I primi passi nell'amministrazione del sistema

Simone Piccardi

8 agosto 2002

Indice

1 Gli editor **1**
1.1 Introduzione . . . 1
1.2 Gli editor . . . 1
1.3 Editor: `emacs` (e `xemacs`) . . . 1
1.4 La sintassi di emacs . . . 2
1.5 Editor: `vi` . . . 3
1.6 La sintassi di `vi` . . . 3
1.7 Editor: `joe` . . . 5
1.8 Editor: jed . . . 6
1.9 Editor: `pico` e `nano` . . . 6
1.10 Gli altri editor . . . 6

2 I principali file di configurazione **7**
2.1 Una panoramica generale . . . 7
2.2 Il file `/etc/fstab` . . . 8
2.3 Il file `/etc/inittab` . . . 10
2.4 Il file `/etc/crontab` . . . 11
2.5 Il file `/etc/mtab` . . . 12
2.6 Il file `/etc/passwd` . . . 12
2.7 Il file `/etc/hosts` . . . 13
2.8 Il file `/etc/modules` . . . 14
2.9 Il file `/etc/ld.so.conf` . . . 14
2.10 I file `/etc/logrotate.conf` e `/etc/logrotate.d/` . . . 14
2.11 Il file `/etc/profile` . . . 16
2.12 Il file `rc.local` . . . 16
2.13 `/etc/securetty` . . . 16
2.14 Il file `/etc/lilo.conf` . . . 16
2.15 Il file `/etc/syslog.conf` . . . 20

1 Gli editor

In questa prima parte faremo una panoramica dei vari editor disponibili su GNU/Linux, essi infatti sono il principale strumento usato da tutti gli amministratori di sistema.

Lo scopo di questa prima parte è quello di mettere il lettore in grado di cavarsela con tutti i principali editor disponibili in tutte le distribuzioni, con il tempo e l'esperienza ognuno finirà con l'adottarne uno.

1.1 Introduzione

Come certamente avete già sentito nella lezione sulla shell (vedi **??**) Linux nasce come tutti gli Unix con l'interfaccia a linea di comando, e tutte le funzionalità del sistema sono accessibili fin da quel livello.

Questo è particolarmente vero per quanto riguarda l'amministrazione di sistema, infatti nonostante l'esistenza di alcuni tool grafici che permettono le più comuni operazioni di amministrazione, in genere questi ultimi mettono a disposizione solo un limitato insieme di opzioni, e non danno mai il livello di controllo che si può raggiungere operando direttamente sui file di configurazione.

Inoltre quando un eventuale problema di qualche tipo su disco, o il classico `rm` dato un po' troppo allegramente da root vi avrà danneggiato qualche file essenziale o bloccato il sistema all'avvio, potrete sempre usare una distribuzione su dischetto per rimettere le cose a posto, ma lì i tool non grafici saranno disponibili.

Una delle caratteristica fondamentali di Linux (come di ogni Unix) è infatti quella di tenere tutte le configurazioni in file di testo, accessibili e modificabili con un qualunque editor, ed in genere raccolti nella directory `/etc/`.

Per questo lo strumento principale di ogni amministratore di sistema è *l'editor di testi*. Lo scopo di questa lezione è allora quello di dare una infarinatura su come operare sui principali file configurazione del sistema e dei programmi più importanti. Inoltre dato che l'editor assume un ruolo centrale, faremo anche una panoramica sui principali editor che potete trovare su un sistema unix alcuni dei quali hanno comandi non immediati.

1.2 Gli editor

Come detto l'editor è lo strumento principale dell'amministrazione sotto Unix. Per questo in questa sezione daremo una breve panoramica sull'uso dei più comuni. Inoltre restando nell'ottica dell'amministrazione base parleremo esclusivamente di editor accessibili da console, e quindi utilizzabili anche attraverso connessioni remote.

L'editor di testi è stata una delle prime applicazioni sviluppate sotto Unix e come per molte altre applicazione di uso generale ne esistono molti, dai più elementari, come `ed`, un editor di linea che opera, come dice il nome, solo sulle linee di testo, ereditato dai tempi dei primi terminali, ai più complessi, come `emacs` che viene considerato da alcuni (esagerati) un secondo sistema operativo.

In questa sezione comunque non entreremo nei dettagli dell'uso dei singoli editor, ci limiteremo a esporre quali sono i comandi base per leggere, modificare e scrivere su un file di testo. La scelta dell'editor è comunque una scelta personale, che genera spesso clamorose *guerre di religione* fra le fazioni dei sostenitori dei diversi editor (particolarmente virulente sono quelle fra i sostenitori di emacs e vi).

1.3 Editor: `emacs` (e `xemacs`)

Per molti `emacs` è l'editor. Sicuramente è il più potente: dentro `emacs` si può davvero fare di tutto. Esempi sono: navigare fra i file e in internet, leggere la posta e le news, programmare

(con evidenziazione della sintassi e shortcut ai costrutti principali di qualunque linguaggio), fare debug, scrivere dispense come queste, giocare (a tetrix o a qualche avventura testuale), farsi psicanalizzare dal doctor.

Qualunque cosa sia possibile fare con del testo con `emacs` si fa, e dato che l'editor è programmabile c'é certamente qualcuno che ha creata una serie opportuna di comandi per rendere le cose più veloci, aggiungere automatismi, ecc.

Tutto questo ha ovviamente un costo, ed infatti i detrattori di emacs ne lamentano la pesantezza (di certo non lo troverete sulle distribuzioni su dischetto), ma data la diffusione e la potenza dello strumento ne parleremo, considerato poi che molti altri editor ne hanno copiato la sintassi e le modalità d'uso, o forniscono modalità di comando *compatibili*.

Inoltre `emacs`, ed il suo cugino `xemacs` che nacque proprio per questo, sono editor usabili direttamente anche dall'interfaccia grafica, nel qual caso vengono forniti all'utente gli usuali menù a tendina in cui si potranno trovare buona parte delle funzionalità di cui essi dispongono.

1.4 La sintassi di emacs

Come per tutti gli editor una sessione di emacs si inizia con il comando `emacs nomefile`, dato il comando si otterrà una finestra come quella mostrata in fig. 1: nella prima riga si trova il menu dei comandi (cui si accede con F10), ad esso segue la sezione principale, (vuota nell'esempio) dove compare il testo del file ed in cui ci si muove con le frecce, una barra di stato (quella che comincia per `---`) in cui compaiono varie informazioni (il nome del file, se sono state fatte modifiche, la modalità[1]; e nella la riga finale il cosiddetto *minibuffer*, dove compaiono brevi messaggi (come nell'esempio, che riporta la scritta `(New file)`) ed in cui si scrivono i comandi. In certi casi, in corrispondenza dei comandi inviati, la sezione principale può venire automaticamente divisa in 2 parti per permettere di inserire o mostrare ulteriori informazioni.

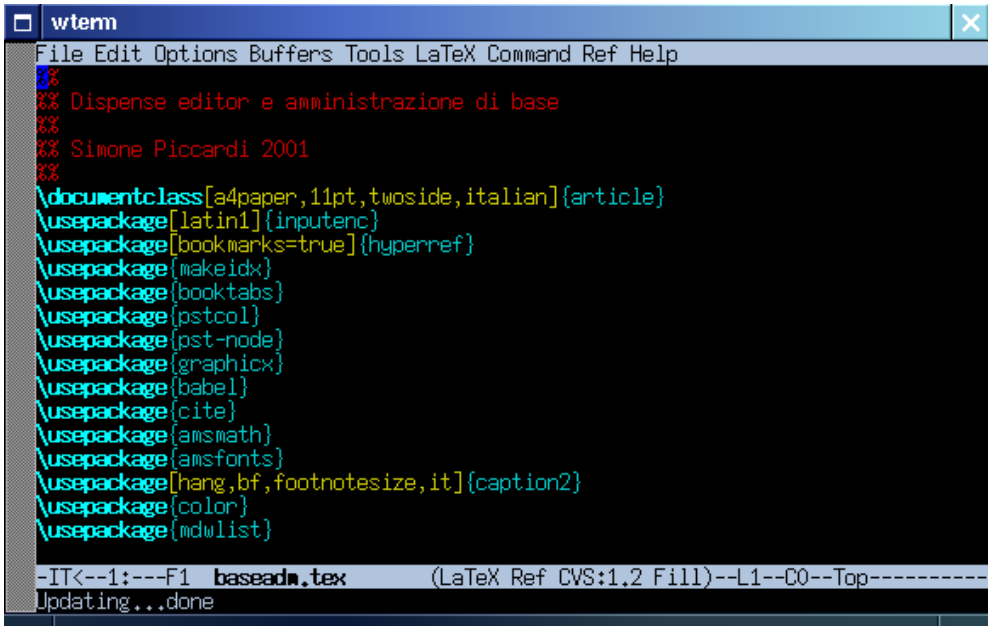

Figura 1: Schermata di avvio dell'editor `emacs`.

Data la complessità dell'editor esistono due modificatori per dare i comandi, control (`C-`) e alt (`M-` da *meta*, che può essere sostituito dall'escape), inoltre la maggior parte dei comandi è data con due combinazioni di tasti successive.

[1] essendo `emacs` un editor programmabile esso può essere usato in modalità diverse a seconda del tipo di file che si usa, provvedendo in ciascun caso diverse serie di combinazioni di tasti

Quella che segue è la lista dei comandi principali:

- aprire un file: `C-x C-f`
- salvare un file: `C-x C-s`
- salvare con nome: `C-x C-w nomefile`
- uscire: `C-x C-c` chiede se salvare le eventuali modifiche
- undo: `C-_`, `C-`
  o `C-x u`, ripetuto torna ulteriormente indietro
- seleziona: `C-x spazio` all'inizio e poi spostarsi con le frecce
- taglia: `C-w` sulla regione selezionata
- incolla: `C-y`
- cancella: `C-d` in avanti e `backspace` indietro
- ricerca: `C-s testo` ricerca incrementale sul testo specificato, `C-s` cerca il successivo `C-r` cerca il precedente.
- help: `C-h ?` poi scegliere nella finestra quello che si vuole
- annulla (comando): `C-g` o `ESC ESC ESC`

1.5 Editor: vi

È stato uno dei primi editor evoluti presenti in un sistema Unix. Deriva dagli editor di linea e ne eredita alcune caratteristiche, in particolare il fatto di essere un editor *modale*, in cui cioè i comandi e il loro effetto dipendono dalla corrente *modalità di operazioni* in cui si trova il programma.

Questa caratteristica lo rende senz'altro il meno intuitivo e più difficile da usare per il novizio, i fan(atici) tendono invece a considerarla utile in quando, secondo loro, con l'abitudine renderebbe più veloce le operazioni. Succede spesso però che al primo impatto non si riesca neanche ad uscire dall'editor.

Al contrario di `emacs` (di cui è il principale concorrente) `vi` è soltanto un editor, non fornisce pertanto nessuna delle funzionalità più evolute di emacs (come la possibilità di fare debug i programmi facendo riferimento diretto al codice che si sta editando) ma resta comunque un editor molto potente.

Il principale vantaggio di `vi` è che essendo molto leggero, lo si trova installato praticamente su qualunque sistema. Inoltre alcune versioni più moderne, come `vim`, hanno introdotto alcune delle capacità di emacs, come l'evidenziazione della sintassi. Non è detto però che quest'ultimo sia sempre disponibile al posto del `vi` normale (e di certo non lo è su una distribuzione di recupero).

1.6 La sintassi di vi

Come accennato `vi` è un editor modale, il comando `vi nomefile`, apre il file in modalità comando in una finestra principale (mostrata in fig. 2), dove compare il testo (essendo il file nuovo nella figura le righe vuote sono sostituite da `~`) ed in cui ci si muove con le frecce, lasciando libera l'ultima linea, usata per dare i comandi o ricevere le informazioni (nel caso il fatto che il file è nuovo e che si è sulla prima riga).

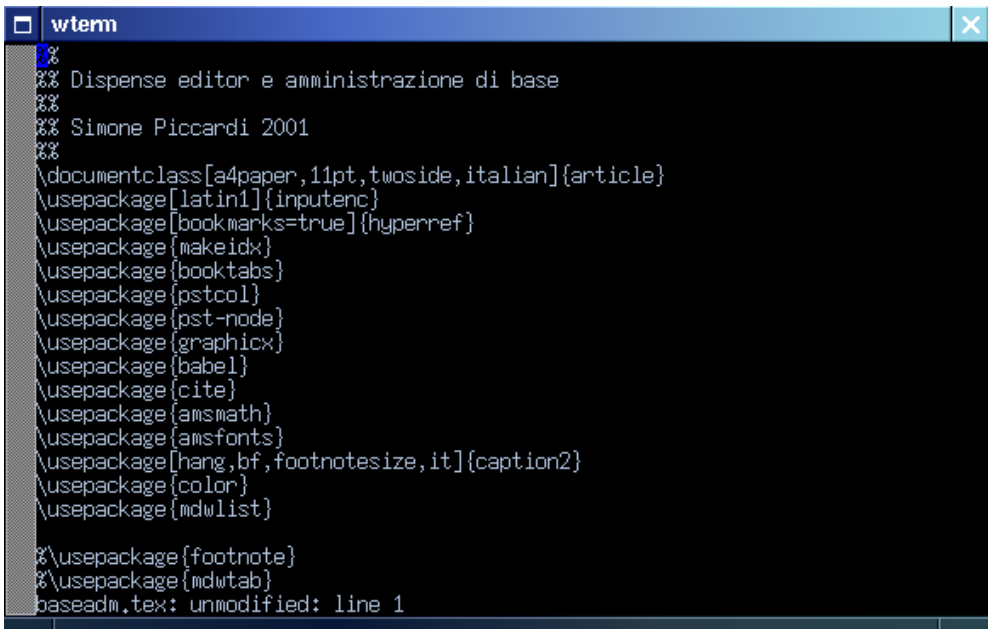

Figura 2: Schermata di avvio dell'editor `vi`.

Tutti i comandi di `vi` sono eseguiti con pressioni di singoli tasti, ma la possibilità di dare il comando dipende dalla modalità in cui si trova l'editor al momento. I modi sono sostanzialmente due: comando ed inserimento, quando in modalità comando occorre scrivere qualcosa, questo viene mostrato nella riga finale.

Una delle cose da capire in `vi` è che una volta che si è entrati in modalità inserimento non è possibile dare più alcun comando (cosa che spesso rende impossibile ai neofiti uscire da `vi`) occorrerà prima tornare in modalità comando, cosa che può essere fatta soltanto premendo il tasto di escape.

Quella che segue è la lista dei comandi principali:

- aprire un file: `:ex file`
- salvare un file: `:w`
- salvare con nome: `:w nomefile`
- uscire: `:q` ma non esce se ci sono modifiche, nel caso `:qw` le salva, `:q!` le scarta
- undo: `u` solo sull'ultima modifica
- taglia: `yy` taglia una riga
- incolla: `p` incolla la riga
- cancella: `d` il singolo carattere, ma `dd` l'intera riga, per ripetere la cancellazione di caratteri usare `del`
- ricerca: `/testo`
- help: `:h`
- annulla (comando): `ESC`
- per scrivere: `i` entra in modalità inserimento, quando si è finito occorre premere `ESC`

1.7 Editor: joe

È un editor leggero e potente che usa la sintassi di wordstar, offre una serie di caratteristiche avanzate ed un help in linea per i vari comandi, che lo rendono piuttosto facile da usare.

Il comando `joe nomefile`, apre il file in una finestra principale, mostrata in fig. 3, dove una riga di stato in testa, seguita dalla sezione principale in cui compare il testo (nell'esempio viene mostrata la scritta `New File`) ed in cui ci si muove con le frecce, l'ultima linea viene usata per mostrare i messaggi e per dare i comandi o ricevere le informazioni, e scompare quando non è più necessaria.

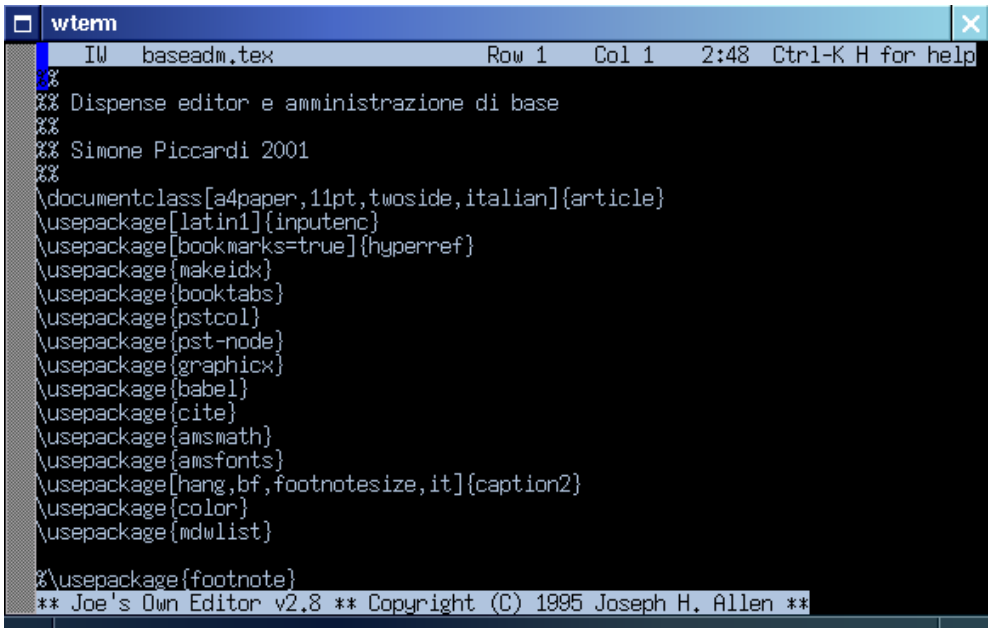

Figura 3: Schermata di avvio dell'editor `joe`.

Quella che segue è la lista dei comandi principali:

- aprire un file: `C-k e`
- salvare un file: `C-k d RET`
- salvare con nome: `C-k d nomefile`
- uscire: `C-k x`esce e salva, `C-c` esce ed eventualmente scarta
- undo: `C-_`, redo: `C-^`
- seleziona: `C-k b` all'inizio e poi spostarsi con le frecce e dare `C-k k` alla fine
- taglia e incolla: `C-k m` muove la regione selezionata sulla posizione attuale
- taglia `C-k y`
- copia `C-k c`copia la regione, la si può spostare poi con `C-k m`
- cancella: `C-k y` in avanti e `backspace` indietro
- ricerca: `C-k f testo`cerca il testo, `C-L` cerca l'occorrenza successiva
- help: `C-k h` e compare una finestra in alto coi comandi principali
- annulla (comando): `C-c`

1.8 Editor: jed

È un editor scritto come reimplementazione di emacs in C, molto più leggero (ma anche molto meno potente). Supporta anch'esso un linguaggio di programmazione interno e può utilizzare diverse sintassi per i comandi (emacs, wordstar, edt). Fornisce anche un'ampia capacità di evidenziazione della sintassi in console.

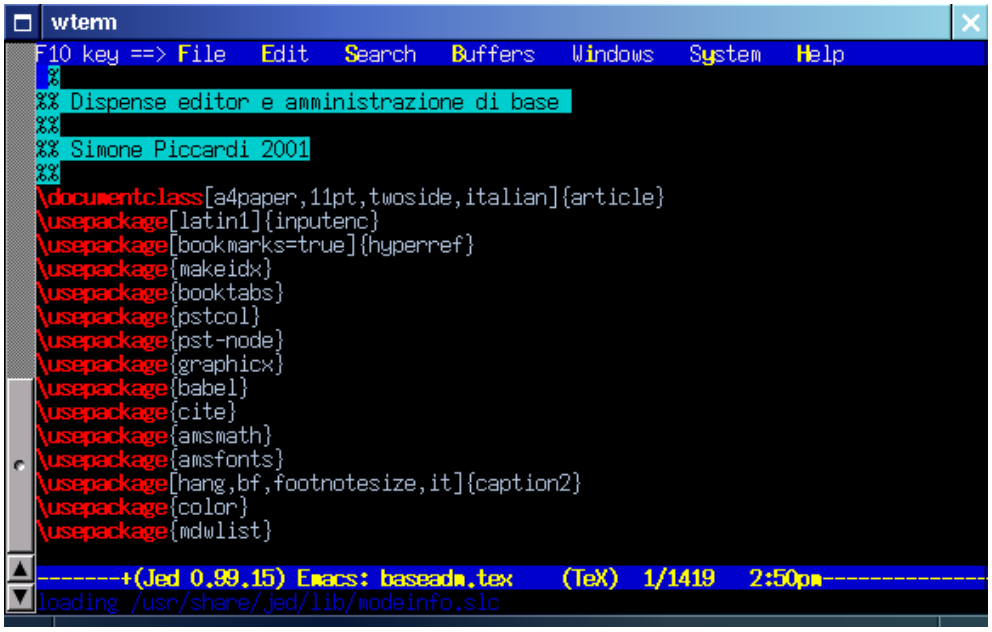

Figura 4: Schermata di avvio dell'editor `joe`.

Essendo in sostanza un clone di emacs non staremo a ripeterne i comandi, infatti di default accetta la sintassi di emacs e basta rifarsi al precedente paragrafo. Essendo un editor leggero e potente si trova su varie distribuzioni su dischetto.

1.9 Editor: `pico` e `nano`

Un programma di posta testuale molto popolare, `pine`, provvedeva al suo inerno anche un editor, `pico` appunto. Data la maggiore utilizzabiltà rispetto a `vi` esso si è diffuso parecchio, ed è diventato un programma a parte.

Dato che `pine` non è software libero, non lo è neanche `pico`, ma ne è stato realizzato un clone completamente libero chiamato `nano` identico dal punto di vista dei comandi principali, ma più leggero e con qualche capacità in più.

Il programma è dotato di help in linea nelle due linee terminale dello schermo (una delle sue maggiori caratteristiche di successo), che spiega i principali comandi, pertanto è inutile ripeterli qui, per vederli basterà invocare `nano nomefile`.

1.10 Gli altri editor

Tutti gli editor citati sono in grado di funzionare in un terminale o dalla console, ma esistono tutta una serie di editor *grafici* come quelli inseriti in *Gnome* e *KDE* che non sono spiegati qui in quanto l'interfaccia grafica è in grado di dare accesso alle funzionalità base con i soliti menù.

Fra questi però mi preme segnalare uno dei più evoluti: `nedit` che dispone di una serie di funzionalità molto avanzate (come la evidenziazione della sintassi) e di un linguaggio di programmazione interna che lo rendono molto potente, pur restando anche facile da usare.

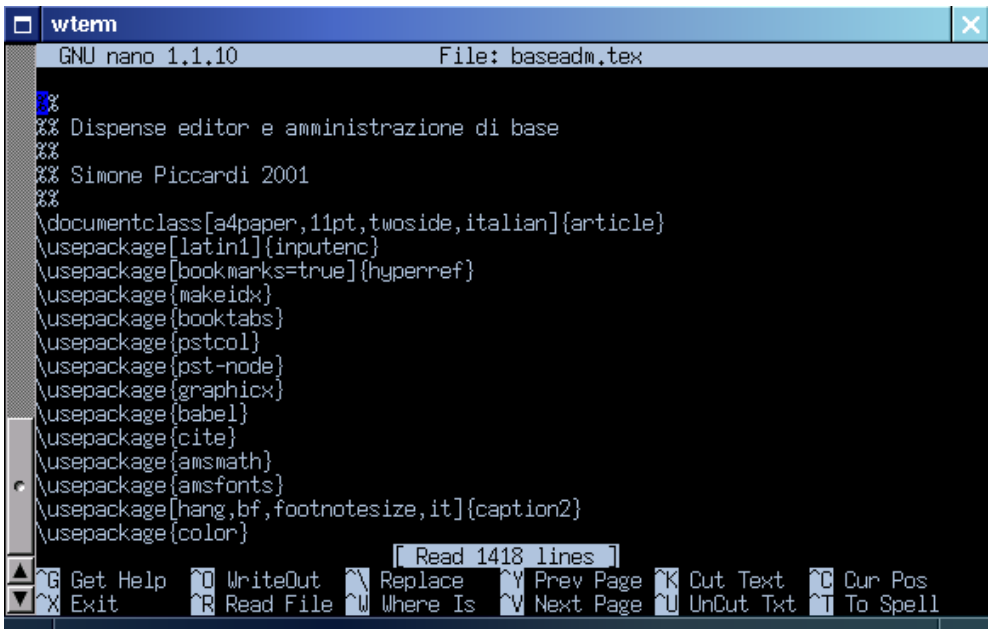

Figura 5: Schermata di avvio dell'editor `nano`.

Esistono comunque anche versioni grafiche di alcuni degli editor precedenti (come `gvim` per vi), mentre come già accennato sia emacs che xemacs sono usabili anche sotto X.

2 I principali file di configurazione

In questa sezione, dopo una panoramica sui concetti base della configurazione dei programmi sotto Linux, prenderemo in esame i principali file di configurazione, descrivendone formato, caratteristiche e proprietà pricipali.

2.1 Una panoramica generale

A differenza di Windows che tiene tutte le configurazioni in un unico file binario (il registro) le sole cose univoche che ha Linux nei confronti dei file di configurazione è che essi sono tenuti nella directory `/etc/` e che sono file di testo.

Questo deriva in buona parte dell'architettura di Unix, in cui i servizi non sono effettuati dal kernel, ma da opportuni programmi (che non è affatto detto siano sempre gli stessi). Ciò comporta ad esempio che i formati dei file di configurazione possano essere i più vari, ed anche se esistono convenzioni generali abbastanza seguite, come ignorare le righe vuote o considerare il carattere `#` l'inizio di un commento, non è detto che esse vengano sempre applicate.

Se da una parte tutto questo può spaventare, vista la mole di file che produce un comando come:

```
[root@roke /etc]# ls -l
total 1584
drwxr-xr-x    3 root     root         4096 Aug 21  2000 CORBA
drwxr-xr-x    3 root     root         4096 Aug 21  2000 GNUstep
-rw-r--r--    1 root     root         4172 Feb 15 01:27 Muttrc
drwxr-xr-x    2 root     root         4096 Feb 26 21:21 Net
drwxr-xr-x   16 root     root         4096 Feb 28 23:47 X11
-rw-r--r--    1 root     root         1660 Feb 26 21:21 adduser.conf
-rw-r--r--    1 root     root           44 Mar 10 02:33 adjtime
```

...

ha invece il grande vantaggio che le modifiche ad un singolo file non hanno alcun modo di influenzare gli altri, e che essendo i file di testo è possibile effettuare ricerche ed operazioni complesse con i soliti comandi di shell.

Una seconda cosa di cui bisogna tenere conto è che Unix è multiutente, per cui è molto spesso possibile per ciascun utente scegliere i settaggi che si ritengono più appropriati per un programma mettendo un opportuno file di configurazione nella propria home directory. In genere questi file sono invisibili (iniziano cioè con un `.`) ed hanno lo stesso nome del loro analogo di `/etc/` valido per tutto il sistema.

Questa è una forma molto potente e pulita di consentire a ciascun utente di personalizzare le sue scelte senza dover andare a scomodare i settaggi generali del sistema.

Veniamo allora ad esaminare i principali file di configurazione. È da tenere presente che per molti di essi è disponibile una man page che ne spiega il formato, accessibile con `man nomefile`, o nel caso di omonimia con un comando o una funzione, con `man 5 nomefile`. Qui faremo una descrizione sommaria, per questo vale sempre la pena controllare la suddetta documentazione, che contiene tutti i dettagli. Inoltre non tratteremo qui i file né che riguardano la connessione ad internet e la rete, né quelli di X, che saranno affrontati nelle rispettive lezioni.

2.2 Il file `/etc/fstab`

Questo file, il cui nome sta per *file system table* contiene le indicazioni dei filesystem disponibili per il sistema. È il file che il programma di avvio `init` va a leggere quando deve montare i dischi ed è pertanto di importanza fondamentale.

Il formato del file è molto semplice, ogni linea definisce un filesystem che sta su un device, linee vuote o che iniziano per `#` vengono ignorate, i campi di ogni linea sono separati da spazi o tabulatori. La man page `man fstab` ne spiega i dettagli.

Vediamone un esempio:

```
# /etc/fstab: static file system information.
#
# file system mount point     type    options                        dump  pass
/dev/hdb5     /               ext2    defaults,errors=remount-ro      0       1
/dev/hdb6     none            swap    sw                              0       0
proc          /proc           proc    defaults                        0       0
/dev/fd0      /floppy         auto    defaults,user,noauto            0       0
/dev/cdrom    /cdrom          iso9660 defaults,ro,user,noauto         0       0
/dev/sr0      /mnt/cdrom      iso9660 defaults,ro,user,noauto         0       0
/dev/hdb1     /boot           ext2    rw                              0       2
/dev/hda1     /mnt/win        vfat    defaults,user,noauto            0       0
/dev/hdc4     /mnt/zip        auto    defaults,user,noauto            0       0
```

Come si può notare ogni linea contiene 6 campi, che indicano rispettivamente il device su cui si trovano i file, la directory in cui questo device deve essere montato, il tipo di filesystem, le opzioni di montaggio, se effettuare un dump del filesystem e l'ordine in cui eseguire il check del medesimo all'avvio.

Il primo campo descrive il device su cui sta il file system da montare, nel caso in questione si hanno due hard disk (`/dev/hda` e `/dev/hdb` con varie partizioni), un floppy (`/dev/fd0`), uno zip (`/dev/hdc4`), un CDROM ed un masterizzatore scsi (`/dev/cdrom` e `/dev/sr0` (nel caso del CDROM si è usato un link simbolico). Quando si vorrà aggiungere un altro device e montarlo occorrerà allora determinare qual'è il suo corrispondente file di dispositivo e inserirlo in questo campo.

Inoltre avendo abilitato il supporto nel kernel è possibile montare il filesystem `/proc` che contiene una serie di file virtuali generati al volo dal kernel per accedere ad alcune delle informazioni interne. Se ci fossero stati dei file montati via NFS (cioè file condivisi sulla rete) si sarebbero avuti anche campi del tipo:

```
firenze.linux.it:/ /mnt/nfs     nfs    defaults,user,noauto          0       0
```

Il secondo campo indica il *mount point* cioè la directory dove i file del nuovo dispositivo saranno resi disponibili. Se il filesystem non deve essere montato, come nel caso della partizione di swap, si usa la parola chiave `none`.

Il terzo campo indica il tipo di filesystem che sta sul device che si vuole montare. I tipi più comuni sono *ext2* (il filesystem standard di Linux), *vfat* (il filesystem di Windows), *iso9660* (il filesystem dei CDROM), ma ne esistono altri non usati nell'esempio caso come *nfs*, o *hfs* che è il filesystem del MacOS. Si noti come in un caso sia presente invece la parola chiave `swap` ad indicare che in quel caso il dispositivo non contiene un filesystem, ma va usato per la swap. È inoltre possibile usare la parola chiave `auto` per dire al sistema di cercare di determinare automaticamente il tipo di filesystem.

Il quarto campo indica le opzioni con cui si può montare il filesystem, esse devono essere indicate con una lista di valori separati da virgole, i valori possibili sono indicati dalla man page del comando `mount`:

- `user` consente anche agli utenti normali di montare il filesystem
- `nouser` non consente agli utenti normali di montare il filesystem
- `dev` consente l'uso di file di dispositivo sul filesystem
- `nodev` non consente l'uso di file di dispositivo sul filesystem
- `auto` tutti i filesystem con questa opzione citati in `fstab` vengono montati dal comando `mount -a` che viene eseguito all'avvio del sistema
- `noauto` il filesystem deve essere montato esplicitamente
- `exec` consente l'esecuzione di programmi sul filesystem
- `noexec` non consente l'esecuzione di programmi sul filesystem
- `suid` consente che i bit suid e sgid abbiano effetto
- `nosuid` non consente che i bit suid e sgid abbiano effetto
- `sync` tutto l'I/O sul filesystem deve essere sincrono
- `async` tutto l'I/O sul filesystem deve essere asincrono
- `ro` monta il filesystem in read-only
- `rw` monta il filesystem in read-write
- `default` usa le opzioni di default: `rw`, `suid`, `dev`, `exec`, `auto`, `nouser`, e `async`.

Nel caso si usi l'opzione default la successiva specificazione di un'altra opzione soprassiede il valore di default.

Gli ultimi due campi sono relativi alla manutenzione del filesystem, in genere il primo viene posto a zero, a meno di non aver predisposto `dump` per il backup, l'ultimo indica la sequenza con

cui all'avvio viene lanciato il comando `fsck` per controllare lo stato dei dischi, uno zero indica che il controllo non deve essere eseguito.

Dal punto di vista dell'amministrazione base si ha a che fare con `/etc/fstab` tutte le volte che si aggiunge un disco, o un nuovo dispositivo, o si cambiano le partizioni. In questo caso occorre identificare qual'è il file di dispositivo da usare e scegliere nel filesystem una directory su cui montarlo. Deve poi essere specificato il filesystem da usare (o `auto` se si vuole tentare il riconoscimento automatico).

Nell'esempio si noti come per zip e floppy si sia consentito agli utenti di montare il filesystem, ma si sia disabilitato il montaggio all'avvio, e pure il controllo dello stato del filesystem, dato che non è detto che il floppy o lo zip siamo sempre nel driver.

Lo stesso vale per il CDROM e il masterizzatore, per i quali si è pure aggiunto l'opzione di montaggio in read-only. Si noti inoltre l'opzione speciale per il filesystem di root, per il quale si è indicato di rimontare il filesystem in read only nel caso di errori.

Nel caso di disco fisso andrà poi scelto se montarlo all'avvio o meno, e in questo caso usare il sesto campo per indicare in quale ordine rispetto agli altri dovrà essere effettuato il controllo del filesystem (il primo deve essere il filesystem usato come radice).

2.3 Il file `/etc/inittab`

Questo file è usato da `init` per stabilire quali sono i processi che devono essere avviati all'avvio e durante le normali operazioni del sistema per le distribuzioni che usano la procedura di avvio in stile System V (in sostanza tutte con l'eccezione di Slackware).

La procedura di avvio di System V (che era una versione di Unix) prevede che il sistema possa porsi un una serie di *runlevel* una sorta di stato del sistema in cui sono attivati o meno certi servizi. È cura del programma di avvio `init` leggere questo file ed attivare il runlevel scelto facendo partire i programmi che esso specifica.

I runlevel validi sono i numeri 0-6, normalmente 0 è usato per fermare il sistema, 6 per il reboot, 1 per andare in single user mode (la modalità di recupero in cui può entrare nel sistema solo root). Per gli altri runlevel le caratteristiche possono variare da distribuzione a distribuzione (con o senza servizi di rete, avvio direttamente da X, etc.).

Il formato del file è molto semplice, la man page, accessibile con `man inittab` ne spiega i dettagli. Le linee vuote o che iniziano per `#` vengono ignorate, le altre devono essere del formato:

```
id:runlevels:action:process
```

dove `id` deve essere una sequenza unica di 1-4 caratteri, `runlevel` la lista dei runlevel a cui la azione specificata dalla parola chiave `action` si applica e `process` il programma che deve essere lanciato. I dettagli al solito si trovano con `man inittab`.

Un esempio di `/etc/inittab` è il seguente:

```
# The default runlevel.
id:2:initdefault:# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS
# /etc/init.d executes the S and K scripts upon change
# of runlevel.
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
```

```
l6:6:wait:/etc/init.d/rc 6
# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
# What to do when the power fails/returns.
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop
# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

Questo file viene letto da `init` e le azioni specificate vengono eseguite nella procedura di startup (e ad ogni cambio di run level), se il numero di runlevel corrisponde con almeno uno di quelli indicati nel secondo campo. In genere l'esecuzione comporta l'esecuzione del programma (o script) indicato dall'ultimo campo secondo diverse modalità. Ad esempio si può eseguirlo una sola volta attendendo che esso sia concluso prima di passare all'azione successiva (come viene fatto con l'opzione `wait` con gli script rc che tirano su i servizi di ciascun runlevel), o metterlo subito in esecuzione e rilanciarlo ogni volta che viene terminato (come viene fatto con `respwan` per `getty` per avere i terminali sulla console).

Il problema con questo file è che il significato di queste azioni non è di immediata comprensione, in quanto alcune sono indipendenti dal runlevel scelto, come per `powerwait`, `powerfailnow`, `powerokwait`, e altre come `initdefault` settano proprio il runlevel di default. Inoltre il campo id può dover essere soggetto a restrizioni come nel caso delle righe con `getty`.

In genere l'utente alle prime armi non ha molto da fare con questo file, l'unica cosa che gli può servire è cambiare la linea dell'azione `initdefault` per cambiare il run level di default a cui ci si trova dopo l'avvio, ad esempio per passare dal login da console a quello su X (sempre che questo sia previsto dalla distribuzione, per RedHat questo significa mettere un 5 al posto di 3, in Debian invece non esiste).

Un seconda azione che si può volere modificare (o disabilitare) è la reazione alla combinazione di tasti ctrl-alt-del che nell'esempio è specificata dall'azione `ctrlaltdel`, ed invoca il programma `shutdown` per riavviare il computer.

2.4 Il file /etc/crontab

Questo file contiene l'elenco dei comandi periodici del sistema, esso viene usato dal programma `crond` (uno dei demoni che viene usualmente lanciato all'avvio del sistema) per eseguire una serie di azioni periodiche di manutenzione del sistema.

In genere si deve intervenire su questo file solo quando si vuole o cambiare uno degli orari a cui le operazioni di default vengono eseguite o per inserire un nuovo comando periodico.

Il formato del file segue la solita regola di ignorare righe vuote ed inizianti per `#`, ogni riga deve contenere una assegnazione di una variabile di ambiente o la specificazione di una azione periodica.

L'azione viene specificata da una serie di 7 campi separati da spazi o tabulatori, i primi

cinque indicano la periodicità con cui il comando indicato nell'ultimo campo viene eseguito, il sesto l'utente usato per eseguire il comando.

I cinque campi della periodicità indicano rispettivamente minuto (da 0 a 60), ora (da 1 a 24), giorno del mese (da 0 a 31), mese dell'anno (da 1 a 12), giorno della settimana (da 0 a 7, ma accetta anche valori tipo Mon, Thu, etc.). Se il tempo corrente corrisponde a tutti i valori ivi specificati il comando viene eseguito, l'utilizzo del carattere `*` vale da wildcard e si usa per indicare un valore qualsiasi.

Il demone `crond` di Linux supporta poi alcune estensioni non presenti in altri Unix, si può usare una lista (separata da virgole) per indicare più valori o il carattere / per indicare un range (ad esempio */2 nel primo campo implica ogni due minuti). Dettagli ulteriori, come sempre, nella man page (accessibile con `man 5 crontab`).

Un esempio è il seguente:

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the 'crontab'
# command to install the new version when you edit this file.
# This file also has a username field, that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user      command
25 6  * * *   root    test -e /usr/sbin/anacron || run-parts --report /etc/cron.daily
47 6  * * 7   root    test -e /usr/sbin/anacron || run-parts --report /etc/cron.weekly
52 6  1 * *   root    test -e /usr/sbin/anacron || run-parts --report /etc/cron.monthly
```

in cui sono riportate le azioni standard che vengono eseguite tutti i giorni alle ore 6:25, tutte le domeniche alle 6:27 e tutti i primi giorni del mese alle 6:52.

A meno di non avere esigenze molto particolari, il contenuto standard di questo file già prevede una serie di azioni giornaliere, settimanali e mensili che vengono eseguite in maniera automatica. Queste azioni sono eseguite (attraverso il comando `run-parts`) dagli script presenti nelle directory elencate nell'esempio precedente, per cui se non si hanno esigenze specifiche non è il caso di intervenire su questo file ma di aggiungere lo script direttamente in `/etc/cron.daily/`, `/etc/cron.weekly/`, `/etc/cron.monthly/`.

Come ultima nota si tenga conto che per i crontab di ciascun utente (quelli che si possono specificare con il comando `crontab -e`) il formato è identico con l'eccezione del sesto campo, che nel caso non è necessario.

2.5 Il file `/etc/mtab`

Questo file contiene l'elenco dei filesystem montati. Viene usato da alcuni programmi per leggere questa informazione, ma viene generato automaticamente e deve essere lasciato stare.

2.6 Il file `/etc/passwd`

Questo è il file che contiene l'elenco degli utenti. Su tutti i sistemi più nuovi esso non contiene più le password cifrate da cui ha avuto il nome che sono invece spostate in `/etc/shadow` che non è leggibile dagli utenti.

Benché ci siano gli opportuni programmi di shell (`adduser`, `passwd`, `chfn`, `chsh`) per la gestione normale in casi di emergenza può essere utile editare a mano il file (se ad esempio si è corrotto e lo si vuole ripristinare).

Il formato del file è molto semplice, per ogni utente deve essere presente una riga composta da 7 campi separati dal carattere `:`, non sono ammessi né commenti né righe vuote.

Il significato dei sette campi è il seguente:

- Nome di login (user name)
- Password criptata (opzionale)
- Identificatore numerico dell'utente
- Identificatore numerico del gruppo
- Nome e cognome dell'utente (con eventuali altri campi di commento, separati da virgole)
- Home directory dell'utente
- Shell di default

Un esempio può essere:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:100:sync:/bin:/bin/sync
games:x:5:100:games:/usr/games:/bin/sh
man:x:6:100:man:/var/cache/man:/bin/sh
...
piccardi:x:1000:1000:Simone Piccardi,,,:/home/piccardi:/bin/bash
gdm:x:100:101:Gnome Display Manager:/var/lib/gdm:/bin/false
```

La presenza di una `x` indica che sono attive le shadow password. Posto che per cambiare shell o le informazioni del quinto campo è opportuno usare i comandi di shell, in caso di emergenza può essere necessario editare questo file, ad esempio se si è persa la password di root, nel qual caso occorrerà far partire il computer con un disco di recupero e togliere la `x` lasciando il campo vuoto, così che la richiesta della password venga disabilitata al successivo riavvio, in modo da poter entrare per ripristinarla.

2.7 Il file /etc/hosts

Questo file serve per associare nomi e numeri IP. Conviene specificare in questo file la risoluzione dei nomi delle macchine a cui si accede più di frequente in modo da evitare di effettuare la richiesta di risoluzione del nome via DNS. Se si è attivata un'interfaccia di rete è sempre utile definire anche il nome della propria macchina, altrimenti di nuovo sarebbe invocato il DNS, che in caso di mancanza di connessione può portare anche a lunghi ritardi (dovuti all'attesa di una risposta dal DNS) nella partenza dei programmi più comuni.

Il formato del file è molto semplice, ogni linea definisce una associazione fra indirizzo numerico e indirizzo simbolico, le linee vuote o che iniziano per `#` vengono ignorate. La man page fornisce una descrizione completa.

Le singole righe hanno il formato:

```
numero(IP) hostname alias
```

Un esempio di questo file è:

```
127.0.0.1       localhost
192.168.1.1     roke.earthsea.ea        roke

# The following lines are desirable for IPv6 capable hosts
# (added automatically by netbase upgrade)

::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

2.8 Il file /etc/modules

Questo file contiene la lista dei moduli che devono essere caricati all'avvio del sistema. Il formato del file è sempre lo stesso, ogni linea deve contenere il nome di un modulo; le linee vuote o che iniziano per # vengono ignorate.

Un possibile esempio di questo file è:

```
# /etc/modules: kernel modules to load at boot time.
#
# This file should contain the names of kernel modules that are
# to be loaded at boot time, one per line.  Comments begin with
# a #, and everything on the line after them are ignored.
#ide-floppy
auto
#
# I2C adapter drivers
i2c-isa
i2c-ali15x3
# I2C chip drivers
w83781d
eeprom
```

2.9 Il file /etc/ld.so.conf

Questo file contiene la lista delle directory che contengono le librerie dinamiche usate dai programmi oltre alle canoniche /lib e /usr/lib.

Pertanto se si installa una nuova libreria dai sorgenti e /usr/local/lib non compare in /etc/ld.so.conf sarà necessario aggiungerlo e poi fare girare il programma ldconfig per aggiornare i link alle librerie dinamiche disponibili, in modo che il linker dinamico possa utilizzarle.

Un esempio di questo file è il seguente:

```
/usr/X11R6/lib/Xaw3d
/usr/X11R6/lib/neXtaw
/usr/local/lib
/usr/X11R6/lib
```

2.10 I file /etc/logrotate.conf e /etc/logrotate.d/

Questo file e la directory citata contengono i settaggi per la rotazione dei vari file di log del sistema.

I file di log (la cui produzione è governata dal demone `syslogd` sulla cui configurazione torneremo dopo) sono una delle caratteristiche più utili di un sistema unix in quanto vi sono registrati messaggi (errori, avvertimenti, notifiche, ecc.) dai vari servizi e sono di importanza fondamentale per capire le ragioni di eventuali malfunzionamenti.

Il problema coi file di log è che tendono a crescere di dimensione finendo fuori controllo e riempiendo la directory `/var/log/` in cui normalmente risiedono. Per questo esiste il programma `logrotate` che lanciato su base periodica (di solito da `crond` nei comandi eseguiti in `/etc/cron.weekly`) gestisce la rotazione (con tanto di eventuale compressione, rimozione delle versioni troppo vecchie, mail di avviso all'amministratore) dei file di log specificati.

Il formato del file è sempre lo stesso, le linee vuote o che iniziano per `#` vengono ignorate. Il comando `man logrotate` fornisce una descrizione completa delle varie opzioni del file. Nel file `/etc/logrotate.conf` ci sono solo le opzioni generali, le varie opzioni per ciascun servizio vengono messe nella directory `/etc/logrotate.d/` in cui ogni pacchetto inserisce un opportuno file di configurazione per `logrotate` all'installazione; questa viene poi incluso dall'apposita direttiva `include /etc/logrotate.d`.

Un esempio di questo file è il seguente, i commenti spiegano in maniera molto chiara il significato delle varie opzioni.

```
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

# send errors to root
errors root

# create new (empty) log files after rotating old ones
create

# uncomment this if you want your log files compressed
#compress

# RPM packages drop log rotation information into this directory
include /etc/logrotate.d

# no packages own wtmp or btmp -- we'll rotate them here
/var/log/wtmp {
    monthly
    create 0664 root utmp
}

/var/log/btmp {
    missingok
    monthly
    create 0664 root utmp
    rotate 1
}

# system-specific logs may be configured here
```

2.11 Il file /etc/profile

Questo si può considerare il file di configurazione della shell, in realtà è lo script che viene lanciato ogni volta che si apre una shell di login. L'argomento è stato trattato nella relativa sezione **??** e a questa vi rimando.

2.12 Il file rc.local

In genere questo file si trova fra gli script di avvio (in /etc/rc.d o in /etc/) e viene eseguito alla fine della procedura di startup, assume un po' il significato di quello che è l'autoexec.bat del DOS, contiene i comandi che si vogliono eventualmente dare dopo che tutti i servizi sono partiti. Trattandosi di uno script di shell non si tratta propriamente di un file di configurazione.

2.13 /etc/securetty

Questo file contiene la lista delle console da cui si può collegare root. Viene usato dal programma login, che legge da esso i nomi dei device di terminale (le tty) dai quali è consentito l'accesso a root.

Un esempio di questo file è il seguente:

```
# Standard consoles
tty1
tty2
tty3
tty4
tty5
tty6
# Same as above, but these only occur with devfs devices
vc/1
vc/2
vc/3
vc/4
vc/5
vc/6
```

Dato che le console virtuali non sono indicate in questo file non è normalmente possibile fare telnet da root su questa macchina. Benché sia possibile consentirlo aggiungendo una riga, non è assolutamente una buona idea per cui se volete farlo dovrete studiarvelo da soli.

Il formato del file è sempre lo stesso, ogni linea definisce un nome di device dal quale è possibile il login, le linee vuote o che iniziano per # vengono ignorate. La man page fornisce una descrizione completa.

2.14 Il file /etc/lilo.conf

Questo è il file di configurazione che contiene le opzioni usate dal boot loader *lilo* scritte nel settore di avvio del disco. Si tenga conto che, contrariamente agli altri file, non basta modificarlo perché i cambiamenti diventino effettivi, ma bisogna anche fare girare il programma lilo che registra i cambiamenti.

In genere si ha a che fare con questo file tutte le volte che si ricompilare un kernel. Infatti per poter lanciare un nuovo kernel occorre specificarne la posizione al bootloader in modo che esso possa caricarlo.

Il formato del file è sempre lo stesso le linee vuote o che iniziano per # vengono ignorate. Le altre linee indicano una opzione. La man page contiene una lista completa delle opzioni disponibili.

Un esempio di questo file è il seguente:

```
# /etc/lilo.conf - See: 'lilo(8)' and 'lilo.conf(5)',
# ---------------       'install-mbr(8)', '/usr/share/doc/lilo/',
#                       and '/usr/share/doc/mbr/'.

# +-----------------------------------------------------------------+
# |                        !! Reminder !!                           |
# |                                                                 |
# | Don't forget to run 'lilo' after you make changes to this       |
# | conffile, '/boot/bootmess.txt', or install a new kernel.  The   |
# | computer will most likely fail to boot if a kernel-image        |
# | post-install script or you don't remember to run 'lilo'.        |
# |                                                                 |
# +-----------------------------------------------------------------+

# Support LBA for large hard disks.
#
lba32

# Specifies the boot device.  This is where Lilo installs its boot
# block.  It can be either a partition, or the raw device, in which
# case it installs in the MBR, and will overwrite the current MBR.
#
boot=/dev/hda

# Specifies the device that should be mounted as root. ('/')
#
root=/dev/hdb5

# Enable map compaction:
# Tries to merge read requests for adjacent sectors into a single
# read request. This drastically reduces load time and keeps the
# map smaller.  Using 'compact' is especially recommended when
# booting from a floppy disk.  It is disabled here by default
# because it doesn't always work.
#
# compact

# Installs the specified file as the new boot sector
#
install=/boot/boot.b

# Specifies the location of the map file
#
map=/boot/map

# You can set a password here, and uncomment the 'restricted' lines
```

```
# in the image definitions below to make it so that a password must
# be typed to boot anything but a default configuration.  If a
# command line is given, other than one specified by an 'append'
# statement in 'lilo.conf', the password will be required, but a
# standard default boot will not require one.
#
# This will, for instance, prevent anyone with access to the
# console from booting with something like 'Linux init=/bin/sh',
# and thus becoming 'root' without proper authorization.
#
# Note that if you really need this type of security, you will
# likely also want to use 'install-mbr' to reconfigure the MBR
# program, as well as set up your BIOS to disallow booting from
# removable disk or CD-ROM, then put a password on getting into the
# BIOS configuration as well.  Please RTFM 'install-mbr(8)'.
#
# password=tatercounter2000

# Specifies the number of deciseconds (0.1 seconds) LILO should
# wait before booting the first image.
#
delay=30

# You can put a customized boot message up if you like.  If you use
# 'prompt', and this computer may need to reboot unattended, you
# must specify a 'timeout', or it will sit there forever waiting
# for a keypress.  'single-key' goes with the 'alias' lines in the
# 'image' configurations below.  eg: You can press '1' to boot
# 'Linux', '2' to boot 'LinuxOLD', if you uncomment the 'alias'.
#
# message=/boot/bootmess.txt
        prompt
#       single-key
#       delay=100
        timeout=30

# Specifies the VGA text mode at boot time. (normal, extended, ask,  mode)
#
 vga=ask
# vga=0x31a
#
#vga=normal

# Kernel command line options that apply to all installed images go
# here.  See: The 'boot-prompt-HOWO' and 'kernel-parameters.txt' in
# the Linux kernel 'Documentation' directory.
#
# append=""

# Boot up Linux by default.
```

```
#
default=linux244

#       restricted
#       alias=1

image=/boot/vmlinuz-2.2.17_old
        label=LinuxOLD
        read-only
        optional
#       restricted
#       alias=2

image=/boot/vmlinuz-2.4.3
        label=linux24
        read-only
        optional

image=/boot/vmlinuz-2.4.4
        label=linux244
        read-only
        optional

image=/boot/vmlinuz-2.2.18
        label=linux
        read-only
        optional

image=/boot/vmlinuz-2.2.19
        label=linuxnew
        read-only
        optional


# If you have another OS on this machine to boot, you can uncomment the
# following lines, changing the device name on the 'other' line to
# where your other OS' partition is.
#
other=/dev/hda1
        label=dos
#       restricted
#       alias=3
```

Le opzioni più importanti sono:

- `image` indica un immagine del kernel da usare nel boot, prende un pathname valido
- `other` Identifica un diverso bootloader, da specificare con il device indicante la partizione su cui lo si è installato, nel caso il bootloader di Windows
- `label` associa un'etichetta con la quale riconoscere le varie immagini o bootloader
- `root` indica quale partizione montare come radice

- `default` indica quale bootloader o immagine del kernel sarà fatta partire di default
- `prompt` indica di aspettare l'immissione di una label all'avvio, permettendo la scelta fra diversi kernel/sistemi
- `delay` indica il tempo in cui aspetta al prompt prima di avviare l'immagine di default

2.15 Il file `/etc/syslog.conf`

È il file di configurazione per il demone `syslogd` che registra i messaggi di sistema. Il formato del file è sempre lo stesso, ogni linea definisce una regola di registrazione, le linee vuote o che iniziano per `#` vengono ignorate. La man page fornisce una descrizione completa.

Ogni regola è costituita da due campi separati da spazi o tabulatori; i primo campo è detto *selettore*, il secondo *azione*. Il campo selettore è costituito da due parti, il *servizio* e la *priorità*, separate da un punto.

Il *servizio* identifica una categoria di servizi di sistema dei quali si vuole registrare i messaggi, e può essere una delle seguenti parole chiave: `auth`, `authpriv`, `cron`, `daemon`, `kern`, `lpr`, `mail`, `mark`, `news`, `syslog`, `user`, `uucp` per i servizi standard, ci sono poi i servizi da `local0` a `local7` lasciati a disposizione dell'utente.

La *priorità* può avere uno dei seguenti valori: `debug`, `info`, `notice`, `warning`, `error`, `crit`, `alert`, `emerg` e identifica l'importanza del messaggio, tutti i messaggi di priorità superiore od uguale a quella indicata verranno registrati.

Oltre a queste parole chiave `syslogd` riconosce alcune estensioni, un asterisco `*` seleziona tutte i servizi o tutte le priorità mentre la parola `none` li esclude tutti, una `,` permette di elencare una lista di servizi per la stessa priorità, o viceversa una lista di priorità per un servizio.

Si possono infine associare più selettori ad una stessa azione separandoli con il `;` ed ulteriori estensione di questa sintassi sono date dal segno `=` che permette di registrare solo una specifica priorità o `!` per escludere una specifica priorità.

L'*azione* è un termine astratto per descrivere come si registrano i messaggi, il caso più comune è un file, in questo caso esso dovrà essere specificato dal pathname completo, con un `-` opzionale davanti per impedire che questo venga sincronizzato ad ogni messaggio, lasciando spazio per la bufferizzazione.

La potenza di `syslogd` è comunque quella di permettere di effettuare le registrazioni in maniera estremamente flessibile, ad esempio se si premette un `|` al nome del file si può usare una named pipe, oppure si può mandare l'output su una console specificando un device tty, si può poi mandare tutto quanto ad una macchina remota usando il carattere `@` prima di un hostname. Si possono poi mandare i messaggi a liste di utenti identificati per username e separate da virgole, infine un `*` manderà i messaggi a chiunque sia loggato.

Un esempio di questo file è il seguente, preso dal mio sistema:

```
#  /etc/syslog.conf     Configuration file for syslogd.
#
#                       For more information see syslog.conf(5)
#                       manpage.

#
# First some standard logfiles.  Log by facility.
#

auth,authpriv.*                 /var/log/auth.log
*.*;auth,authpriv.none          -/var/log/syslog
#cron.*                         /var/log/cron.log
```

```
daemon.*                        -/var/log/daemon.log
kern.*                          -/var/log/kern.log
lpr.*                           -/var/log/lpr.log
mail.*                          /var/log/mail.log
user.*                          -/var/log/user.log
uucp.*                          -/var/log/uucp.log

#
# Logging for the mail system. Split it up so that
# it is easy to write scripts to parse these files.
#
mail.info                       -/var/log/mail.info
mail.warn                       -/var/log/mail.warn
mail.err                        /var/log/mail.err
#
# Some 'catch-all' logfiles.
#
*.=debug;\
   auth,authpriv.none;\
   news.none;mail.none     -/var/log/debug
*.=info;*.=notice;*.=warn;\
   auth,authpriv.none;\
   cron,daemon.none;\
   mail,news.none          -/var/log/messages

#
# Emergencies are sent to everybody logged in.
*.emerg                         *

#
# I like to have messages displayed on the console, but only on a virtual
# console I usually leave idle.
#
#daemon,mail.*;\
#       news.=crit;news.=err;news.=notice;\
#       *.=debug;*.=info;\
#       *.=notice;*.=warn       /dev/tty8

# The named pipe /dev/xconsole is for the 'xconsole' utility.  To use it,
# you must invoke 'xconsole' with the '-file' option:
#
#    $ xconsole -file /dev/xconsole [...]
#
# NOTE: adjust the list below, or you'll go crazy if you have a reasonably
#      busy site..
#
daemon.*;mail.*;\
      news.crit;news.err;news.notice;\
      *.=debug;*.=info;\
```