sbin/Netwo
avahi     3433  0.0  0.1   2936  1424 ?        Ss   May15   4:14 avahi-daemon: r
avahi     3434  0.0  0.0   2552   180 ?        Ss   May15   0:00 avahi-daemon: c
root      3441  0.0  0.0   2908   536 ?        Ss   May15   0:00 /usr/sbin/Netwo
root      3457  0.0  0.0   1752   452 ?        Ss   May15   0:02 /usr/sbin/inetd
root      3477  0.0  0.0   4924   512 ?        Ss   May15   0:02 /usr/sbin/sshd
ntp       3507  0.0  0.0   4144   764 ?        Ss   May15   0:00 /usr/sbin/ntpd
root      3521  0.0  0.0   1976   724 ?        Ss   May15   0:02 /sbin/mdadm --m
daemon    3540  0.0  0.0   1828   280 ?        Ss   May15   0:00 /usr/sbin/atd
root      3547  0.0  0.0   2196   720 ?        Ss   May15   0:00 /usr/sbin/cron
root      3590  0.0  0.0   1572   372 tty2     Ss+  May15   0:00 /sbin/getty 384
root      3591  0.0  0.0   1576   372 tty3     Ss+  May15   0:00 /sbin/getty 384
root      3592  0.0  0.0   1572   372 tty4     Ss+  May15   0:00 /sbin/getty 384
root      3593  0.0  0.0   1572   372 tty5     Ss+  May15   0:00 /sbin/getty 384
root      3595  0.0  0.0   1576   372 tty6     Ss+  May15   0:00 /sbin/getty 384
```


Gestione dei processi

- Anche se tutti questi processi fossero utili, sarebbe importante
  - sapere che origine hanno
  - sapere come terminarli, evitando che ricompaiano
  - **processi inutili non consumano solo risorse, offrono anche opportunità di attacco**
- Banali e fondamentali:
  - *man* è il vostro migliore amico, seguito da Internet.
  - *ps*, *top*, *kill*, ... sono efficaci per individuare e risolvere problemi istantanei, ma non garantiscono che non si ripresenteranno
- Ci sono tre fonti primarie di processi (oltre agli utenti)
  - Pianificatori periodici e sporadici
  - Demoni di gestione degli eventi
  - **Procedure di avvio del sistema**

Esecuzioni pianificate

- L'esecuzione periodica di programmi è compito di *cron*
  - ogni utente ha la propria *cron table* (*crontab*),
    - guardate in `/var/spool/cron` per trovarle
  - i task di sistema sono spesso raccolti in `/etc/crontab`
    - tipicamente preconfigurato per l'esecuzione di script a periodicità di uso comune
    - `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly`, `/etc/cron.monthly`
  - `/etc/crontab` si può editare direttamente, per le tabelle utente meglio usare

    `crontab -e [-u username]`

- L'esecuzione singola in un istante preciso è compito di `at`
  - `atq` per elencare i job in attesa
  - `atrm` per rimuoverli

Event managers / IPC systems

- Dbus è un'architettura di Inter-Process Communication
  - Nata per uniformare la comunicazione tra elementi delle interfacce desktop
  - Curiosate in `/etc/dbus-1/` per vedere i file di configurazione
  - In `/etc/dbus-1/event.d` sono collocati gli script di avvio dei sottosistemi gestiti
- Udev ha rimpiazzato devfs come **event manager** per la creazione istantanea dei device special file quando un nuovo dispositivo viene connesso; ora è parte di systemd
  - In `/etc/udev/rules.d` sono configurate le regole evento → azione
  - Es. `70-persistent-net.rules`

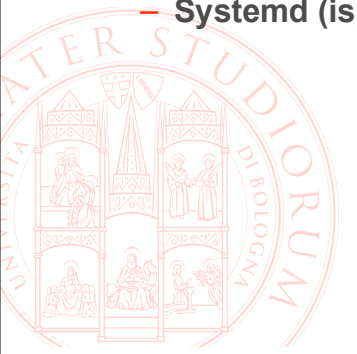
```
# PCI device 0x10ec:0x8168 (r8169)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="d0:67:e5:18:d9:e4", ATTR{dev_id}=="0x0", ATTR{type}=="1",
KERNEL=="eth*", NAME="eth0"
```

alla comparsa nel subsystem net di una scheda con MAC=d0:67:e5:18:d9:e4 le assegna nome eth0

Inizializzazione e attività in background (demoni)

- *init* è il primo processo avviato dal kernel
  - Gestisce i *runlevel*
    - stati di funzionamento del sistema definiti dal sottoinsieme di servizi attivi
  - Orchestra la sequenza corretta di eventi per raggiungere un runlevel
  - Intercetta e gestisce alcuni eventi
    - es. ctrl-alt-canc, terminazione anomala di processi,
  - Spegne il sistema in modo ordinato
- Tre varianti principali
  - (storico) SystemV-style initialization
  - Upstart (Canonical, 2006-2014)
  - Systemd (ispirazione RedHat, 2010-oggi)

utile da conoscere per l'attuale mix imprevedibile di distribuzioni moderne e tradizionali



sysvinit

- `/sbin/init` dell'originale SystemV Unix
  - configurato dal file `/etc/inittab`
  - *inittab* specifica il default runlevel
    - `id:2:initdefault:`
  - ma se la keyword `single` viene passata come parametro al kernel dal boot loader, questo settaggio è scavalcato e il sistema parte in ***single user mode*** (runlevel 1)
    - `~~:S:wait:/sbin/sulogin`
  - *init* avvia i virtual terminal e i gestori delle console su linea seriale (può sembrare un arcaismo, ma nel mondo IoT è tornato alla ribalta)
    - `1:2345:respawn:/sbin/getty 38400 tty1`
    - `2:23:respawn:/sbin/getty 38400 tty2`
    - `...`
    - `T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100`
    - `T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttyS3`

processi avviati da sysvinit

- *init* è in senso astratto responsabile per tutti i processi che girano sul sistema, ma in particolare due attività possono essere direttamente ricondotte ad esso
  - linee del tipo

    ```
    l0:0:wait:/etc/init.d/rc 0
    ```

    pilotano il processo di avvio *System-V-style*
    - wait = esecuzione sequenziale
    - quando il runlevel obiettivo è 'N', `rc` esegue
      - ogni programma con nome che inizia per 'S' in `/etc/rcN.d/` col parametro `start`
      - ogni programma con nome che inizia per 'K' in `/etc/rcN.d/` col parametro `stop`
    - per evitare l'inutile duplicazione degli script di avvio e arresto dei demoni, questi sono tutti raccolti in `/etc/init.d/`, e symlinked dalle 7 directory `/etc/rcN.d/`
  - linee del tipo

    ```
    x:5:respawn:/usr/X11/bin/gdm
    ```

    avviano il programma specificato come 4° campo, e *init* monitora il processo per riavviarlo (respawn) se termina

Controllo del sistema con sysvinit

- Configurazione persistente (applicata automaticamente all'avvio)
  - `chkconfig` (RedHat) o `update-rc.d` (Debian) configurano i runlevel gestendo i symlink nelle 7 directory
- Verifica del runlevel attivo
  - `runlevel`
  - restituisce il precedentemente attivo e l'attuale
- Cambio di runlevel
  - `telinit` *N*
- Avvio, arresto e verifica dello stato dei singoli servizi
  - `/etc/init.d/scriptname {start|stop|status}`
  - supportati da alcuni script `reload`, `restart`, `condrestart`, ...

Upstart (principalmente Ubuntu)

- Un rimpiazzo per *init* basato sulla logica a eventi
  - Inizializzazione dei sottosistemi parallela e non bloccante
  - Gestione omogenea di tutti gli eventi asincroni
    - Aggiunta e rimozione di hardware
    - Avvio e arresto di processi
  - Inizializzazione multi-stadio (es. rilevazione hardware → caricamento firmware → attivazione device → rilevazione delle caratteristiche del device)
  - In prospettiva, integrazione dei pianificatori (cron, at)
- Distribuzioni principali che lo adotta(va)no
  - Ubuntu 6.10 – 14.10
  - Fedora 9 – 14
  - Debian (opzione)
  - Nokia's Maemo platform
  - Palm's WebOS
  - Google's Chromium OS
  - Google's Chrome OS

Qualche concetto di base su upstart

- Filosofia (dal sito):
  - Task e Servizi sono avviati e arrestati in seguito a eventi
  - Il completamento di un avvio/arresto genera a sua volta un evento
  - Gli eventi possono essere ricevuti da qualsiasi processo sul sistema
  - I Servizi possono essere riavviati se terminano inaspettatamente
  - La supervisione e il riavvio di un demone è gestita anche nel caso sia un processo figlio separato dal progenitore
  - La comunicazione avviene via D-Bus
- Operativamente
  - La directory `/etc/init` contiene un file per definire ogni attività
  - Il demone `init` continua ad essere l'orchestratore del sistema
    - ogni modifica ai file di configurazione è rilevata via inotify e applicata in tempo reale
  - Il comando `initctl` interagisce con le attività mandando segnali appropriati (documentati nei sorgenti in `event.h`) a `init` (via sotto-comandi):
    - `start / stop / status`
    - `list / emit / reload-configuration`

Systemd (ispirato da RedHat – ora molto diffuso)

- Che aspetti affronta systemd?
  - Dipendenze tra servizi
  - Avvio a richiesta di servizi
  - Logging precoce
  - Conservazione dell'output dei demoni
  - Tracciamento dei cgroup (per controllo preciso risorse hardware)
  - Tracciamento e gestione dei mount point
  - Snapshots di sistema e loro ripristino
  - Gestione delle impostazioni globali come hostname, locale, ecc.
  - Ambiente deterministico di esecuzione dei servizi
  - Aggiornamenti del sistema offline (al riavvio)
  - Processo di boot più rapido e senza shell interattive

Systemd

- Systemd si propone di sostituire
  - init (etc.)
  - udev
  - pm-utils
  - inetd
  - acpid
  - crond/atd
  - ConsoleKit
  - automount
  - watchdog
  - syslog

Systemd – termini

- Diversi tipi di [control] unit i cui nomi seguono la convenzione `name.type`
- `type` può essere:
  - `Service`: controllo e monitoraggio dei demoni
  - `Socket`: attivazione di canali IPC di ogni tipo (file, net socket, Unix socket)
  - `Target`: gruppo di unit che **rimpiazza il concetto di runlevel**
  - `Device`: punti di accesso ai dispositivi, creati dal kernel in seguito a interazioni con l'hardware
    - filesystem-related: `Mounts`, `Automounts`, `Swap`
  - `Snapshots`: stato salvato del sistema
  - `Timers`: attività legate al tempo (→ cron, at)
  - `Paths`: monitoraggio del contenuto di una directory via inotify
  - `Slices`: gestione delle risorse via cgroup
  - `Scopes`: raggruppamento di processi per miglior organizzazione

Systemd – dove trovare le definizioni delle unit

- "libreria" di definizioni di riferimento
  - `/lib/systemd/system`
- File forniti dai mantainer dei diversi pacchetti software
  - Quasi sempre link alle definizioni di riferimento
  - `/usr/lib/systemd/system`
- File con le personalizzazioni
  - prioritari rispetto alle definizioni di sistema sopra elencate
  - `/etc/systemd/system`

Systemd – operazioni base

- Controllo a run time dei servizi
  - `systemctl {start|stop|status|restart|reload} servicename`
    - ...intuitivo
    - output molto descrittivo dello stato
      - stato corrente ed elenco dei passi fatti per raggiungerlo
      - process tree
      - righe di log rilevanti
    - "`-H [hostname]`" si connette a un host remoto via ssh
- Configurazione persistente dei servizi al boot
  - `systemctl {enable|disable|mask|unmask} servicename`
    - `disable` lascia disponibile la possibilità di usare manualmente `start`
    - `mask` "neutralizza" l'intera definizione della unit, impedendo anche il controllo manuale

Systemd – verifica della configurazione

- Solo qualche esempio
  - `systemctl list-units`
    - mostra tutte le *unit* gestite (di tutti i tipi elencati!)
  - `systemctl -t type`
    - es.: `systemctl -t timers`
    - mostra tutte le unit attive del tipo specificato
  - `systemctl list-unit-files [-t type]`
    - es.: `systemctl list-unit-files -t services`
    - mostra tutte le unit installate del tipo specificato
  - `systemctl --state state`
    - es.: `systemctl --state failed`
    - mostra tutte le unit che si trovano nello stato specificato

Systemd – avvio

- I runlevel sono rimpiazzati dai target
  `/etc/inittab` non è più utilizzato
  - il target di default è visualizzabile/impostabile con
    `systemctl get-default`
    `systemctl set-default [target]`
    - es.: `systemctl set-default graphical.target`
- Equivalenze
  - Esplorate `/lib/systemd/system`

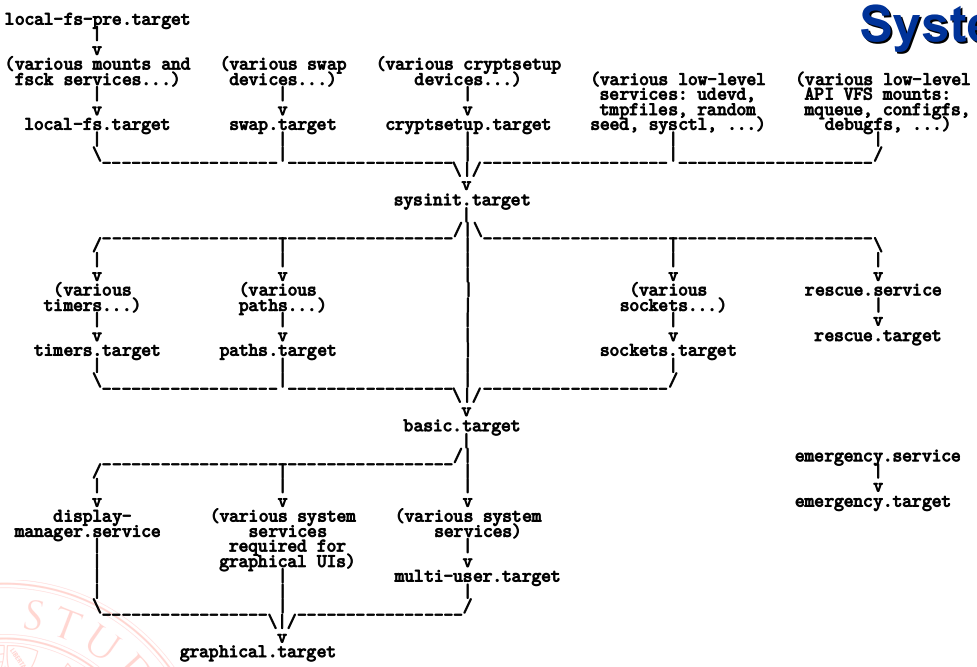
Runlevel	Systemd Target	Description
0	poweroff.target, runlevel0.target	System halt
1	rescue.target, runlevel1.target	Single user mode
3 (2,4)	multi-user.target, runlevel3.target	Multi-user, non graphical
5	graphical.target, runlevel5.target	Multi-user, graphical
6	reboot.target, runlevel6.target	System reboot

Systemd – avvio

- Cosa fa un target? Dalla man page `systemd.target (5)`:
  "Target units […] exist merely to group units via **dependencies** (useful as boot targets), and to establish **standardized names** for synchronization points used in dependencies between units."
- Dipendenze = automazione robusta
  - Sysvinit = sequenziale → lento, nessuna gestione errori
  - Systemd = parallelo condizionato → ogni unit parte non appena sono rispettati i vincoli espressi dalle direttive:
    - `Requires` – elenco di altre unit da avviare quando questa è avviata/fermata: se l'avvio di tali unit fallisce, questa viene arrestata; si può configurare la relazione temporale (dopo, prima, simultaneamente)
    - `Wants` – versione più soft di Requires (il fallimento delle dipendenze non blocca l'avvio di questa unit)
    - `Conflicts` – vincolo negativo per rendere unit mutuamente esclusive
    - `OnFailure` – unit da avviare quando questa fallisce
    - `RequiredBy / WantedBy` – crea automaticamente entry Requires/Wants nelle unit elencate quando questa viene installata
    - `Restart` – riavvia il servizio in caso di terminazione; è l'equivalente del respawn di inittab, ma con una varietà ricca di ulteriori sotto-parametri per controllare sotto quali condizioni effettivamente eseguire il riavvio

https://www.freedesktop.org/software/systemd/man/systemd.service.html

Systemd – avvio



- Unit speciali
  - Alcune unit sono predefinite con nomi fissi e funzioni fondamentali
    - Principalmente target, e alcune slice (vedi `systemd.special(7)` e `bootup(7)`)
    - Es. punti di controllo della sequenza di boot, che punta a `default.target`
      - `default.target` sarà un link a uno dei "veri" target disponibili

Cheat sheet

	SysVinit (Debian) (RedHat)	Upstart	Systemd
Start service	/etc/init.d/name start	service name start	systemctl start name
Stop service	/etc/init.d/name stop	service name stop	systemctl stop name
Status check	/etc/init.d/name status	service name status	systemctl status name
Enable service start at boot	update-rc.d name enable chkconfig name on	rm /etc/init/name.override	systemctl enable name
Inhibit service start at boot	update-rc.d name disable chkconfig name off	echo manual > /etc/init/name.override	systemctl disable name
List installed services	ls /etc/init.d chkconfig --list	service --status-all && initclt list	systemctl list-unit-files -t services
List services starting at boot	ls /etc/rc**X**.d/S* chkconfig --list | grep **X**:on	Give up and upgrade to systemd	systemctl list-unit-files -t services --state=enabled

X = runlevel di default

service e *systemctl* sono stati introdotti dai rispettivi sistemi ma sono wrapper retrocompatibili (in alcuni sistemi c'è un misto di demoni gestiti in 2 o tutti e 3 i modi!) es. se systemctl start name non trova la unit name, prova service name start, così come questo proverebbe /etc/init.d/name start in caso di assenza di configurazione upstart

Assunzione standard:
i servizi installati sono configurati per partireal boot