

Amministrare GNU/Linux

Simone Piccardi
piccardi@truelite.it

Copyright © 2004-2005 Simone Piccardi Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with Front-Cover Texts: “Truelite Srl <http://www.truelite.it> info@truelite.it”, and with no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Questa documentazione libera è stata sviluppata all'interno delle attività formative effettuate da Truelite S.r.l. Il materiale è stato finanziato nel corso della realizzazione dei corsi erogati dall'azienda, e viene messo a disposizione di tutti sotto licenza GNU FDL.

Questo testo, insieme al resto della documentazione libera realizzata da Truelite S.r.l., viene distribuito su internet all'indirizzo:

<http://svn.truelite.it/truedoc>

dove saranno pubblicate nuove versioni ed aggiornamenti.



Società italiana specializzata nella fornitura di servizi, consulenza e formazione esclusivamente su GNU/Linux e software libero.

Per informazioni:

Truelite S.r.l

Via Monferrato 6,

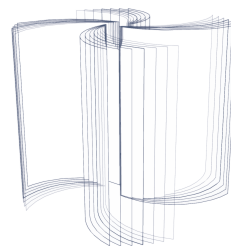
50142 Firenze.

Tel: 055-7879597

Fax: 055-7333336

e-mail: info@truelite.it

web: <http://www.truelite.it>



Indice

1	L'architettura di un sistema GNU/Linux	1
1.1	L'architettura del sistema.	1
1.1.1	L'architettura di base.	1
1.1.2	Il funzionamento del sistema	3
1.1.3	Alcune caratteristiche specifiche di Linux	4
1.2	L'architettura dei file	5
1.2.1	Il <i>Virtual File System</i> e le caratteristiche dei file.	6
1.2.2	L'architettura di un filesystem e le proprietà dei file	9
1.2.3	L'organizzazione delle directory ed il <i>Filesystem Hierarchy Standard</i> . . .	14
1.2.4	La gestione dell'uso di dischi e volumi	20
1.3	L'architettura dei processi	25
1.3.1	Le proprietà dei processi	26
1.3.2	I segnali	35
1.3.3	Priorità	37
1.3.4	Sessioni di lavoro e <i>job control</i>	39
1.4	Il controllo degli accessi	41
1.4.1	Utenti e gruppi	41
1.4.2	I permessi dei file	43
1.4.3	I permessi speciali	44
1.4.4	La gestione dei permessi dei file	46
1.4.5	Altre operazioni privilegiate	47
2	La shell e i comandi	51
2.1	L'interfaccia a linea di comando.	51
2.1.1	La filosofia progettuale	51
2.1.2	Le principali shell	52
2.1.3	Introduzione alla sintassi della riga di comando	53
2.1.4	Funzionalità interne della shell	55
2.1.5	La redirectione dell'I/O	63
2.1.6	Scripting elementare	67
2.2	I comandi dei file	74
2.2.1	Caratteristiche comuni	75
2.2.2	I comandi per le ricerche sui file	75
2.2.3	I comandi per visualizzare il contenuto dei file	82
2.2.4	I comandi per suddividere il contenuto dei file	83
2.2.5	I comandi per filtrare il contenuto dei file	85
2.2.6	Altri comandi dei file	88
2.3	Altri comandi	89
2.3.1	I comandi per la documentazione	89
2.3.2	I comandi per la gestione dei tempi	92

2.3.3	I comandi di ausilio per la redirectione	94
2.4	Gli editor di testo	95
2.4.1	Introduzione	95
2.4.2	Un editor evoluto: emacs o xemacs	96
2.4.3	Un editor di base, vi	99
2.4.4	Gli altri editor	101
3	La configurazione dei servizi di base	105
3.1	I file di configurazione	105
3.1.1	Una panoramica generale	105
3.1.2	La gestione delle librerie condivise	106
3.1.3	Il <i>Name Service Switch</i>	108
3.1.4	I file usati dalla procedura di <i>login</i>	110
3.1.5	La configurazione del sistema delle pagine di manuale	111
3.2	Altri file	113
3.2.1	Il file rc.local e la directory rc.boot	113
3.2.2	La directory /etc/skel ed il file /etc/shells	113
3.2.3	Il file /etc/updatedb.conf	114
3.3	I servizi di base	114
3.3.1	Il servizio <i>cron</i>	115
3.3.2	Il servizio <i>at</i>	116
3.3.3	Il servizio <i>syslog</i>	117
3.3.4	Il sistema di rotazione dei file di log	120
3.4	L' <i>X Window System</i>	122
3.4.1	Introduzione a <i>X Window System</i>	122
3.4.2	La configurazione del server X	123
3.4.3	L'avvio del server	129
3.4.4	L'uso di X Window dal lato client	130
3.5	Il sistema di stampa	132
3.5.1	Introduzione generale	132
3.5.2	Il sistema di stampa in stile BSD	133
3.5.3	La configurazione della stampa con LPRng	135
3.5.4	Il <i>Common Unix Printing System</i>	139
4	Amministrazione ordinaria del sistema	147
4.1	Archiviazione e backup	147
4.1.1	Criteri generali per il backup	147
4.1.2	Il comando tar	149
4.1.3	Il comando cpio	152
4.1.4	I comandi dump e restore	153
4.2	La gestione dei pacchetti software	155
4.2.1	L'installazione diretta	155
4.2.2	La gestione dei pacchetti con rpm	157
4.2.3	Il sistema di gestione dei pacchetti APT	158
4.3	La gestione di utenti e gruppi	160
4.3.1	Una visione generale	160
4.3.2	I comandi per la gestione di utenti e gruppi	161
4.3.3	Il database di utenti e gruppi	164
4.3.4	Il <i>Pluggable Authentication Method</i>	168

5	Amministrazione straordinaria del sistema	171
5.1	La gestione di kernel e moduli	171
5.1.1	Le versioni del kernel	171
5.1.2	Sorgenti e <i>patch</i>	172
5.1.3	La ricompilazione del kernel	175
5.1.4	La gestione dei moduli	185
5.2	La gestione dei dischi e dei filesystem	191
5.2.1	Alcune nozioni generali	191
5.2.2	Il partizionamento	193
5.2.3	La creazione di un filesystem	197
5.2.4	Controllo e riparazione di un filesystem	201
5.2.5	La gestione della <i>swap</i>	205
5.3	La gestione dell'avvio del sistema	207
5.3.1	L'avvio del kernel	207
5.3.2	L'uso di <i>LILO</i>	209
5.3.3	L'uso di GRUB	213
5.3.4	Il sistema di inizializzazione alla SysV	215
5.4	La gestione di interfacce e periferiche	219
5.4.1	Gestione delle interfacce di espansione	219
5.4.2	Gestione delle interfacce SCSI	226
5.4.3	Gestione delle interfacce seriali	231
5.4.4	Gestione delle interfacce USB	232
6	Amministrazione avanzata del sistema	237
6.1	L'utilizzo del RAID	237
6.1.1	Introduzione	237
6.1.2	Il RAID su Linux	240
6.1.3	Il RAID software	240
6.2	Il sistema del <i>Logical Volume Manager</i>	244
6.2.1	Introduzione	244
6.2.2	La gestione dei <i>volumi fisici</i>	246
6.2.3	La gestione dei <i>gruppi di volumi</i>	247
6.2.4	La gestione dei <i>volumi logici</i>	249
6.2.5	Il ridimensionamento dei filesystem	251
6.3	Le quote disco	252
6.3.1	Visione generale	252
6.3.2	Configurazione del sistema delle quote	253
6.3.3	Gestione delle quote di utenti e gruppi	255
7	L'amministrazione di base delle reti	259
7.1	Un'introduzione ai concetti fondamentali delle reti.	259
7.1.1	L'estensione	259
7.1.2	La topologia	260
7.1.3	I protocolli	261
7.2	Il TCP/IP.	264
7.2.1	Introduzione.	265
7.2.2	Gli indirizzi IP	267
7.2.3	L'instradamento o <i>routing</i>	270
7.2.4	I servizi e le porte.	271
7.3	La configurazione di base	273
7.3.1	L'assegnazione degli indirizzi ed il comando ifconfig	273

7.3.2	L'impostazione dell'instradamento ed il comando route	276
7.3.3	La configurazione automatica all'avvio del sistema.	280
7.4	Il sistema della risoluzione dei nomi	283
7.4.1	Introduzione al <i>resolver</i>	284
7.4.2	I file di configurazione del <i>resolver</i>	285
7.4.3	La gestione locale dei nomi	287
7.4.4	La gestione degli altri nomi di rete	288
7.5	I comandi diagnostici	290
7.5.1	Il comando ping	290
7.5.2	I comandi traceroute ed mtr	291
7.5.3	Il comando netstat	293
7.5.4	Il protocollo ARP ed il comando arp	295
7.5.5	I servizi RPC	297
7.6	I client dei servizi di base	298
7.6.1	I comandi telnet e netcat	298
7.6.2	Il comando ftp	301
7.6.3	I comandi finger e whois	302
7.7	Le connessioni di rete con PPP	303
7.7.1	Cenni sul protocollo PPP	304
7.7.2	Il demone pppd	304
7.7.3	I meccanismi di autenticazione	306
8	La gestione dei servizi di base	309
8.1	I programmi di ausilio alla gestione dei servizi di base	309
8.1.1	I servizi elementari e i super-demoni	309
8.1.2	Il super-demone inetd	310
8.1.3	Il super-demone xinetd	312
8.1.4	I TCP wrappers	316
8.2	L'assegnazione dinamica degli indirizzi IP	318
8.2.1	I protocolli RARP, BOOTP e DHCP	319
8.2.2	Uso del servizio DHCP dal lato client	321
8.2.3	La configurazione di un server DHCP	323
8.3	Il servizio SSH	326
8.3.1	Il server sshd	326
8.3.2	I comandi ssh ed scp	328
8.3.3	Autenticazione a chiavi	329
8.4	Il protocollo NFS	331
8.4.1	Il server NFS	331
8.4.2	NFS sul lato client	333
8.5	La condivisione dei file con Samba	334
8.5.1	La configurazione di Samba come server	334
8.5.2	L'impostazione degli utenti	336
8.5.3	L'uso di Samba dal lato client	337
9	Il servizio DNS	339
9.1	Il funzionamento del servizio DNS	339
9.1.1	Introduzione	339
9.1.2	I comandi host e dig	340
9.2	La gestione di un server DNS	342
9.2.1	Il server named	342
9.2.2	Il file named.conf	343

9.2.3	La configurazione base	344
9.2.4	La configurazione di un dominio locale.	346
9.3	Configurazioni avanzate	351
9.3.1	La delegazione di una zona	351
9.3.2	La gestione di un secondario	351
A	Sinossi dei comandi principali	353
A.1	Comandi per la gestione dei file	353
A.2	Comandi per la gestione dei processi	354
A.3	I permessi dei file	354
A.4	Comandi per la localizzazione dei file	354
A.5	Comandi per la documentazione	355
A.6	Comandi per la gestione dei tempi	355
A.7	Comandi di archiviazione e compressione	355
A.8	Gestione dei pacchetti	356
A.9	I comandi diagnostici	356
A.10	I client dei servizi base	356
B	Indice degli argomenti per LPI	357
B.1	Argomenti LPI 101	357
B.2	Argomenti LPI 102	358
C	GNU Free Documentation License	359
C.1	Applicability and Definitions	359
C.2	Verbatim Copying	360
C.3	Copying in Quantity	360
C.4	Modifications	361
C.5	Combining Documents	362
C.6	Collections of Documents	363
C.7	Aggregation With Independent Works	363
C.8	Translation	363
C.9	Termination	363
C.10	Future Revisions of This License	363

Capitolo 1

L'architettura di un sistema GNU/Linux

1.1 L'architettura del sistema.

Prima di addentrarci nei dettagli dell'amministrazione di un sistema GNU/Linux, conviene fornire un quadro generale per introdurre i vari concetti su cui si basa l'architettura di questo sistema, che è basata su quella, consolidatasi in 30 anni di impiego, dei sistemi di tipo Unix.

Il fatto che questa architettura abbia una certa età fa sì che spesso i detrattori di GNU/Linux ne denuncino la presunta mancanza di *innovatività*, ma anche le case hanno da secoli le stesse basi architettoniche (porte, muri e tetti), ma non per questo non esiste innovazione. Il vantaggio della architettura di Unix infatti è quello di aver fornito una solida base per la costruzione di sistemi affidabili ed efficienti, consolidata e corretta in decenni di utilizzo, tanto che ormai è divenuto comune il detto che *chi non usa l'architettura Unix è destinato a reinventarla*.

1.1.1 L'architettura di base.

Contrariamente ad altri sistemi operativi, GNU/Linux nasce, come tutti gli Unix, come sistema multitasking e multiutente. Questo significa che GNU/Linux ha una architettura di sistema che è stata pensata fin dall'inizio per l'uso contemporaneo da parte di più utenti. Questo comporta conseguenze non del tutto intuitive nel caso in cui, come oggi sempre più spesso accade, esso venga usato come stazione di lavoro da un utente singolo.

Il concetto base dell'architettura di ogni sistema Unix come GNU/Linux è quello di una rigida separazione fra il *kernel* (il *nucleo* del sistema), cui si demanda la gestione delle risorse hardware, come la CPU, la memoria, le periferiche e i *processi*, (le unità di esecuzione dei programmi), che nel caso vanno dai comandi base di sistema, agli applicativi, alle interfacce per l'interazione con gli utenti.

Lo scopo del kernel infatti è solo quello di essere in grado di eseguire contemporaneamente molti processi in maniera efficiente, garantendo una corretta distribuzione fra gli stessi della memoria e del tempo di CPU, e quello di fornire le adeguate interfacce software per l'accesso alle periferiche della macchina e le infrastrutture di base necessarie per costruire i servizi. Tutto il resto, dall'autenticazione all'interfaccia utente, viene realizzato usando processi che eseguono gli opportuni programmi.

Questo si traduce in una delle caratteristiche essenziali su cui si basa l'architettura dei sistemi Unix: la distinzione fra il cosiddetto *user space*, che è l'ambiente a disposizione degli utenti, in cui vengono eseguiti i processi, e il *kernel space*, che è l'ambiente in cui viene eseguito il kernel. I due ambienti comunicano attraverso un insieme di interfacce ben definite e standardizzate; secondo una struttura come quella mostrata in fig. 1.1.

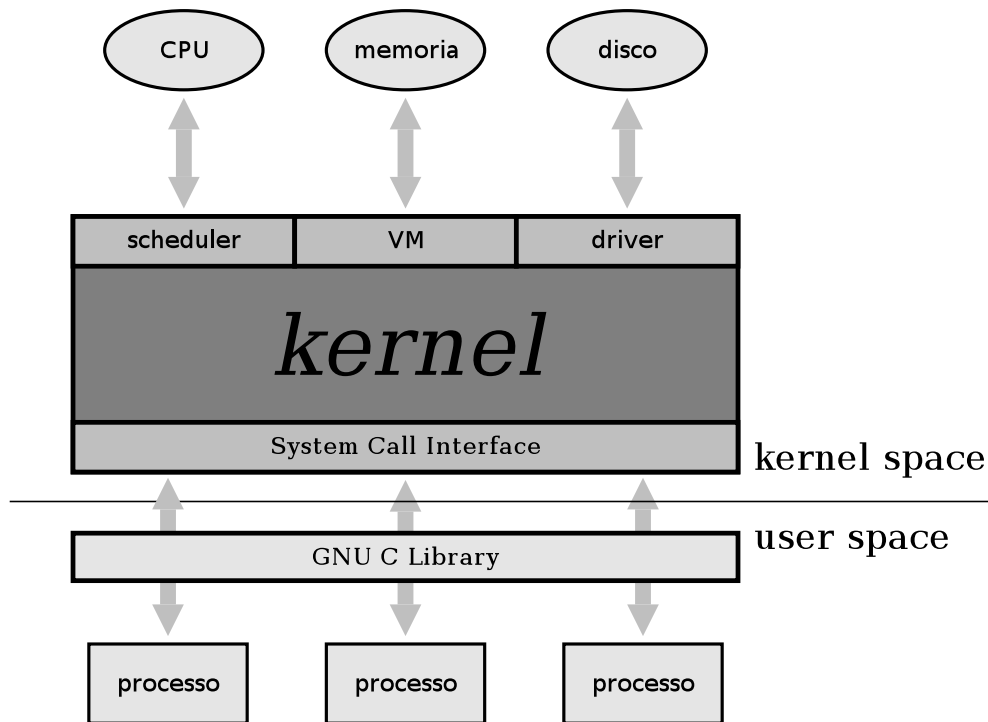


Figura 1.1: Schema della struttura del sistema operativo GNU/Linux.

Questa architettura comporta che solo il kernel viene eseguito in modalità privilegiata, ed è l'unico a poter accedere direttamente alle risorse dell'hardware. I normali programmi invece verranno eseguiti in modalità protetta, in un ambiente virtuale, l'*user space*, in cui essi vedono se stessi come se avessero piena disponibilità della CPU e della memoria, ma in cui possono accedere alle periferiche e alle altre funzionalità messe a disposizione del kernel solo attraverso una serie di *funzioni di sistema* standardizzate,¹ le cosiddette *system call*, che vengono utilizzate attraverso l'interfaccia fornita dalla libreria di sistema (la *GNU C library* di fig. 1.1).

In sostanza quello che succede è che da un certo punto di vista l'unico "vero" programma che viene eseguito è il kernel che si incarica di costruire questo ambiente virtuale in cui fare girare gli altri programmi. Una parte del kernel, quella indicata in fig. 1.1 come *scheduler*, si incaricherà della gestione del tempo di processore e provvederà a decidere volta per volta qual'è il processo che deve essere eseguito in un determinato momento (realizzando così il multitasking).

Una seconda parte, quella indicata in fig. 1.1 come *VM*, (sigla che sta per *Virtual Memory*) si occuperà invece di gestire l'uso della memoria disponibile. La *memoria virtuale* è uno dei sottosistemi più importanti del kernel² perché è quella che fa in modo che ogni processo veda uno spazio di indirizzi proprio che poi viene rimappato³ nella memoria fisica effettivamente presente, così che sia impossibile che un processo possa accedere alla memoria di un altro processo. La *memoria virtuale* si incarica anche di gestire, in caso di esaurimento della RAM, l'eventuale spostamento delle pagine di memoria meno usate su uno opportuno spazio disco (lo *swap*, che tratteremo in sez. 5.2.5) evitando di fermare l'esecuzione di un processo per una temporanea mancanza di memoria. Torneremo brevemente sull'argomento in sez. 1.3.1.

Infine c'è un'ultima parte del kernel, indicata in fig. 1.1 con l'indicazione generica *driver*,⁴ che

¹lo standard in questo caso si chiama POSIX.1, ma oltre quelle dello standard Linux supporta alcune *system call* ulteriori relative a sue estensioni specifiche.

²e oggetto di continui rifacimenti, in quanto critico per tutte le prestazioni del sistema.

³questo viene in genere realizzato con l'ausilio delle MMU (*Memory Management Unit*) dei microprocessori, una descrizione più dettagliata del funzionamento della *memoria virtuale* può essere trovata nella sezione 2.2 di [1].

⁴questa in realtà non è un unico sottosistema come le precedenti, ma un insieme di varie parti diverse, specifiche

si incaricherà di accedere alle periferiche per conto dei programmi. Questa, come vedremo meglio in sez. 1.2, permette di definire una interfaccia di accesso generica per qualunque dispositivo, cui spesso si fa riferimento dicendo che in un sistema unix-like *tutto è un file*.

La conseguenza più importante di questa separazione fra *user space* e *kernel space* è che in questo modo non è possibile che un singolo programma possa disturbare l'azione di un altro programma o del kernel stesso, e questo è il principale motivo della stabilità di un sistema Unix nei confronti di altri sistemi in cui i processi non hanno di questi limiti, o vengono, per vari motivi, eseguiti all'interno del kernel.

1.1.2 Il funzionamento del sistema

Per illustrare meglio la distinzione fra kernel space e user space prendiamo in esame brevemente in esame la procedura di avvio del sistema, su cui torneremo in dettaglio in sez. 5.3.4. All'accensione del computer viene eseguito il programma che sta nel BIOS; questo dopo aver fatto i suoi controlli interni esegue la procedura di avvio del sistema. Nei PC tutto ciò viene effettuato caricando dal dispositivo indicato nelle impostazioni del BIOS un apposito programma, il *bootloader*,⁵ che a sua volta recupera (in genere dal disco) una immagine del kernel che viene caricata in memoria ed eseguita.

Una volta che il controllo è passato al kernel questo, terminata la fase di inizializzazione (in cui ad esempio si esegue una scansione delle periferiche disponibili, e si leggono le tabelle delle partizioni dei vari dischi) si incaricherà di montare (vedi sez. 1.2.2) il filesystem su cui è situata la *directory radice* (vedi sez. 1.2.3), e farà partire il primo processo. Per convenzione questo processo si chiama *init*, ed è il programma di inizializzazione che a sua volta si cura di far partire tutti gli altri processi che permettono di usare il sistema.

Fra questi processi ci saranno ad esempio quelli che forniscono i vari servizi di rete, quelli che eseguono vari compiti di amministrazione, così come quelli che si occupano di chiedere nome e password dell'utente che si vuole collegare,⁶ e che una volta completato il *collegamento* (procedura che viene chiamata *login*) lanciano altri programmi per mettere a disposizione dell'utente l'interfaccia da cui inviare i comandi, che potrebbe essere sia una shell a riga di comando (argomento che tratteremo in dettaglio in cap. 2) che una delle tante interfacce grafiche disponibili (argomento che riprenderemo in sez. 3.4).

È da rimarcare poi come anche tutti i programmi che un utente di un sistema GNU/Linux può utilizzare una volta che si è collegato non hanno niente di diverso da quelli appena citati. Tutti i programmi funzionano allo stesso modo: vengono eseguiti dal kernel come processi ed eseguono le loro operazioni attraverso le opportune *system call* che esso mette a disposizione. Da questo punto di vista eseguire sulla shell un programma per vedere la lista dei file non ha niente di diverso dall'eseguire in ambiente grafico un programma di scrittura o un programma di fotoritocco, o dal lanciare un server web che viene eseguito anche quando nessuno è collegato al sistema.

Questo significa ad esempio che il kernel di per sé non dispone di primitive per tutta una serie di operazioni, come la copia di un file,⁷ che altri sistemi operativi (come Windows) hanno al loro interno: tutte le operazioni di normale amministrazione di un sistema GNU/Linux sono sempre realizzate tramite degli opportuni programmi.

per il tipo di periferica in questione.

⁵questo è un programma speciale, il cui solo compito è quello di far partire un sistema operativo, in genere ogni sistema ha il suo, nel caso di Linux per l'architettura PC i due principali sono LILO e GRUB, che vedremo in sez. 5.3.

⁶in realtà se la cosa è fatta da console i programmi sono due, il primo chiede l'utente e poi chiama il secondo che chiede la password, torneremo su questo in sez. 3.1.4 e sez. 5.3.4.

⁷questa infatti viene eseguita usando semplicemente le funzioni che permettono di leggere e scrivere il contenuto di un file, leggendo l'originale e scrivendo sulla copia.

Tutto ciò ci dice anche che benché costituisca il cuore del sistema, il kernel da solo sarebbe assolutamente inutile, così come sarebbe inutile da solo il motore di una automobile, senza avere le ruote, lo sterzo, la carrozzeria, e tutto il resto. Per avere un sistema funzionante dal punto di vista di un utente normale infatti occorre avere, oltre al kernel, anche tutti i programmi che gli permettano di eseguire le varie operazioni con i dischi, i file, le periferiche.

Per questo al kernel vengono sempre uniti degli opportuni programmi di gestione per il sistema e tutta una serie di programmi applicativi, ed è l'insieme di questi e del kernel che costituisce un sistema funzionante. Di solito i rivenditori, o anche gruppi di volontari, come nel caso di Debian, si preoccupano di raccogliere in forma coerente i programmi necessari, per andare a costruire quella che viene chiamata una *distribuzione*. Sono in genere queste distribuzioni (come Debian, Mandrake, RedHat, Slackware⁸), quelle che si trovano sui CD con i quali si installa quello che, con una semplificazione molto brutale, viene chiamato solo "Linux".

Il gruppo principale di questi programmi, e le librerie di base che essi e tutti gli altri programmi usano, derivano dal progetto GNU della Free Software Foundation: è su di essi che ogni altro programma è basato, ed è per questo che è più corretto riferirsi all'intero sistema come a GNU/Linux, dato che Linux indica solo una parte, il kernel, che benché fondamentale non costituisce da sola un sistema operativo.

Si tenga presente infine che anche se il kernel tratta tutti i programmi allo stesso modo, non tutti hanno la stessa importanza. Nell'esempio appena fatto abbiamo accennato ad un programma particolare, *init*, che ha un ruolo privilegiato in quanto è quello che si occupa dell'inizializzazione del sistema quando questo viene fatto partire. Anche *init* però alla fine non è che un programma che usa le system call e viene eseguito dal kernel come un qualunque altro processo; la sua unica peculiarità infatti è quella di essere lanciato per primo direttamente dal kernel.

Questo ci porta ad un'altra caratteristica fondamentale dell'architettura dei sistemi unix-like (ci torneremo in dettaglio in sez. 1.3) che è quella per cui qualunque processo può a sua volta avviarne di nuovi.⁹ È questo che permette l'avvio del sistema eseguendo un unico programma di inizializzazione come *init*, dato che questo potrà poi lanciare altri programmi, che a loro volta ne potranno lanciare degli altri ancora, fino a fornire tutte le funzionalità richieste per il funzionamento del sistema.

Benché sia possibile per usi particolari (ad esempio in sistemi *embedded*¹⁰ che devono svolgere un solo compito) far partire un qualunque altro programma al posto di *init*,¹¹ in pratica tutti i sistemi Unix usano questo specifico programma per gestire l'avvio del sistema, ed è a seconda degli ulteriori programmi che *init* mette in esecuzione (tratteremo l'argomento in sez. 5.3.4) che alla fine della procedura di avvio ci si troverà davanti ad un terminale a caratteri o ad una interfaccia grafica, e si avrà, a seconda di quanto deciso (ed installato) dall'amministratore, un server di posta, un server web, una workstation, ecc.

1.1.3 Alcune caratteristiche specifiche di Linux

Benché Linux stia diventando il più diffuso, esistono parecchi altri kernel unix-like, sia liberi che proprietari, nati nella tumultuosa e complessa evoluzione che dallo Unix originario della AT/T ha portato alla nascita di una miriade di sistemi derivati (BSD, Solaris, AIX, HP-UX, Digital Unix, IRIX, solo per citare i più noti) che si innestano tutti in due rami principali, quelli derivati dal sistema sviluppato dalla AT/T, detto SysV (da *System V*, l'ultima versione ufficiale) e quelli derivati dal codice sviluppato all'università di Berkley, detto BSD (da *Berkley Software*

⁸in rigoroso ordine alfabetico!

⁹nel qual caso si dice che il primo processo è il *padre* degli altri, che a loro volta sono chiamati *figli*.

¹⁰si chiamano così i sistemi destinati all'esecuzione di compiti specifici, come quelli dei telefonini, dei videoregistratori, ecc.

¹¹vedremo in sez. 5.3.4 come in casi di emergenza si può lanciare al suo posto una shell.

Distribution). La prima caratteristica distintiva di Linux è che esso è stato riscritto da zero, per cui non è classificabile in nessuno di questi due rami e prende invece, a seconda dei casi, le migliori caratteristiche di ciascuno di essi.

Un'altra delle caratteristiche peculiari di Linux rispetto agli altri kernel unix-like è quella di essere *modulare*; Linux cioè può essere esteso (torneremo su questo in sez. 5.1.4) inserendo a sistema attivo degli ulteriori “pezzi”, i *moduli*, che permettono di ampliare le capacità del sistema (ad esempio fargli riconoscere una nuova periferica). Questi possono poi essere tolti dal sistema in maniera automatica quando non sono più necessari: un caso tipico è quello del modulo che permette di vedere il floppy, caricato solo quando c'è necessità di leggere un dischetto ed automaticamente rimosso una volta che non sia più in uso per un certo tempo.

In realtà è sempre possibile costruire un kernel Linux comprensivo di tutti i moduli che servono, ottenendo quello che viene chiamato un kernel *monolitico* (come sono i kernel degli altri Unix); questo permette di evitare il ritardo nel caricamento dei moduli al momento della richiesta, ma comporta un maggiore consumo di memoria (dovendo tenere dentro il kernel anche codice non utilizzato), ed una flessibilità nettamente inferiore in quanto si perde la capacità di poter specificare eventuali opzioni al momento del caricamento, costringendo al riavvio in caso di necessità di cambiamenti.

Per contro in certi casi l'uso dei moduli può degradare leggermente le prestazioni (quasi sempre in maniera non avvertibile) e può dar luogo a conflitti inaspettati (che con un kernel monolitico avrebbero bloccato il sistema all'avvio). Questi problemi oggi sono sempre più rari, in ogni caso non è possibile utilizzare i moduli nel caso in cui la funzionalità da essi fornite siano necessarie ad avviare il sistema.

Una seconda peculiarità di Linux è quella del *Virtual File System* (o VFS). Un concetto generale presente in tutti i sistemi Unix (e non solo) è che lo spazio su disco su cui vengono tenuti i file di dati è organizzato in quello che viene chiamato un *filesystem* (tratteremo l'amministrazione dei *filesystem* in sez. 5.2). Lo spazio disco grezzo è normalmente¹² suddiviso in settori contigui di dimensione fissa, ma all'interno del sistema questo viene organizzato in maniera tale da permettere il rapido reperimento delle informazioni memorizzate su questi settori, anche quando queste sono essere sparse qua e là sul disco; si ha così quello che l'utente vede come un singolo file.

Quello che contraddistingue Linux è che l'interfaccia per la lettura del contenuto del filesystem è stata completamente virtualizzata, per cui inserendo gli opportuni moduli nel sistema diventa possibile accedere con la stessa interfaccia (e, salvo limitazioni della realizzazione, in maniera completamente trasparente all'utente) ai più svariati tipi di filesystem, a partire da quelli usati da Windows e dal DOS, dal MacOS, e da tutte le altre versioni di Unix.

Dato che essa gioca un ruolo centrale nel sistema, torneremo in dettaglio sull'interfaccia dei file (e di come possa essere usata anche per altro che i file di dati) in sez. 1.2; quello che è importante tenere presente da subito è che la disponibilità di una astrazione delle operazioni sui file rende Linux estremamente flessibile, dato che attraverso di essa è in grado di supportare con relativa facilità, ed in maniera nativa, una varietà di filesystem superiore a quella di qualunque altro sistema operativo.

1.2 L'architettura dei file

Un aspetto fondamentale della architettura di GNU/Linux è quello della gestione dei file, esso deriva direttamente da uno dei criteri base della progettazione di tutti i sistemi Unix, quello espresso dalla frase “*everything is a file*” (cioè *tutto è un file*), per cui l'accesso ai file e alle periferiche è gestito attraverso una interfaccia identica.

¹²nel senso che le interfacce hardware per i dischi consentono l'accesso diretto al contenuto di questi settori.

Inoltre, essendo in presenza di un sistema multiutente e multitasking, il kernel deve anche essere in grado di gestire l'accesso contemporaneo allo stesso file da parte di più processi, e questo viene fatto usando un disegno specifico nella struttura delle interfacce di accesso, che è uno dei punti di maggior forza della architettura di un sistema Unix.

1.2.1 Il *Virtual File System* e le caratteristiche dei file.

Come accennato in sez. 1.1.3 i file sono organizzati sui dischi all'interno di filesystem. Perché i file diventino accessibili al sistema un filesystem deve essere *montato* (torneremo su questo in sez. 1.2.4). Questa è una operazione privilegiata (che normalmente può fare solo l'amministratore) che provvede ad installare nel kernel le opportune interfacce (in genere attraverso il caricamento dei relativi moduli) che permettono l'accesso ai file contenuti nel filesystem.

Come esempio consideriamo il caso in cui si voglia leggere il contenuto di un CD. Il kernel dovrà poter disporre sia delle interfacce per poter parlare al dispositivo fisico (ad esempio il supporto SCSI, se il CDROM è SCSI), che di quelle per la lettura dal dispositivo specifico (il modulo che si interfaccia ai CDROM, che è lo stesso che questi siano su SCSI, IDE o USB), sia di quelle che permettono di interpretare il filesystem ISO9660 (che è quello che di solito viene usato per i dati registrati su un CDROM) per estrarne il contenuto dei file.

Allo stesso modo se si volessero leggere i dati su un dischetto occorrerebbe sia il supporto per l'accesso al floppy, che quello per poter leggere il filesystem che c'è sopra (ad esempio *vfat* per un dischetto Windows e *hfs* per un dischetto MacOS).

Come accennato nell'introduzione a questa sezione, uno dei criteri fondamentali dell'architettura di un sistema Unix è quello per cui *tutto è un file* e che altro non significa che si può accedere a tutte le periferiche¹³ con una interfaccia identica a quella con cui si accede al contenuto dei file. Questo comporta una serie di differenze nella gestione dei file rispetto ad altri sistemi.

Anzitutto in un sistema Unix tutti i file di dati sono uguali (non esiste la differenza fra file di testo o binari che c'è in Windows, né fra file sequenziali e ad accesso diretto che c'era nel VMS). Inoltre le estensioni sono solo convenzioni, e non significano nulla per il kernel, che legge tutti i file di dati alla stessa maniera, indipendentemente dal nome e dal contenuto.

In realtà il sistema prevede tipi diversi di file, ma in un altro senso; ad esempio il sistema può accedere alle periferiche, attraverso dei file speciali detti *device file* o *file di dispositivo*. Così si può suonare una canzone scrivendo su `/dev/dsp`, leggere l'output di una seriale direttamente da `/dev/ttyS0`, leggere direttamente dai settori fisici del disco rigido accedendo a `/dev/hda`, o fare animazioni scrivendo su `/dev/fb0` (questo è molto più difficile da fare a mano). Un elenco dei vari tipi oggetti visti come file dal kernel è riportato in tab. 1.1, ognuno di questi fornisce una funzionalità specifica, sempre descritta in tabella.

Altri tipi di file speciali sono le *fifo* ed i *socket*, che altro non sono che dei canali di comunicazione messi a disposizione dei processi perché questi possano parlare fra loro. Dato che i processi sono completamente separati deve essere il kernel a fornire le funzionalità che permettano la comunicazione. Questi file speciali sono due modalità per realizzare questa comunicazione; aprendo una *fifo* un processo può scrivervi sopra ed un altro processo leggerà dall'altro capo quanto il primo ha scritto, niente verrà salvato su disco, ma passerà tutto attraverso il kernel che consente questa comunicazione come attraverso un *tubo*. I *socket* fanno la stessa cosa ma consentono una comunicazione bidirezionale, in cui il secondo processo può scrivere indietro, ed il primo leggere, quando invece per le *fifo* il flusso dei dati è unidirezionale.

La possibilità di avere tutti questi tipi di file speciali è dovuta al fatto che, come accennavamo in sez. 1.1.3, in Linux l'accesso ai file viene eseguito attraverso una interfaccia unificata, il *Virtual*

¹³con la sola eccezione delle interfacce ai dispositivi di rete, che non rientrano bene nell'astrazione e sono gestiti in maniera diversa.

Tipo di file		Descrizione
<i>regular file</i>	file regolare	- un file che contiene dei dati (l'accezione normale di file).
<i>directory</i>	cartella o direttorio	d un file che contiene una lista di nomi associati a degli <i>inode</i> .
<i>symbolic link</i>	collegamento simbolico	l un file che contiene un riferimento ad un altro file o directory.
<i>char device</i>	dispositivo a caratteri	c un file che identifica una periferica ad accesso a caratteri.
<i>block device</i>	dispositivo a blocchi	b un file che identifica una periferica ad accesso a blocchi.
<i>fifo</i>	"coda"	p un file speciale che identifica una linea di comunicazione unidirezionale.
<i>socket</i>	"presa"	s un file speciale che identifica una linea di comunicazione bidirezionale.

Tabella 1.1: I vari tipi di file riconosciuti da Linux

File System, che definisce una serie di operazioni generiche che sono le stesse per tutti gli oggetti gestiti tramite essa; è questo che implementa la filosofia del *tutto è un file*.

Le principali operazioni sono riportate in tab. 1.2; ogni oggetto del sistema visto attraverso il *Virtual File System* definisce la sua versione di queste operazioni. Come si può notare sono definite sia operazioni generiche come la lettura e la scrittura, che più specialistiche come lo spostamento all'interno di un file.¹⁴ Quando si utilizzano le *system call* per accedere ad un file sarà compito del kernel chiamare l'operazione relativa ad esso associata (che sarà ovviamente diversa a seconda del tipo di file), o riportare un errore quando quest'ultima non sia definita (ad esempio sul file di dispositivo associato alla seriale non si potrà mai utilizzare l'operazione di spostamento *llseek*).

Funzione	Operazione
<i>open</i>	apre il file.
<i>read</i>	legge dal file.
<i>write</i>	scrive sul file.
<i>llseek</i>	si sposta all'interno del file.
<i>ioctl</i>	accede alle operazioni di controllo.
<i>readdir</i>	legge il contenuto di una directory.

Tabella 1.2: Principali operazioni sui file definite nel VFS.

Il *Virtual File System* è anche il meccanismo che permette al kernel di gestire tanti filesystem diversi; quando uno di questi viene montato è compito il kernel utilizzare per le varie *system call* le opportune operazioni in grado di accedere al contenuto di quel particolare filesystem; questa è la ragione principale della grande flessibilità di Linux nel supportare i filesystem più diversi, basta definire queste operazioni per un filesystem per poterne permettere l'accesso da parte delle varie *system call* secondo la stessa interfaccia.

Uno dei comandi fondamentali per la gestione dei file è **ls** (il cui nome deriva da *LiSt file*), il comando mostra l'elenco dei file nella directory corrente. Usando l'opzione **-l** è possibile ottenere una lista estesa, in cui compaiono varie proprietà del file; ad esempio:

¹⁴ed altre ancora più complesse che non sono state riportate.

```

piccardi@oppish:~/filetypes$ ls -l
total 1
brw-r--r--  1 root    root      1,  2 Jul  8 14:48 block
crw-r--r--  1 root    root      1,  2 Jul  8 14:48 char
drwxr-xr-x  2 piccardi piccardi 48 Jul  8 14:24 dir
prw-r--r--  1 piccardi piccardi  0 Jul  8 14:24 fifo
-rw-r--r--  1 piccardi piccardi  0 Jul  8 14:24 file
lrwxrwxrwx  1 piccardi piccardi  4 Jul  8 14:25 link -> file

```

ci mostra il contenuto di una directory dove si sono creati i vari tipi di file¹⁵ elencati in tab. 1.1, si noti come la prima lettera in ciascuna riga indichi il tipo di file, anche questo secondo la notazione riportata nella terza colonna della stessa tabella.

Il comando `ls` è dotato di innumerevoli opzioni, che gli consentono di visualizzare le varie caratteristiche dei file, delle quali il tipo è solo una. Altre caratteristiche sono i tempi di ultimo accesso, modifica e cambiamento (su cui torneremo fra poco), le informazioni relative a permessi di accesso e proprietari del file (che vedremo in dettaglio in sez. 1.4.2), la dimensione, il numero di hard link (che vedremo in sez. 1.2.2).

Il comando prende come parametro una lista di file o directory, senza opzioni viene mostrato solo il nome del file (se esiste) o il contenuto della directory specificata. Le opzioni sono moltissime, e le principali sono riportate in tab. 1.3; l'elenco completo è riportato nella pagina di manuale accessibile con il comando `man ls`.

Opzione	Significato
-l	scrive la lista in formato esteso.
-a	mostra i file <i>invisibili</i> .
-i	scrive il numero di <i>inode</i> (vedi sez. 1.2.2).
-R	esegue la lista ricorsivamente per tutte le sottodirectory.
-c	usa il tempo di ultimo cambiamento del file.
-u	usa il tempo di ultimo accesso al file.
-d	mostra solo il nome e non il contenuto quando riferito ad una directory, e non segue i link simbolici.

Tabella 1.3: Principali opzioni del comando `ls`.

Una convenzione vuole che i file il cui nome inizia per un punto (".") non vengano riportati nell'output di `ls`, a meno di non richiederlo esplicitamente con l'uso dell'opzione `-a`; per questo tali file sono detti *invisibili*. Si tenga presente comunque che questa *non* è una proprietà dei file e non ha nulla a che fare con le modalità con cui il kernel li tratta (che sono sempre le stesse), ma solo una convenzione usata e rispettata dai vari programmi in user space.

L'opzione `-l` permette di mostrare una lista in formato esteso in cui vengono riportate molte informazioni concernenti il file. Abbiamo visto in precedenza un esempio di questa lista, e come il primo carattere della prima colonna indichi tipo di file, mentre il resto della colonna indica i *permessi* del file, secondo una notazione su cui torneremo in sez. 1.4.2. Il secondo campo indica il numero di *hard link* al file (su questo torneremo in sez. 1.2.2), mentre il terzo ed il quarto campo indicano rispettivamente utente e gruppo proprietari del file (anche questo sarà trattato in sez. 1.4.2). Il quinto campo indica la dimensione, usualmente riportata in byte. Il sesto campo è il tempo di *ultima modifica* del file e l'ultimo campo il nome del file.

In un sistema unix-like i tempi dei file (mantenuti automaticamente dal kernel quando opera su essi) sono tre ed hanno un significato diverso rispetto a quanto si trova in altri sistemi operativi. Il tempo mostrato di default da `ls` è il *tempo di ultima modifica* (o *modification time*) che

¹⁵con l'eccezione dei *socket*, questi ultimi infatti non sono di norma utilizzati dai comandi di shell, ma vengono creati direttamente dai programmi che li usano (il caso più comune è X window), non esiste pertanto un comando che permetta di crearne uno in maniera esplicita.

corrisponde all'ultima volta che è stato modificato il contenuto di un file. Si badi bene che questo tempo riguarda solo il *contenuto* del file, se invece si operano delle modifiche sulle proprietà del file (ad esempio si cambiano i permessi) varia quello che viene chiamato *tempo di ultimo cambiamento* (il *change time*) che viene visualizzato con l'opzione `-c`. Infine tutte le volte che si accede al contenuto del file viene cambiato il *tempo di ultimo accesso* (o *access time*) che può essere visualizzato con l'opzione `-u`. Si noti infine come in un sistema unix-like *non* esista un tempo di creazione del file.

1.2.2 L'architettura di un filesystem e le proprietà dei file

Come già accennato Linux (ed ogni sistema unix-like) organizza i dati che tiene su disco attraverso l'uso di un filesystem. Una delle caratteristiche di Linux rispetto agli altri Unix è quella di poter supportare, grazie al VFS, una enorme quantità di filesystem diversi, ognuno dei quali ha una sua particolare struttura e funzionalità proprie. Per questo non entreremo nei dettagli di un filesystem specifico, ma daremo una descrizione a grandi linee che si adatta alle caratteristiche comuni di qualunque filesystem usato su sistema unix-like.

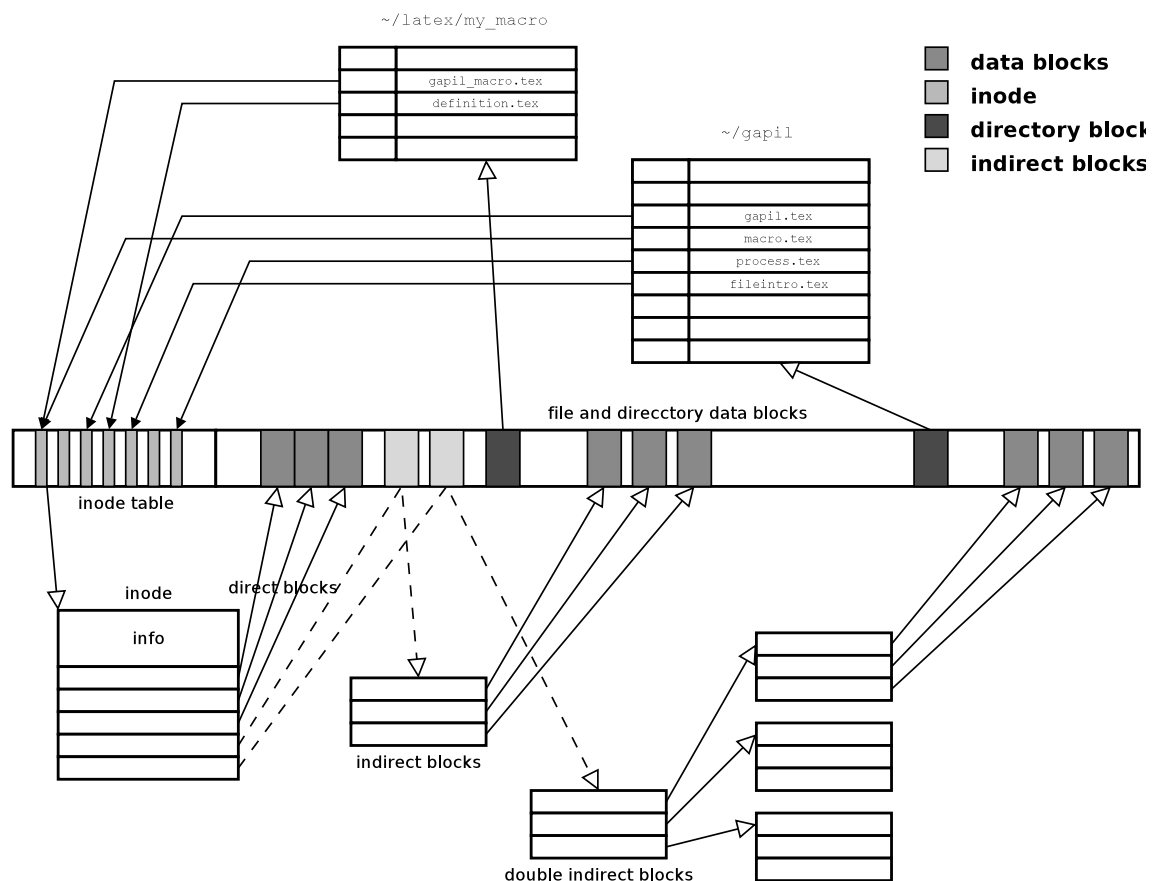


Figura 1.2: Strutturazione dei dati all'interno di un filesystem.

Se si va ad esaminare con maggiore dettaglio come è strutturata l'informazione all'interno di un singolo filesystem, possiamo esemplificare la situazione con uno schema come quello esposto in fig. 1.2, da cui si evidenziano alcune delle caratteristiche di base di un filesystem, sulle quali è bene porre attenzione visto che sono fondamentali per capire il funzionamento dei comandi che manipolano i file e le directory.

La struttura che identifica univocamente un singolo file all'interno di un filesystem è il cosiddetto *inode*: ciascun file è associato ad un *inode* in cui sono mantenute tutte le informazioni

che lo riguardano, come il tipo (fra quelli di tab. 1.1), i permessi di accesso, utente e gruppo proprietario, le dimensioni, i tempi, ed anche tutti i riferimenti ai settori del disco (i *blocchi fisici*) che contengono i dati; le informazioni che il comando `ls` fornisce provengono dall'*inode*.

L'unica informazione relativa al file non contenuta nell'*inode* è il suo nome; infatti il nome di un file non è una proprietà del file, ma semplicemente una etichetta associata ad un *inode*. Le directory infatti non *contengono* i file, ma sono dei file speciali (di tipo *directory*, così che il kernel può trattarle in maniera diversa) il cui contenuto è semplicemente una lista di nomi a ciascuno dei quali viene associato un numero di *inode* che identifica il file cui il nome fa riferimento.

Come mostrato in fig. 1.2 si possono avere più voci in directory diverse che puntano allo stesso *inode*. Questo introduce il concetto di *hard link*: due file che puntano allo stesso *inode* sono fisicamente lo stesso file, nessuna proprietà specifica, come permessi, tempi di accesso o contenuto permette di distinguerli, in quanto l'accesso avviene per entrambi attraverso lo stesso *inode*.

Siccome uno stesso *inode* può essere referenziato in più directory, un file può avere più nomi, anche completamente scorrelati fra loro. Per questo ogni *inode* mantiene un contatore che indica il numero di riferimenti (detto *link count*) che gli sono stati fatti; questo viene mostrato, nell'esempio del precedente risultato di `ls`, dal valore numerico riportato nel secondo campo, e ci permette di dire se un file ha degli *hard link* (anche se non indica dove sono).

Il comando generico che permette di creare dei link è `ln` che prende come argomenti il file originale ed il nome del link. La differenza rispetto a Windows o MacOS è che in un sistema unix-like i link sono di due tipi, oltre agli *hard link* appena illustrati infatti esistono anche i cosiddetti *collegamenti simbolici*, o *symbolic link* (quelli più simili ai collegamenti di Windows o agli alias del MacOS), come il file `link` mostrato in precedenza.

Per creare un *hard link* basta usare direttamente il comando `ln` (da *LiNk file*), che di default crea questo tipo di link. Così potremo creare il file `hardlink` come *hard link* al file `file` visto in precedenza con il comando:

```
piccardi@oppish:~/filetypes$ ln file hardlink
```

e adesso potremo verificare che:

```
piccardi@oppish:~/filetypes$ ls -l
total 1
brw-r--r-- 1 root root 1, 2 Jul 8 14:48 block
crw-r--r-- 1 root root 1, 2 Jul 8 14:48 char
drwxr-xr-x 2 piccardi piccardi 48 Jul 8 14:24 dir
prw-r--r-- 1 piccardi piccardi 0 Jul 8 14:24 fifo
-rw-r--r-- 2 piccardi piccardi 0 Jul 8 14:24 file
-rw-r--r-- 2 piccardi piccardi 0 Jul 8 14:24 hardlink
lrwxrwxrwx 1 piccardi piccardi 4 Jul 8 14:25 link -> file
```

e si noti come adesso il secondo campo mostri per `file` e `hardlink` un valore pari a due. Usando l'opzione `-li` di `ls` possiamo anche stampare per ciascun file il numero di *inode* ottenendo:

```
piccardi@oppish:~/filetypes$ ls -li
total 1
2118 brw-r--r-- 1 root root 1, 2 Jul 8 14:48 block
2120 crw-r--r-- 1 root root 1, 2 Jul 8 14:48 char
15 drwxr-xr-x 2 piccardi piccardi 48 Jul 8 14:24 dir
2115 prw-r--r-- 1 piccardi piccardi 0 Jul 8 14:24 fifo
2117 -rw-r--r-- 2 piccardi piccardi 0 Jul 8 14:24 file
2117 -rw-r--r-- 2 piccardi piccardi 0 Jul 8 14:24 hardlink
2116 lrwxrwxrwx 1 piccardi piccardi 4 Jul 8 14:25 link -> file
```

e come si può notare **file** e **hardlink** hanno lo stesso numero di *inode*, pari a 2117.

Il problema con gli *hard link* è che le directory contengono semplicemente il numero di *inode*, per cui si può fare riferimento solo ad un *inode* nello stesso filesystem della directory, dato che su un altro filesystem lo stesso numero identificherà un *inode* diverso. Questo limita l'uso degli *hard link* solo a file residenti sul filesystem corrente, ed il comando **ln** darà un errore se si cerca di creare un *hard link* ad un file posto in un altro filesystem.

Per superare questa limitazione sono stati introdotti i link simbolici, che vengono creati usando l'opzione **-s** del comando **ln**; ad esempio si è creato il link simbolico **link** dell'esempio precedente con il comando:

```
piccardi@oppish:~/filetypes$ ln -s file link
```

In questo caso viene creato un nuovo file, di tipo *symbolic link*, che avrà un suo diverso *inode*, come mostrato nell'esempio precedente, ed il cui contenuto è il percorso (torneremo sulla notazione che esprime i *pathname* dei file fra poco) per il file a cui esso fa riferimento, che a questo punto può essere in qualsiasi altro filesystem. È compito del kernel far sì che quando si usa un link simbolico si vada poi ad usare il file che questo ci indica.

Oltre a **-s** il comando **ln** prende una serie di altre opzioni le principali delle quali sono riportate in tab. 1.4. La lista completa è riportata nella pagina di manuale accessibile attraverso il comando **man ln**.

Opzione	Significato
-s	crea un link simbolico.
-f	forza la sovrascrittura del nuovo file se esso esiste già.
-i	richiede conferma in caso di sovrascrittura.
-d	crea un <i>hard link</i> ad una directory (in Linux questa non è usabile).
-b	esegue un backup della destinazione se questa esiste già.

Tabella 1.4: Principali opzioni del comando **ln**.

Una seconda caratteristica dei link simbolici è la possibilità di creare dei link anche per delle directory. Questa capacità infatti, sebbene teoricamente possibile anche per gli *hard link*, in Linux non è supportata per la sua pericolosità, è possibile infatti creare dei *link loop* se si commette l'errore di creare un link alla directory che contiene il link stesso; con un link simbolico questo errore può essere corretto in quanto la cancellazione del link simbolico rimuove quest'ultimo, e non il file referenziato, ma con un *hard link* non è più possibile fare questa distinzione e la rimozione diventa impossibile.

La possibilità di creare dei link alle directory tuttavia è estremamente utile, infatti qualora si voglia accedere ai file in una directory attraverso un percorso diverso, ad esempio a causa di un programma che cerca dei file di configurazione in una locazione diversa da quella usuale, piuttosto che dover spostare tutti i file basterà creare un link simbolico e si sarà risolto il problema.

Oltre agli *hard link* la struttura di un filesystem unix-like ha ulteriori conseguenze non immediate da capire per chi proviene da sistemi operativi diversi. La presenza degli *hard link* e l'uso degli *inode* nelle directory infatti comporta anche una modalità diversa nella cancellazione dei file e nello spostamento degli stessi.

Il comando per la cancellazione di un file è **rm** (da *ReMove file*), ma la funzione usata dal sistema per effettuare questo compito si chiama in realtà **unlink** ed essa, come ci dice il nome, non cancella affatto i dati del file, ma si limita ad eliminare la relativa voce da una directory e decrementare il numero di riferimenti presenti nell'*inode*.

Solo quando il numero di riferimenti ad un *inode* si annulla, i dati del file vengono effettivamente rimossi dal disco dal kernel. In realtà oltre ai riferimenti mostrati da **ls** il kernel mantiene anche un'altra lista di riferimenti per ciascun processo che sta accedendo al file, per cui anche

se si cancella un file da una directory, ma resta attivo un processo che lo utilizza, lo spazio disco non viene rilasciato, e fintanto che il processo non avrà finito i dati resteranno, anche se solo per lui, disponibili.

Il comando `rm` prende come argomenti una lista di file da cancellare; se si usa l'opzione `-i` il comando chiede di confermare la cancellazione, mentre con l'opzione `-f` si annulla ogni precedente `-i` ed inoltre non vengono stampati errori per file non esistenti. Infine l'opzione `-R` (o `-r`) permette la cancellazione ricorsiva di una directory e di tutto il suo contenuto, ed è pertanto da usare con estrema attenzione, specie se abbinata con `-f`.¹⁶ La lista completa delle opzioni è riportata nella pagina di manuale, accessibile con il comando `man rm`.

Come accennato la struttura di un filesystem unix-like comporta anche una diversa concezione dell'operazione di spostamento dei file, che nel caso è identica a quella di cambiamento del nome. Il comando per compiere questa operazione infatti è unico e si chiama `mv`, da *MoVe file*.

Infatti fintanto che si “sposta” un file da una directory ad un'altra senza cambiare filesystem, non c'è nessuna necessità di spostare il contenuto del file e basta semplicemente che sia creata una nuova voce per l'*inode* in questione rimuovendo al contempo la vecchia: esattamente la stessa cosa che avviene quando gli si cambia nome (nel qual caso l'operazione viene effettuata all'interno della stessa directory). Qualora invece si debba effettuare lo spostamento ad un filesystem diverso diventa necessario prima copiare il contenuto e poi cancellare l'originale.

Il comando `mv` ha due forme, e può prendere come argomenti o due nomi di file o una lista di file seguita da una directory. Nel primo caso rinomina il primo file nel secondo, cancellando quest'ultimo qualora esista già, nel secondo caso sposta tutti i file della lista nella directory, sempre cancellando eventuali file con lo stesso nome presenti in essa. Le principali opzioni sono riportate in tab. 1.5, l'elenco completo è riportato nella pagina di manuale, accessibile con il comando `man mv`.

Opzione	Significato
<code>-f</code>	forza la sovrascrittura del nuovo file se esso esiste già.
<code>-i</code>	richiede conferma in caso di sovrascrittura.
<code>-u</code>	esegue lo spostamento solo se la destinazione è più vecchia della sorgente (o non esiste).
<code>-b</code>	esegue un backup della destinazione se questa esiste già.

Tabella 1.5: Principali opzioni del comando `mv`.

Dato che il comando si limita a cambiare una voce associata ad un numero di *inode* all'interno di una directory, fintanto che lo spostamento avviene all'interno dello stesso filesystem, i tempi dei file non vengono modificati dall'uso di `mv`. Quando però lo spostamento avviene fra filesystem diversi viene copiato il contenuto e cancellato il file originario, pertanto in teoria dovrebbero risultare modificati anche i tempi di ultimo accesso e modifica. In realtà il comando provvede a ripristinare questi tempi (come le altre caratteristiche del file) al valore del file originario, ma non può fare nulla per ripristinare il tempo di ultimo cambiamento.¹⁷ Pertanto in quel caso si potrà notare, usando `ls -lc`, che questo è cambiato e corrisponde al momento dello spostamento.

Qualora invece si voglia duplicare un file il comando da usare è `cp` (da *CoPy file*). Come per `mv` il comando può prendere come argomenti o due nomi di file o una lista di file seguita da una

¹⁶uno dei sistemi più efficaci per distruggere una installazione è un `rm -fR` eseguito come amministratore nel posto sbagliato, e anche se come utente non si può danneggiare il sistema, è comunque molto semplice spazzare via tutti i propri dati.

¹⁷il kernel fornisce delle *system call* che permettono di cambiare i tempi di ultimo accesso e modifica di un file, che il comando `mv` può usare per ripristinare i tempi precedenti, ma non ne esistono per cambiare il tempo di ultimo cambiamento, che corrisponderà pertanto al momento in cui il nuovo file è stato creato; questa è una misura di sicurezza che permette sempre di verificare se un file è stato modificato, anche se si cerca di nascondere le modifiche.

directory; nel primo caso effettua una copia del primo file sul secondo, nel secondo copia tutti i file della lista nella directory specificata.

Dato che il comando funziona copiando il contenuto di un file su un nuovo file creato per l'occasione, i tempi di ultima modifica, accesso e cambiamento di quest'ultimo corrisponderanno al momento in cui si è eseguita l'operazione. Inoltre il file sarà creato con i permessi standard dell'utente che ha lanciato il comando, che risulterà anche il suo proprietario. Se si vogliono preservare invece le caratteristiche del file originale occorrerà usare l'opzione `-p`.¹⁸

Opzione	Significato
<code>-f</code>	forza la sovrascrittura della destinazione se essa esiste già.
<code>-i</code>	richiede conferma in caso di sovrascrittura.
<code>-p</code>	preserva tempi, permessi e proprietari del file.
<code>-l</code>	crea degli hard link al posto delle copie.
<code>-s</code>	crea dei link simbolici al posto delle copie.
<code>-d</code>	copia il link simbolico invece del file da esso indicato.
<code>-r</code>	copia ricorsivamente tutto il contenuto di una directory.
<code>-R</code>	identico a <code>-r</code> .
<code>-a</code>	combina le opzioni <code>-dpr</code> .
<code>-L</code>	segue sempre i link simbolici.
<code>-b</code>	esegue un backup della destinazione se questa esiste già.
<code>-u</code>	esegue la copia solo se la destinazione è più vecchia della sorgente (o non esiste).

Tabella 1.6: Principali opzioni del comando `cp`.

Si tenga presente poi che nel caso di link simbolici il comando copia il file indicato tramite il link, se invece si vuole copiare il link stesso occorrerà usare l'opzione `-d`. Il comando permette inoltre di creare degli hard link invece che delle copie usando l'opzione `-l` e dei link simbolici usando l'opzione `-s`. Una lista delle principali opzioni è riportata in tab. 1.6, l'elenco completo è riportato nella pagina di manuale, accessibile attraverso il comando `man cp`.

Fino ad ora nel descrivere gli argomenti da dare ai vari comandi che abbiamo trattato, si è parlato dei nomi dei file o delle directory senza entrare troppo nei dettagli su quale fosse il formato in cui questi vengono espressi. Negli esempi infatti si sono specificati dei semplici nomi, ma questo dava per scontati alcuni concetti che in realtà non lo sono affatto.

La convenzione usata in tutti i sistemi unix-like è che i nomi dei file sono indicati con un *pathname* o *percorso*, che descrive il cammino che occorre fare nell'albero dei file per raggiungere il file passando attraverso le varie directory; i nomi delle directory sono separati da delle `/`. Il percorso può essere indicato (vedi tab. 1.7) in maniera assoluta, partendo dalla directory radice,¹⁹ ed indicando tutte le directory che si devono attraversare, o in maniera relativa, partendo dalla cosiddetta *directory di lavoro corrente*. Nel primo caso il pathname inizierà con `/`, mentre nel secondo no.

Esempio	Formato
<code>/home/piccardi/gapil/gapil.tex</code>	assoluto
<code>gapil/gapil.tex</code>	relativo

Tabella 1.7: Formato dei pathname assoluti e relativi.

Finora, specificando solo dei nomi semplici, abbiamo sempre fatto l'assunto di operare appunto nella *directory di lavoro corrente*. Questa directory è una proprietà specifica di ogni processo, che viene ereditata dal padre alla sua creazione (questo argomento è trattato in sez. 1.3.1) ed indica appunto la directory a partire dalla quale vengono risolti, per quel processo, i pathname

¹⁸per poterlo fare però occorre eseguire il comando come amministratore.

¹⁹torneremo sulla struttura dell'albero e sul concetto di radice in sez. 1.2.1

relativi. Questo vuol dire che si possono indicare i file presenti in quella directory direttamente, senza doverne specificare il pathname completo a partire dalla radice.

Quando si entra nel sistema la directory di lavoro corrisponde alla *home*²⁰ dell'utente; essa può essere cambiata con il comando `cd` (da *Change Directory*) seguito dal pathname della directory in cui ci si vuole spostare, mentre la si può stampare a video con il comando `pwd` (da *Print Work Directory*).

Si tenga presente poi che ciascuna directory contiene sempre²¹ almeno due voci: la directory `."` che fa riferimento a se stessa, e che usata all'inizio di un pathname indica con la directory di lavoro corrente, e la directory `.."`, che fa riferimento alla directory in cui l'attuale è contenuta, e che all'inizio di un pathname indica la directory sovrastante quella corrente. In questo modo anche con dei pathname relativi si possono fare riferimenti a directory poste in sezioni diverse dell'albero dei file, risalendo lo stesso con l'uso della directory `.."`. Si noti come entrambe queste due voci (dato che iniziano per `."`) siano *invisibili*.

Inoltre, come vedremo in sez. 2.1.3, la shell quando deve passare dei pathname ai comandi che operano su file e directory (come `cd`, `cp`, ecc.) riconosce alcuni caratteri speciali, ad esempio il carattere `~` viene usato per indicare la home dell'utente corrente, mentre con `~username` si indica la home dell'utente `username`; `cd` poi riconosce il carattere `-` che indica il ritorno alla precedente directory di lavoro, torneremo su questo con qualche dettaglio in più in sez. 2.1.4.

Come accennato nella sezione precedente le directory sono dei file, anche se sono dei file speciali il cui compito è solo contenere elenchi di nomi di altri file. Per questo per la creazione di una directory è previsto un comando apposito, `mkdir`, che crea la (o le) directory passate come argomento. Queste saranno specificate con quello che sarà il loro pathname, in forma assoluta o relativa. Perché il comando abbia successo ovviamente la parte di percorso che deve contenere la directory che si vuole creare deve esistere, se non è così si può forzare la creazione di tutte le directory indicate nel percorso con l'opzione `-p`.

Come per la creazione è necessario un comando apposito, `rmdir`, anche per la rimozione di una directory (`rm`, a meno di non usare l'opzione `-R` che però cancella tutto anche il contenuto, rimuove solo i file). Di nuovo il comando prende il nome di una o più directory vuote da cancellare. Se le directory non sono vuote il comando fallisce (vuote significa che non ci deve essere niente a parte le due directory `."` e `.."`). Anche in questo caso si può usare l'opzione `-p` che cancella tutto un percorso di directory (che comunque devono essere tutte vuote).

1.2.3 L'organizzazione delle directory ed il *Filesystem Hierarchy Standard*

Una delle caratteristiche peculiari di un sistema unix-like è che l'albero delle directory è unico; non esistono cioè i vari dischi (o volumi) che si possono trovare in altri sistemi, come su Windows, sul MacOS o sul VMS. All'avvio il kernel *monta*²² quella che si chiama la *directory radice* (o *root directory*) dell'albero, che viene indicata con `/`, tutti i restanti dischi, il CDROM, il floppy ed qualunque altro dispositivo contenente file, verranno poi *montati* (vedi sez. 1.2.4) successivamente in opportune sotto-directory della radice.

Come per il processo `init`, che non è figlio di nessun altro processo e viene lanciato direttamente dal kernel, anche la directory radice non è contenuta in nessuna altra directory²³ e, come accennato in sez. 1.1.1, viene montata direttamente dal kernel in fase di avvio. Per questo

²⁰ogni utente ha una sua directory personale nella quale può tenere i suoi file, che viene chiamata così, tratteremo la questione più avanti in sez. 1.2.3.

²¹è compito del comando di creazione di una nuova directory far sì che esse siano sempre presenti.

²²l'operazione di rendere visibili ai processi i file contenuti all'interno di un filesystem facendoli comparire all'interno nell'albero delle directory viene detta appunto *montare* il filesystem.

²³nel suo caso infatti la directory `.."` è identica a `.` ed indica sempre la radice stessa dato che al di sopra non c'è niente.

motivo la directory radice viene ad assumere un ruolo particolare, ed il filesystem che la supporta deve contenere tutti i programmi di sistema necessari all'avvio (compreso `init`).

Un esempio di questa struttura ad albero, che al contempo ci mostra anche i contenuti delle directory principali, può essere ottenuto con il comando `tree`. Se chiamato senza parametri questo comando mostra l'albero completo a partire dalla directory corrente, scendendo in tutte le directory sottostanti; usando l'opzione `-L` si può specificare il numero massimo di livelli a cui scendere, per cui andando su / avremo qualcosa del tipo:

```
piccardi@oppish:~$ cd /
piccardi@oppish:/$ tree -L 2
.
|-- bin
|   |-- arch
...
|-- boot
|   |-- System.map-2.4.20
...
|-- cdrom
|-- dev
|   |-- MAKEDEV -> /sbin/MAKEDEV
...
|-- etc
|   |-- GNUstep
...
|-- floppy
|-- home
|   '-- piccardi
|-- initrd
|-- lib
|   |-- cpp -> /usr/bin/cpp-2.95
...
|-- lost+found
|-- mnt
|   '-- usb
|-- opt
|-- proc
|   |-- 1
...
|-- root
|-- sbin
|   |-- MAKEDEV
...
|-- tmp
|   '-- ssh-XXBiWAR1
|-- usr
|   |-- X11R6
|   |-- bin
|   |-- doc
|   |-- games
|   |-- include
|   |-- info
|   |-- lib
|   |-- local
|   |-- sbin
|   |-- share
|   '-- src
|-- var
|   |-- backups
|   |-- cache
|   |-- lib
|   |-- local
|   |-- lock
|   |-- log
|   |-- mail
|   |-- opt
|   |-- run
|   |-- spool
|   '-- tmp
'-- vmlinuz -> boot/vmlinuz-2.2.20-idepci
```


e questo ci mostra il contenuto sommario primi due livelli dell'albero, con un esempio dei file e delle sottodirectory presenti in una distribuzione Debian.

L'organizzazione dell'albero delle directory è standardizzata in maniera molto accurata da un documento che si chiama *Filesystem Hierarchy Standard* (abbreviato in FHS), a cui tutte le distribuzioni si stanno adeguando.²⁴ Lo standard descrive in dettaglio la struttura dell'albero delle directory e il relativo contenuto, prevedendo una divisione molto rigorosa che permette una notevole uniformità anche fra distribuzioni diverse; si organizzano così in maniera meticolosa ed ordinata dati, programmi, file di configurazione, documentazione, file degli utenti, ecc.

Directory	Contenuto
/bin	comandi essenziali
/boot	file statici necessari al <i>bootloader</i>
/dev	file di dispositivo
/etc	file di configurazione della macchina
/lib	librerie essenziali e moduli del kernel
/mnt	<i>mount point</i> per filesystem temporanei
/opt	pacchetti software aggiuntivi
/sbin	comandi di sistema essenziali
/tmp	file temporanei
/usr	gerarchia secondaria
/var	dati variabili

Tabella 1.8: Sottodirectory di / obbligatorie per qualunque sistema.

In particolare le directory vengono suddivise sulla base di alcuni criteri fondamentali; il primo è quello della possibilità di contenere file il cui contenuto può essere modificato (nel qual caso il filesystem che le contiene deve essere montato in lettura/scrittura) o meno (nel qual caso il filesystem può essere montato in sola lettura); il secondo è quello della possibilità di contenere file (come i programmi di sistema) che possono essere condivisi (ad esempio utilizzando un filesystem di rete) fra più stazioni di lavoro o file che invece sono locali e specifici alla macchina in questione, il terzo criterio è quello di contenere o meno comandi o file (configurazioni e file di dispositivo) che sono necessari all'avvio del sistema, e che pertanto devono essere situati sul filesystem usato per la directory radice, dato che essi non sarebbero disponibili se posti in filesystem diversi che possono essere montati solo dopo che il sistema è partito.

Lo standard prevede che debbano essere necessariamente presenti le sottodirectory di / specificate in tab. 1.8, mentre quelle di tab. 1.9 sono obbligatorie soltanto qualora si siano installati i sottosistemi a cui essi fanno riferimento (utenti, /**proc** filesystem, diversi formati binari).²⁵

Directory	Contenuto
/lib<qual>	librerie in formati alternativi
/home	home directory degli utenti
/root	home directory di <i>root</i>
/proc	filesystem virtuale con le informazioni sul sistema

Tabella 1.9: Sottodirectory di / obbligatorie solo in presenza dei relativi sottosistemi.

Un elenco delle specifiche delle caratteristiche e del contenuto di ciascuna delle sottodirectory di / è riportato di seguito; per alcune di esse, come /**usr** e /**var**, sono previste delle ulteriori sottogerarchie che definiscono ulteriori dettagli dell'organizzazione dei file.

²⁴al momento della stesura di questo testo la versione corrente è la 2.3, rilasciato come parte delle specifiche LSB (*Linux Standard Base*) 2.0 che sono in corso di rilascio; nella attuale versione 1.3 delle LSB è utilizzata la versione 2.2.

²⁵le eventuali /lib<qual> contengono le versioni delle librerie di sistema in formati binari diversi; le /lib alternative sono state usate al tempo della transizione dei programmi dal formato *a.out* ad *ELF*, oggi sono usate principalmente per quei sistemi (come gli AMD-64) che supportano diversi formati binari (32 o 64 bit).

- /bin** Contiene i comandi essenziali del sistema (usati sia dall'amministratore che dagli utenti, come `ls`), che devono essere disponibili anche quando non ci sono altri filesystem montati, ad esempio all'avvio o quando si è in *single user mode* (vedi sez. 5.3.4). Non deve avere sottodirectory e non può stare su un filesystem diverso da quello della radice.
- /boot** Contiene tutti i file necessari al procedimento di boot (immagini del kernel, ramdisk, ecc.) eccetto i file di configurazione ed i programmi per l'impostazione del procedimento stesso (che vanno in `/sbin`). Può stare su qualunque filesystem purché visibile dal *bootloader* (vedi sez. 5.3.2).
- /dev** Contiene i file di dispositivo, che permettono l'accesso alle periferiche. Deve stare sullo stesso filesystem della radice, a meno che non si sia installato nel kernel il supporto per il `devfs`, che permette di trasferire il contenuto di questa directory su un apposito filesystem virtuale.
- /etc** Contiene i file di configurazione del sistema e gli script²⁶ di avvio. Non deve contenere programmi binari e non può stare su un filesystem diverso da quello della radice. I file possono essere raggruppati a loro volta in directory; lo standard prevede solo che, qualora siano installati, siano presenti le directory `/etc/opt` (per i pacchetti opzionali), `/etc/X11` (per la configurazione di *X Window*, vedi sez. 3.4) e `/etc/sgml` (per la configurazione di SGML e XML).
- /home** Contiene le *home directory* degli utenti, la sola parte del filesystem (eccetto `/tmp`) su cui gli utenti hanno diritto di scrittura. Può essere montata su qualunque filesystem.
- /lib** Contiene le librerie condivise essenziali, usate dai programmi di `/bin` e `/sbin`, e deve essere sullo stesso filesystem della radice. Qualora sia stato installato un kernel modulare (vedi sez. 5.1.4) i moduli devono essere installati in `/lib/modules`.
- /mnt** Contiene i *mount point* (vedi sez. 1.2.4) per i filesystem temporanei ad uso dell'amministratore di sistema (i filesystem di periferiche permanenti come i floppy o il CDROM possono essere tenuti sia in questa directory che direttamente sotto `/`). Normalmente è vuota e deve essere creata direttamente sotto la radice.
- /opt** Contiene eventuali pacchetti software aggiuntivi. Può essere su qualunque filesystem. Un pacchetto deve installarsi nella directory `/opt/package` dove *package* è il nome del pacchetto. All'amministratore è riservato l'uso di alcune directory opzionali: `/opt/bin`, `/opt/doc`, `/opt/include`, `/opt/info`, `/opt/lib` e `/opt/man`. File variabili attinenti ai suddetti pacchetti devono essere installati in `/var/opt` ed i file di configurazione in `/etc/opt`, nessun file attinente ai pacchetti deve essere installato al di fuori di queste directory.
- /proc** È il *mount point* standard del filesystem virtuale `proc`. Questo è un filesystem speciale che permette di accedere a tutta una serie di variabili interne al kernel (relative a parametri e impostazioni di tutti tipi) con l'interfaccia dei file. Così se si vogliono informazioni sugli interrupt ed i canali di DMA (vedi sez. 5.4.1) utilizzati dal sistema si potranno leggere i file `/proc/interrupts` e `/proc/dma`, mentre si potranno impostare varie caratteristiche del sistema scrivendo nei file sotto `/proc/sys`.
- /root** È la *home directory* dell'amministratore. Di norma la si mantiene nello stesso filesystem della radice.

²⁶gli *script*, su cui torneremo in sez. 2.1.6, sono un po' gli equivalenti (come potrebbe esserlo una Ferrari in confronto ad una 500) in ambito Unix dei file `.bat` del DOS, una lista di comandi messi in un file (in realtà i tratta di un vero e proprio linguaggio di programmazione) e fatti eseguire automaticamente.

- /sbin** Contiene i programmi essenziali per l'amministrazione del sistema (come `init`). Deve stare sullo stesso filesystem della radice. Vanno messi in questa directory solo i programmi essenziali per l'avvio del sistema, il recupero e la manutenzione dei filesystem.
- /tmp** La directory viene usata per mantenere file temporanei. Viene cancellata ad ogni riavvio, ed i programmi non devono assumere che i file siano mantenuti fra due esecuzioni successive.
- /usr** È la directory principale che contiene tutti i file ed i dati non variabili che possono essere condivisi fra più stazioni. Di solito viene montata su un filesystem separato rispetto a `/` e può essere montata in sola lettura. Prevede una ulteriore gerarchia di directory in cui i vari file vengono organizzati; lo standard richiede obbligatoriamente le seguenti:

- bin** Contiene i programmi usati dall'utente installati direttamente dal sistema (o dalla distribuzione originale). Non può essere ulteriormente suddivisa.
- include** Contiene tutti gli header file usati dal compilatore e dai programmi C e C++.
- lib** Contiene le librerie relative ai programmi di **bin** e **sbin**.
- local** Contiene una replica della gerarchia di **/usr** dedicata ai file installati localmente dall'amministratore. In genere qui vengono installati i programmi compilati dai sorgenti e tutto quello che non fa parte della distribuzione ufficiale.
- sbin** Contiene le utilità di sistema non essenziali per l'avvio, ad uso dell'amministratore.
- share** Contiene una gerarchia in cui sono organizzati tutti i dati che non dipendono dalla architettura hardware: **man** per le pagine di manuale, **dict** per i dizionari, **doc** per la documentazione, **games** per i dati statici dei giochi, **info** per i file del relativo sistema di help, **terminfo** per il database con le informazioni sui terminali, **misc** per tutto quello che non viene classificato nelle altre.

mentre sono obbligatorie solo se i relativi pacchetti sono installati, le seguenti directory:

- X11R6** Contiene la gerarchia dei file relativi ad *X Window* (vedi sez. 3.4).
- games** Contiene i binari dei giochi.
- src** Contiene i sorgenti dei pacchetti.

- /var** Contiene i file variabili: le directory di spool, i file di log, i dati transienti e temporanei, in modo che **/usr** possa essere montata in sola lettura. È preferibile montarla in un filesystem separato; alcune directory non possono essere condivise. Anche in questo caso i file sono organizzati in una ulteriore gerarchia standardizzata che prevede le seguenti sottodirectory:

- cache** Dati di appoggio per le applicazioni.
- lib** Informazioni variabili sullo stato del sistema.
- local** Dati variabili relativi ai pacchetti di **/usr/local**.
- lock** File di lock.
- opt** File variabili per i pacchetti di **/opt**.
- run** Dati relativi ai processi in esecuzione.
- spool** Directory per i dati di spool di varie applicazioni (stampanti, posta elettronica, news, ecc.).
- tmp** File temporanei non cancellati al riavvio del sistema.

In fig. 1.3 è riportata una rappresentazione grafica della struttura generale delle directory prevista dal FHS, (si è mostrata solo una parte delle directory previste). I dettagli completi sulla struttura (così come le specifiche relative ad i contenuti delle varie directory, possono essere reperiti sul documento ufficiale di definizione del FHS, disponibile all'indirizzo: <http://www.pathname.com/fhs/>.

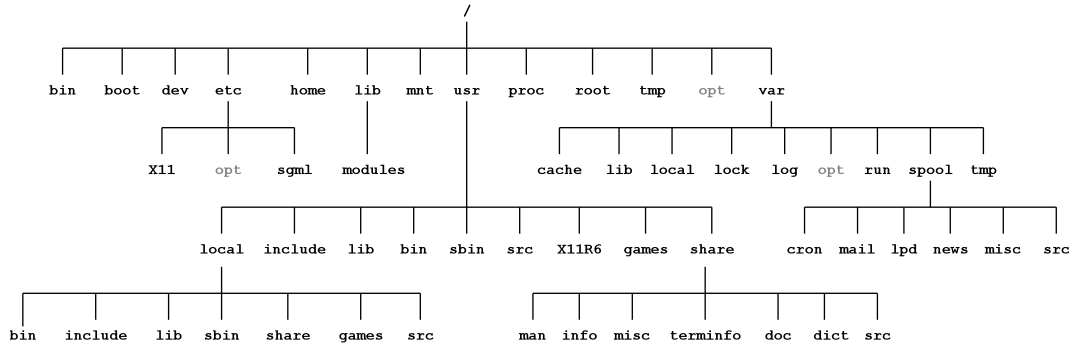


Figura 1.3: Struttura tipica delle directory, secondo il Filesystem Hierarchy Standard.

L'importanza del *Filesystem Hierarchy Standard* diventa evidente quando si vanno ad esaminare le strategie partizionamento dei dischi. In tal caso infatti occorrerà stabilire quali directory dovranno andare sul filesystem usato come radice, e quali altre directory porre su altre partizioni.

È evidente infatti che alcune directory (come `/usr` ed `/opt`) possono essere mantenute su partizioni e filesystem diversi rispetto alla directory radice. È pertanto utile separare queste due directory che, contenendo file comuni di norma identici per le diverse installazioni, possono essere montate in sola lettura e non inserite nei backup (in quanto è possibile sempre ripristinarle dall'installazione), o addirittura montate via rete e condivise fra più macchine.

La situazione è invece del tutto diversa per directory come `/home` e `/var`. Anche in questo caso è opportuno separarle dalle altre directory, ma in questo caso è necessario l'accesso in scrittura e le informazioni variabili non saranno necessariamente condivisibili (ad esempio non lo sono `/var/run` e `/var/lock` che contengono informazioni sui processi locali). Inoltre essendo qui contenuti la gran parte dei dati del sistema (le altre directory sono solo `/root` per i file personali dell'amministratore e `/etc` per le configurazioni) queste dovranno essere sottoposte a regolare backup.

Si tenga inoltre presente che alcune di queste directory (ad esempio `/proc`) devono essere lasciate vuote sul disco; esse infatti servono solo come riferimento per montare i relativi filesystem virtuali. Non ha quindi alcun senso effettuare backup del contenuto di queste directory in quanto esse presentano solo una interfaccia di accesso (che permette però l'uso dei normali comandi per i file) a variabili interne del kernel create dinamicamente.

1.2.4 La gestione dell'uso di dischi e volumi

Una delle caratteristiche di GNU/Linux che disorientano maggiormente chi proviene da altri sistemi operativi è la presenza di un unico albero delle directory, come illustrato in sez. 1.2.3. Non esistendo il concetto di volume o disco come entità separata, questo significa (come accennato in sez. 1.2.3) che i nuovi dischi devono essere inseriti in maniera opportuna all'interno dell'albero, in modo che il loro contenuto possa essere visto all'interno delle opportune directory, con quell'operazione che si chiama *montaggio* del disco.

Allora, a parte la directory radice che viene montata dal kernel all'avvio,²⁷ tutti gli altri *volumi*, che siano filesystem contenuti in partizioni diverse dello stesso disco o in altri dischi,

²⁷e come vedremo in sez. 5.3 la definizione di quale sia il dispositivo su cui si trova il filesystem che contiene la radice è una delle impostazioni fondamentali relative all'avvio del sistema.

CDROM, floppy o qualunque altra forma di supporto che contiene un filesystem, devono essere montati successivamente.

Il comando che permette di montare un disco è `mount`, che di norma si limita ad invocare la omonima *system call* del kernel;²⁸ nella modalità standard esso viene sempre invocato nella forma:

```
mount -t filesystem_type /dev/device /path/to/dir
```

dove l'opzione `-t` serve ad indicare il tipo di filesystem contenuto nel device `/dev/device` (indicato tramite il suo file di dispositivo in `/dev`) e `/path/to/dir` indica la directory, detta *mount point*, in cui esso verrà *montato*, cioè all'interno della quale verrà reso accessibile il contenuto del filesystem.

Il comando richiede la conoscenza del tipo di filesystem presente nel dispositivo che si vuole montare; l'elenco dei principali filesystem supportati è riportato in tab. 1.10; è possibile comunque usare anche un meccanismo di ricerca automatico, che viene attivato usando `auto` come tipo di filesystem. In questo caso viene effettuato automaticamente un controllo se nel dispositivo è presente uno dei filesystem riportati nella prima parte (fino alla riga orizzontale) di tab. 1.10.

Se il riconoscimento non riesce viene effettuato un ulteriore controllo: prima viene letto `/etc/filesystem` e, se questo non esiste, `/proc/filesystem` per eseguire una prova con tutti quelli ivi elencati. In genere si usa `/etc/filesystem` se si vuole cambiare l'ordine in cui il controllo viene effettuato,²⁹ si può poi indicare l'uso ulteriore di `/proc/filesystem` terminando `/etc/filesystem` con un asterisco (*).

Si tenga presente che per poter usare `/proc/filesystem` occorre che il filesystem virtuale `/proc` (che abbiamo già incontrato in sez. 1.2.3) sia stato preventivamente montato. Un esempio del formato del file, è il seguente:

```
nodev    rootfs
nodev    bdev
nodev    proc
nodev    sockfs
nodev    tmpfs
nodev    shm
nodev    pipefs
          ext2
nodev    ramfs
nodev    devpts
nodev    usbdevfs
nodev    usbfs
          iso9660
```

in cui i filesystem virtuali sono marcati dalla parola chiave `nodev`, e non vengono usati nel procedimento di ricerca automatica appena illustrato.

Ciascun filesystem è dotato di caratteristiche proprie, ed in generale è possibile gestirle attraverso l'opzione `-o` di `mount`, che permette di specificare dei valori delle opzioni che controllano alcune modalità di funzionamento del filesystem che si va a montare. Alcune di queste opzioni, riportate in tab. 1.11, sono disponibili in generale, altre sono invece specifiche per ciascun tipo di filesystem, e ci torneremo più avanti. Infine alcune delle opzioni, in particolare `auto`, `user`, `users` e `defaults` e relative negazioni, hanno significato solo quando usate nel quarto campo di `/etc/fstab` (su cui torneremo fra breve). Più opzioni possono essere specificate simultaneamente scrivendole tutte di seguito separate da virgole (senza spazi in mezzo).

²⁸si tenga infine presente che per alcuni filesystem (in particolare per quelli di rete come `nfs` e `smbfs`) per l'esecuzione del comando non è sufficiente la chiamata alla omonima *system call*, ma devono essere usati dei programmi ausiliari, questi vengono lanciati con l'invocazione automatica di un corrispondente programma `/sbin/mount.TYPE`.

²⁹questo resta utile per provare prima `vfat` di `msdos`, evitando che venga usato quest'ultimo quando è disponibile il primo, perdendo la relativa informazione.

Tipo	Descrizione
adfs	<i>Acorn Disc Filing System</i> , il filesystem del sistema operativo RiscOS.
cramfs	<i>Compressed ROM File System</i> , un filesystem su ROM per sistemi embedded.
ext2	<i>Second Extended File System</i> , il filesystem standard di Linux.
ext3	<i>Second Extended File System</i> (filesystem standard di Linux) in versione <i>journalled</i> .
hfs	<i>Hierarchy File System</i> , il filesystem del MacOS (non MacOS X).
hpfs	<i>? File System</i> , il filesystem di OS/2.
iso9660	Il filesystem dei CD-ROM, secondo lo standard ISO 9660.
jfs	<i>Journalling File System</i> , il filesystem <i>journalled</i> della IBM portato su Linux.
minix	<i>Minix File System</i> , il filesystem del sistema operativo Minix.
ntfs	<i>NT File System</i> , il filesystem di Windows NT.
qnx4	<i>QNX4 File System</i> , il filesystem usato da QNX4 e QNX6.
reiserfs	<i>Reiser File System</i> un filesystem <i>journalled</i> per Linux.
romfs	Il filesystem
ufs	<i>Unix File System</i> , il filesystem usato da vari Unix derivati da BSD (SunOS, FreeBSD, NetBSD, OpenBSD e MacOS X).
vxfs	<i>Veritas VxFS File System</i> , filesystem standard di UnixWare disponibile anche su HP-UX e Solaris.
xfs	<i>? File System</i> , il filesystem di IRIX, portato dalla SGI su Linux.
beefs	<i>BeOS File System</i> , il filesystem del sistema operativo BeOS.
msdos	Il filesystem elementare usato dall'MSDOS.
vfat	Il filesystem FAT usato da Windows 95/98.
proc	Filesystem virtuale che fornisce informazioni sul sistema.
shm	Filesystem virtuale che fornisce l'accesso ai segmenti di memoria condivisa.
devpts	Filesystem virtuale per consentire un accesso efficiente ai terminali virtuali.
usbdevfs	Filesystem virtuale contenente le informazioni relative al bus USB.
nfs	<i>Network File System</i> , filesystem per la condivisione di file attraverso la rete attraverso il protocollo NFS creato da Sun.
coda	<i>Coda? File System</i> , filesystem distribuito su rete che supporta funzionalità evolute come autenticazione, replicazione e operazioni disconesse.
smbfs	<i>SMB File System</i> , filesystem usato per montare le directory condivise di Windows.

Tabella 1.10: Principali filesystem disponibili su Linux e relativi nomi per l'opzione **-t** di **mount**.

Oltre a specificare delle modalità di funzionamento coi valori riportati in tab. 1.11 l'opzione **-o** consente anche di effettuare alcune operazioni speciali; ad esempio usando l'opzione **remount** diventa possibile rimontare al volo un filesystem già montato senza smontarlo, per cambiare alcune delle opzioni precedenti. Uno degli usi più comuni per questa opzione è quello di rimontare in lettura/scrittura un filesystem che si è montato in sola lettura per poterci effettuare un controllo.

Un'altra opzione molto utile è **loop**, che consente di montare il filesystem dal contenuto di un file (ovviamente il file deve contenere un filesystem completo di un qualche tipo). Così ad esempio, se **geomorphix.iso** è l'immagine di un CD si potrà accedere al contenuto in maniera trasparente montandolo come se fosse su un CD con il comando:

```
mount -t iso9660 -o loop geomorphix.iso /cdrom
```

l'interfaccia (detta *loopback*) inoltre consente anche di montare un filesystem opportunamente cifrato, nel qual caso si dovrà specificare l'opzione **encryption** per indicare l'algoritmo di cifratura usato e l'opzione **keybits** per specificare la lunghezza (in bit) della chiave; la password di accesso verrà chiesta sul terminale.

Valore	Significato
async	tutto l'I/O sul filesystem viene eseguito in maniera asincrona, cioè le funzioni di scrittura ritornano ed è il kernel che si incarica di eseguire la effettiva scrittura dei dati nel momento più opportuno (è il valore di default).
atime	Aggiorna il valore del tempo di ultimo accesso ai file presenti sul filesystem (è il valore di default).
auto	tutti i filesystem con questa opzione citati in fstab vengono montati dal comando mount -a (che in genere è quello che viene eseguito all'avvio del sistema).
default	usa le opzioni di default: rw , suid , dev , exec , auto , nouser , e async .
dev	consente l'uso di file di dispositivo presenti nel filesystem (è il valore di default).
exec	consente l'esecuzione di programmi presenti sul filesystem (è il valore di default).
noatime	non aggiorna il valore del tempo di ultimo accesso al file (utile quando si vogliono evitare ulteriori accessi al disco).
noauto	il filesystem deve essere montato esplicitamente (viene ignorato dall'opzione -a).
nodev	non consente l'uso di file di dispositivo presenti sul filesystem.
noexec	non consente l'esecuzione di programmi presenti sul filesystem.
nosuid	non consente che i bit suid e sgid (vedi sez. 1.4.3) abbiano effetto.
ro	monta il filesystem in sola lettura.
rw	monta il filesystem in lettura e scrittura (è il valore di default).
suid	consente che i bit suid e sgid abbiano effetto (è il valore di default).
sync	tutto l'I/O sul filesystem deve essere sincrono (vale a dire che le funzioni di scrittura prima di proseguire aspettano che i dati vengano scritti su disco).
dirsync	esegue in maniera sincrona le operazioni che comportano una scrittura sulle directory (creazione di link, creazione, spostamento e cancellazione di file, creazione e cancellazione di directory e file di dispositivo).
user	consente anche ad un utente normale di montare il filesystem, il nome utente viene scritto su /etc/mtab e solo lui potrà smontarlo, comporta come restrizioni le opzioni noexec , nosuid , e nodev , se non soprusse esplicitamente con exec , suid , e dev .
nouser	solo l'amministratore può montare il filesystem (è il valore di default).
users	consente a qualunque utente di montare o smontare il filesystem, con le stesse restrizioni di user .

Tabella 1.11: Valori per l'opzione **-o** di **mount** disponibili per qualunque tipo di filesystem.

Altre opzioni possibili per **mount** sono **-L**, che permette di specificare una partizione invece che attraverso il corrispondente file di dispositivo attraverso una etichetta (che deve essere stata impostata in fase di partizionamento, vedi sez. 5.2.2), **-v** che aumenta la prolissità dei messaggi, **-w** e **-r** che sono abbreviazioni per **-o rw** e **-o ro**. L'elenco completo è riportato nella pagina di manuale accessibile con **man mount**.

Come accennato nel funzionamento di **mount** è fondamentale il file **/etc/fstab**, il cui nome sta per *file system table*. Questo può essere visto come una specie di file di configurazione del comando. Il formato del file è molto semplice: ogni linea definisce un filesystem da montare, ed è composta da sei campi. Linee vuote o che iniziano per **#** vengono ignorate, i campi di ogni linea sono separati da spazi o tabulatori. La pagina di manuale **man fstab** ne spiega i dettagli; un esempio del suo contenuto è:

```
# /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

Il primo campo descrive il dispositivo su cui sta il filesystem 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).

Avendo abilitato il supporto nel kernel è stato possibile montare anche il filesystem /proc; non avendo questo nessun dispositivo (è completamente virtuale) viene montato usando come dispositivo la parola chiave **proc**. Se ci fossero stati dei file montati via NFS (cioè file condivisi sulla rete) si sarebbero avuti anche righe del tipo:

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

in cui si indica come dispositivo la directory remota da montare. Infine se si sono usate le etichette per le partizioni si possono usare queste ultime al posto dei nomi di dispositivo.

Il secondo campo del file 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* (vedi sez. 5.2.5), si usa la parola chiave **none**.

Il terzo campo indica il tipo di filesystem che sta sul dispositivo che si vuole montare, i vari tipi si sono già riportati in tab. 1.10. Si noti poi come per /dev/hdb6 sia presente la parola chiave **swap** ad indicare che in quel caso il dispositivo non contiene un filesystem, ma viene usato per la *swap* (vedi sez. 5.2.5).

Il quarto campo indica le opzioni con cui si può montare il filesystem, riportate in tab. 1.11. Nel caso si usi l'opzione **defaults** la successiva specificazione di un'altra opzione soprassiede il valore di default. Più opzioni vanno specificate in fila, separate da virgole e senza spazi interposti.

Gli ultimi due campi sono relativi alla manutenzione del filesystem, il quinto campo indica se effettuare il *dump*³⁰ del filesystem ed in genere viene lasciato a 0 (per attivarlo occorre usare invece 1) mentre il sesto campo 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.

L'uso principale di /etc/fstab è il controllo del comportamento del comando **mount -a**, che viene utilizzato nella procedura di avvio del sistema per montare automaticamente tutte le directory del sistema (ad esempio /var, /usr e /home) che sono state installate su filesystem separati rispetto alla radice. In questo caso è necessario marcare la riga relativa con l'opzione **auto**; per i filesystem che non devono essere montati invece (ad esempio CD-ROM e floppy) si deve specificare l'opzione **noauto**.

Il file permette inoltre di semplificare l'uso di **mount** poiché per i filesystem in esso elencati il comando può essere invocato specificando solo il *mount point*. Inoltre con questa sintassi consente l'uso di **mount** anche agli utenti normali,³¹ i quali potranno montare un dispositivo qualora si siano specificate le opzioni **user** o **users** nella riga relativa. In questo modo si può permettere agli utenti di montare i propri CD e floppy, senza però consentirgli di modificare il *mount point* o le opzioni di montaggio.

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).

³⁰è un valore utilizzabile solo per i filesystem (attualmente ext2 e ext3) che supportano il comando di backup **dump** (vedi sez. 4.1.4); se attivato con un valore non nullo verranno salvate le informazioni che consentono a **dump** di eseguire i backup incrementali, per maggiori dettagli si faccia riferimento alla pagina di manuale del comando.

³¹l'operazione è privilegiata e può essere effettuata in modo generico solo dall'amministratore.

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 siano 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 sola lettura 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).

Un altro file collegato all'uso di `mount` è `/etc/mtab`, che contiene l'elenco dei filesystem montati. Viene usato da alcuni programmi per leggere questa informazione, ma viene generato automaticamente e non deve essere modificato; lo si può sostituire con un link a `/proc/mounts` che mantiene le stesse informazioni.

Si tenga presente che quando si monta un filesystem su una directory un eventuale contenuto di quest'ultima viene *oscurato* dal contenuto del nuovo filesystem, e non sarà più possibile accedervi fintanto che questo non viene smontato. Se però si sono aperti dei file in essa presenti questi continueranno a funzionare regolarmente (in quanto sono visti attraverso il loro inode, si ricordi quanto detto in sez. 1.2.2). Inoltre a partire dal kernel 2.4 diventa possibile *impilare* più operazioni di mount sulla stessa directory, che presenterà il contenuto dell'ultimo filesystem montato (valendo quanto detto prima per il contenuto dei precedenti).

In maniera analoga a come lo si è montato, quando non si ha più la necessità di accedere ad un filesystem, questo potrà essere *smontato*. In questo modo diventa possibile rimuovere (nel caso di kernel modulare) le eventuali risorse aggiuntive, e liberare la directory utilizzata per il montaggio per il riutilizzo³².

Il comando in questo caso è `umount`³³ che prende come parametro sia il *mount point* che il file di dispositivo e distacca il relativo filesystem dall'albero dei file. Si tenga presente che fintanto che il filesystem è utilizzato questa operazione non viene permessa; questo significa che se si hanno processi che hanno aperto dei file contenuti nel filesystem, o la cui directory di lavoro (vedi sez. 1.3.1) è ivi contenuta non si potrà smontare il filesystem. Per ovviare a questo problema, nelle condizioni in cui è comunque indispensabile smontare filesystem, si può usare l'opzione `-f` che forza l'operazione. Dal kernel 2.4.11 è inoltre disponibile un *lazy umount*, attivabile con l'opzione `-l`, che distacca immediatamente il filesystem (impedendo ogni ulteriore accesso allo stesso) ma esegue le successive operazioni di pulizia solo quando tutte le risorse occupate vengono liberate.

Una sintassi alternativa per il comando è l'uso dell'opzione `-a`, che smonta tutti i filesystem elencati in `/etc/mtab` (tranne, a partire dalla versione 2.7, `/proc`), in questo caso ovviamente non è necessario indicare quale dispositivo smontare, ma si può restringere le operazioni a tutti i filesystem di un determinato tipo, specificando quest'ultimo con l'opzione `-t`. L'elenco completo delle opzioni del comando è disponibile nella pagina di manuale, accessibile con `man umount`.

1.3 L'architettura dei processi

In questa sezione prenderemo in esame l'architettura della gestione dei processi, che costituiscono l'entità fondamentale con cui il kernel permette l'esecuzione dei vari programmi. Vedremo come i processi sono organizzati in forma gerarchica, quali sono caratteristiche e proprietà che ciascuno di essi porta con sé, e come vengono gestiti all'interno del sistema.

³²con le ultime versioni di kernel in realtà questo non è più necessario, in quanto si possono *impilare* più montaggi sulla stessa directory; è comunque necessario smontare un device rimovibile come il floppy o il CD, prima di poterlo estrarre e sostituire.

³³si dice che la `n` si sia persa nei meandri delle prime implementazioni di Unix.

1.3.1 Le proprietà dei processi

Come accennato in sez. 1.1.2 una delle caratteristiche principali dell'architettura dei processi in un sistema unix-like è che qualunque processo questo può creare nuovi processi; in questo contesto il processo originale viene chiamato *padre*, mentre i processi da lui creati vengono detti *figli*. La caratteristica distintiva è che tutti i processi presenti nel sistema possono essere creati solo in questo modo, e pertanto tutti i processi avranno un padre; l'unica eccezione è quella di *init*, il processo iniziale che, venendo lanciato direttamente dal kernel all'avvio, non è figlio di nessun altro processo.

Questa caratteristica permette di classificare i processi in una gerarchia ad albero basata sulla relazione padre-figlio; in questa gerarchia *init* viene a ricoprire nel sistema un ruolo speciale, come radice dell'albero. Questa classificazione può essere stampata con il comando **ps tree** che evidenzia in maniera grafica l'*albero genealogico* dei processi presenti nel sistema, con un risultato del tipo:

```
init--atd
| -bdf flush
| -bonobo-moniker-
| -cron
| -evolution-addre
| -evolution-alarm
| -evolution-calen
| -evolution-execu
| -evolution-mail---evolution-mail---4*[evolution-mail]
| -gconfd-2
| -6*[getty]
| -inetd---famd
| -junkbuster
| -kalarmd
| -kapmd
| -kdeinit--artsd
|     | -evolution
|     | -gabber
|     | -kdeinit---xvncviewer
|     | -kdeinit
|     | -kdeinit---bash---bash
|     | -kdeinit---bash--emacs
|     | | '-xpdf---xpdf.bin
|     | -kdeinit---bash---pstree
|     | '-kdeinit---bash---ssh
| -8*[kdeinit]
| -kdeinit---mozilla-bin---mozilla-bin---4*[mozilla-bin]
| -kdm--XFree86
|     | '-kdm---kde3--kwrapper
|     | '-ssh-agent
| -keventd
| -khubd
| -klogd
| -korgac
| -kreiserfsd
| -ksensors
| -ksoftirqd_CPU0
| -kswapd
| -kupdated
| -lockd---rpciod
| -master--cleanup
|     | -pickup
|     | -proxymap
|     | -qmgr
|     | -smtp
```

```

|          |-smtpd
|          '-trivial-rewrite
|-oafd
|-sshd
|-syslogd
|-wombat
|-wwwoffled
'-xfs

```

dove si può notare che, come dicevamo all'inizio, alla radice dell'albero c'è `init`.

Si tenga presente che questa architettura, in cui qualunque processo può creare degli altri processi, è molto diversa da quella di altri sistemi operativi in cui spesso l'operazione di lanciare un nuovo processo è privilegiata e non può essere eseguita da un programma qualsiasi.

Una seconda differenza rispetto ad altri sistemi multiutente come il VMS o NT, è che in Linux la creazione di un processo e l'esecuzione di un programma sono due operazioni separate, gestite da due *system call* diverse, la prima delle quali crea un nuovo processo, identico al padre, il quale va ad usare la seconda per eseguire un altro programma.³⁴

Il meccanismo però permette anche, con una modalità di funzionamento comune con i server di rete, di scrivere un programma unico che esegue più processi, in cui il padre esegue la parte che si occupa di ricevere le richieste, e per ciascuna di esse fa eseguire³⁵ ad un figlio creato appositamente le operazioni necessarie a fornire le relative risposte.

Il comando che permette di ottenere la lista dei processi attivi nel sistema è `ps`; questo è uno dei comandi fondamentali presenti fin dalle prime versioni di Unix; per questo si sono accavallate anche diverse sintassi per le opzioni, derivate dalle varie versioni che del comando sono state realizzate nel tempo. In Linux il comando supporta la maggior parte delle opzioni esistenti, in genere indicate da lettere singole: quelle derivate da SysV devono essere precedute da un `-`, quelle derivate da BSD non devono essere precedute da un `-`, mentre le estensioni GNU usano un `--` ed una forma estesa.

Si tenga presente che in molti casi viene usata la stessa lettera, con significati diversi, sia nel formato BSD che in quello SysV, nel caso si vogliano combinare le due opzioni di questi formati alternativi allora non vale la sintassi usuale (che tratteremo in dettaglio in sez. 2.1.3) di specificare più opzioni con lettere consecutive dopo il `-`, e ciascuna andrà ripetuta a parte.

Il comando non richiede nessun argomento, e se eseguito su un terminale senza specificare nessuna opzione `ps` mostra l'elenco dei processi appartenenti all'utente che ha eseguito il comando attivi su quel terminale; avremo cioè qualcosa del tipo:

```

piccardi@monk:~/truedoc/corso$ ps
  PID TTY          TIME CMD
 31203 pts/0    00:00:00 bash
 18957 pts/0    00:00:43 emacs
 21348 pts/0    00:00:09 xpdf.bin
 22913 pts/0    00:00:00 ps

```

che mostra, nell'ultima colonna marcata `CMD`, come sul terminale siano presenti la shell, il comando stesso, più l'editor ed il visualizzatore di PDF utilizzati per la realizzazione di queste dispense. Tralasciamo per ora il significato delle altre colonne, ci torneremo fra poco.

Specificando l'opzione `a` verranno visualizzati anche i processi lanciati da altri utenti, purché facenti riferimento ad un terminale,³⁶ mentre con l'opzione `x` si visualizzano tutti i processi non

³⁴per i dettagli del meccanismo si può consultare la sezione 3.2 di [1].

³⁵ovviamente usando una istruzione condizionale, essendovi la possibilità di riconoscere se si sta eseguendo il padre o il figlio.

³⁶prima dell'introduzione delle interfacce grafiche questa era la modalità per vedere i processi interattivi lanciati dagli utenti, in quanto questi potevano essere solo lanciati da un terminale.

associati ad un terminale; infine l'opzione **f** permette di mostrare la gerarchia dei processi. Così si può ottenere un elenco completo dei processi con un comando del tipo:

```
[piccardi@hogen piccardi]$ ps axf
  PID TTY          STAT       TIME COMMAND
    6 ?           SW          0:00 [kupdated]
    5 ?           SW          0:00 [bdf flush]
    4 ?           SW          0:00 [kswapd]
    3 ?           SWN         0:00 [ksoftirqd_CPU0]
    1 ?           S           0:03 init [2]
    2 ?           SW          0:00 [keventd]
    7 ?           SW          0:00 [kjournald]
   76 ?           SW          0:00 [kjournald]
  106 ?           S           0:00 /sbin/portmap
  168 ?           S           0:00 /sbin/syslogd
  171 ?           S           0:00 /sbin/klogd
  176 ?           S           0:00 /usr/sbin/named
  180 ?           S           0:00 /sbin/rpc.statd
  327 ?           S           0:00 /usr/sbin/gpm -m /dev/psaux -t ps2
  332 ?           S           0:00 /usr/sbin/inetd
  344 ?           S           0:00 lpd Waiting
  435 ?           S           0:00 /usr/lib/postfix/master
  437 ?           S           0:00 \_ pickup -l -t fifo -c
  438 ?           S           0:00 \_ qmgr -l -t fifo -u -c
  448 ?           S           0:00 /usr/sbin/sshd
  908 ?           S           0:00 \_ /usr/sbin/sshd
  909 pts/0        S           0:00 \_ -bash
  919 pts/0        R           0:00 \_ ps axf
  474 ?           S           0:00 /usr/sbin/atd
  477 ?           S           0:00 /usr/sbin/cron
  484 tty2         S           0:00 /sbin/getty 38400 tty2
  485 tty3         S           0:00 /sbin/getty 38400 tty3
  486 tty4         S           0:00 /sbin/getty 38400 tty4
  487 tty5         S           0:00 /sbin/getty 38400 tty5
  488 tty6         S           0:00 /sbin/getty 38400 tty6
  635 ?           SN          0:00 /usr/sbin/junkbuster /etc/junkbuster/config
  672 ?           SN          0:00 /usr/sbin/wwwoffled -c /etc/wwwoffle/wwwoffle.conf
  907 tty1         S           0:00 /sbin/getty 38400 tty1
```

dato che la combinazione delle opzioni **a** e **x**, essendo queste complementari, seleziona tutti i processi attivi nel sistema. L'uso dell'opzione **r** permette invece di restringere la selezione ai soli processi in esecuzione effettiva (cioè nello stato **R**, che spiegheremo fra poco).

Come già questi primi esempi ci mostrano, in realtà lo scopo di **ps** non è semplicemente quello di fornire la lista dei processi in esecuzione, quanto quello di mostrare le loro caratteristiche; la descrizione del significato di quest'ultime andrà allora di pari passo con la spiegazione dell'output del comando.

Per ogni processo attivo infatti il kernel mantiene tutta una serie di caratteristiche ed identificatori usati per il controllo delle varie operazioni che esso può compiere, insieme ad una serie di informazioni relative alle risorse da esso utilizzate. Già in questi due primi esempi il comando ci mostra alcune informazioni di base sufficienti ad introdurre alcune delle proprietà essenziali dei processi.

La prima colonna dei precedenti esempi, marcata **PID**, mostra il cosiddetto *process id* del processo. Questo è il numero che il kernel utilizza per identificare univocamente ciascun processo; questo numero viene assegnato alla creazione del processo, ed è unico fintanto che il processo resta attivo. È questo il numero che si deve usare tutte le volte che si vuole fare riferimento ad uno specifico processo.

La seconda colonna, marcata **TTY**, mostra il nome del terminale di controllo del processo. Ogni processo interattivo è sempre associato ad un terminale di controllo, che corrisponde ap-

punto al terminale da cui il processo riceve i dati in ingresso (in genere dalla tastiera) e sul quale scrive il suo output (in genere lo schermo). Se il processo non è interattivo, o non lo è attraverso l'interfaccia a riga di terminale (come un programma che usa l'interfaccia grafica) non esiste un terminale di controllo e la colonna riporta come valore “?”.

La terza colonna, marcata **STAT**, riporta un'altra informazione fondamentale: lo stato del processo. Il sistema infatti prevede cinque possibili stati diversi per i processi, i cui valori sono riportati in tab. 1.12, torneremo su questo a breve, in quanto ci permette di spiegare varie caratteristiche dell'architettura dei processi. Seguono infine la colonna **TIME** che indica il tempo di CPU usato finora dal processo e la colonna **COMMAND** che riporta la riga di comando usata per lanciare il programma.

Per capire il significato del campo **STAT** occorrono alcune spiegazioni generali sull'architettura della gestione dei processi. Come sistema multitasking Linux è in grado di eseguire più processi contemporaneamente, in genere però un processo, pur essendo attivo, non è assolutamente detto che sia anche in esecuzione. Il caso più comune infatti è quello in cui il programma è in attesa di ricevere dati da una periferica; in tal caso il kernel pone il programma in stato di *sleep* e lo toglie dalla lista di quelli che hanno bisogno del processore (che sono identificati invece dallo stato *runnable*).

Stato	STAT	Descrizione
<i>runnable</i>	R	Il processo è in esecuzione o è pronto ad essere eseguito (cioè è in attesa che gli venga assegnata la CPU).
<i>sleep</i>	S	Il processo è in attesa di un risposta dal sistema, ma può essere interrotto da un segnale.
<i>uninterruptible sleep</i>	D	Il processo è in attesa di un risposta dal sistema (in genere per I/O), e non può essere interrotto in nessuna circostanza.
<i>stopped</i>	T	Il processo è stato fermato con un SIGSTOP , o è tracciato.
<i>zombie</i>	Z	Il processo è terminato ma il suo stato di terminazione non è ancora stato letto dal padre.

Tabella 1.12: Elenco dei possibili stati di un processo in Linux, nella colonna **STAT** si è riportata la corrispondente lettera usata dal comando **ps** nell'omonimo campo.

Si noti allora come nell'elenco precedente tutti i processi, eccetto lo stesso comando **ps axf**, fossero in stato di *sleep*. In genere un processo entra in stato di *sleep* tutte le volte che si blocca nell'esecuzione di una *system call* che richiede una qualche forma di accesso non immediato a dei dati (caso classico è la lettura dell'input da tastiera).

Come dettagliato in tab. 1.12 gli stati di *sleep* sono due, contraddistinti dalle lettere S e D. Il più comune è il primo; l'esecuzione della maggior parte delle *system call* infatti può essere interrotta da un segnale (i segnali saranno trattati in sez. 1.3.2), per cui se un processo si blocca nell'esecuzione di una di queste esso può essere comunque terminato, in alcuni casi (in genere questo avviene quando la *system call* sta gestendo la risposta ad un interrupt hardware) questo non è possibile ed allora si ha uno stato di *uninterruptible sleep*, in questo caso se il processo resta bloccato³⁷ nello stato D non può più essere terminato se non con il riavvio della macchina. Un processo in stato D comunque non avrà nessun effetto né sugli altri processi né sul sistema, che continuerà a funzionare senza problemi, a parte quello di non poter liberare le risorse occupate dal processo.

Lo stato di *stopped* è relativo ai processi la cui esecuzione è stata fermata per una richiesta dell'utente (per fare questo, come vedremo in sez. 1.3.2, si ha a disposizione un segnale apposito), nel qual caso semplicemente il processo non viene eseguito (ma resta in memoria e può riprendere

³⁷in genere questo avviene per un qualche errore nella gestione della periferica nel kernel, ad esempio se si monta un disco USB e poi si lo si estrae dal bus senza smontarlo e si ha la sventura di farlo in un momento poco opportuno.

l'esecuzione in qualunque momento) fintanto che non lo si fa ripartire (anche per questo è previsto un altro segnale).

Infine c'è lo stato di *zombie*, il cui significato può essere compreso solo ritornando con maggiori dettagli sulla relazione fra processo padre e gli eventuali figli. La relazione padre/figlio infatti è ben definita finché entrambi i processi sono attivi nel sistema, ma se uno dei due termina cosa accade? Alla terminazione di un processo il kernel provvede ad una serie di compiti di pulizia; tutti i file aperti dal processo vengono chiusi, la memoria utilizzata viene liberata e con essa tutte le risorse occupate dal processo. Resta il problema di come notificare l'avvenuta conclusione del processo, ad esempio se questa è stata regolare o è stata dovuta ad un qualche errore, ma soprattutto il problema è a chi fare questa notifica.

Dato che in un sistema Unix tutto viene fatto con i processi, la scelta è stata quella per cui è compito del padre ricevere lo stato di uscita del figlio e controllare se tutto è andato bene o c'è stato qualche errore; quando un processo termina al padre viene inviato uno speciale segnale che lo avvisa del fatto, e lui deve invocare una apposita *system call* per ricevere lo stato di uscita. Questo ad esempio è il meccanismo con cui la shell riceve lo stato di uscita dei comandi che si sono eseguiti.

Se questo non viene fatto comunque il processo figlio si conclude regolarmente e tutte le risorse che occupava nel sistema vengono rilasciate, resta allocata soltanto una voce nella tabella dei processi che contiene le informazioni per riportare lo stato di uscita. Questo è quello che in gergo viene chiamato uno *zombie*, cioè un processo che non esiste più, perché è terminato, ma che mostra una voce con lo stato Z nella lista fornita da **ps** e che di nuovo non può essere terminato, per il semplice fatto che lo è già. È pertanto compito di chi scrive programmi che creano processi figli curarsi della ricezione del loro stato di uscita, ed infatti i programmi ben scritti non presentano mai questo problema.

Di per sé la presenza di uno *zombie* non è un grave problema, in quanto esso non occupa (a differenza di un processo bloccato in stato D) nessuna risorsa nel sistema, tranne la voce mantenuta nell'output di **ps**. Però uno *zombie* occupa comunque una voce nella tabella dei processi e pertanto se il numero degli *zombie* cresce si può rischiare di saturare quest'ultima, rendendo inutilizzabile il sistema. Vedremo fra poco però che, a differenza dei processi in stato D, per gli *zombie* è possibile risolvere il problema e far sì che essi siano terminati regolarmente senza dover riavviare il sistema.

Per capire come comportarsi con gli *zombie* si deve considerare un altro caso della gestione del funzionamento dei processi: quello in cui è il padre che termina per primo. Se accade questo chi è che riceverà lo stato di terminazione dei figli? Dato che i processi sono eseguiti in maniera del tutto indipendente un caso come questo è assolutamente naturale, e si dice che il figlio diventa *orfano*. Per questo il sistema controlla, durante le operazioni di terminazione di un processo, se questo ha dei figli, e nel caso assegna a questi ultimi **init** come nuovo padre.³⁸ Si dice allora che **init** *adotta* i figli dei processi che terminano, e sarà lui che gestirà, alla terminazione di questi ultimi, la ricezione del loro stato di uscita.

Perciò, dato che **init** è scritto bene e sa gestire la ricezione dello stato di uscita, tutto quello che serve fare per eliminare gli *zombie* dal sistema è renderli orfani terminando il processo che li ha generati.³⁹ In questo modo essi saranno adottati da **init** che si curerà immediatamente di riceverne lo stato di uscita liberando la voce che occupavano nella tabella dei processi.

Si noti nella lista precedente la presenza di alcuni processi con una lettera aggiuntiva W nella colonna **STAT**, con un nome del comando fra parentesi quadre e con un PID molto basso; questi non sono processi effettivamente lanciati da **init** quanto dei processi interni al kernel (e da

³⁸Si vedrà cioè, ad una successiva esecuzione di **ps** con opzioni che permettano di visualizzare il PID del padre, che questo è diventato 1.

³⁹per poterlo fare però occorre avere un processo in grado di eseguire il comando, cosa non facile da ottenere se si è già esaurita la tabella dei processi.

esso utilizzati) per la gestione di alcuni compiti interni, come lo scarico su disco dei buffer, la generazione di eventi dovuti all'inserimento di periferiche (usato da USB e PCMCIA), ecc. Nel caso si usa la lettera W per indicare che i processi non usano memoria in user space, il che permette di identificarli. Le altre lettere associate al campo STAT sono ">", "N" e "L" e indicano rispettivamente una priorità maggiore o minore di quella standard (torneremo sulla priorità in sez. 1.3.3) e la presenza di pagine di memoria bloccate.⁴⁰

Le opzioni di visualizzazione di `ps` sono moltissime, e qui potremo prendere in esame solo le principali. Restando nell'ambito della sintassi BSD una delle più usate è `u` che stampa una lista con le informazioni più rilevanti riguardo l'utente, altre opzioni sono `v` che stampa informazioni relative all'uso della memoria virtuale, e `s` che stampa informazioni sui segnali. Infine l'opzione `o` permette all'utente di specificare un suo formato, con l'uso di una serie di direttive `%X`, dove `X` indica quale proprietà del processo si vuole far comparire nella lista (secondo una tabella di valori riportata nella pagina di manuale).

Se invece si usa la sintassi SysV le opzioni più usate sono `-e`, che permette di selezionare tutti processi, e `-f` che permette di avere una lista con più informazioni; in tal caso infatti si avrà come uscita del comando:

```
parker:/home/piccardi# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0 Aug12 ?        00:00:03 init
root           2        1  0 Aug12 ?        00:00:00 [keventd]
root           3        1  0 Aug12 ?        00:00:00 [ksoftirqd_CPU0]
root           4        1  0 Aug12 ?        00:00:00 [kswapd]
root           5        1  0 Aug12 ?        00:00:00 [bdfldush]
root           6        1  0 Aug12 ?        00:00:00 [kupdated]
root           7        1  0 Aug12 ?        00:00:01 [kjournald]
root          43        1  0 Aug12 ?        00:00:00 [kapmd]
root         101        1  0 Aug12 ?        00:00:00 [eth0]
daemon       106        1  0 Aug12 ?        00:00:00 /sbin/portmap
root         168        1  0 Aug12 ?        00:00:00 /sbin/syslogd
root         171        1  0 Aug12 ?        00:00:00 /sbin/klogd
root         175        1  0 Aug12 ?        00:00:00 /usr/sbin/named
root         179        1  0 Aug12 ?        00:00:00 /sbin/rpc.statd
root         203        1  0 Aug12 ?        00:00:00 /usr/sbin/inetd
daemon       214        1  0 Aug12 ?        00:00:00 lpd Waiting
root         310        1  0 Aug12 ?        00:00:00 /usr/lib/postfix/master
postfix      314       310  0 Aug12 ?        00:00:40 qmgr -l -t fifo -u -c
root         319        1  0 Aug12 ?        00:00:00 /usr/sbin/sshd
root         322        1  0 Aug12 ?        00:00:00 /usr/sbin/cron
root         325        1  0 Aug12 tty1      00:00:00 /sbin/getty 38400 tty1
root         326        1  0 Aug12 tty2      00:00:00 /sbin/getty 38400 tty2
root         327        1  0 Aug12 tty3      00:00:00 /sbin/getty 38400 tty3
root         328        1  0 Aug12 tty4      00:00:00 /sbin/getty 38400 tty4
root         329        1  0 Aug12 tty5      00:00:00 /sbin/getty 38400 tty5
root         330        1  0 Aug12 tty6      00:00:00 /sbin/getty 38400 tty6
postfix      2046       310  0 11:13 ?        00:00:00 pickup -l -t fifo -u -c
root         2047       319  0 12:24 ?        00:00:00 /usr/sbin/sshd
piccardi     2049      2047  0 12:24 ?        00:00:00 /usr/sbin/sshd
piccardi     2050      2049  0 12:24 pts/0    00:00:00 -bash
root         2054      2050  0 12:24 pts/0    00:00:00 bash
root         2087      2054  0 12:34 pts/0    00:00:00 ps -ef
```

E come si vede questa versione riporta una serie di dati in più. Anzitutto notiamo che la prima colonna, UID, riporta un nome utente. Ciascun processo infatti mantiene le informazioni riguardanti l'utente che ha lanciato il processo ed il gruppo cui questo appartiene, anche queste

⁴⁰quest'ultima è una caratteristica avanzata che va al di là di quanto affrontabile in questa sede, per una trattazione si può fare riferimento alla sezione 2.2.7 del secondo capitolo di GaPiL [1].

sono mantenute nella forma dei relativi UID (*user id*) e GID (*group id*).⁴¹ Ogni processo in realtà mantiene diverse versioni di questi identificatori,⁴² ma quelle che poi sono significative sono l'*effective user id* e l'*effective group id*, utilizzati per il controllo di accesso (che vedremo in sez. 1.4), che sono appunto quelli che corrispondono alla colonna UID (e ad una eventuale colonna GID) ed il *real user id* e l'*real group id*, che identificano l'utente che ha lanciato il processo,⁴³ che invece vengono identificati da sigle come RUSER e RGROUP, se indicati con il nome o con RUID e RGID se numerici.

La seconda colonna del nostro esempio riporta di nuovo il PID di ciascun processo, mentre la terza colonna, PPID, indica il *parent process id*, ogni processo infatti mantiene il PID del padre, in modo da poter ricostruire agevolmente la genealogia dei processi. Questa ci permette anche, in caso di *zombie*, di identificare il processo responsabile della produzione degli stessi.

La quarta colonna, C, indica, in un sistema multiprocessore, l'ultima CPU usata dal processo. La quinta colonna, STIME, indica invece il momento in cui il comando è stato lanciato. Le altre colonne sono analoghe a quelle già viste in precedenza per la sintassi BSD. Un elenco delle principali informazioni relative ai processi che vengono riportate da `ps` sono elencate in tab. 1.13, indicate sempre in base al nome usato come intestazione della colonna che ne visualizza il valore. Per l'elenco completo delle opzioni, informazioni e dati visualizzabili con `ps` si può fare riferimento alla pagina di manuale del comando, accessibile con `man ps`.

Proprietà	Descrizione
PID	PID (<i>process ID</i>) del processo.
PPID	PID (<i>process ID</i>) del padre del processo.
UID	<i>effective user id</i> del processo.
GID	<i>effective group id</i> del processo.
CMD	linea di comando con cui è stato lanciato il processo.
STAT	stato del processo.
NI	valore di <i>nice</i> (vedi sez. 1.3.3) del processo.
TTY	terminale di riferimento del processo.
SID	SID o <i>session id</i> (vedi sez. 1.3.4) del processo.
PGID	<i>process group id</i> (vedi sez. 1.3.4) del processo.
%CPU	percentuale del tempo di CPU usato rispetto al tempo reale di esecuzione.
%MEM	percentuale della memoria fisica utilizzata dal processo.
C	ultima CPU utilizzata dal processo (ha senso solo in un sistema multiprocessore).
START	orario dell'avvio del processo.
TIME	tempo totale di CPU utilizzato dall'avvio del processo.
USER	<i>effective user id</i> del processo.
RUSER	<i>real user id</i> del processo.
GROUP	<i>effective group id</i> del processo.
RGROUP	<i>real group id</i> del processo.
COMMAND	riga di comando con la quale si è avviato il processo.

Tabella 1.13: Le principali proprietà dei processi e ed il nome della relativa colonna nella visualizzazione effettuata da `ps`.

Infine ci sono alcune proprietà dei processi, non direttamente visualizzabili con `ps`, che comunque sono importanti e vanno menzionate. Come accennato in sez. 1.2.2, ogni processo ha una directory di lavoro rispetto alla quale risolve i pathname relativi; anche questa è una ca-

⁴¹come vedremo in sez. 1.4.1 il sistema identifica ogni utente e gruppo nel sistema con un numero, detti appunto *user id* e *group id*.

⁴²Linux usa ben quattro gruppi di identificatori diversi per gestire il controllo di accesso, secondo uno schema che va al di là di quanto è possibile spiegare qui, gli interessati possono trovare una trattazione dell'argomento nella sezione 3.3 di [1].

⁴³di norma questi coincidono con gli identificatori del gruppo *effective*, ma vedremo in sez. 1.4.3 che esistono casi in cui questo non avviene.

ratteristica del processo, che viene ereditata nella creazione di un processo figlio. Lo stesso vale per la *directory radice*, infatti benché di norma questa coincida con la radice del sistema, essa può essere cambiata con il comando `chroot`, in modo da restringere un processo in una sezione dell'albero dei file. Per questo motivo ogni processo porta con sé, oltre alla directory di lavoro corrente, anche l'indicazione della directory che considera come radice, e rispetto alla quale risolve i pathname assoluti.

Come si può notare `ps` si limita a stampare la lista dei processi attivi al momento della sua esecuzione. Se si vuole tenere sotto controllo l'attività del sistema non è pratico ripetere in continuazione l'esecuzione di `ps`. Per questo ci viene in aiuto il comando `top`, che stampa una lista di processi, aggiornandola automaticamente in maniera periodica.

Il comando opera normalmente in maniera interattiva, a meno di non averlo lanciato con l'opzione `-b`, che lo esegue in modalità batch, consentendo la redirectione dell'output; in tal caso di solito si usa anche l'opzione `-n` per specificare il numero di iterazioni volute. Il comando ristampa la lista ogni secondo, a meno di non aver impostato un intervallo diverso con l'opzione `-d`, che richiede un parametro nella forma `ss.dd` dove `ss` sono i secondi e `dd` i decimi di secondo. Infine l'opzione `-p` permette di osservare una lista di processi scelta dall'utente, che deve specificarli tramite una lista di PID. Per la lista completa delle opzioni si faccia riferimento al solito alla pagina di manuale, accessibile con `man top`.

Quando opera in modalità interattiva il comando permette di inviare dei comandi da tastiera. I più rilevanti sono `h` e `?` che mostrano un help in linea dei comandi disponibili, e `q` che termina il programma. Un esempio di output del comando è il seguente:

```
top - 13:06:35 up 6:04, 6 users, load average: 0.04, 0.98, 1.00
Tasks: 82 total, 1 running, 81 sleeping, 0 stopped, 0 zombie
Cpu(s): 1.3% user, 1.3% system, 0.0% nice, 97.4% idle
Mem: 256180k total, 252828k used, 3352k free, 8288k buffers
Swap: 524280k total, 85472k used, 438808k free, 110844k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3605	piccardi	17	0	10052	9.8m	4512	S	1.6	3.9	1:03.01	emacs
3729	piccardi	16	0	1124	1124	896	R	1.3	0.4	0:01.00	top
1	root	8	0	548	516	516	S	0.0	0.2	0:02.73	init
2	root	8	0	0	0	0	S	0.0	0.0	0:00.57	keventd
3	root	19	19	0	0	0	S	0.0	0.0	0:00.01	ksoftirqd_CPU0
4	root	9	0	0	0	0	S	0.0	0.0	0:00.66	kswapd
5	root	9	0	0	0	0	S	0.0	0.0	0:00.00	bdflood
6	root	9	0	0	0	0	S	0.0	0.0	0:00.06	kupdated
83	root	9	0	0	0	0	S	0.0	0.0	0:00.01	khud
186	root	9	0	968	932	836	S	0.0	0.4	0:00.00	dhclient
190	daemon	9	0	500	452	436	S	0.0	0.2	0:00.04	portmap
289	root	9	0	912	852	836	S	0.0	0.3	0:00.63	syslogd
295	root	9	0	1196	528	504	S	0.0	0.2	0:00.14	klogd
320	root	9	0	824	768	740	S	0.0	0.3	0:00.02	inetd
324	root	9	0	864	784	756	S	0.0	0.3	0:00.00	lpd
330	root	9	0	724	684	636	S	0.0	0.3	0:49.00	pbbUTTONSD

In testa vengono sempre stampate cinque righe di informazioni riassuntive sul sistema: nella prima riga viene riportato ora, *uptime*,⁴⁴ numero di utenti e carico medio della macchina; nella seconda riga le statistiche sul totale dei processi; nella terza le statistiche di utilizzo della CPU, nelle ultime due le statistiche di uso della memoria fisica e della *swap*.

A queste informazioni generiche seguono, dopo una riga lasciata vuota che serve per gestire l'input in interattivo, le informazioni sui processi, ordinati per uso decrescente della CPU (vengono cioè mostrati i più attivi), evidenziando (con la stampa un grassetto) quelli trovati in stato *runnable*. Le informazioni riportate di default sono il PID del processo (colonna omonima),

⁴⁴cioè il tempo trascorso da quando il sistema è stato avviato.

l'utente cui esso appartiene (colonna **USER**); la priorità ed il valore di *nice* (torneremo su questi in sez. 1.3.3) rispettivamente nelle colonne **PR** e **NI**.

Seguono i dati dell'uso della memoria nelle colonne **VIRT**, **RES** e **SHR**; per capire queste quantità occorre dare qualche dettaglio in più sul sistema della *memoria virtuale* cui abbiamo già accennato in sez. 1.1.1; in quella occasione abbiamo detto come sia compito del kernel mappare lo spazio (virtuale) degli indirizzi di memoria di un processo nella memoria fisica effettivamente disponibile. La memoria usata da un processo è sostanzialmente suddivisa in due parti,⁴⁵ il codice del programma e i dati usati dallo stesso.

Il programma per essere eseguito dovrà avere il codice ed i relativi nella memoria fisica, e la quantità di memoria attualmente usata a questo scopo è quella che è indicata dalla colonna **RES** e viene detta *residente*, che sarà la somma della parte usata per i dati (indicata da **DATA**) e della parte usata per il codice (indicata da **CODE**). Una parte di quest'ultima (ad esempio il codice delle librerie) sarà condivisa con altri processi, e questa viene indicata dalla colonna **SHR**.

Ci sarà però anche una parte di memoria che al momento non è in uso, e che, per liberare della memoria fisica a favore di altri programmi che devono essere eseguiti, è stata temporaneamente *parcheeggiata* su una area di disco a questo dedicata (detta *swap*) da cui può essere ripresa in caso di necessità. Allora il totale della memoria vista dal programma, che a questo punto è *virtuale*, in quanto non corrisponde a della RAM direttamente accessibile, è quello espresso dalla colonna **VIRT**, che sarà la somma sia della parte *residente* (quella di **RES**) che di quella che è stata *parcheeggiata* nell'area di *swap*.

Come altre informazioni presenti nella stampa di default il comando riporta lo stato del processo (colonna **S**), le percentuali di utilizzo di CPU e memoria (colonne **%CPU** e **%MEM**), il tempo trascorso dall'avvio del programma (colonna **TIME+**) ed il comando usato per lanciarlo (colonna **COMMAND**), il cui significato è analogo a quello già visto per **ps**.

In generale le informazioni riportate nelle colonne stampate da **top** sono simili a quelle di **ps**, anche in alcuni casi esse vengono indicate diversamente. Le principali proprietà mostrate di **top** sono riportate in tab. 1.14, per quelle comuni non presenti in detta tabella si faccia riferimento alle omonime di **ps** riportate in tab. 1.13.

Proprietà	Descrizione
SHR	ammontare della <i>memoria condivisa</i> , rappresenta la memoria potenzialmente condivisibile con altri processi.
SWAP	ammontare delle pagine della <i>memoria virtuale</i> di un processo.
CODE	ammontare della memoria fisica utilizzata dal processo per il suo codice eseguibile.
DATA	ammontare della memoria fisica utilizzata dal processo per i suoi dati.
RES	ammontare della memoria fisica usata dal processo (uguale a CODE + DATA).
VIRT	ammontare totale della <i>memoria virtuale</i> usata dal processo include tutto il codice, i dati e le librerie condivise più le pagine che sono state messe su <i>swap</i> (uguale a SWAP + RES).
S	stato del processo (analogo allo STAT di ps).
TIME+	tempo di CPU utilizzato dall'avvio (analogo al TIME di ps , ma con granularità fino al centesimo di secondo).

Tabella 1.14: Proprietà dei processi e nome della relativa colonna nella visualizzazione effettuata da **top**.

Una delle caratteristiche di **top** è che se viene usato in modalità interattiva diventa possibile dare una serie di comandi da tastiera, ad esempio con **k** si può inviare un segnale (di default è **SIGTERM**, vedi sez. 1.3.2) mentre con **r** si può cambiare la priorità (vedi sez. 1.3.3) di un processo. Con il comando **u** si possono selezionare i processi di un utente, con **c**, si può alternare fra la stampa del nome comando e della riga completa, con **d** cambiare il periodo di aggiornamento

⁴⁵si sta usando una descrizione brutalmente semplificata, per una trattazione dettagliata dell'argomento si può fare riferimento alla sezione 2.2 di [1].

dei risultati. L'elenco completo, oltre che nella pagina di manuale, può essere stampato a video con **h**.

Si noti che nelle righe iniziali **top** riporta anche delle statistiche complessive sull'uso della memoria, queste possono essere ottenute separatamente tramite il comando **free**, che mostra un riassunto generale del tipo:

```
piccardi@monk:~/truedoc/corso$ free
              total        used        free      shared    buffers     cached
Mem:      775444      759364      16080           0       213276      316908
-/+ buffers/cache:      229180      546264
Swap:      498004       34708      463296
```

La prima riga riporta l'uso della memoria fisica, mentre l'ultimo quello della *swap* (se questa è attiva, torneremo sull'argomento in sez. 5.2.5); la riga centrale ci dice quanto della memoria viene utilizzata per i buffer. Si noti che in genere la RAM libera è sempre molto poca, questo è corretto, in quanto non ha senso lasciare inutilizzata la RAM, ed allora viene impiegata dai buffer del kernel (per gestire più efficacemente il trasferimento dei dati verso i dispositivi) e per mantenere dati temporanei, come riportato nelle colonne **buffer** e **cache**.⁴⁶

Il comando non ha argomenti e prende come opzioni **-b**, **-k** e **-m** per stampare i dati di utilizzo rispettivamente in byte, kilobyte (il default) e megabyte. I dettagli sono al solito nella pagina di manuale accessibile con **man free**.

1.3.2 I segnali

Benché i processi siano di norma entità separate e completamente indipendenti fra di loro, esistono molti casi in cui è necessaria una qualche forma di comunicazione. La forma più elementare di comunicazione fra processi è costituita dai segnali, che sono usati anche direttamente dal kernel per comunicare ai processi una serie di eventi o errori (come l'uso inappropriato della memoria o una eccezione aritmetica).

Come dice la parola un segnale è una specie di avviso che viene inviato ad un processo; e non contiene nessuna informazione oltre al fatto di essere stato inviato. In genere i segnali vengono utilizzati per notificare ai processi una serie di eventi (abbiamo accennato in sez. 1.3.1 che uno di essi viene utilizzato per notificare la terminazione di un processo figlio), e possono essere anche inviati a mano attraverso l'uso del comando **kill**. Ciascun segnale è identificato da un numero ed un nome simbolico; la lista dei segnali disponibili può essere ottenuta semplicemente con:

```
piccardi@oppish:~$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL
 5) SIGTRAP     6) SIGABRT     7) SIGBUS      8) SIGFPE
 9) SIGKILL    10) SIGUSR1    11) SIGSEGV    12) SIGUSR2
13) SIGPIPE    14) SIGALRM    15) SIGTERM    17) SIGCHLD
18) SIGCONT    19) SIGSTOP    20) SIGTSTP    21) SIGTTIN
22) SIGTTOU    23) SIGURG     24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH    29) SIGIO
30) SIGPWR     31) SIGSYS     32) SIGRTMIN    33) SIGRTMIN+1
34) SIGRTMIN+2 35) SIGRTMIN+3 36) SIGRTMIN+4 37) SIGRTMIN+5
38) SIGRTMIN+6 39) SIGRTMIN+7 40) SIGRTMIN+8 41) SIGRTMIN+9
42) SIGRTMIN+10 43) SIGRTMIN+11 44) SIGRTMIN+12 45) SIGRTMIN+13
46) SIGRTMIN+14 47) SIGRTMIN+15 48) SIGRTMAX-15 49) SIGRTMAX-14
50) SIGRTMAX-13 51) SIGRTMAX-12 52) SIGRTMAX-11 53) SIGRTMAX-10
54) SIGRTMAX-9  55) SIGRTMAX-8  56) SIGRTMAX-7  57) SIGRTMAX-6
58) SIGRTMAX-5  59) SIGRTMAX-4  60) SIGRTMAX-3  61) SIGRTMAX-2
62) SIGRTMAX-1 63) SIGRTMAX
```

⁴⁶la colonna **shared** veniva usata per indicare la memoria condivisa fra più processi, adesso è obsoleta e deve essere ignorata.

I segnali effettivamente usati dal sistema sono i primi 31, gli altri sono chiamati *real time signal* ed hanno un uso specialistico che va al di là di quello che possiamo affrontare qui.⁴⁷ In generale ciascuno di essi ha un compito o un significato specifico, alcuni di essi vengono generati automaticamente in caso di errori del programma; ad esempio il segnale **SIGSEGV** viene inviato dal kernel per segnalare ad un processo una *Segment Violation*, cioè un accesso illegale alla memoria;⁴⁸ altri vengono inviati direttamente dall'utente, come **SIGKILL** che causa l'immediata terminazione di un processo.

Gran parte dei segnali (tutti eccetto **SIGKILL** e **SIGSTOP**, che hanno un comportamento speciale) possono essere *intercettati* dal processo, che può eseguire una opportuna funzione al loro arrivo. Ad esempio il segnale **SIGTERM** (che è quello che il comando **kill** invia di default) serve per richiedere la terminazione immediata di un processo, ma il segnale può essere intercettato per eseguire delle operazioni di pulizia come ad esempio cancellare dei file temporanei prima dell'uscita.

Se un segnale non viene intercettato viene eseguita una azione di default che è specifica di ciascuno di essi. Nella maggior parte dei casi l'azione consiste nella terminazione immediata del processo; questa può però avvenire in due modi, o con una semplice uscita o con l'uscita eseguita insieme alla creazione, nella directory di lavoro corrente del processo, di un file **core** che contiene una copia dello spazio della memoria del processo (viene per questo detto un *core dump*) che può essere usato con un programma di debug per vedere in che punto c'è stata l'interruzione. Infine alcuni segnali (come **SIGCHLD**, che è quello che viene usato per notificare al padre la terminazione di un figlio) di default vengono ignorati,⁴⁹ il programma cioè continua ad essere eseguito senza nessuna conseguenza.

Tipici segnali che causano una semplice uscita sono **SIGTERM**, il default del comando **kill**, e **SIGINT** che è associato al carattere di interruzione dato dalla tastiera (vedi sez. 1.3.4), un segnale che invece produce un *core dump* è **SIGQUIT**, così come tutti i segnali relativi ad errori di programmazione come **SIGSEGV**, **SIGFPE**, ecc. Un elenco dettagliato dei segnali, del loro significato e delle azioni di default si può trovare nella sezione 9.2 di [1].

In generale il comando **kill** permette di inviare un segnale ad un processo qualunque, specificando come parametro il PID di quest'ultimo. Come accennato il segnale inviato di default è **SIGTERM**, ma si può inviare qualunque altro segnale specificandone numero o nome preceduto da un **-** come opzione; ad esempio:

```
kill -9 1029
kill -SIGKILL 1029
kill -KILL 1029
kill -s SIGKILL 1029
```

sono modalità equivalenti di inviare il segnale **SIGKILL** al processo con PID 1029. Oltre a **-s** e **-l** il comando **kill** accetta le opzioni **-L** (sinonimo di **-l**) e **-V** che ne stampa la versione. Per una descrizione accurata delle opzioni si faccia al solito riferimento alla pagina di manuale accessibile con **man kill**.

Nel caso specifico, dato che **SIGKILL** non è intercettabile e la sua azione di default è la terminazione del processo, l'effetto di questo comando è di terminare senza possibilità di scampo il processo in questione. Se infatti **SIGTERM** viene intercettato può risultare inefficace qualora il

⁴⁷per una spiegazione più dettagliata al riguardo si può fare riferimento alla sezione 9.4.6 del nono capitolo di GaPiL.

⁴⁸questo è un'altro dei vantaggi della memoria virtuale, se per un errore di programmazione un processo cerca di scrivere su una parte dello spazio degli indirizzi che non corrisponde a nessuna locazione di memoria associata al processo stesso, il sistema se ne accorge ed invia questo segnale che ne causa la terminazione; si impedisce così che un accesso sbagliato possa andare a sovrascrivere la memoria di altri processi.

⁴⁹nel caso di **SIGCHLD** non è proprio il caso di farlo, altrimenti, come spiegato in sez. 1.3.1, ci si ritroverà con degli *zombie*.

processo venga bloccato anche nell'esecuzione della funzione di gestione; dato che non è possibile intercettare **SIGKILL** si ha a disposizione un mezzo infallibile⁵⁰ per terminare un processo impazzito, e questo senza alcuna conseguenza per gli altri processi o per il sistema.

Infine invece del PID si può inviare un segnale ad un intero *process group* (sui *process group* torneremo più avanti in sez. 1.3.4) usando un valore negativo come parametro; il valore **-1** inoltre ha il significato speciale di indicare tutti i processi correnti eccetto **init** ed il processo che esegue il comando.

Un comando alternativo a **kill**, e più facile da usare, è **killall** che invece di richiedere un numero di PID funziona indicando il nome del programma, ed invia il segnale (specificato con la stessa sintassi di **kill**) a tutti i processi attivi con quel nome. Le opzioni principali disponibili sono riportate in tab. 1.15. Al solito la documentazione completa è nella pagina di manuale, accessibile con **man killall**.

Opzione	Significato
-g	invia il segnale al <i>process group</i> del processo.
-e	richiede una corrispondenza esatta anche per nomi molto lunghi (il comando controlla solo i primi 15 caratteri).
-i	chiede una conferma interattiva prima di inviare il segnale.
-l	stampa la lista dei nomi dei segnali .
-w	attende la terminazione di tutti i processi cui ha inviato il segnale.

Tabella 1.15: Principali opzioni del comando **killall**.

I segnali sono un meccanismo di comunicazione elementari fra processi, è cioè possibile utilizzarli per dare delle istruzioni (molto semplici, come quella di terminare l'esecuzione) ad un processo. Uno dei segnali più usati è ad esempio **SIGHUP**, che viene utilizzato per dire ai *demoni*⁵¹ di sistema di rileggere il proprio file di configurazione.

Alcuni segnali (come **SIGSTOP** e **SIGTERM**) sono associati al controllo del terminale e vengono inviati attraverso opportune combinazioni di tasti; torneremo su questo in sez. 1.3.4 quando affronteremo le questioni relative al controllo di sessione.

1.3.3 Priorità

Abbiamo visto in sez. 1.3.1 che una delle proprietà dei processi che si può modificare con **top** è la priorità. In realtà, come accennato in tale occasione, quello che di norma si può modificare non è tanto la priorità di un processo, quanto il suo valore di *nice*.

La gestione delle priorità in un sistema unix-like infatti è abbastanza complessa dato che esistono due tipi di priorità: statiche e dinamiche. Di norma le priorità statiche vengono utilizzate solo per i cosiddetti processi *real-time*,⁵² i processi ordinari (tutti, l'uso di processi *real-time* è riservato a pochi usi specialistici) hanno priorità statica nulla e il loro ordine di esecuzione viene stabilito solo in base alle priorità dinamiche.

Una priorità statica più alta comporta che un processo verrà sempre eseguito prima di ogni altro processo a priorità più bassa. Il che vuol dire che se si lancia un processo a priorità statica alta che non fa I/O non si potrà fare più nulla nel sistema fintanto che questo non si

⁵⁰quasi infallibile, infatti come accennato in sez. 1.3.1 non è possibile inviare segnali ad un processo in stato **D** perché non può riceverli, ed è inutile inviarli ad un processo in stato **Z** perché in realtà esso non esiste più.

⁵¹sono chiamati così i programmi che girano in *background* senza essere associati ad un terminale, che servono a fornire vari servizi, come ad esempio i server di rete, il sistema per il log e le esecuzioni periodiche dei comandi, e molti altri.

⁵²questo nome è in realtà fuorviante, Linux, almeno nella sua versione standard, non è un sistema operativo *real-time*. Se si ha necessità di usare un sistema effettivamente *real-time* occorre usare una versione del kernel opportunamente modificata come RTAI o RT-Linux.

sarà concluso (e se c'è un errore e il programma si blocca in un ciclo si dovrà necessariamente riavviare la macchina, non avendo modo di eseguire nient'altro). La cosa non vale quando il processo deve fare I/O perché in tal caso anche il processo real-time viene messo in stato di sleep, ed in quel momento altri processi possono essere eseguiti, lasciando così la possibilità di interromperlo. Se più processi hanno la stessa priorità statica l'ordine di esecuzione dipende dalla politica di *scheduling* scelta, che può essere di tipo *Round Robin*, in cui i processi girano a turno per un tempo fisso, e *First In First Out*, in cui vengono eseguiti nella sequenza in cui sono stati lanciati.

Nell'uso normale comunque non si ha bisogno di usare le priorità statiche (che come accennato sono anche molto rischiose in caso di errori di programmazione), e ci si affida al normale procedimento di *scheduling*, basato su un sistema di priorità dinamiche che permette di ottenere quella che usualmente viene chiamata la “*fairness*” nella distribuzione del tempo di CPU. Tralasciando i dettagli possiamo dire che le priorità dinamiche sono caratterizzate da un valore iniziale, che è quello che poi si chiama *nice*, che di default è nullo; più un processo viene eseguito, più il valore di priorità dinamica aumenta, e lo *scheduler*⁵³ mette sempre in esecuzione, fra tutti quelli in stato *runnable*, il processo che ha una priorità dinamica più bassa.

Questo fa sì che anche i processi che partono da un valore di *nice* più alto (che viene chiamato così appunto perché i processi che lo usano sono più “*gentili*” nei confronti degli altri) ottengano alla fine una possibilità di essere eseguiti, secondo quello che appunto è il meccanismo chiamato *fairness*.

Il comando che permette di modificare il valore di *nice* di un processo è appunto **nice**, che deve essere usato quando si lancia un programma, nella forma:

```
nice [OPTION] [COMMAND [ARG]...]
```

in cui si fa seguire a **nice** la linea di comando di cui si vuole cambiare la priorità. L'opzione principale è **-n** che permette di specificare un valore di *nice* da applicare al programma. Se non si specifica nulla viene applicato un valore di 10. Si ricordi che valori positivi corrispondono ad una diminuzione della priorità. L'amministratore può anche usare valori negativi, aumentando così la priorità. I valori possibili sono fra 19 e -20.

Il comando **nice** può essere usato solo quando si avvia un programma, se si vuole cambiare la priorità di un programma già in esecuzione si può usare il comando **renice**; questo ha una sintassi diversa, del tipo:

```
renice priority [[-p] pid ...] [[-g] pgrp ...] [[-u] user ...]
```

In questo caso la priorità si indica immediatamente come valore numerico e si può specificare il processo (o i processi) a cui applicare il cambiamento in tre modi diversi. Con l'opzione **-p** si può specificare un processo singolo, indicandone il PID; con l'opzione **-u** si possono selezionare tutti i processi di un singolo utente; infine con **-g** si possono indicare tutti i processi di uno stesso gruppo (che come vedremo in sez. 1.3.4 corrispondono ai comandi eseguiti su una stessa riga di shell) specificandone il *process group*.

Al solito si applica la restrizione che solo l'amministratore può applicare valori negativi, ed aumentare così la priorità di un processo. Si tenga conto inoltre che, a differenza di **nice**, i valori di **renice** sono relativi al valore di *nice* attuale. Pertanto una volta che si è diminuita la priorità di un processo aumentandone il valore di *nice* un utente normale non potrà tornare indietro.

⁵³si ricordi (vedi sez. 1.1.1) che lo *scheduler* è la parte di kernel che decide quale processo deve essere posto in esecuzione.

1.3.4 Sessioni di lavoro e *job control*

La gestione delle sessioni di lavoro è uno dei punti più oscuri della interfaccia a linea di comando di un sistema unix-like. Essa origina dagli albori del sistema, quando ci si poteva collegare solo attraverso un terminale, che era l'unica modalità di interazione con il sistema. Questo allora ha portato a tracciare una distinzione fra i processi interattivi (che quindi erano associati ad un terminale) e quelli non interattivi slegati da un terminale (che abbiamo già trovato nell'output di `ps`, nella colonna TTY). In genere questi ultimi sono tradizionalmente chiamati *demoni*, e sono programmi utilizzati dal sistema per compiere una serie di compiti di utilità, (come spedire la posta, eseguire lavori periodici, servire pagine web, ecc.) anche quando nessun utente è collegato ad un terminale. Per questo lavorano come suol dirsi “in *background*” e non hanno nessun terminale di riferimento.

Avendo a disposizione un solo terminale, se questo fosse stato occupato da un solo processo alla volta, non si sarebbero potute sfruttare le capacità di multitasking del sistema, per questo venne introdotta un'interfaccia, quella delle sessioni di lavoro e del *job control*, che permettesse di lanciare ed usare più processi attraverso un solo terminale. Oggi, con la possibilità di avere più console virtuali, e con l'interfaccia grafica che non fa riferimento ad un terminale, questo problema non c'è più, ma l'interfaccia è rimasta e mantiene comunque una sua utilità, ad esempio in presenza di una connessione diretta via modem in cui si ha a disposizione un solo terminale, e si devono eseguire compiti diversi.

Questo però ha comportato che la distinzione fra processi interattivi e non non sia più così semplice, non potendosi più basare sulla semplice presenza o meno di un terminale di controllo perché, con l'introduzione delle interfacce grafiche, si possono eseguire anche programmi con cui è possibile interagire, (attraverso di esse), senza che questi siano associati a nessun terminale. Comunque è sempre possibile identificare questi ultimi, in quanto discenderanno dal processo che gestisce l'accesso attraverso l'interfaccia grafica.

Si tenga comunque presente che dal punto di vista del kernel non esiste nessuna differenza fra processi interattivi e non, esso si limita a mettere a disposizione alcune risorse che possono essere utilizzate per realizzare il controllo di sessione, (in sostanza degli identificatori aggiuntivi come il *session id* ed il *process group id* e le funzioni per impostarne il valore, e l'invio dei segnali relativi alla gestione), che poi viene realizzato completamente in user space, secondo la modalità classica ereditata dai primi Unix.

Per capire meglio il significato di tutto questo, e della differenziazione fra processi interattivi e non, occorre illustrare una caratteristica fondamentale dell'interfaccia a riga di comando. La *shell* infatti, nel momento in cui lancia un programma, si cura sempre di automaticamente tre file, che sono immediatamente disponibili.⁵⁴ Essi sono rispettivamente lo *standard input*, lo *standard output* e lo *standard error*. Dato che questi sono i primi tre file aperti e lo sono in questa sequenza, ad essi vengono assegnati rispettivamente i *file descriptor* 0, 1 e 2.⁵⁵

Convenzionalmente⁵⁶ un programma legge il suo input dal primo file descriptor, scrive l'output sul secondo, e gli eventuali errori sul terzo. Quando un processo è interattivo (cioè è stato lanciato direttamente da una shell interattivamente) tutti e tre questi file sono associati al terminale su cui si stava operando, e l'interfaccia dei terminali (cioè quella dei dispositivi di questo tipo) fa sì che in lettura il terminale fornisca quanto scritto sulla tastiera e che quanto scritto sul terminale venga stampato sullo schermo.

La gestione delle sessioni di lavoro deriva direttamente dalla procedura di login su terminale.

⁵⁴torneremo su questo in sez. 2.1.5.

⁵⁵come accennato in sez. 1.2.2 il sistema mantiene anche una lista dei file aperti dai vari processi, ciascun processo a sua volta ha una lista dei file che ha aperto, il cui indice è appunto un numero chiamato *file descriptor*, per maggiori dettagli riguardo a questa problematica si può consultare la sezione 6.1 di [1].

⁵⁶si tenga presente che è solo una convenzione, si può tranquillamente scrivere un programma che si comporta in modo diverso, scrivendo e leggendo da un file qualunque.

Negli output di **ps** mostrati in sez. 1.3.1 si può notare come sia presente il processo **getty** associato ai sei terminali virtuali (da **tty1** a **tty6**) presenti sullo schermo. Questo è il processo che cura in generale la procedura di login, dando l'avvio ad una sessione di lavoro.

Esso stampa un messaggio di benvenuto preso dal file **/etc/issue** (vedi sez. 3.1.4). Poi stampa la linea “**login:** ” ed attende l'immissione sul terminale di un nome di login, che viene passato al programma **login** che si cura di chiedere la password ed effettuare l'autenticazione. Se questa ha successo è **login** che si incarica di impostare il valore del *session id* per il processo in corso, cambiare il proprietario dello stesso all'utente che si è collegato, e lanciare una shell di login.⁵⁷

A questo punto si ha a disposizione una riga di comando (torneremo in dettaglio sull'interfaccia a riga di comando in sez. 2.1) e tutti processi lanciati tramite la shell saranno identificati dallo stesso *session id*; le informazioni relative alla sessione possono essere visualizzate con l'opzione **-j** di **ps**, lanciando il comando dal terminale in cui si sta scrivendo queste dispense otteniamo:

```

piccardi@anarres:~$ ps -je f
  PID  PGID  SID TTY      STAT   TIME COMMAND
    1    0    0  ?        S      0:02 init [2]
...
 4857  4527  4527  ?        S      0:00 gnome-terminal
 4858  4527  4527  ?        S      0:00 \_ gnome-pty-helper
 4859  4859  4859 pts/1    S      0:00 \_ bash
...
 5936  5936  5936  ?        S      0:00 /usr/sbin/sshd
 8005  8005  8005  ?        S      0:00 \_ sshd: piccardi [priv]
 8007  8005  8005  ?        S      0:35 \_ sshd: piccardi@pts/2
 8009  8009  8009 pts/2    S      0:00 \_ -bash
 8013  8013  8009 pts/2    S      0:01 \_ xmms
 8014  8013  8009 pts/2    S      0:00 | \_ xmms
 8015  8013  8009 pts/2    S      0:00 | \_ xmms
 8016  8013  8009 pts/2    S      0:00 | \_ xmms
 8504  8013  8009 pts/2    S      0:00 | \_ xmms
 8505  8013  8009 pts/2    S      0:00 | \_ xmms
 8037  8037  8009 pts/2    S      0:00 \_ /usr/bin/perl -w /usr/bin
 8039  8037  8009 pts/2    S      0:03 | \_ xdvi.bin -name xdvi s
 8040  8037  8009 pts/2    S      0:02 | \_ gs -sDEVICE=x11 -
 8382  8382  8009 pts/2    S      1:07 \_ emacs struttura.tex
 8506  8506  8009 pts/2    R      0:00 \_ ps -je f
...

```

In questo caso si può notare come la shell sia stata ottenuta non da **getty** ma tramite una sessione di rete fatta partire da **sshd**, sul terminale virtuale **pts/2**. È inoltre presente un'altra sessione, associata al terminale virtuale **pts/1**, creata da **gnome-terminal** all'interno di una sessione X. Tutti i programmi che si sono lanciati dalla nostra shell (**xmms**, **xdvi**, **emacs** e lo stesso **ps**) hanno lo stesso SID.

La seconda caratteristica della gestione del job control è che quando si lancia una linea di comando inoltre tutti i processi avviati all'interno della stessa riga (nel caso le varie istanze di **xmms**, o i vari programmi avviati nell'esecuzione di **xdvi**) vengono assegnati allo *process group*, e identificati pertanto dallo stesso valore della colonna PGID.

Come si può notare tutti questi comandi fanno riferimento allo stesso terminale, e vengono eseguiti insieme; essi infatti sono stati lanciati in *background*. Questa è un'altra funzionalità della shell, che fa sì, quando si termina una linea di comando con il carattere **&**, che i processi

⁵⁷di nuovo il programma è sempre lo stesso, e gran parte delle distribuzioni usa come shell di default la **bash**, ma una shell può essere eseguita anche all'interno di uno script non interattivo, o lanciata come *sub-shell* all'interno di una sessione, e per distinguere questo tipo di situazioni, che richiedono comportamenti diversi, il comando viene lanciato con le opportune opzioni, che contraddistinguono poi quella che viene appunto chiamata una *shell di login* e una *shell interattiva*.

vengano mandati in esecuzione sospendendo momentaneamente l'accesso al terminale. Se, come accade per `xms`, `xdvi` e `emacs` il processo non ha bisogno del terminale (nel caso perché `sshd` esegue il *forwarding* la sessione X, per cui l'interazione avviene attraverso l'interfaccia grafica) questi continuano ad essere eseguiti senza accedere più al terminale. Quando però un processo in *background* dovesse tentare di effettuare una lettura o una scrittura sul terminale, verrebbe automaticamente inviato un segnale (rispettivamente `SIGTTIN` e `SIGTTOU`) a tutti i processi nello stesso *process group*, la cui azione di default è quella di fermare i processi. Così se ad esempio si fosse lanciato in background l'editor `jed` (che opera solo tramite terminale) questo si sarebbe immediatamente fermato, e alla successiva operazione sulla shell, avremmo avuto un avviso, con un qualcosa del tipo:

```
piccardi@anarres:~/gapil$ jed prova &
[3] 8657
piccardi@anarres:~/gapil$

[3]+  Stopped                  jed prova
```

Un altro modo di mandare un processo in background è attraverso l'uso del segnale `SIGSTOP`, che è associato alla combinazione di tasti `C-z`,⁵⁸ questo ferma il processo, che può essere fatto ripartire in background con il comando `bg`.⁵⁹ Oltre a `SIGSTOP`, quando un processo è associato ad un terminale di controllo, si possono mandare altri segnali con delle opportune combinazioni di tasti: `C-c` invia un `SIGINT` e `C-\` invia un `SIGQUIT`.

L'elenco dei processi in *background* può essere stampato con il comando `jobs`, che mostra la lista ed il relativo stato di ciascuno di essi. Un processo può essere riassociato al terminale (cioè messo in *foreground*) con il comando `fg`, questo può prendere come parametro sia il PID del processo (indicato direttamente) che il numero di job stampato da `jobs`, che deve essere preceduto da un carattere `"%"`, sintassi che può essere usata anche per il comando `kill`, quando questo è realizzato anch'esso come comando interno della shell (come avviene nel caso della `bash`).

1.4 Il controllo degli accessi

Come già detto e ripetuto più volte, Linux è nato come sistema multiutente e nella sua architettura è nativa la possibilità di avere utenti diversi che lavorano sulla stessa macchina. Questo ovviamente comporta la necessità di un meccanismo di controllo degli accessi, che permetta di restringere le capacità dei singoli in maniera che questi non possano recare danni (volontariamente o involontariamente che sia) agli altri o al sistema.

1.4.1 Utenti e gruppi

Essendo nato come sistema multiutente, ogni kernel di tipo unix-like come Linux deve fornire dei meccanismi di identificazione dei vari utenti sulla base dei quali poi possa venire imposto un adeguato controllo degli accessi e delle operazioni che questi possono fare.

La struttura di sicurezza tradizionale di un sistema unix-like è estremamente semplice, tanto da essere in certi casi considerata troppo primitiva⁶⁰ e prevede una distinzione fondamentale fra

⁵⁸ cioè l'uso del tasto *control* insieme alla lettera *z*, la notazione è formalizzata in sez. 2.4.2.

⁵⁹ attenzione, questo, come i seguenti `fg` e `jobs`, è un comando interno della shell, e non corrisponde a nessun eseguibile su disco, vedremo la differenza in sez. 2.1.3.

⁶⁰ sono previste estensioni specifiche, come quelle di SELinux, incorporate nel kernel a partire dalle versioni di sviluppo 2.5.x, che implementano il *Mandatory Access Control* e la capacità di definire in maniera molto più dettagliata i privilegi di accesso, sottoponendo anche l'amministratore ad ulteriori restrizioni; questo necessita ovviamente di un bel po' di lavoro amministrativo in più per cui non è detto che sia sempre utilizzata.

l'amministratore del sistema, tradizionalmente identificato dall'username **root**, per il quale non viene effettuato nessun controllo, e tutti gli altri utenti per i quali invece vengono effettuati vari controlli previsti per le operazioni che li richiedono.

In genere ogni utente è identificato con un numero, lo *user-ID* o *uid*. Questo è nullo per l'amministratore e diverso da zero per tutti gli altri utenti. I controlli vengono effettuati sulla base di questo numero, *se* è diverso da zero si verifica se corrisponde a quello per cui la richiesta è consentita, e nel caso si nega o concede l'accesso. Si noti il *se*, qualora l'*uid* sia nullo il controllo non viene neanche effettuato: è per questo che **root** è sempre in grado di compiere qualunque operazione.⁶¹

Per questo motivo tutti i sistemi unix-like prevedono una procedura di autenticazione che permette di riconoscere l'utente che si collega al sistema. Questa nella sua forma più elementare è fatta dal programma **login**, che richiede all'utente il nome che lo identifica di fronte al sistema (detto *username*) ed una password che ne permette di verificare l'identità (torneremo sulla gestione di questi aspetti in sez. 4.3).

Gli utenti poi possono venire raggruppati in *gruppi*, ogni gruppo ha un nome (detto *group name*) ed un relativo identificatore numerico, il *group-ID* o *gid*. Inoltre ogni utente è sempre associato almeno un *gruppo*, detto *gruppo di default*; di norma si fa sì che questo contenga solo l'utente in questione e abbia nome uguale all'username. Un utente può comunque appartenere a più gruppi, che possono così venire usati per permettere l'accesso ad una serie di risorse comuni agli utenti dello stesso gruppo.

Ogni processo, quando viene lanciato, eredita dal padre l'informazione relativa all'utente che lo ha creato (e a tutti i gruppi cui questo appartiene). In questo modo una volta entrati nel sistema tutti i programmi verranno eseguiti per conto dell'utente che ha effettuato il login. Il sistema può così provvedere al controllo degli accessi, e porre una serie di limitazioni a quello che l'utente può fare, impedendo l'esecuzione di tutte le operazioni non consentite; si evita così che utenti diversi possano danneggiarsi fra loro o danneggiare il sistema.

Il comando che permette di verificare chi è l'utente che lo sta eseguendo (identificando così le proprie credenziali) è **whoami**, che stampa il nome di login (comunemente detto *username*) con qualcosa del tipo:

```
piccardi@monk:~/truedoc/corso$ whoami
piccardi
```

Per verificare invece di quali gruppi si fa parte si può usare il comando **groups**; questo, invocato da un utente normale, non vuole parametri e stampa i gruppi cui questo appartiene, ad esempio:

```
piccardi@anarres:~/gapil$ groups
piccardi cdrom audio
```

se invece il comando viene usato dall'amministratore può prendere come parametro un username, nel qual caso stampa i gruppi cui appartiene detto utente. Entrambi i comandi non hanno opzioni se non quelle standard GNU che vedremo in sez. 2.2.1.

Infine il comando **id** permette di stampare in generale tutti i vari identificatori associati ad un processo, sia di gruppi che degli utenti, sia reali che effettivi ed in forma sia letterale che numerica. Il comando prende molte opzioni che permettono di specificare nei dettagli quali informazioni stampare, al solito si può fare riferimento alla pagina di manuale, accessibile con **man id**, per la documentazione completa.

⁶¹e per questo è da evitare assolutamente l'uso di **root** per qualunque compito che non sia strettamente connesso all'amministrazione del sistema.

1.4.2 I permessi dei file

Anche la gestione dei permessi dei file è tutto sommato piuttosto semplice; esistono estensioni⁶² che permettono di renderla più sottile e flessibile, ma nella maggior parte dei casi il controllo di accesso standard è più che sufficiente. In sostanza per il sistema esistono tre livelli di privilegi:

- i privilegi dell'utente (indicati con la lettera **u**, dall'inglese *user*).
- i privilegi del gruppo (indicati con la lettera **g**, dall'inglese *group*).
- i privilegi di tutti gli altri (indicati con la lettera **o**, dall'inglese *other*).

e tre permessi base:

- permesso di lettura (indicato con la lettera **r**, dall'inglese *read*).
- permesso di scrittura (indicato con la lettera **w**, dall'inglese *write*).
- permesso di esecuzione (indicato con la lettera **x**, dall'inglese *execute*).

il cui significato è abbastanza ovvio.

Ogni file è associato ad un utente, che è detto *proprietario* del file e ad un gruppo; per ciascuno dei tre livelli di privilegio (utente, gruppo e altri) è possibile specificare uno dei tre permessi; questi vengono riportati nella versione estesa dell'output del comando `ls`:

```
[piccardi@gont gapil]$ ls -l sources/
drwxr-sr-x  2 piccardi piccardi    128 Jan  4 00:46 CVS
-rw-r--r--  1 piccardi piccardi   3810 Sep 10 00:45 ElemDaytimeTCPClient.c
-rw-r--r--  1 piccardi piccardi   4553 Sep  9 19:39 ElemDaytimeTCPUncServ.c
-rw-r--r--  1 piccardi piccardi   3848 Sep 10 00:45 ElemDaytimeTCPServer.c
-rw-r--r--  1 piccardi piccardi   3913 Sep 10 00:45 ElemEchoTCPClient.c
-rw-r--r--  1 piccardi piccardi   4419 Sep  9 19:39 ElemEchoTCPServer.c
-rw-r--r--  1 piccardi piccardi   9534 Oct 14 17:05 ErrCode.c
-rw-r--r--  1 piccardi piccardi   4103 Oct 26 23:40 ForkTest.c
-rw-r--r--  1 piccardi piccardi   1052 Jan  1 12:53 Makefile
-rw-r--r--  1 piccardi piccardi   3013 Jan  4 00:44 ProcInfo.c
-rw-r--r--  1 piccardi piccardi   3904 Sep 10 00:45 SimpleEchoTCPClient.c
-rw-r--r--  1 piccardi piccardi   4409 Sep 10 00:45 SimpleEchoTCPServer.c
-rw-r--r--  1 piccardi piccardi   1748 Sep 10 00:45 SockRead.c
-rw-r--r--  1 piccardi piccardi   1678 Sep 10 00:45 SockWrite.c
-rw-r--r--  1 piccardi piccardi   2821 Jan  4 00:44 TestRen.c
-rwxr-xr-x  1 piccardi piccardi  28944 Jan  1 13:11 getparam
-rw-r--r--  1 piccardi piccardi   4416 Jan  4 00:44 getparam.c
-rw-r--r--  1 piccardi piccardi   3018 Jan  4 00:44 test_fopen.c
-rw-r--r--  1 piccardi piccardi   7404 Jun 10 2001 wrappers.h
```

che ci mostra nella prima colonna i permessi secondo lo schema riportato in fig. 1.4. La lettura dell'output di questo comando ci permette di vedere che i sorgenti dei programmi contenuti nella directory in oggetto hanno quelli che sono in genere i permessi di default per i file di dati, e cioè il permesso di scrittura solo per il proprietario, quello di lettura per tutti quanti (proprietario, gruppo e altri) e l'assenza del permesso di esecuzione.

Nel caso di un programma invece (come il `getparam` nell'esempio) i permessi di default sono diversi, ed il permesso di esecuzione è abilitato per tutti; lo stesso vale per le directory, dato che in questo caso il permesso di esecuzione ha il significato che è possibile attraversare la directory quando si scrive un pathname.

In generale un utente può effettuare su un file solo le azioni per le quali ha i permessi, questo comporta ad esempio che di default un utente può leggere i file di un altro ma non può modificarli o cancellarli; il proprietario di un file può sempre modificarne i permessi (con il comando `chmod`,

⁶²come le ACL (*Access Control List*), introdotte ufficialmente a partire dai kernel 2.5.x.

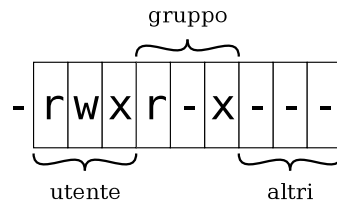


Figura 1.4: Legenda dei permessi dell'output di `ls`.

vedi sez. 1.4.4) per allargarli o restringerli (ad esempio i file della posta elettronica normalmente non hanno il permesso di lettura).

Al solito tutto questo vale per tutti gli utenti eccetto l'amministratore che non è soggetto a nessun tipo di restrizione e può eseguire qualunque operazione. In genere poi ogni distribuzione fa sì che tutti i file di configurazione ed i programmi installati nel sistema appartengano a `root`, cosicché diventa impossibile per un utente normale poter danneggiare (accidentalmente o meno) gli stessi⁶³.

1.4.3 I permessi speciali

Quelli illustrati in sez. 1.4.2 sono i permessi standard applicati ai file, esistono però altri tre *permessi speciali*. Ciascun permesso in realtà corrisponde ad un bit in una apposita parola mantenuta nell'inode (vedi sez. 1.2.2) del file. I bit usati per i permessi sono in realtà 12, i primi nove sono quelli già illustrati in fig. 1.4, gli altri tre vengono rispettivamente chiamati, dal loro nome *suid bit*, *sgid bit* e *sticky bit*.

Per i file normali tutti questi permessi hanno significato⁶⁴ solo al caso di programmi eseguibili.⁶⁵ I primi due servono per modificare il comportamento standard del sistema, che quando esegue un programma lo fa con i permessi dell'utente che lo ha lanciato. Per consentire l'uso di privilegi maggiori impostando il *suid bit* o lo *sgid bit* si consente al programma di essere eseguito rispettivamente con i privilegi dell'utente e del gruppo cui questo appartiene.

In questo modo si può far eseguire anche ad un utente normale dei compiti che nel caso generico sono riservati al solo amministratore. Questo ad esempio è il modo in cui funziona il comando `passwd`, che se usato da `root` non ha restrizioni, ma se usato da un utente normale consente di modificare la password, ma solo la propria e fornendo prima quella corrente. Per poter fare questo occorre comunque l'accesso in scrittura al file della password (proprietà di `root`) che viene garantito con l'uso del *suid bit*.

Lo *sticky bit* è al giorno d'oggi sostanzialmente inutilizzato, serviva nelle prime versioni di Unix, quando memoria e disco erano risorse molto scarse e gli algoritmi per la memoria virtuale poco raffinati, a privilegiare l'uso della *swap* mantenendovi permanentemente i programmi usati più di frequente. Ha assunto però, come vedremo a breve, un significato speciale per le directory.

La situazione completa dei bit associati ai permessi è illustrata in fig. 1.5. Data corrispondenza diretta con il contenuto dell'inode, e considerato il fatto che i gruppi di permessi sono raggruppati naturalmente a gruppi di tre bit, una notazione comune è quella di indicarli direttamente con il valore numerico di questa parola espressa in notazione ottale, così che ogni cifra corrisponde, a partire dalla meno significativa, ai permessi per tutti gli altri, per il gruppo, e per il proprietario. Per cui i permessi di fig. 1.5, corrispondenti ai valori già mostrati in fig. 1.4, si

⁶³ questa è una delle ragioni per cui i virus sono molto più difficili da fare con Linux: non essendo possibile ad un utente normale modificare i file di sistema, un virus potrà al più infettare i file di quell'utente, cosa che rende molto più difficile la sua diffusione.

⁶⁴ con l'eccezione del cosiddetto *mandatory locking*, una estensione ripresa da SysV, che viene attivato impostando lo *sgid bit* su un file non eseguibile.

⁶⁵ nel caso si cerchi di attivarli per uno script essi vengono comunque, come misura di sicurezza, disattivati.

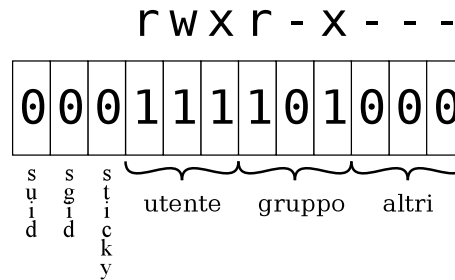


Figura 1.5: Schema dei bit dei permessi mantenuti nell'inode.

possono specificare direttamente con la cifra **640**. Volendo specificare uno dei permessi speciali occorrerà ovviamente usare anche una quarta cifra, secondo lo specchietto di fig. 1.5.

Al contrario di quanto avviene con i permessi normali, i permessi speciali non hanno a disposizione una posizione a parte nell'output di **ls**, ma quando attivati cambiano il valore della lettera associata al permesso di esecuzione. In particolare se il *suid bit* o lo *sgid bit* sono attivi verrà rispettivamente mostrata una **s**⁶⁶ nel permesso di esecuzione per utente e gruppo. Lo *sticky bit* invece modifica la lettera che indica il permesso di esecuzione per tutti in una **t**.⁶⁷

Finora abbiamo parlato di permessi applicati ai file normali, ma gli stessi permessi possono essere anche applicati ad una directory⁶⁸ nel qual caso essi vengono ad assumere un significato diverso. Se infatti è immediato dare un senso al leggere una directory (per mostrare i file che vi sono elencati), o allo scriverla (per aggiungere nuovi file e directory), non è immediato capire cosa possa significare il permesso di esecuzione.

Questo infatti viene ad assumere un ruolo particolare, ed indica che la directory può essere *attraversata* nella risoluzione del nome di un file. Se cioè manca il permesso di esecuzione, si potrà leggere il contenuto di una directory, ma non si potrà più accedere a quello delle eventuali sottodirectory, mentre nel caso opposto, pur non potendone mostrare il contenuto, si potrà accedere ai file che essa contiene (purché se ne conosca il nome, ovviamente).

Inoltre, come accennato in precedenza, lo *sticky bit* assume per le directory un significato particolare: se esso viene attivato pur in presenza di un permesso di scrittura per tutti, solo gli utenti proprietari di un file potranno cancellarlo; questo è ad esempio la modalità in cui viene protetta la directory **/tmp**, in cui tutti possono scrivere ma in cui, per la presenza dello *sticky bit*, gli utenti non possono cancellare i file creati dagli altri.

Infine per le directory l'uso dello *sgid bit* viene utilizzato per far sì che i nuovi file creati in una di esse abbiano come gruppo proprietario non quello del processo che li ha creati, ma quello che è proprietario della directory stessa. Questo permette al gruppo proprietario della directory di mantenere automaticamente la proprietà dei file in essa creati.

Si noti come a differenza di altri sistemi operativi (ad esempio il VMS) non esista un permesso specifico per cancellare un file. Questa è un'altra caratteristica, derivata dall'architettura dei file illustrata in sez. 1.2.2, che spesso lascia perplesso chi si accosta le prime volte ad un sistema unix-like, dato che sarebbe intuitivo associare la cancellazione di un file al suo permesso di scrittura.

Se però ci si ricorda cosa significa in realtà la cancellazione di un file si capisce subito che in effetti i permessi di un file non contano niente, dato che questa consiste solo nella rimozione di un riferimento *dalla directory*. Il che comporta che per cancellare un file tutto quello che serve

⁶⁶o una **S** qualora il corrispondente permesso di esecuzione non sia attivo.

⁶⁷che come per i precedenti diventa una **T** qualora il corrispondente permesso di esecuzione non sia attivo.

⁶⁸ed in generale a tutti gli altri file speciali presenti sul filesystem: hanno un significato speciale però solo per le directory; per *fifo* e file di dispositivo contano solo i permessi di lettura e scrittura, mentre per i link simbolici essi vengono ignorati.

è il permesso di scrittura *sulla directory* che lo contiene, dato che è questa che viene modificata; i permessi del file sono completamente ignorati.

1.4.4 La gestione dei permessi dei file

Inizieremo a trattare il problema della gestione dei permessi dei file, esaminando con qualche dettaglio in più sulle modalità con cui nuovi file e directory vengono creati. Il comportamento di default dei programmi infatti sarebbe quello di creare i nuovi file con i permessi di lettura e scrittura attivati per *tutti* dove con tutti qui si vuole intendere sia il proprietario, che il gruppo, che tutti gli altri, cioè un permesso numerico in forma ottale di **666**. Per le directory invece viene attivato anche il permesso di esecuzione, per cui il valore corrispondente sarebbe **777**.

Ovviamente lasciare aperti i permessi di scrittura a chiunque non è cosa molto saggia dal punto di vista della sicurezza. Per questo motivo il kernel mantiene anche, per ogni processo, una proprietà specifica, chiamata *umask*, che viene ereditata nella creazione di un processo figlio. La *umask* specifica una maschera di bit, nella stessa forma mostrata in fig. 1.5, che serve, nella creazione di un nuovo file o directory, a cancellare dai permessi ad esso assegnati tutti i bit in essa specificati.

In genere la *umask* viene impostata negli script eseguiti al *login* grazie al comando `umask`. Dato che il valore viene ereditato nella creazione dei processi figli è sufficiente farlo all'inizio perché quanto scelto sia mantenuto anche in seguito. Un utente può comunque modificare il valore di questo parametro invocando il comando `umask` specificando come argomento un nuovo valore⁶⁹ con qualcosa del tipo:

```
umask 027
```

Il valore dell'argomento di `umask` può essere specificato sia in forma numerica, come nel caso precedente, che in forma simbolica, nel qual caso si deve specificare quali permessi *conservare*. Se invocato senza parametri il comando permette di visualizzare il valore corrente, e con l'opzione `-S` si stampa il valore in forma simbolica anziché numerica. Di norma un utente personalizza questo valore negli script di avvio della shell (vedi sez. 2.1.6).

Allora se un processo ha una *umask* di **022** (il valore di default) significa che il permesso di scrittura per il gruppo e tutti gli altri saranno automaticamente cancellati alla creazione di un nuovo file o directory. Questo vuol dire che i nuovi file saranno creati con permessi **644** (tutti i permessi per il proprietario e sola lettura per il gruppo e gli altri), mentre le nuove directory saranno create con permessi **755** (di nuovo tutti i permessi per il proprietario ma solo lettura e attraversamento per il gruppo e gli altri). Se ad esempio si fosse voluto cancellare tutti i permessi per gli altri, si sarebbe dovuta usare una *umask* di **027**.

Il comando che permette di modificare direttamente i permessi di file e directory è `chmod`. Nella sua forma più elementare esso prende come parametri il valore (espresso in forma numerica ottale) dei permessi seguito dal nome (o dalla lista) del file. I permessi possono anche essere espressi in forma simbolica, usando le lettere elencate in sez. 1.4.2; in tal caso essi possono essere espressi nella forma:

```
[gruppo] [operatore] [permesso]
```

Il gruppo specifica a quale gruppo di permessi si fa riferimento, esso può essere `u` per indicare il proprietario (*user*), `g` per indicare il gruppo (*group*) e `o` per indicare gli altri (*others*), più il valore `a` che indica tutti quanti (da *all*), si possono specificare più gruppi usando più lettere.

L'operatore indica il tipo di modifica che si vuole effettuare, e può essere `+` per aggiungere un permesso, `-` per toglierlo, e `=` per impostarlo esattamente al valore fornito. Quest'ultimo può

⁶⁹si tenga presente che questo è un comando interno della shell (vedremo il significato di ciò in sez. 2.1.3), dato che l'invocazione di un altro programma non servirebbe a nulla, in quanto la *system call* che esegue l'operazione opera solo sul processo corrente.

essere **w** per la scrittura (*write*), **r** per la lettura *read* e **x** per l'esecuzione *execute*. Inoltre per gruppo e proprietario si può usare **s** per impostare i corrispondenti *sgid* e *suid*, mentre con **t** si può impostare lo *sticky*.

Così ad esempio **a+x** aggiunge il permesso di esecuzione per tutti (cosa di solito necessaria quando si scrive uno script), mentre **o-r** toglie il permesso di lettura per gli altri, **g+w** aggiunge il permesso di scrittura al gruppo e **u+s** attiva il *suid bit*. Si possono anche combinare più espressioni scrivendole di fila separate da virgole, così ad esempio con **g+w,o-r** si abilita la scrittura per il gruppo, e si disabilita la lettura per tutti gli altri.

Si tenga presente che un utente può cambiare i permessi solo dei file che gli appartengono, deve cioè esserne proprietario, il permesso di scrittura non basta. Inoltre quando si usa il programma con un link simbolico verranno cambiati i permessi del programma referenziato dal link e varranno le credenziali (utente e gruppo) di quest'ultimo.

Il comando supporta anche l'uso dell'opzione **-R** che, quando lo si applica ad una directory, esegue la modifica dei permessi ricorsivamente per tutto il tratto di albero dei file contenuto nella stessa. La versione GNU/Linux del comando supporta anche l'opzione **--reference=FILE** che permette di prendere i permessi di un altro file. Per la descrizione completa del comando e di tutte le opzioni al solito si faccia riferimento alla pagina di manuale accessibile con **man chmod**.

Oltre ai permessi può essere necessario anche cambiare proprietario e gruppo di un file (o di una directory). Per questo si usano rispettivamente i due comandi **chown** e **chgrp**; il primo cambia il proprietario ed il secondo il gruppo. Entrambi prendono come primo argomento l'utente (o il gruppo per **chgrp**) da assegnare, seguito dalla lista dei file su cui operare. Le opzioni sono simili a quelle di **chmod**, ed in particolare **-R** esegue dei cambiamenti ricorsivamente su tutto il contenuto di una directory e **--reference** usa i valori di un altro file. Inoltre **chown** supporta per il nome utente anche la sintassi **username.group** che permette di cambiare anche il gruppo.

Ovviamente assegnare utente e gruppo ad un file è una operazione privilegiata, si tenga presente che su Linux⁷⁰ un utente normale non può assegnare ad un altro utente i suoi file, e può soltanto cambiare solo il gruppo dei file che gli appartengono, ed assegnarli soltanto ad altri gruppi cui egli stesso appartiene. Solo l'amministratore ha la capacità piena di cambiare proprietario e gruppo di un file qualunque.

1.4.5 Altre operazioni privilegiate

Oltre all'accesso ai file esistono altre operazioni che devono sottostare al controllo di accesso. Una di queste è la possibilità di inviare segnali ad un processo, cosa che un utente può fare solo con i processi che appartengono a lui, restrizione che non vale (come tutte le altre) per l'amministratore che può inviare segnali a qualunque processo.

Altre operazioni privilegiate sono quelle che riguardano la rete (che tratteremo a partire da cap. 7). Solo l'amministratore può attivare e disattivare interfacce, modificare la tabella di instradamento dei pacchetti, applicare o rimuovere regole al sistema di *firewall*. All'amministratore inoltre è riservato l'allocazione delle prime 1024 porte usate dai protocolli UDP e TCP, per cui di norma soltanto lui è in grado di lanciare demoni che usano una di queste porte (dette per questo *riservate*) come un server web, di posta o un DNS.

Un'altra operazione riservata, come già visto in sez. 1.2.4, è il montaggio dei filesystem, anche se è possibile delegare in parte la possibilità di eseguire questa operazione agli utenti. Ma l'esecuzione generica del comando **mount** con parametri qualunque può essere effettuata soltanto dall'amministratore.

Infine soltanto l'amministratore è in grado di creare i file speciali relativi ai dispositivi,

⁷⁰questa è una caratteristica di sicurezza usata anche da BSD, ma che non è detto sia presente su tutti gli Unix.

cioè solo root può usare il comando `mknod` per creare un file di dispositivo. Questo prende come parametri il nome del file seguito da una lettera, che indica il tipo di file speciale, e può essere “p” per indicare una *fifo*, “b” per indicare un dispositivo a blocchi e “c” per indicare un dispositivo a caratteri. Qualora si crei un file di dispositivo (che sia a blocchi o a caratteri è lo stesso) devono poi essere specificati di seguito il *major number* ed il *minor number* ⁷¹ che lo identificano univocamente.⁷²

Si noti che creare un file di dispositivo è una azione diversa dall’accedere al dispositivo sottostante, cosa che invece è regolata dai permessi di quest’ultimo (come file). Pertanto se si è stati poco accorti e si è permesso agli utenti l’accesso in scrittura a `/dev/hda`, anche se questi non possono creare un file di dispositivo, potranno comunque scriverci sopra ed ad esempio saranno tranquillamente in grado di ripartizionare il disco.

Si tenga presente però che per un utente è comunque possibile creare una *fifo* usando il comando dedicato `mkfifo`, che prende come argomento il nome della stessa. Il comando supporta anche l’opzione `-m` che permette di specificare la maschera dei permessi (con la stessa sintassi di `chmod`) da applicare alla stessa.

Infine alcuni filesystem supportano delle operazioni speciali, che non rientrano nella normale gestione dei file, attraverso quelli che vengono chiamati gli *attributi*. Questi sono una serie di capacità aggiuntive, fornite dai filesystem che le supportano, che possono essere attivata o meno per ciascun file.⁷³ Alcuni di questi attributi (in particolare quelli che permettono di imporre delle speciali restrizioni all’accesso) possono essere impostati soltanto dall’amministratore, questo è il motivo per cui si è scelto di trattare questo argomento in questa sezione.

Facendo riferimento ad estensioni che non è detto siano presenti su tutti i filesystem⁷⁴ gli attributi dei file non possono essere visualizzati con il comando `ls`, per visualizzarli esiste un comando apposito, `lsattr`.

Il comando è analogo a `ls` sia per l’uso degli argomenti, che per l’uso delle opzioni `-R`, `-a` e `-d` che hanno lo stesso significato visto in tab. 1.3. Il comando stampa a video l’elenco dei file, preceduto dal relativo valore degli attributi speciali, pertanto se lo eseguiamo sui file di queste dispense otterremo qualcosa del tipo:

```
[piccardi@gont corso]$ lsattr *.tex
----- advadmin.tex
----- appendici.tex
----- config.tex
----- corso.tex
----- netadmin.tex
----- netbase.tex
----- netinter.tex
----- ordadmin.tex
----- ringraziamenti.tex
----- shell.tex
----- stradmin.tex
----- struttura.tex
```

⁷¹questi due numeri sono il meccanismo con cui storicamente vengono identificati i dispositivi all’interno del kernel (così come gli *inode* identificano un file); il nome sotto `/dev` è solo una etichetta, potrebbe essere qualunque (anche se poi molti script non funzionerebbero), quello che indica al kernel quale dispositivo usare quando si accede a quel file sono questi due numeri.

⁷²per una lista completa delle corrispondenze di questi numeri con i vari dispositivi si può fare riferimento al file `devices.txt` distribuito con i sorgenti del kernel nella directory `Documentation`.

⁷³come per tutte le altre proprietà di un file, anche la lista degli attributi attivi viene mantenuta all’interno dell’*inode*.

⁷⁴queste estensioni sono state introdotte con *ext2*, ma sono supportate anche dagli altri i filesystem più comuni usati con Linux, come *ext3* e *reiserfs*.

che ci mostra come nessun attributo speciale sia stato impostato.⁷⁵ Quando uno di essi è attivo questo viene indicato nell'elenco dalla presenza della lettera che lo identifica (vedi tab. 1.16) al posto del carattere “-”.

Il comando che permette di impostare gli attributi speciali è **chattr**, questo prende come primo argomento una stringa che identifica quali attributi attivare o disattivare, e come argomenti successivi una lista di file. Il comando supporta anche l'opzione **-R** per eseguire ricorsivamente le operazioni quando nella lista dei file si è inclusa una directory.

Attributo	Significato
A	blocca l'aggiornamento del tempo di ultimo accesso.
a	attiva il cosiddetto <i>append flag</i> , ⁷⁶ che consente la scrittura solo in coda al file, il contenuto corrente non può essere modificato. Solo l'amministratore può attivare o disattivare questo attributo.
c	attiva la compressione trasparente del contenuto del file.
D	richiede che il contenuto della directory sia salvato su disco in maniera sincrona.
d	esclude il file dalla registrazione dei dati necessari ad eseguire il backup con dump (vedi sez. 4.1.4).
E	segnala un errore nella compressione quando questa è stata attivata; l'attributo può essere solo letto da lsattr .
I	segnala che una directory viene indicizzata con gli <i>hash tree</i> (una funzionalità avanzata trattata in sez. 5.2.3); l'attributo può essere solo letto da lsattr .
i	attiva il cosiddetto <i>immutable flag</i> , il file non può essere cancellato o rinominato, non si possono creare hard link né modificarne il contenuto. ⁷⁷
j	richiede che tutti i contenuti del file siano prima scritti sul giornale (tratteremo il <i>journalling</i> dei filesystem in sez. 5.2.3). Solo l'amministratore può attivare o disattivare questo attributo.
s	richiede che quando un file viene cancellato tutti i blocchi che contenevano i suoi dati siano riscritti su disco azzerandone il contenuto. ⁷⁸
S	richiede che il contenuto del file sia salvato su disco in maniera sincrona.
u	richiede che alla cancellazione il contenuto del file sia salvato in modo da poterne consentire il recupero.
X	permette di accedere direttamente al contenuto di un file compresso disabilitando la decompressione trasparente.
Z	indica che un file compresso è in uno stato inconsistente (<i>dirty</i>); l'attributo può essere solo letto da lsattr .

Tabella 1.16: Gli attributi speciali dei file.

La stringa che permette di specificare quali attributi attivare o disattivare è analoga a quella usata con **chmod** per i permessi, ciascun attributo è identificato da una lettera, riportata in tab. 1.16,⁷⁹ per attivare un attributo occorrerà farla precedere dal carattere “+”, mentre se si vuole cancellare un attributo già impostato si dovrà usare il carattere “-”; come per **chmod** si possono specificare insieme anche più attributi.

⁷⁵questo è il caso normale, gli attributi speciali infatti vanno impostati esplicitamente, i file vengono sempre creati senza che nessuno di essi sia presente.

⁷⁶è cioè possibile aprire un file in scrittura soltanto se lo si fa nel cosiddetto *append mode*, per una trattazione di questo argomento si può consultare il cap. 6 di [1].

⁷⁷in sostanza, oltre ad impedirne la rimozione dalla directory in cui si trova è impossibile modificare sia il contenuto del file che dell'inode, pertanto anche tutte le altre caratteristiche del file (permessi, proprietario, ecc.) non possono essere modificate.

⁷⁸di default infatti il kernel si limita a marcare i blocchi come liberi, ma non cancella il precedente contenuto, che potrebbe pertanto essere riletto.

⁷⁹non si sono menzionati nella tabella alcuni attributi sperimentali, per i quali si rimanda alla lettura della pagina di manuale di **chattr**.

Come accennato alcuni attributi, che solo l'amministratore può impostare, permettono di utilizzare delle speciali restrizioni di accesso, ad esempio con “a” si attiva l'*append flag* che consente soltanto di aggiungere dati ad un file mentre il contenuto corrente non può essere modificato; questo può essere utilizzato per salvaguardare i file di log (vedi sez. 3.3.3) da eventuali modifiche accidentali o meno.⁸⁰

Un altro attributo che permette di attivare una protezione speciale è “i” che abilita l'*immutable flag* che finché è attivo impedisce qualunque tipo di modifica al file. E si noti questo vale anche per le operazioni richieste dall'amministratore stesso, un file immutabile potrà essere cancellato o modificato soltanto se prima ne viene rimosso il relativo attributo. Di nuovo questa funzionalità permette di proteggere file essenziali in modo da impedirne ogni forma di modifica.

Potremo allora attivare alcuni di questi attributi (ad esempio l'*append flag* per un file di log, o l'*immutable flag* per `/etc/fstab`), in questo caso basterà eseguire i comandi:

```
[root@gont corso]# chattr +a /var/log/auth.log
[root@gont corso]# chattr +i /etc/fstab
```

e potremo controllare lo stato degli attributi ottenendo che:

```
[root@gont corso]# lsattr /var/log/auth.log /etc/fstab
-----a----- /var/log/auth.log
-----i----- /etc/fstab
```

a questo punto si potrà anche verificare che non è più possibile modificare `/etc/fstab`:

```
[root@gont corso]# touch /etc/fstab
touch: cannot touch '/etc/fstab': Permission denied
```

nonostante il comando sia stato eseguito con privilegi di amministratore.

⁸⁰in genere i file di log sono la prima cosa che un intruso cerca di modificare, con la presenza di questo attributo questo gli viene impedito fintanto che l'attributo non viene rimosso; benché nelle operazioni ordinarie questo sia sempre possibile se si hanno i privilegi di amministratore, esistono delle modalità operative (dette *capabilities*) in cui è possibile cedere questa capacità e rendere pertanto impossibile una successiva rimozione dell'attributo.

Capitolo 2

La shell e i comandi

2.1 L'interfaccia a linea di comando.

I sistemi Unix nascono negli anni '70, ben prima della nascita delle interfacce grafiche, quando l'unico modo di interagire con il computer era attraverso dei terminali, se non addirittura delle semplici telescriventi. Per cui anche se oggi sono disponibili delle interfacce grafiche del tutto analoghe a quelle presenti in altri sistemi operativi nati in tempi più recenti, l'interfaccia a riga di comando resta di fondamentale importanza, dato che 30 anni di storia e migliaia di persone che ci han lavorato sopra per migliorarla, la hanno resa la più potente e flessibile interfaccia utente disponibile.

2.1.1 La filosofia progettuale

Come per il sistema, anche le caratteristiche dell'interfaccia a riga di comando derivano da alcune scelte progettuali precise. Arnold Robbins spiega molto chiaramente questa filosofia in un articolo riportato anche nella pagina *info* (vedi sez. 2.3.1) del pacchetto dei **coreutils** GNU. In sostanza la filosofia progettuale della shell e dei comandi a riga di comando si può capire facendo ricorso ad una analogia, che riprenderemo da quell'articolo.

Molte persone utilizzano un coltellino svizzero, dato che questo permette di avere in solo oggetto un discreto insieme di attrezzi diversi: coltello, forbici, cacciavite, seghetto, cavatappi. Però è molto difficile vedere un professionista usare il coltellino svizzero per il suo lavoro. Un professionista ha bisogno di attrezzi professionali, e un carpentiere non costruisce una casa con un coltellino svizzero, ma con tanti attrezzi ciascuno dei quali è specializzato nello svolgere un compito specifico.

Le persone che han progettato l'interfaccia a riga di comando erano appunto dei professionisti, che sapevano bene che anche se fare un programma unico per tutti i compiti poteva essere attraente per l'utente finale, che deve conoscere solo quello, in pratica questo sarebbe stato difficile da scrivere, mantenere e soprattutto estendere. Per cui da professionisti pensarono ai programmi come a degli attrezzi, e piuttosto che il coltellino svizzero realizzarono l'equivalente della cassetta degli attrezzi (la “*Unix toolbox*”), con in testa un criterio fondamentale: che ciascun programma facesse una sola cosa, nel miglior modo possibile.

Questa è la caratteristica fondamentale dei programmi base di un sistema unix-like come GNU/Linux. Ogni comando¹ è progettato per eseguire un compito preciso: **ls** mostra la lista dei file, **ps** la lista dei processi, **cp** copia un file, **chmod** cambia i permessi, **man** mostra le pagine di manuale, ecc. I comandi hanno uno scopo preciso e precise funzionalità; le opzioni sono limitate

¹ne abbiamo incontrati già diversi nel corso della trattazione delle caratteristiche del sistema in cap. 1, ne faremo una trattazione sistematica fra breve.

e comunque specifiche allo scopo del comando, e sono descritte dettagliatamente nella relativa pagina di manuale.

Il passo successivo fu quello di costruire anche un meccanismo che permettesse di combinare insieme i vari programmi, cosicché divenisse possibile eseguire, con una opportuna combinazione, anche dei compiti che nessuno di essi era in grado di fare da solo. Questo aveva il grande vantaggio, rispetto all'approccio del programma universale, di non dover attendere che l'autore dello stesso si decidesse a programmare la funzione in più che serviva e che non era stata prevista all'inizio.

Questo è il ruolo della *shell*, cioè del programma che implementa l'interfaccia a riga di comando; è attraverso di essa che, concatenando vari comandi, si può costruire l'equivalente di una catena di montaggio, in cui il risultato di un comando viene inviato al successivo, riuscendo a compiere compiti complessi con grande velocità e flessibilità, e spesso fare anche cose che gli autori dei singoli programmi neanche si sarebbero immaginati.

2.1.2 Le principali shell

La modalità tradizionale con cui si utilizza l'interfaccia a riga di comando è, come accennato in sez. 1.3.4 quando un utente, una volta completata la procedura di autenticazione, inizia una sessione di lavoro su un terminale. Questo significa semplicemente che il programma **login** conclude il suo compito lanciando la *shell* assegnata all'utente (che come vedremo in sez. 4.3.3 è specificata dall'ultimo campo di `/etc/passwd`). Oggi con le interfacce grafiche si hanno molte altre modalità di accesso ad un terminale (ad esempio attraverso una finestra che contiene un terminale virtuale), in ogni caso, una volta predisposta l'opportuna interfaccia di accesso, verrà comunque lanciata una shell.

Si ricordi comunque che per il kernel, secondo la filosofia fondamentale di Unix illustrata in sez. 1.1.1, la *shell* resta un programma come tutti gli altri; essa ha però un compito fondamentale, che è quello di fornire l'interfaccia che permette di lanciare altri programmi. Inoltre è sempre la shell che permette di usufruire di tutta una serie di ulteriori funzionalità messe a disposizione dal kernel, come il controllo di sessione visto in sez. 1.3.4.

Dato che la shell è un programma come gli altri, essa può essere realizzata in diversi modi, ed in effetti nel tempo sono state realizzate diverse shell. Anche in questo caso ci sono stati due filoni di sviluppo, il primo deriva dalla prima shell creata, la *Bourne shell*, chiamata così dal nome del suo creatore. La *Bourne shell* è la shell più antica e le sue funzionalità sono anche state standardizzate dallo standard POSIX.2. Il secondo filone deriva da un'altra shell, realizzata con una sintassi alternativa, più simile a quella del linguaggio C, e chiamata per questo *C shell*.

Ciascuno di questi due filoni ha dato vita a varie versioni con funzionalità più o meno avanzate; un breve elenco delle varie shell disponibili anche su GNU/Linux è il seguente:

- *Bourne shell e derivate.*

- **La Bourne shell.** La prima shell di Unix, in genere utilizzata semplicemente con il comando **sh**. Non viene praticamente più usata. In GNU/Linux è sostituita da **bash**² o da **ash**. Sugli altri sistemi che rispettano lo standard POSIX, è di norma sostituita da **ksh**.
- **La Bourne-Again SHell.** La *bash* è la shell di riferimento del progetto GNU. Il suo nome è un gioco di parole sul nome della *Bourne shell*, in sostanza una shell *rinata*. Viene utilizzata con il comando **bash**. Incorpora molte funzionalità avanzate, come la storia dei comandi (detta *history*), l'auto-completamento dell'input sulla linea di comando (per comandi, nomi di file e qualunque altra cosa, date le opportune

²che quando viene invocata come **sh** fornisce esclusivamente le funzionalità previste dallo standard POSIX.2, disabilitando le varie estensioni di cui è dotata.

estensioni), editing di linea, costrutti di programmazione complessi e molto altro (praticamente di tutto, si vocifera sia anche in grado di fare il caffè).

- **La Korn Shell** La Korn shell (dal nome dell'autore) è stata la prima ad introdurre la history (l'accesso ai comandi precedenti) e l'editing della linea di comando. Ha il grosso difetto che gran parte delle funzionalità avanzate non vengono attivate di default, per cui occorre un ulteriore lavoro di configurazione per utilizzarla al meglio. Viene utilizzata con il comando `ksh`. Non viene usata su GNU/Linux dato che `bash` ne ha tutte le caratteristiche; è però utile conoscerne l'esistenza dato che è facile trovarla su altri Unix.
- **La ash.** Una shell *minimale*, realizzata in poche decine di kilobyte di codice sorgente. Viene utilizzata con il comando `ash`. Ha molti comandi integrati, occupa poca RAM e poco spazio disco, ed ha poche funzioni (ma è conforme allo standard POSIX.2). Viene usata spesso nei dischetti di installazione o recupero, può essere utile per sistemi dove si fa un grosso uso di script perché è più veloce di `bash`.
- **La Z shell.** Un'altra shell avanzata. Viene utilizzata con il comando `zsh`. Offre praticamente le stesse funzioni della Korn shell, ed altre funzionalità avanzate, come il completamento di comandi, file e argomenti, che però trovate anche nella `bash`.

- ***C shell e derivate.***

- **La C shell.** Utilizza una sintassi analoga a quella del linguaggio C. Viene utilizzata con il comando `csh`. In GNU/Linux non è disponibile essendo sostituita da `tcsh`.
- **La tcsh.** È una evoluzione della *C shell*, alla quale aggiunge history e editing di linea e varie funzionalità avanzate. Viene utilizzata con il comando `tcsh`. Si trova su vari Unix proprietari, ma è poco diffusa su GNU/Linux, pur essendo disponibile.

Dato che è il principale strumento di lavoro di un amministratore professionista, la scelta della shell è spesso una questione strettamente personale. Qui parleremo però solo di `bash`, che è la shell utilizzata in praticamente tutte le distribuzioni di GNU/Linux, e probabilmente è anche la più potente e flessibile fra quelle disponibili. L'unico motivo per volerne usare un'altra infatti è solo perché siete maggiormente pratici con quella, nel qual caso probabilmente non avete bisogno di leggere questo capitolo.

Il riferimento più immediato per il funzionamento della `bash` è la sua pagina di manuale, accessibile al solito con `man bash`. Probabilmente questa è la più lunga fra tutte le pagine di manuale.³ Per questo in seguito faremo riferimento, quando necessario, alle varie sezioni in cui essa è divisa. Per le funzionalità più avanzate esiste anche un ottimo manuale libero [2], tradotto pure in italiano.

2.1.3 Introduzione alla sintassi della riga di comando

Come accennato lo scopo della shell è quello di implementare l'interfaccia a riga di comando, cioè mettere a disposizione dell'utente un meccanismo con cui questi possa essere in grado di mettere in esecuzione i vari programmi che vuole utilizzare. Pertanto il compito principale della shell è quello di leggere una riga di comando dalla tastiera, riconoscere qual è il programma che volete mettere in esecuzione, ricostruire i vari argomenti da passare al programma stesso, e poi eseguirlo. La forma generica di un qualunque comando è nella forma:

```
comando -o ption argomento1 --altra-opzione argomento2 --riopzione=valore
```

³sul mio sistema conta la bellezza di 5266 righe, ed in effetti più che una pagina è un manuale!

da scrivere sulla stessa riga⁴ e terminare con la pressione del tasto di invio.

La sintassi prevede cioè che si scriva sempre all'inizio della riga (eventuali spazi antistanti verranno comunque ignorati) il nome del comando da eseguire, seguito da opzioni ed argomenti. Si tenga presente che la shell, per identificare il comando e separarlo da opzioni e argomenti (e separare questi ultimi fra di loro) usa caratteri vuoti come lo spazio o il tabulatore. La presenza cioè di uno (o più) spazi o tabulatori⁵ dice che si sta passando dal nome del comando ad un argomento o opzione, o da un argomento/opzione al successivo.

Per questo il nome di un comando non può contenere spazi, inoltre è la stragrande maggioranza dei comandi usa nomi scritti esclusivamente in lettere minuscole (si ricordi che un sistema unix-like è case-sensitive), i soli caratteri non alfabetici utilizzati (molto poco) sono il “-” o il “_” ed eventualmente i numeri, gli altri infatti, come vedremo fra breve, vengono interpretati dalla shell assumendo significati speciali, e pertanto di nuovo non possono essere inseriti nel nome del comando.

La differenza fra opzioni e argomenti è che le prime attivano funzionalità o modalità di operazione specifiche del comando, mentre i secondi indicano gli oggetti (solitamente dei file) su cui questo opera. In realtà il meccanismo è un po' più complesso, e può essere compreso nei dettagli solo con un po' di nozioni di programmazione,⁶ quello che in effetti fa la shell è identificare il programma da usare sulla base del nome del comando, e poi spezzare l'intera riga in una lista di stringhe che verranno passate al suddetto programma come argomenti iniziali. Questa scansione viene eseguita su tutta la riga, utilizzando gli spazi vuoti come separatori e senza distinguere fra nome del comando, opzioni e argomenti; tutto il contenuto sarà suddiviso in una lista di stringhe, che poi programma dovrà utilizzare al suo interno. Si può inoltre dire alla shell di utilizzare come separatore caratteri diversi dagli spazi vuoti, indicando quelli da usare con la variabile (tratteremo le variabili di shell più avanti) `IFS`.

Gran parte dei programmi per gestire gli argomenti e le opzioni passate dalla linea di comando utilizzano una serie di funzioni fornite dalla libreria di fondamentale del sistema, la *GNU C library* di fig. 1.1.⁷ Questo comporta, per i programmi che usano queste funzionalità, la presenza di un formato della riga di comando abbastanza uniforme, mostrato appunto nell'esempio precedente, con la possibilità di distinguere fra opzioni e argomenti, e di specificare questi in un ordine qualunque. Si tenga presente però che questo non è sempre vero, dato che non tutti i programmi fanno ricorso a queste funzionalità, per cui troveremo anche in seguito svariate eccezioni (ad esempio in cui l'ordine conta, e le opzioni vanno specificate prima degli argomenti).

Nel comportamento comune comunque le opzioni sono sempre costituite da un carattere “-” seguito da una lettera (in rari casi anche numeri o caratteri non alfabetici), cui può seguire (separato o meno da uno spazio) un eventuale valore passato come parametro (non tutte le opzioni lo prevedono, per alcune va specificata solo la lettera dell'opzione). Una forma alternativa per le opzioni, in uso principalmente con i programmi del progetto GNU che utilizzano le funzionalità citate e anch'essa mostrata nell'esempio precedente, è quella in cui l'opzione è scritta in forma estesa, e allora viene introdotta da un “--”, in questo caso, qualora essa necessiti di un

⁴il terminale in genere va a capo da solo o fa scorrere la riga se lo spazio disponibile sullo schermo finisce, si può però scrivere un comando su più righe utilizzando il carattere “\” per proteggere la pressione del tasto di invio (che altrimenti passerebbe la linea di comando alla shell) e andare su una nuova riga senza che la linea di comando venga ad essere interrotta.

⁵qualora si vada a capo su una nuova riga con l'uso della “\” illustrato nella nota precedente, anche il carattere di a capo viene considerato come uno spazio vuoto.

⁶per i curiosi questo argomento è trattato nei dettagli nella sezione 2.3 di [1].

⁷la funzionalità si chiama `getopt`, e permette l'uso di una sintassi standard per le opzioni, la scansione automatica delle stesse indipendentemente dall'ordine in cui le si specificano, la possibilità di identificare opzioni con parametri o senza, la capacità di passare al programma, una volta terminata la scansione della lista di stringhe ottenuta dalla shell, solo quelle contenenti gli argomenti restanti; per una trattazione più dettagliata si può fare riferimento alla sez. 2.3.2 di [1].

parametro, il relativo valore deve essere assegnato con l'uso di un "=" (torneremo su questo in sez. 2.2.1).

Così se per esempio scrivete sul vostro terminale una riga di comando contenente un qualcosa del tipo:

```
piccardi@anarres:~$ rm -f pippo pluto paperino
```

la shell capirà che volete invocare il comando `rm` ed individuerà il file che contiene il relativo programma su disco e lo lancerà passandogli l'intera riga, spezzata nelle cinque stringhe delimitate dagli spazi. La prima stringa contiene il nome del programma e di norma viene ignorata,⁸ le restanti invece verranno analizzate dal programma; così `-f` verrà identificata come opzione mentre, non necessitando questa di nessun parametro, `pippo`, `pluto` e `paperino` verranno considerati come argomenti, che nel caso indicano i file da cancellare.

Si tenga conto poi che benché la filosofia di Unix sia quella di utilizzare un apposito comando per effettuare ciascun compito specifico, e che il caso più comune sia quello appena illustrato in cui la shell interpreta la riga di comando per lanciare il programma che le avete chiesto, esistono alcune funzionalità che essa vi fornisce direttamente tramite alcuni *comandi interni* (detti anche *built-in*); in questo caso la shell si accorgerà che non è necessario eseguire nessun programma a parte ed opererà direttamente sulla base di quanto dite sulla linea di comando. Di norma i comandi interni sono quelli che servono a eseguire impostazioni relative al funzionamento stesso della shell, o a impostare proprietà generali che essa deve trasmettere ai programmi che essa lancia.

Un esempio classico di questi comandi interni è `cd`; questo comando, che serve a modificare la directory di lavoro corrente della shell, non può essere eseguito con un programma esterno, dato che nell'esecuzione quest'ultimo potrebbe solo cambiare la directory di lavoro corrente per sé stesso, ma non quella della shell. Alcuni di questi comandi interni, ad esempio quelli relativi al controllo di sessione, li abbiamo già incontrati in sez. 1.3.4, altri li vedremo più avanti.

Infine occorre sottolineare che quanto appena detto illustra solo il caso più elementare della sintassi usata dalla shell, quello in cui volete lanciare un comando scrivendo direttamente quest'ultimo con le relative opzioni ed argomenti. Vedremo nel corso delle successive sezioni, in cui introdurremo le altre funzionalità della shell, come in realtà sulla linea di comando si possano effettuare anche altre operazioni, e come il suo utilizzo possa essere modificato attraverso l'uso di opportuni caratteri di controllo.

2.1.4 Funzionalità interne della shell

Oltre a quello generico di lanciare programmi, già l'esempio precedente con `rm` ci mostra come la shell esegua di per sé anche molti altri compiti. Il primo che prenderemo in esame è quello della visualizzazione del *prompt*, cioè di quella scritta che compare sulla sinistra della linea di comando, a fianco della quale, quando non avete scritto nulla, lampeggia il cursore e che serve ad avvisarvi che la shell è in attesa di ricevere dei comandi da eseguire.

Nel caso della `bash` il *prompt* è completamente personalizzabile, e può contenere diverse informazioni. Quello che viene stampato come *prompt* è stabilito da una variabile di shell `PS1`,⁹ (tratteremo delle variabili di shell poco più avanti) nel cui contenuto, oltre ai normali caratteri che saranno stampati inalterati, si possono inserire una serie di caratteri di controllo (i principali dei quali sono riportati in tab. 2.1) che verranno automaticamente *espansi* in un determinato valore

⁸alcuni programmi invece usano questa stringa per modificare il loro comportamento, ad esempio `gzip`, un compressore di file, può essere invocato anche come `gunzip` (in genere entrambi i comandi sono presenti come hard link allo stesso file eseguibile) nel qual caso decomprimerà invece di comprimere.

⁹in realtà in tutto le variabili che controllano l'aspetto del prompt sono 4, ed oltre `PS1` ci sono anche `PS2`, `PS3` e `PS4`; dettagliare il loro scopo va oltre le possibilità di questa introduzione, ma la loro descrizione si trova nella pagina di manuale, nella sezione `Shell Variables`.

(come data, utente, stazione, ecc.). L'elenco completo è disponibile nella pagina di manuale, alla sezione **PROMPTING**.

Opzione	Significato
\d	la data in formato tipo: Tue May 26.
\H	in nome della stazione.
\u	lo <i>username</i> dell'utente.
\w	il path completo della directory di lavoro.
\W	il nome della directory di lavoro.
\\$	un # per l'amministratore, un \$ per gli altri.
!	la posizione nella history del comando.
\t	il tempo corrente in formato HH:MM:SS (24h).
\T	il tempo corrente in formato HH:MM:SS (12h).

Tabella 2.1: Principali caratteri di controllo usati nella visualizzazione del prompt della shell.

Due esempi possibili per il prompt sono quello usato da Debian, che prevede per **PS1** un valore pari a `\u@\h:\w\$`, e che produce un prompt del tipo:

```
piccardi@anarres:~/Truelite/documentazione/corso$
```

o quello di RedHat, che usa un valore di `[\u@\h \W]\$`, e produce un prompt del tipo:

```
[root@gont piccardi]#
```

La scelta del prompt dipende dai gusti e dall'uso: se vi trovate a lavorare contemporaneamente con diversi utenti, su computer diversi, allora vi sarà utile sapere con quale utente state operando in un certo terminale; altrimenti potete farne a meno e risparmiare spazio sulla linea di comando con un valore di **PS1** come `"\$"`.

Come ripetuto più volte la vera funzione della shell è quella di semplificarvi la vita; per questo al di là dell'estetica del prompt, essa fornisce una lunga serie di funzionalità generiche che permettono di rendere più efficace l'uso dei comandi. Una delle funzionalità fondamentali è quella delle *variabili di shell*. La shell vi permette cioè di definire e modificare delle variabili, in questo caso la sintassi della riga di comando cambia e potrete utilizzare una assegnazione del tipo:

```
VARIABILE=valore
```

e si noti però che in questo caso non ci sono più spazi a separare il nome della variabile dal suo valore, qualora infatti si mettessero degli spazi la shell interpreterebbe **VARIABILE** come il nome di un comando, e probabilmente riporterebbe un errore non trovando un programma corrispondente.

Per convenzione le variabili di shell si scrivono con lettere maiuscole, ma è appunto solo una convenzione, dovuta al fatto che così è più difficile confonderle con i comandi, ma in realtà è possibile usare qualunque tipo di lettera ed, eccezion fatta per l'iniziale del nome, anche qualunque numero ed il carattere `"_"`.

L'elenco delle variabili di shell già definite (vedremo in sez. 2.1.6 come effettuarne l'impostazione al login o quando si lancia un terminale) si può ottenere con il comando **set**,¹⁰ che ne stampa a video la lista coi relativi valori, mentre per cancellarne una definita in precedenza si può usare il comando¹¹ **unset** seguito dal nome della variabile. La modifica si effettua semplicemente assegnando alla variabile il nuovo valore.

¹⁰in realtà oltre alle variabili **set** stampa anche tutte le funzioni definite nella shell stessa; questo comporta che in certi casi l'output può essere molto lungo ed essendo le variabili stampate per prime si rischia di non riuscire a vederle; si tenga inoltre presente che questo comando viene anche usato, quando invocato con le opportune opzioni, per impostare una lunga serie di altre proprietà della shell; per l'elenco completo di quest'ultime si può fare riferimento alla sua descrizione nella sezione **SHELL BUILTIN COMMANDS** della pagina di manuale.

¹¹anche questo, come il precedente è un comando interno alla shell.

La caratteristica delle variabili di shell è che una volta definite è possibile recuperarne il valore precedendone il nome con il carattere “\$”, così se nella variabile `MAIL` mettete la directory in cui si trova la vostra posta elettronica, potrete guardare il contenuto della directory con un comando del tipo:

```
[piccardi@gont piccardi]$ ls -l $MAIL
-rw-rw----    1 piccardi mail          4136 Aug 25 17:30 /var/mail/piccardi
```

in genere cioè potrete utilizzare qualunque variabile in una linea di comando referenziandola antepoendo un “\$” al suo nome, in tal caso infatti la shell nell’eseguire la scansione della linea sostituirà automaticamente alla variabile il suo contenuto, e passerà questo come argomento (o come opzione, a seconda del valore) al comando che viene eseguito.

Si noti che mentre la shell fornisce la sintassi e gli opportuni comandi interni per assegnare e cancellare la singola variabile, altrettanto non accade per la visualizzazione della stessa. Questo avviene perché questo compito può essere eseguito attraverso l’uso di uno dei tanti comandi specialistici della *Unix toolbox*. In particolare il comando da usare è `echo`, il cui solo compito è stampare in uscita la stringa (o le stringhe) che gli vengono passate come argomento: siccome la shell esegue automaticamente l’espansione delle variabili prima di passare gli argomenti ad un comando, si potrà usare `echo` per leggere il contenuto di una variabile con una riga del tipo:

```
piccardi@anarres:~/Truelite/documentazione/corso$ echo $USER
piccardi
```

dove la variabile `USER` viene prima *espansa* dalla shell nella stringa `piccardi`, dopo di che quest’ultima viene passata come argomento ad `echo`, che, come avrebbe fatto per una qualunque altra stringa che potreste aver scritto direttamente sulla linea di comando, la stamperà a video.

Il comando `echo` è uno dei più elementari e prende due sole opzioni. La prima è `-n`, che evita la stampa del carattere di a capo alla fine della stringa passata come argomento. Questa è utile quando si vuole costruire una riga con più invocazioni e la si usa in genere all’interno di uno *script* (accenneremo agli script in sez. 2.1.6), dato che sul terminale si avrebbe l’interferenza da parte del prompt. La seconda opzione è `-e` che attiva l’interpretazione di una serie di caratteri speciali, la cui espressione, insieme a tutti gli altri dettagli relativi al comando, si trova nella relativa pagina di manuale, accessibile con `man echo`.

La `bash` definisce di suo (o utilizza qualora siano già definite) tutta una serie di variabili (come `IFS`, che abbiamo già incontrato), che permettono sia di controllarne il funzionamento che di accedere ad una serie di informazioni. Un elenco delle principali è riportato in tab. 2.2, l’elenco completo è descritto nella sezione **Shell Variables** della pagina di manuale.

Una delle altre funzionalità che la shell fornisce con l’uso delle variabili, è quella dell’*ambiente* (in inglese *environment*), esso viene utilizzato per poter utilizzare il valore di alcune variabili di shell anche all’interno dei programmi che questa mette in esecuzione.¹² Non tutte le variabili definite nella shell sono però inserite nell’*ambiente* in quanto molte di esse (come `PS1`) sono di interesse esclusivo della shell. Se si vuole inserire una variabile nell’ambiente lo si deve fare esplicitamente usando il comando interno `export` seguito dal nome della variabile, che deve essere già definita.¹³ Lo stesso risultato si può ottenere con l’uso del comando interno `declare`,

¹²in sostanza quando un programma quando viene lanciato con la opportuna chiamata al sistema (l’argomento è trattato nel capitolo due di [1]), deve ricevere una serie di informazioni da chi lo lancia; una parte di queste informazioni sono gli argomenti, che vengono presi direttamente da quanto scritto nella riga di comando con il meccanismo illustrato in sez. 2.1.3, l’altra parte sono appunto le variabili di ambiente, che vengono definite appunto nell’*environment*.

¹³la `bash` supporta anche la definizione della variabile nella stessa linea in cui viene esportata nell’ambiente, con una sintassi del tipo `export VAR=val`; questa però è una estensione che non è supportata da altre shell per cui quando si scrivono script è il caso di evitare questa sintassi per mantenere la compatibilità.

Variabile	Significato
HOSTNAME	Il nome della macchina.
OSTYPE	La descrizione del sistema operativo corrente.
PWD	La directory di lavoro corrente.
GLOBIGNORE	Una lista, separata da “:”, di nomi di file da ignorare nel <i>filename globbing</i> .
HISTFILE	Il file in cui viene memorizzata la storia dei comandi.
HISTFILESIZE	Il massimo numero di linee da mantenere nel file della storia dei comandi.
HISTSIZE	Il numero di comandi da mantenere nella storia dei comandi.
HOME	La home directory dell’utente.
IFS	Il carattere che separa gli argomenti sulla linea di comando.
PATH	La lista delle directory in cui si trovano i comandi.

Tabella 2.2: Principali variabili di shell.

che serve in generale ad assegnare variabili e funzioni (tratteremo queste ultime in sez. 2.1.6) o impostarne gli attributi. Il comando prevede diverse opzioni, riportate in tab. 2.3, ma con l’uso dell’opzione `-x` è del tutto equivalente ad `export`. Per rimuovere una variabile di ambiente si utilizza sempre `unset`.

Opzione	Significato
<code>-a</code>	dichiara una variabile vettoriale.
<code>-f</code>	stampa solo le funzioni.
<code>-i</code>	dichiara una variabile intera.
<code>-r</code>	rende variabile in sola lettura.
<code>-t</code>	attiva il tracing per la funzione.
<code>-x</code>	inserisce la variabile nell’ambiente.
<code>-F</code>	evita che sia stampata la definizione completa delle funzioni.
<code>-p</code>	stampa tutti gli attributi associati ad una variabile o una funzione.

Tabella 2.3: Opzioni del comando interno `declare`.

Per visualizzare le variabili di shell presenti nell’ambiente si può usare lo stesso `export` senza specificare nulla, e si otterrà la lista delle variabili presenti nell’ambiente precedute dalla dichiarazione `declare -x`; la lista viene cioè stampata in un formato pronto per essere salvato su un file ed eseguito come script (tratteremo gli script in sez. 2.1.6) per ricreare l’ambiente.

Se si vuole solo una semplice lista di nomi con relativo valore si può usare il comando `env`, questo in generale permette di eseguire un altro comando, da passare come argomento, con un ambiente modificato rispetto a quello della shell; con l’opzione `-i` (o `--ignore`) infatti l’ambiente viene completamente svuotato,¹⁴ mentre con una serie di opzioni `-u` (o `--unset`) si possono cancellare altrettante variabili di ambiente (da specificare come parametro per l’opzione). Se si invoca `env` senza argomenti il comando si limita a stampare la lista delle variabili di ambiente, nella classica forma `VARIABILE=valore`.

Le variabili di ambiente sono di grande importanza perché sono usate in moltissimi casi per controllare alcuni comportamenti predefiniti dei programmi. Alcune impostazioni e valori di uso generale vengono allora memorizzati in una serie di variabili che di norma ogni shell deve definire, come `USER`, che indica il nome dell’utente corrente, `HOME` che ne indica la *home directory*, `TERM` che specifica il tipo di terminale su cui si sta operando, `EDITOR` che indica quale programma usare come editor predefinito, `PATH` che specifica la lista delle directory dove cercare i comandi, ecc.

¹⁴dato che molti comandi rispondono ai valori delle variabili di ambiente, è in genere una misura di sicurezza fornire un ambiente noto e ben definito cancellando preventivamente quanto già presente, onde evitare di ottenere risposte non previste quando si esegue il comando in un ambiente che può essere stato “*manomesso*”.

La variabile `PATH` è di importanza particolare perché è direttamente connessa ad una delle funzionalità della shell su cui finora abbiamo sorvolato dandola per scontata: quella del cosiddetto *path search*, cioè del fatto che essa vi permette di associare un nome scritto sulla linea di comando ad un programma da lanciare. In genere¹⁵ questo nome deve corrispondere al nome del file che contiene il programma che la shell deve eseguire. In sostanza, come visto nell'esempio iniziale di sez. 2.1.3, voi potete sempre scrivere solo il nome del file al posto del pathname completo: se cioè volete leggere una pagina di manuale, basterà scrivere `man` e non sarà necessario specificare `/usr/bin/man`.

Si può fare così perché la shell esegue automaticamente una ricerca nel cosiddetto *PATH dei comandi*; cioè dato un nome sulla linea di comando la shell cerca un file da eseguire con quel nome nella lista di directory contenute nella variabile d'ambiente `PATH`. Se volete evitare questo comportamento, ad esempio perché il programma che volete lanciare non è in una di quelle directory e non sarebbe trovato, è necessario specificare, sia in forma assoluta che relativa, un pathname per accedere al comando;¹⁶ per questo in genere per lanciare un comando (o uno script) nella directory corrente si usa la sintassi:

```
piccardi@monk:~/Truelite$ ./comando
```

La variabile `PATH` ha la forma di una lista di pathname di directory, separati fra loro da un carattere di due punti,¹⁷. Un esempio del valore di questa variabile potrebbe essere il seguente:

```
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
```

come è presa dalla definizione standard per un utente comune su Debian. Si noti come in questo caso siano indicate solo le directory che, nella standardizzazione del filesystem illustrata in sez. 1.2.3, contengono i programmi di uso per l'utente; sono pertanto assenti¹⁸ directory come `/sbin`, `/usr/sbin`, ecc. che usualmente vengono inserite dentro `PATH` soltanto per l'amministratore.

Si tenga conto che `PATH` è comunque modificabile dall'utente, che può personalizzarla a piacere, ed aggiungere un'altra directory, come ad esempio la directory corrente che di norma non vi è mai inclusa.¹⁹ Il dubbio casomai può essere sul come farlo senza doversi riscrivere tutto da capo il contenuto precedente; per questo ci viene di nuovo in aiuto la sintassi della riga di comando per cui è possibile ridefinire una qualunque variabile con una espressione del tipo:

```
piccardi@anarres:~/Truelite$ PATH=$PATH:./
```

dove di nuovo si è usato la capacità della shell di *espandere* il valore delle variabili, che funziona anche all'interno della sintassi usata per assegnare le stesse. Si noti come una riga del genere non esegua nessun programma, limitandosi ad utilizzare esclusivamente le funzionalità interne

¹⁵vedremo a breve come si possono creare dei comandi personalizzati attraverso l'uso degli *alias*, nel qual caso il nome può essere del tutto arbitrario.

¹⁶la shell riconosce questo quando nel nome del comando compare il carattere `./` che indica una directory.

¹⁷è un formato comune usato nelle variabili di ambiente tutte le volte che si deve specificare una lista di directory in cui effettuare ricerche, lo incontreremo spesso, ad esempio in sez. 3.1.2 per le directory dove sono mantenute le librerie di sistema.

¹⁸questo è il motivo per cui alcuni comandi *“non funzionano”* quando chiamati da utente normale, in realtà non è che non funzionano, solo non vengono trovati i relativi programmi, che potrebbero tuttavia essere eseguiti ugualmente (con privilegi ridotti ovviamente) usando un pathname assoluto o inserendo le directory in cui si trovano all'interno di `PATH`.

¹⁹questa scelta ha una ragione precisa: se tenete dentro `PATH` la directory corrente un altro utente potrebbe mettere nelle directory cui ha accesso una versione *“opportunamente”* modificata di un comando di sistema, che verrebbe eseguita al posto dell'originale, col rischio di guai seri qualora ciò avvenisse per l'amministratore; è vero che si inserisce la directory corrente in coda a `PATH` i comandi standard hanno la precedenza, ma qualcuno potrebbe sempre creare un programma `ls` e aspettarvi al varco per un comune errore di battitura di `ls ...`

della shell. Una delle caratteristiche peculiari di **bash** è quella di essere in grado di vedere immediatamente i nuovi programmi non appena si ridefinisce il valore di **PATH**, mentre con altre shell (ad esempio la **csh**) può essere necessario utilizzare un comando apposito come **rehash**.

Come complemento alla gestione del *path search* la shell fornisce il comando interno **which** che, dato il nome di un programma presente nel *path search*, ne stampa a video il pathname assoluto, ad esempio potremo ottenere il pathname del comando **ls** con:

```
piccardi@monk:~/Truelite$ which ls
/bin/ls
```

in questo modo è possibile anche capire quale sia effettivamente il programma che viene usato qualora ne esistano più versioni in alcune delle directory contenute all'interno di **PATH**.²⁰ Se non viene trovato nessun programma corrispondente non sarà stampato nulla.

Si tenga presente però che **which** funziona controllando se il nome passato come argomento compare nella lista dei programmi contenuti nelle directory elencate in **PATH**, se si indica il nome di un comando interno della shell questo non viene considerato, per questo se si vuole capire se si sta usando un comando interno o un programma,²¹ si può usare un altro comando interno, **type**, ottenendo qualcosa del tipo:

```
piccardi@monk:~/Truelite$ type export
export is a shell builtin
```

Un'ulteriore funzionalità che la shell mette a disposizione è quella degli *alias*: prima ancora di cercare se il nome di un comando corrisponde ad un programma presente in una delle directory indicate in **PATH**, la shell controlla se esso corrisponde ad un *alias*; ci permette così di ideare nuovi comandi, definire abbreviazioni per quelli più usati, o ridefinire lo stesso nome di un comando per farlo comportare in maniera diversa. Un *alias* si crea con il comando interno **alias** associando un nome ad un altro comando (che può a sua volta essere stato definito in un altro *alias*) con una sintassi del tipo:

```
alias ll='ls --color=auto -l'
```

ma si può anche definire una abbreviazione con:

```
alias l='ls -l'
```

o ridefinire un comando esistente con:

```
alias rm='rm -i'
```

In questo modo nel primo caso si definisce una versione colorata e “*prolissa*” di **ls**, nel secondo una abbreviazione per l'uso di **ls -l** e nel terzo si ridefinisce **rm** in modo che di default sia richiesta conferma nella cancellazione dei file. Per cancellare un *alias* si può usare il comando **unalias** seguito dal nome dello stesso. Qualora invece si esegua il comando **alias** senza specificare nulla questo mostrerà la lista degli *alias* già definiti.

Un'altra funzionalità di grande utilità fornita dalla shell è il cosiddetto *filename globbing*, si può cioè operare su insiemi di file con nomi simili passando ai comandi che dovranno operare su di essi degli argomenti che li indicano in gruppo tramite i cosiddetti *caratteri jolly* (in inglese *wildcard*). Se cioè all'interno di una riga di comando si usa come argomento un nome che contiene uno di questi caratteri jolly, la shell nell'eseguirne la scansione verificherà quali file ci sono nella

²⁰la regola è che viene usato sempre il primo programma che viene trovato nella scansione delle directory del *path search*; questa avviene da sinistra a destra nell'ordine in cui le directory sono elencate all'interno di **PATH**.

²¹o un *alias*.

directory corrente, e se ne troverà di corrispondenti *espanderà* il nome contenente il carattere jolly nella lista dei file.²²

Come nel caso del DOS o del VMS, il carattere “*” indica un numero arbitrario di caratteri qualsiasi mentre il carattere “?” indica un solo carattere qualunque. Ma oltre a queste due *wild-card* elementari la shell ne supporta di più sofisticate; così si possono indicare elenchi di possibili alternative per un singolo carattere, ponendole fra due parentesi quadre. Le alternative possono essere espresse con una lista dei caratteri voluti messi uno di seguito all’altro, mentre si possono specificare degli intervalli (corrispondenti a tutti i caratteri compresi fra i due estremi) usando due caratteri separati da un “-”. Si può inoltre invertire la selezione dei caratteri specificati fra parentesi quadre precedendo la lista o l’intervallo con un carattere di “~”. Infine come ultima modalità di espansione si può utilizzare il carattere “~” per indicare la home directory dell’utente corrente, mentre si può usare la notazione `~username` per indicare la home directory di un altro utente.

Per capire meglio facciamo alcuni esempi di *filename globbing*: supposto che per essi venga usata la solita estensione `.pdf`, si potranno vedere tutti i documenti PDF presenti in una directory con una riga di comando come:

```
piccardi@anarres:~/Truelite/documentazione/corso$ ls *.pdf
Struttura.pdf baseadm.pdf corso.pdf netadmin.pdf
```

mentre si potranno selezionare tutti i file il cui nome è lungo esattamente sette caratteri con una riga di comando del tipo:

```
piccardi@anarres:~/Truelite/documentazione/corso$ ls ???????
fdl.aux fdl.tex
```

se invece vogliamo vedere i file i cui nomi finiscono in `c` o in `g` potremo usare:

```
piccardi@anarres:~/Truelite/documentazione/corso$ ls *[cg]
Struttura.log baseadm.log corso.log netadmin.log ringraziamenti.log
Struttura.toc baseadm.toc corso.toc netadmin.toc texput.log
```

mentre per selezionare i file PDF il cui nome inizia con una lettera maiuscola potremo usare:

```
piccardi@anarres:~/Truelite/documentazione/corso$ ls [A-Z]*.pdf
Struttura.pdf
```

infine se ci interessano i file Latex il cui nome non inizia per lettera maiuscola potremo usare:

```
piccardi@anarres:~/Truelite/documentazione/corso$ ls [^A-Z]*.tex
baseadm.tex corso.tex netadmin.tex shell.tex
config.tex fdl.tex ringraziamenti.tex struttura.tex
```

Si tenga presente che il meccanismo dell’espansione è quello descritto per cui il risultato della stessa è che viene passato al comando come argomento (o come argomenti, se sono più di uno) il nome (o i nomi) del file che corrisponde, per cui ad esempio se si fosse eseguita la riga di comando:

```
piccardi@anarres:~/Truelite/documentazione/corso$ ls ???
Entries Repository Root
```

²²questo significa che al programma sarà passato, invece di un singolo argomento, la lista dei nomi dei file corrispondenti all’espansione dei caratteri jolly.

si sarebbe ottenuto un risultato forse inaspettato, dato che non riporta file di tre lettere. Il risultato è questo perché l'unico nome di file di tre lettere presente nel caso dell'esempio è quello della directory `CVS`, per cui dall'espansione del *filename globbing* otteniamo una riga di comando equivalente a `ls CVS`, quindi ci verrà stampato, secondo la sintassi di `ls`, il contenuto di tale directory.

Il *filename globbing* e l'uso delle variabili di ambiente sono funzionalità molto utili, però talvolta presentano degli inconvenienti; ad esempio se si volesse passare ad un comando come argomento un file il cui nome contiene uno dei caratteri jolly, o il carattere “\$” usato per referenziare le variabili si avrebbe il problema che la shell cercherebbe di espandere la relativa espressione. Un inconveniente analogo ci sarebbe per i nomi dei file che contengono degli spazi: dato che lo spazio viene usato per separare fra di loro gli argomenti, un tale nome verrebbe spezzato in due con il risultato di passare al comando due argomenti invece di uno.

Per risolvere questi problemi esistono dei caratteri che permettono di disabilitare del tutto o in parte le capacità della shell di interpretare in maniera speciale altri caratteri. Uno di questi è la barra trasversa “\” che anteposta ad un carattere ne disabilita l'interpretazione, così si può usare un asterisco in un argomento indicandolo con “*” o si può utilizzare uno spazio proteggendolo con “\ ” o specificare la stessa barra trasversa con “\\”.

Una modalità alternativa è quella che permette di specificare degli argomenti che vanno considerati come una stringa letterale (risolvendo anche così il problema degli spazi) e per i quali non si vuole che la shell effettui delle espansioni. Questo può essere fatto in due modi, il primo, che disabilita il *filename globbing* e l'interpretazione degli spazi, è quello di definire la stringa racchiudendola fra virgolette (“”), così che al suo interno si possano usare spazi, asterischi e punti interrogativi. In questo modo però resta attiva l'espansione di eventuali variabili di shell tramite il carattere “\$”, si può eliminare anche questa se al posto delle virgolette si usano gli apici singoli ('); in tal caso la stringa sarà interpretata in maniera assolutamente letterale.

Un'altra funzionalità molto utile della shell che si attiva con una sintassi simile alla specificazione delle stringhe, è quella che viene chiamata *command expansion*; si può cioè usare come argomento nella riga di comando il risultato di un altro comando, racchiudendo quest'ultimo fra due apici inversi (‘), o usando la sintassi equivalente `$(comando)`; ad esempio se nel file `elenco` si è scritto una lista di file si potrà effettuare la cancellazione con una linea di comando come:

```
piccardi@anarres:~/Truelite/documentazione/corso$ rm 'cat elenco'
rm: cannot lstat 'pippo.tex': No such file or directory
rm: cannot lstat 'vecchio.tex': No such file or directory
```

che nel caso fallisce perché i file elencati non esistono. In questo caso quello che succede è che la shell esegue il comando racchiuso fra apici inversi, e mette il suo output sulla riga di comando, come se lo si fosse scritto manualmente. Questo ci mette a disposizione un mezzo estremamente potente per collegare fra loro i comandi: possiamo usare il risultato di uno di essi per generare gli argomenti di un altro.

Simile alla *command expansion* è la cosiddetta *arithmetic expansion* che permette di valutare delle semplici espressioni aritmetiche, per farlo si usa una sintassi del tipo di “`$(())`” all'interno della quale si inserisce l'espressione da calcolare; il risultato del calcolo sarà riportato direttamente come argomento, e potremo avere un qualcosa del tipo:

```
piccardi@monk:~/Truelite/documentazione/corso$ echo $((5+7))
12
```

e considerato che all'interno dell'espressione si possono usare delle variabili di shell, si capisce anche come da questa espansione si possano trarre ulteriori potenzialità di utilizzo.

L'ultima funzionalità generica della shell che prenderemo in esame è quella della *storia dei comandi*, la cosiddetta *history*, che permette di accedere, usando i tasti di freccia in alto e in

basso, alle linee di comando eseguite in precedenza. La **bash** infatti mantiene in memoria ogni linea di comando eseguita e poi quando esce salva l'elenco in un apposito file, che di default è `.bash_history` (nella home dell'utente), ma che può essere cambiato in qualunque altro file specificato dalla variabile `HISTFILE`. Nel file vengono salvate fino ad un numero di righe massimo specificato attraverso la variabile `HISTSIZE`.²³

Diventa così possibile non solo rieseguire i comandi precedenti, navigando avanti ed indietro lungo la storia con i tasti di freccia in alto e in basso, ma è anche possibile sia effettuare una ricerca incrementale nella *history* utilizzando la combinazione **C-r** seguita dal testo della ricerca (che comparirà all'interno di un apposito *prompt*), o richiamare con l'uso del carattere `!` una specifica voce, indicandola sia per numero che tramite le prime lettere.

Con il comando `history` inoltre si può visualizzare l'intero contenuto della *history*, in cui ogni riga è numerata progressivamente, con il valore che si può usare con `!` per richiamarla. Così ad esempio per richiamare righe di comando precedenti si potrà fare qualcosa come:

```
piccardi@anarres:~/Truelite/documentazione/corso$ which chown
/bin/chown
piccardi@anarres:~/Truelite/documentazione/corso$ ls -l $(!wh)
ls -l $(which chown)
-rwxr-xr-x    1 root    root      19948 Aug 19 03:44 /bin/chown
```

mentre se si vuole usare direttamente il numero, si potrà verificarlo con:

```
piccardi@anarres:~/Truelite/documentazione/corso$ history
...
523  ls [^A-Z]*.tex
524  which chown
525  ls -l $(which chown)
526  history
```

e richiamare un comando con:

```
piccardi@anarres:~/Truelite/documentazione/corso$ !526
...
523  ls [^A-Z]*.tex
524  which chown
525  ls -l $(which chown)
526  history
527  history
```

si tenga infine conto che il carattere `!`, come mostrato nel primo esempio viene espanso quando usato nella linea di comando, e per usarlo in maniera letterale occorre o proteggerlo con la barra rovesciata, o metterlo in una stringa delimitata da apici singoli. Per maggiori dettagli riguardo tutto l'argomento si può fare riferimento alla sezione **HISTORY EXPANSION** del manuale della shell.

2.1.5 La redirectione dell'I/O

Abbiamo voluto dedicare una sezione separata a questa funzionalità fondamentale della shell perché è quella che ci fornisce la vera potenza dell'interfaccia a riga di comando, permettendoci di combinare fra loro i vari comandi. In sostanza il meccanismo che ci permette di costruire quella catena di montaggio cui accennavamo in sez. 2.1.1 è appunto quello della redirectione dell'I/O.

Come accennato in sez. 1.3.4 quando la shell lancia un comando uno dei suoi compiti è aprire preventivamente 3 file, lo *standard input*, lo *standard output* e lo *standard error*, associati ai primi

²³e si può anche indicare una dimensione massima del file con `HISTFILESIZE`.

tre file descriptor (0, 1 e 2). Questi nel caso di shell interattiva sono associati al terminale su cui essa è eseguita, e pertanto corrispondono alla tastiera per i dati in ingresso e allo schermo per i dati in uscita.

Seguendo una convenzione comune, tutti i comandi unix si aspettano di ricevere i loro dati in ingresso sullo *standard input* (e non da uno specifico file o dispositivo, a meno che questo non sia previsto come argomento), e scrivono i loro dati in uscita sullo *standard output*.²⁴

Prendiamo allora come esempio il comando `cat`, questo (il suo nome, non proprio intuitivo, origina da *conCATenate file*) è un comando elementare serve a leggere uno o più file in ingresso passati come argomenti e a scriverne il contenuto sullo *standard output*. Se non si specifica nessun file come argomento il comando legge di default dallo *standard input*, per cui se eseguiamo:

```
piccardi@anarres:~/Truelite/documentazione/corso$ cat
```

ci troveremo in una situazione in cui il comando è bloccato in attesa che scriviamo qualcosa. Se lo facciamo scrivendo `prova` seguito da invio otterremo qualcosa del tipo:

```
piccardi@anarres:~/Truelite/documentazione/corso$ cat
prova
prova
```

dove il comando reagisce all'invio ristampando sul terminale quanto appena scritto e attendendo nuovo input.²⁵

Usato così il comando non è di grande utilità, in genere non serve a molto scrivere qualcosa sulla tastiera per vederselo ristampare sul terminale ad ogni riga; però se vogliamo vedere il contenuto di un file il modo più immediato per farlo è con un comando del tipo:

```
piccardi@anarres:~$ cat /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

PATH="/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games"

...
```

dato che il comando leggerà il contenuto e lo riverserà sullo *standard output*, ed essendo quest'ultimo collegato al terminale, lo si vedrà sullo schermo.

Fin qui di nuovo, l'utilità del comando è relativa, dato che programmi per visualizzare il contenuto del file ce ne sono altri, magari più sofisticati come `more` e `less`, che permettono anche di vedere il file un pezzo alla volta, e andare avanti e indietro, e non scrivono tutto di file in un colpo solo sul terminale dove poi l'inizio va perso nello scorrimento delle scritte sullo schermo. Il comando comunque prende alcune opzioni per facilitare la visualizzazione, come `-n` che stampa un numero progressivo per ogni riga, `-t` che stampa i tabulatori come `^I`, `-v` che visualizza i caratteri non stampabili, e `-E` che scrive un `$` alla fine di ogni riga. La descrizione completa si trova al solito nella pagine di manuale accessibile con `man cat`.

²⁴lo *standard error* non viene, in caso di operazioni regolari, mai usato; su di esso vengono scritti solo gli eventuali messaggi di errore. Nel caso di shell interattiva questi di nuovo vanno sul terminale e compaiono sullo schermo, insieme ai dati in uscita, ma questo solo perché si è usato lo stesso file dello *standard output*, in generale, ad esempio quando si dirige lo *standard output*, non è così, essendo i due file distinti.

²⁵questo avviene in quanto sul terminale si quello che si chiama I/O **bufferizzato**, per cui i dati in ingresso vengono letti e scritti una riga alla volta, per cui alla conclusione della riga verrà letto quanto appena scritto, ed il comando lo riscriverà sullo *standard output*; per uscire dal comando occorre nel caso indicare che la lettura è conclusa, ad esempio inviando un end-of-file sull'input con `C-d`, o interrompere il programma con un segnale.

La vera utilità del comando, il cui scopo come dice il nome stesso non è quello di mostrare un file, emerge solo quando esso viene unito alla capacità della shell di modificare i file associati a *standard input*, *standard output* e *standard error*, cioè con la *redirezione dell'I/O*. La shell infatti riconosce sulla riga di comando vari operatori, detti appunto di *redirezione*. I due più elementari sono “<” e “>” che permettono rispettivamente di redirigere lo *standard input* e lo *standard output* su un file specificato dall'utente dopo l'operatore.²⁶

Vediamo allora come si usano di questi due operatori, una modalità alternativa di stampare il contenuto di un file con `cat`, è utilizzando un comando del tipo:

```
piccardi@anarres:~/Truelite/documentazione/corso$ cat < /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).
...
```

in cui invece di leggere un file specificato dalla riga di comando, si legge sempre dallo *standard input* che però in questo caso, invece di essere associato al terminale, è stato rediretto dall'operatore “<” sul file `/etc/profile`.

La redirezione dell'output avviene in maniera identica associando lo standard output ad un file, e ad esempio si può copiare il contenuto del file `pippo` sul file `pippo2` con un comando del tipo:

```
cat pippo > pippo2
```

dato che invece che sul terminale adesso il contenuto di `pippo` sarà scritto su `pippo2`. Si tenga conto che se il file su cui si effettua la redirezione non esiste viene creato, se invece esiste viene prima troncato a zero e poi sovrascritto.

Fin qui di nuovo niente di particolarmente utile, dato che nel primo caso si sarebbe potuto stampare direttamente il file, e nel secondo si sarebbe potuto usare `cp`. Però, e qui emerge anche la vera utilità del comando `cat`, con `cp` si può solo copiare il contenuto di un file su un altro, mentre cosa succede se con `cat` specifichiamo una serie di file per poi redirigere l'uscita su di un altro? Per quanto appena detto il comando leggerà il contenuto di ciascuno dei file passati come argomenti, riscrivendolo sullo *standard output*, che nel caso è stato rediretto sul nuovo file, per cui quest'ultimo risulterà appunto essere la *concatenazione* del contenuto di tutti i file passati come argomenti.

Così se si è spezzato un file di grosse dimensioni in una serie di file più piccoli (ad esempio per scriverlo su una serie di floppy) si potrà ricostruirlo con un semplice comando del tipo:

```
cat file1 file2 file3 ... > filecompleto
```

La redirezione dello *standard input* è invece molto utile con i programmi che richiedono l'immissione dei dati da tastiera, specie quando questi vengono eseguiti in uno script. Una volta definiti quali dati sono necessari infatti li si potranno scrivere in un file, e farli leggere al programma semplicemente con la redirezione dello standard input.

Come accennato quando si effettua una redirezione dello *standard output* il file di destinazione viene sovrascritto; occorre pertanto stare molto attenti a non cancellare dei dati. Per questo è bene essere consapevoli della modalità con cui il meccanismo funziona, se infatti si pensasse di aggiungere dei dati in fondo ad un file con un comando del tipo di:

```
cat file addendum > file
```

²⁶in realtà si può effettuare la redirezione su un qualunque file descriptor, specificando il suo numero prima dell'operatore, ad esempio si può redirigere lo *standard error* con l'operatore “2>”. I dettagli sulle varie forme di redirezione possono trovare nella pagina di manuale alla sezione **REDIRECTION**.

si andrebbe incontro ad una sgradita sorpresa. Quello che avviene con la redirectione infatti è che prima viene creato o troncato il file, e poi ci viene scritto sopra il risultato; il che comporta che nell'esempio in questione `cat` alla lettura di `file` lo troverebbe vuoto. Nel caso specifico `cat` è in grado di rilevare la situazione anomala e stampare un errore, ma non è detto che altri programmi (o versioni non GNU di `cat`) siano in grado di farlo; se ad esempio si fosse usato `less` (che usato con la redirectione si comporta esattamente come `cat`) si sarebbe perso il contenuto originale di `file`.

Per ovviare a questo problema è disponibile un altro operatore di redirectione, “>>”, che redirige lo *standard output* su di un file eseguendo però la scrittura in *append* (una modalità di scrittura speciale in cui i dati vengono aggiunti in coda a quelli già presenti). In questo modo si sarebbe potuto realizzare l'aggiunta di informazioni alla fine del file `file` con il comando:

```
cat addendum >> file
```

La redirectione di ingresso ed uscita, per quanto utile, è però marginale rispetto ad una forma di redirectione molto più potente, il cosiddetto *pipelining*, che usa l'operatore “|”. È questo infatti quello che ci permette di eseguire la concatenazione dei comandi cui abbiamo accennato in sez. 2.1.1. Questo operatore infatti permette di collegare l'uscita del comando che lo precede con l'ingresso del successivo attraverso una *pipe*,²⁷ che come indica il nome funziona appunto come *tubo di collegamento* fra i due comandi. Questo significa che si può fornire come dati in ingresso ad un comando quelli prodotti in uscita da un altro. Così si possono lanciare in sequenza una serie di comandi in cui ciascuno elabora i risultati del precedente, fornendo un risultato per il successivo.

Già questo comincia a farci intuire le capacità intrinseche nel funzionamento della riga di comando, ad esempio non è più necessario dover inserire in tutti i comandi una opzione per ordinare in maniera diversa i risultati, basta inviare questi ultimi con una *pipe* al programma `sort` il cui solo scopo è quello di effettuare riordinamenti nelle modalità più varie.

Comando	Significato
<code>cmd < file</code>	redirige lo <i>standard input</i> : legge l'input del comando <code>cmd</code> dal file <code>file</code> .
<code>cmd > file</code>	redirige lo <i>standard output</i> : scrive il risultato del comando <code>cmd</code> sul file <code>file</code> .
<code>cmd >> file</code>	redirige lo <i>standard output</i> : scrive il risultato del comando <code>cmd</code> accodandolo al contenuto del file <code>file</code> .
<code>cmd 2> file</code>	redirige lo <i>standard error</i> : scrive gli errori del comando <code>cmd</code> sul file <code>file</code> .
<code>cmd1 cmd2</code>	<i>pipelining</i> : redirige lo <i>standard output</i> del comando <code>cmd1</code> sullo <i>standard input</i> del comando <code>cmd2</code> .
<code>cmd > file 2>&1</code>	redirige lo <i>standard output</i> e lo <i>standard error</i> : scrive errori e risultati del comando <code>cmd</code> sul file <code>file</code> .
<code>cmd1 2>&1 cmd2</code>	redirige lo <i>standard output</i> e lo <i>standard error</i> del comando <code>cmd1</code> sullo <i>standard input</i> del comando <code>cmd2</code> .

Tabella 2.4: Principali modalità di redirectione.

Vedremo in sez. 2.2 vari esempi di come queste funzionalità possono essere sfruttate per costruire delle vere e proprie *catene di montaggio* in cui si ottiene un risultato finale attraverso la concatenazione di molti comandi, concludiamo questa sezione riportando in tab. 2.4 un riassunto delle principali modalità di redirectione utilizzate nella shell. Compresse alcune delle più esoteriche che permettono di redirigere più file.

²⁷che è del tutto analoga alle *fifo* incontrate in sez. 1.2.1, solo che in questo caso non è associata ad un oggetto nel filesystem, ma esiste solo come file descriptor accessibile dall'interno dei processi interessati.

2.1.6 Scripting elementare

Una delle capacità fondamentali della shell è che, come avveniva per i file `.bat` del DOS, si possono inserire delle sequenze di comandi in dei file per farli eseguire direttamente senza doverli riscrivere tutte le volte. Ma se questo è più o meno tutto quello che si può fare con il DOS, il grande vantaggio di una shell unix che è avete a disposizione un vero e proprio linguaggio di programmazione con tanto di variabili e direttive di iterazione, condizionali, ecc. che vi permette di effettuare anche compiti di notevole complessità, tanto che, per quanto sia poco elegante e piuttosto limitato, viene comunque ampiamente utilizzato, per la sua immediatezza, per automatizzare una grande quantità di compiti per i quali non vale la pena ricorrere ad un linguaggio di più alto livello.

E per questo motivo che la shell, anche se finora ne abbiamo parlato solo come del programma che implementa l'interfaccia a riga di comando, è in realtà uno dei componenti essenziali per far funzionare il sistema,²⁸ e non solo dal punto di vista dell'interfaccia utente: tutti i sistemi di avvio del sistema infatti, come vedremo in sez. 5.3.4, usano proprio degli *script di shell* per eseguire le loro operazioni.

La descrizione dettagliata del linguaggio della shell non è affrontabile adesso, le sue caratteristiche principali si trovano comunque nella sezione **SHELL GRAMMAR** della pagina di manuale, ed un ottima risorsa per quanti vogliano approfondirlo è [2]. Quello che è importante sottolineare qui è che la shell non solo può essere invocata in maniera interattiva, quando è associata ad un terminale ed utilizzata dagli utenti per dare comandi, ma può anche essere lanciata non interattivamente facendole eseguire direttamente quello che appunto si chiama uno *script*, in questo caso la si invoca come un qualunque altro comando, e prende come argomento il file dello script da eseguire, con sintassi del tipo `bash script`.

In realtà gli script possono sfruttare una funzionalità peculiare del *link-loader*²⁹ di Linux (che si trova comunque in tutti i sistemi unix-like), che oltre ad eseguire i programmi nel formato binario standard³⁰ è in grado di eseguire anche gli script, che vengono trattati esattamente allo stesso modo. Questi sono identificati dal *link-loader* come file di testo dotati di permesso di esecuzione la cui prima riga è nella forma:

```
#!/bin/bash
```

in cui cioè si trovano i caratteri `"#!"` seguiti dal pathname di un programma che faccia da interprete. In tal caso il *link-loader* lancerà automaticamente il comando (con le opzioni che si possono specificare sempre sulla prima riga del file) dandogli come argomento il nome file stesso.

Dato che per la shell e per tutti i linguaggi di scripting, come il python o il perl, che usano questa funzionalità il carattere `#` indica l'inizio di una riga di commento, quando si vuole creare uno script basterà scrivere un file di testo contenente tutte le istruzioni necessarie che inizi con questa riga (che verrà ignorata nell'interpretazione dei comandi), e poi renderlo eseguibile.³¹ In questa maniera uno script sarà del tutto equivalente ad un comando binario, ed oltre ad eseguirlo come tale la shell provvederà anche a passargli gli argomenti scritti nella riga di comando, esattamente come avrebbe fatto se di fosse trattato di un eseguibile compilato.

Nel caso degli script però, rispetto quanto avviene sulla riga di comando, si usa spesso una sintassi della shell che prevede l'uso di istruzioni e comandi con una sintassi più complessa di

²⁸si noti infatti che essa deve trovarsi, come richiesto dal FHS trattato in sez. 1.2.3, sotto `/bin`, dato che è essenziale per l'avvio del sistema.

²⁹questo è un programma speciale, che non costituisce un programma a se (ed infatti sta sotto `/lib`), ma che viene lanciato come parte dalla chiamata al sistema che esegue un nuovo programma, che permette di utilizzare le librerie condivise e mettere in esecuzione gli altri programmi.

³⁰in Linux esistono sostanzialmente due formati binari, quello chiamato `a.out`, usato nei kernel delle serie fino alla 1.2, e il formato ELF, usato nelle serie successive.

³¹renderlo eseguibile in questo caso significa attivarne il permesso di esecuzione con `chmod +x`, dal punto di vista del contenuto del file non cambia assolutamente nulla, questo resta un normale file di testo.

quella elementare illustrata in sez. 2.1.3. Una sintassi e delle istruzioni che, per quanto siano utilizzabili in teoria anche sulla riga di comando, sono generalmente poco usate in quanto meno indicate ad un uso interattivo, o dipendenti da funzionalità specifiche degli script.

Una prima sintassi è quella che permette di eseguire più comandi distinti sulla stessa linea; questo può essere fatto in tre modi diversi. La prima modalità prevede che i singoli comandi siano separati fra di loro tramite il carattere “;”, in tal caso si tratta semplicemente di scrivere quello che altrimenti si scriverebbe su linee diverse su una riga unica, usando il “;” per indicare la fine di un comando e l’inizio del successivo: in questo caso essi saranno eseguiti in sequenza da destra a sinistra come al posto del “;” si fosse premuto invio. In realtà questo è di scarsa utilità sulla linea di comando (se non per eseguire più comandi in sequenza senza dover stare ad attendere che ciascuno di essi finisca per lanciare il successivo) ma è utile negli script dove alcuni direttive di programmazione (come i condizionali) usano come parametro una linea di comando, la cui terminazione può essere resa esplicita con questo metodo.³²

La seconda modalità è quella in cui i comandi che si vogliono eseguire vengono separati dall’operatore “&&”, che assume il significato di AND logico. In questo caso il comando successivo all’operatore verrà eseguito solo se il precedente non ha avuto errori. In caso di errori la catena viene immediatamente interrotta. Questo consente allora di eseguire un secondo comando solo se il primo ha avuto successo; in genere lo si usa per verificare se un certo file esiste e poi eseguire un comando.

La terza modalità prevede che i diversi comandi sulla stessa linea vengano separati dall’operatore “||”, che invece assume il significato di OR logico. In questo caso si avrà che il comando successivo all’operatore verrà eseguito soltanto se il precedente ha fallito. Questo consente di eseguire in alternativa due comandi; in genere lo si usa per eseguire le operazioni da effettuare solo in caso di errori.

Queste due ultime modalità fanno riferimento, per stabilire se un comando ha avuto o meno successo, ad una caratteristica comune di tutti i programmi che una volta eseguiti riportano (nel caso specifico alla shell che li ha lanciati) uno stato di uscita che serve ad indicare se le operazioni sono state concluse correttamente. La convenzione è che un valore nullo (cioè 0) significa il successo dell’operazione, mentre un valore non nullo (in genere 1) indica che c’è stato un errore. Inoltre se il programma non si è concluso spontaneamente ma è stato interrotto da un segnale la shell assegna al codice di uscita il valore di $128 + n$, dove n è il numero del segnale (ed ovviamente il comando viene considerato fallito). È con un controllo automatico su questi valori che vengono prese le rispettive decisioni nell’uso dei operatori “&&” e “||” quando la shell concatena dei comandi. È inoltre da specificare che qualora invece si concatenino dei comandi senza condizionali con l’uso del “;” verrà riportato il codice di uscita dell’ultimo comando eseguito.

La presenza di un codice di uscita è molto importante, specie per gli script che possono usare questa informazione per capire se ci sono stati problemi. Per questo la shell, nel ricevere lo stato di uscita di un comando, lo memorizza nella variabile speciale “?”, così che per un comando successivo o all’interno di uno script è possibile ricavare lo stato di uscita dell’ultimo comando eseguito in precedenza accedendo a quest’ultima. Allora potremo verificare che:

```
piccardi@anarres:~/Truelite/documentazione$ ls
CVS README corso internet-server lucidi samba
piccardi@anarres:~/Truelite/documentazione$ echo $?
0
piccardi@anarres:~/Truelite/documentazione$ ls lucidi
ls: lucidi: No such file or directory
piccardi@anarres:~/Truelite/documentazione$ echo $?
1
```

³²è ad esempio il costrutto usato nel condizionale presente nell’esempio di `/etc/profile` illustrato più avanti.

e nel primo caso il comando è stato eseguito correttamente, mentre nel secondo, non esistendo il file `licidi`, no.

Variabile	Significato
#	Il numero degli argomenti passati ad uno script.
*	La lista degli argomenti. Se espansa fra virgolette come "\$*" è equivalente ad una unica stringa formata dal contenuto dei singoli argomenti separato da spazi (o dal carattere di separazione specificato in IFS).
@	La lista degli argomenti. Se espansa fra virgolette come "\$@" è equivalente alla lista dei singoli argomenti.
?	Il codice di uscita dell'ultimo comando eseguito.
\$	Il pid della shell.
!	Il pid del programma in background che è stato eseguito per ultimo.
0	Il nome della shell o dello script.

Tabella 2.5: Principali variabili speciali.

La variabile `?` è una delle *variabili speciali* (descritte nella sezione **Special Parameters** della pagina di manuale), queste variabili vengono mantenute direttamente dalla shell stessa (non ha cioè senso assegnarle direttamente come visto in sez. 2.1.4) che vi memorizza il valore di alcuni parametri interni. Ad esempio quando si lancia uno script con degli argomenti con `$#` si ottiene il numero degli stessi, con `$1` il primo, con `$2` il secondo, ecc. È inoltre possibile ottenere una stringa contenente tutti gli argomenti con `$*`, mentre con `$@` si ottiene la lista degli stessi.³³ L'elenco delle principali *variabili speciali* è riportato in tab. 2.5.

Al di là della semplice logica dei due operatori `&&` e `||`, la shell fornisce anche delle *istruzioni* che consentono di creare costrutti sintattici più complessi, propri di un vero linguaggio di programmazione. Di questi illustreremo solo i più significativi (l'elenco completo è nella sezione **SHELL GRAMMAR** della pagina di manuale). Il primo è quello usato per specificare delle esecuzioni condizionali ed è basato sull'istruzione `if`; la sua forma generica è:

```
if comando; then
    comando
    ...
else
    comando
    ...
fi
```

dove dopo l'`if` si esegue il comando specificato, che è quello che specifica la condizione, e a seconda del suo codice di ritorno verranno eseguiti alternativamente i comandi indicati dopo l'istruzione `then` (se nullo, cioè condizione di riuscita) o l'istruzione `else` (se non nullo, cioè fallimento).

La sintassi può essere più complessa in quanto si può sostituire l'istruzione `else` con un'istruzione `elif` che invece di eseguire una serie di comandi ripete un successivo blocco condizionale, prendendo come argomento un altro comando da usare come condizione, seguito da un altro `then` e da altri comandi; queste istruzioni possono essere ripetute fino a chiudere la sequenza con l'istruzione `else`, o, se non ci sono alternative, chiudere definitivamente il blocco condizionale con l'istruzione `fi`.

³³la differenza fra `$*` e `$` può sembrare non significativa, e se si stampano le due variabili con `echo` non se ne vedrà nessuna; in realtà questa è fondamentale negli script, infatti `$*` nelle istruzioni usate negli script viene considerata come un'unica stringa, mentre `$` viene espansa in una lista, costituita da tanti elementi quanti sono gli argomenti passati allo stesso; questo comporta ad esempio che nel caso si usi la prima in una istruzione `for` si avrà un solo ciclo, mentre con la seconda si avranno tanti cicli quanti sono gli argomenti.

Oltre all'uso di `if` si possono eseguire cicli condizionali tramite dell'uso delle due istruzioni `while` e `until` che ripetono l'esecuzione del comando specificato come argomento fintanto che questo non riporta rispettivamente uno stato di uscita nullo o non nullo, eseguendo ogni volta la lista dei comandi inserita nella sezione seguente delimitata dalle istruzioni `do` e `done`, secondo uno schema del tipo:

```
while comando; do
    comando
    ...
done
```

(e analogo per `until`); in questo modo si possono eseguire dei cicli condizionali, basati sul successo (`while`) o l'insuccesso (`until`) di un determinato comando che governa la ripetizione del ciclo.

Se invece si devono operare delle scelte opzionali su un valore si può usare l'istruzione `case`, che prende come parametro una espressione che possa assumere valori diversi (una variabile, una *command expansion*, ecc.) e che permette di eseguire comandi diversi a seconda dei valori che detta espressione può assumere, con una sintassi generica del tipo di:

```
case expr in
    valore1)
        comando
        ...
    ;;
    valore2)
        comando
        ...
    ;;
    *)
        comando;
        ...
    ;;
esac
```

in cui ogni caso viene terminato da un doppio “;”, ed i singoli valori con cui viene confrontata l'espressione sono terminati da una parentesi chiusa, ed il valore generico è indicato da un `*`.

L'utilità di `case` rispetto all'uso di una serie concatenata di `if` ed `elif`, oltre alla leggibilità, è che i valori con cui effettuare il controllo supportano i caratteri jolly con lo stesso significato che questi assumono nel *filename globbing*, anche se in questo caso vengono applicati alla lista dei valori forniti dall'espressione, piuttosto che ai file. Questo consente allora di raggruppare in maniera flessibile valori simili e trattarli con una unica serie di istruzioni.

Infine l'istruzione `for` permette di compiere dei cicli su un numero stabilito di alternative, definendo una variabile che può essere utilizzata all'interno del ciclo, con una sintassi del tipo:

```
for var in lista ; do
    comando
done
```

dove `var` è la variabile che assumerà per ciascun ciclo ciascuno dei possibili valori risultanti dall'espansione dell'espressione `lista`.

Una delle caratteristiche comuni a tutti i linguaggi, e la shell non fa eccezione, è quella della possibilità di definire delle funzioni cui fare eseguire compiti ripetitivi. Questa capacità aggiunge una ulteriore funzionalità alla linea di comando in quanto oltre ad eseguire i comandi binari e gli

script, o usare un alias per fare riferimento ad un comando, una modalità con cui si possono far eseguire dei comandi alla shell è quello di definire delle funzioni ed eseguire una lista di comandi richiamando semplicemente il nome della funzione.

In generale una funzione è definita all'interno di uno script, e viene introdotta dalla parola chiave **function** (ma lo si può fare anche a linea di comando, è solo molto più scomodo), cui segue in nome della funzione e il costrutto **()**, dopo di che il corpo della funzione è delimitato fra parentesi graffe; in sostanza la forma è sempre qualcosa del tipo:

```
function nome () {
    comando1;
    comando2;
    ...;
}
```

dove ciascuna linea conterrà un comando.

Oltre alle istruzioni interne della shell esistono anche una serie di comandi che sono stati creati proprio per facilitare la scrittura di script. Il primo di questi è **seq**, che permette di stampare sullo standard output una sequenza di numeri. Il comando può essere invocato con un numero di argomenti variabile da uno a tre numeri interi: un solo argomento indica il numero finale a cui fermare la sequenza, partendo da 1 ed incrementando di uno, specificando due argomenti si indicano il valore iniziale ed il valore finale, che viene raggiunto sempre a passi di uno, infine con tre argomenti il secondo viene usato come valore di incremento nell'intervallo che inizia con il primo e finisce col terzo. Questo comando è molto utile quando si vogliono fare dei cicli, lo si può infatti usare per generare una lista di numeri da utilizzare con l'istruzione **for**, i cui valori possono poi essere utilizzati all'interno del ciclo.

Altri due comandi speciali sono **true** e **false**, che non fanno assolutamente nulla se non fornire alla shell un valore di ritorno. Nel caso di **true** si avrà un valore nullo che nelle condizioni implica vero, e viene utilizzato, come illustrato in sez. 2.2.1 per indicare che un comando ha avuto successo. Se invece si usa **false** questo restituisce un valore non nullo, equivalente alla condizione falsa e al fallimento del comando. Questi due programmi vengono usati spesso come shell di default (vedi sez. 4.3.3) per gli account degli utenti di sistema che non devono avere una shell.

Un altro comando utilizzato negli script principalmente per controllare il suo codice di uscita è **test** che è uno dei comandi più comuni utilizzato in combinazione con le *istruzioni* condizionali illustrate in precedenza. Per questo motivo una modalità alternativa per invocare il comando è con il nome **[]** (si può infatti osservare l'esistenza del file **/usr/bin/[]** nel qual caso la linea di comando dovrà essere terminata con una corrispondente **]**). È questo il motivo per cui le condizioni che si trovano spesso negli script di shell richiedono che dopo la parentesi quadra sia sempre presente uno spazio, non si tratta infatti di una sintassi interna della shell ma di un comando esterno, che come tale deve essere riconosciuto.

Il comando può eseguire controlli su una enorme serie di diverse condizioni, specificate attraverso una opportuna espressione, ritornando un valore nullo se la condizione è verificata, e non nullo altrimenti. Il comando inoltre supporta l'uso di alcuni *operatori*, come **!**, che anteposto ad una espressione ne nega il risultato, e la combinazione logica di due espressioni con l'uso delle opzioni **-a** e **-o** (che indicano rispettivamente l'AND e l'OR); in questo modo si possono costruire delle condizioni molto complesse.

Buona parte delle condizioni verificate da **test** fanno riferimento alle proprietà dei file, in questo caso la condizione viene espressa con una opzione seguita dal nome del file cui la si vuole applicare; le principali opzioni utilizzate, ed il relativo significato, sono riportati in tab. 2.6.

Oltre alle condizioni sui singoli file il comando può essere usato per esprimere condizioni relative a due file, in tal caso queste vengono espresse scrivendo questi ultimi come argomenti

Opzione	Significato
-b	il file esiste ed è un dispositivo a blocchi
-c	il file esiste ed è un dispositivo a caratteri
-d	il file esiste ed è una directory
-e	il file esiste
-f	il file esiste ed è un file normale
-h	il file esiste ed è un link simbolico
-O	il file esiste ed è di proprietà dell' <i>effective user ID</i> del processo
-G	il file esiste ed è proprietà dell' <i>effective group ID</i> del processo
-r	il file esiste ed è leggibile
-s	il file esiste ed ha dimensione maggiore di zero
-w	il file esiste ed è scrivibile
-x	il file esiste ed è eseguibile

Tabella 2.6: Opzioni per le condizioni sui file del comando `test`.

separati dalla relativa opzione; con `-ef` si richiede che essi abbiano lo stesso inode, con `-nt` si richiede che il primo sia più recente (in termini di tempo di ultima modifica, vedi sez. 1.2.1) mentre con `-ot` che sia più vecchio (negli stessi termini).

Una ultima serie di condizioni riguarda l'uso di stringhe e numeri; per questi ultimi valgono gli operatori di confronto dei valori come `-eq`, `-ge`, `-gt`, `-le`, `-lt` e `-ne`, il cui significato è banale (uguale, maggiore uguale, maggiore, minore uguale, minore, e diverso). Per le stringhe invece si possono usare gli operatori `"=`" e `"!="` per indicare rispettivamente uguaglianza e disuguaglianza, e l'opzione `-z` serve a verificare se la stringa è vuota.

Oltre ad una sintassi più complessa, l'uso degli script ci pone di fronte ad alcune sottigliezze del funzionamento della shell. Se infatti si lancia uno script esso viene eseguito come tutti gli altri comandi, la shell cioè crea un nuovo processo figlio in cui esegue quella che si chiama una *subshell*³⁴ a cui fa eseguire lo script. Questo vuol dire ad esempio che se nello script si effettuano operazioni che modificano le proprietà della shell (come modifiche alle variabili di ambiente o agli alias) queste modifiche saranno effettive solo per la *subshell*, e non per la shell originaria da cui esso è stato lanciato.

Per cui se si vogliono automatizzare una serie di impostazioni per la shell corrente, non si può pensare di far questo lanciando uno script che contiene i relativi comandi. È però possibile eseguire i comandi di uno script senza usare una *subshell*, direttamente all'interno della shell corrente, usando il comando `source`,³⁵ cui dare come argomento il nome dello script da cui leggerli.

Ora è abbastanza chiaro che, benché il programma sia lo stesso, una shell utilizzata per eseguire uno script necessita di impostazioni diverse rispetto ad una shell usata per gestire la riga di comando da un terminale (ad esempio non serve il *prompt*). Per questo la shell supporta diverse modalità di funzionamento di cui quella usuale a linea di comando avviene attraverso quella che si chiama una *shell interattiva*, in cui essa è associata ad un terminale, e che deve pertanto essere in grado di gestire tutte le problematiche relative al controllo di sessione illustrato in sez. 1.3.4, che invece non sono necessarie quando la si invoca per eseguire uno script.

Un'altra modalità è quella della *shell di login* (vedi sez. 4.3). Questa è la modalità che viene usata dal programma `login` quando dà all'utente un accesso al sistema e che in genere è alla radice di ogni altro programma lanciato dall'utente.³⁶ Queste modalità in realtà cambiano pochissimo, quello che le contraddistingue è che nel caso di shell di login viene sempre

³⁴cioè un'altra istanza della shell, che viene eseguito da un altro processo.

³⁵una sintassi alternativa, più sintetica ma anche più criptica, è quella di usare `.` al posto di `source`.

³⁶in generale un utente può sempre lanciare, dopo il login, varie altre shell interattive, il sistema comunque ne fornisce una di default.

letto ed eseguito (all'interno della shell corrente) il contenuto del file `/etc/profile`,³⁷ in cui l'amministratore di sistema può inserire una serie di impostazioni comuni per tutti gli utenti.

Un esempio di questo file, che può essere considerato una sorta di file di configurazione per la shell, è il seguente (l'esempio è preso dalla versione installata di default su una Debian):

```
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

PATH="/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games"

if [ "$BASH" ]; then
    PS1='\u@\h:\w\$ '
else
    if [ "'id -u'" -eq 0 ]; then
        PS1='# '
    else
        PS1='$ '
    fi
fi

export PATH PS1

umask 022
```

in questo caso si può notare come prima venga impostato la variabile `PATH`, poi, se la shell è una `bash`, il *prompt* usando un valore di `PS1` che sfrutta le funzionalità di tab. 2.1, altrimenti (supponendo una shell generica) vengono usati direttamente i caratteri `#` e `$` a seconda che l'utente sia o meno l'amministratore. Infine viene impostato il valore della *umask* (vedi sez. 1.4.4).

Per permettere agli utenti di personalizzare la loro shell di login, dopo aver letto `/etc/profile` la `bash` cerca nella home degli utenti, in quest'ordine, i file `.bash_profile`, `.bash_login` e `.profile`, ed esegue i comandi nel primo che trova (e per il quale ha il permesso di lettura), senza guardare gli altri. Si noti comunque che nessuno di questi file deve essere eseguibile, dato che non viene eseguito come script, per cui è sufficiente il permesso di lettura. Un esempio di `.bash_profile` è il seguente:

```
# ~/.bash_profile: executed by bash(1) for login shells.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

umask 027

# include .bashrc if it exists

if [ -f ~/.bashrc ]; then
    source ~/.bashrc
fi
```

dove sostanzialmente si modifica la *umask* (vedi sez. 1.4.4) rispetto al valore impostato per il sistema e si utilizzano le impostazioni personali poste in un altro file, `.bashrc`.

L'uso di `.bashrc` per le impostazioni è dovuto al fatto che se la shell è interattiva ma non è di login, (questo vale ad esempio per le shell lanciate dentro un terminale sotto X) al posto dei precedenti file viene letto quest'ultimo. È per questo che conviene inserire in esso le proprie personalizzazioni, ed eseguire un `source` di questo file all'interno di `.bash_profile`, in modo da averle disponibili in tutti i casi. Un esempio è:

³⁷questo vale in generale per tutte le shell derivate dalla Bourne shell, quelle derivate dalla C shell usano `/etc/csh.login`.

```

# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If running interactively, then:
if [ "$PS1" ]; then

    # don't put duplicate lines in the history. See bash(1) for more options
    # export HISTCONTROL=ignoredups

    # enable color support of ls and also add handy aliases
    if [ "$TERM" != "dumb" ]; then
        eval 'dircolors -b'
        alias ls='ls --color=auto'
        #alias dir='ls --color=auto --format=vertical'
        #alias vdir='ls --color=auto --format=long'
    fi

    # some more ls aliases
    #alias ll='ls -l'
    #alias la='ls -A'
    #alias l='ls -CF'

    # set a fancy prompt
    PS1='\u@\h:\w\$ '

    # If this is an xterm set the title to user@host:dir
    #case $TERM in
    #xterm*)
    #    PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME}: ${PWD}\007"'
    #    ;;
    #*)
    #    ;;
    #esac

    # enable programmable completion features (you don't need to enable
    # this, if it's already enabled in /etc/bash.bashrc).
    #if [ -f /etc/bash_completion ]; then
    #    . /etc/bash_completion
    #fi
fi

```

ed in questo caso quello che si fa è definire una serie di *alias*, ed un nuovo prompt.

Si tenga conto che oltre a `.bashrc` per i singoli utenti esiste un equivalente di `/etc/profile` anche per le shell interattive, in cui sono mantenute le impostazioni generali scelte dall'amministratore per tutto il sistema, che è `/etc/bash.bashrc`.

2.2 I comandi dei file

Dato che in un sistema unix-like tutto è un file, è naturale che la maggior parte dei comandi abbia a che fare con le operazioni di manipolazione dei file. Abbiamo già visto nel cap. 1 i comandi elementari che ci permettono la gestione dei file sul filesystem come `cp`, `mv`, `mkdir`, `ln`, `rm`, ecc. In questa sezione affronteremo gli altri comandi che permettono di operare sul contenuto dei file ed eseguire su di essi operazioni più complesse, e dato che buona parte di queste vengono effettuate con la redirectione vedremo anche delle applicazioni di quest'ultima.

2.2.1 Caratteristiche comuni

Benché, come anticipato in sez. 2.1, ogni comando sia specializzato per fare un compito specifico, esistono comunque una serie di caratteristiche comuni, dato che i comandi relativi ai file sono la maggioranza, le tratteremo qui, anche se si applicano in generale a qualunque tipo di comando, e non solo a quelli che riguardano i file. La principale caratteristica comune a quasi tutti i comandi è la gestione delle opzioni che avviene come illustrato in sez. 2.1.3 e alcune di esse, quelle più comuni, tendono ad essere le stesse; ad esempio per la gran parte dei comandi l'opzione `-h` può essere utilizzata per stampare a video una schermata di aiuto che riassume l'uso del comando.

Questa è comunque solo una convenzione, e non è seguita da tutti i comandi, alcuni infatti usano delle sintassi diverse per eredità storiche, (come abbiamo visto in sez. 1.3.1 con `ps`). La convenzione però viene usata da tutti i comandi realizzati all'interno del progetto GNU e da quelli che usano le relative funzionalità delle GNU libc, ed è perciò molto diffusa.

Le opzioni in genere sono di due tipi, dei semplici *switch*, che come degli interruttori attivano o disattivano una certa modalità di funzionamento che si attivano semplicemente scrivendole (ad esempio si usa spesso `-v` per aumentare la *prolissità* dei messaggi del comando), o delle opzioni più complesse che, come in maniera analoga ad un cursore o una manopola, permettono di passare dei valori al comando (come il *process ID* del processo cui applicare un cambiamento di priorità, che si viene dato come valore per l'opzione `-p` di `renice`); in tal caso questo, nel formato che varierà da caso a caso, il valore dovrà essere specificato di seguito all'opzione.

Inoltre comandi del progetto GNU supportano (come accennato in sez. 2.1.3) anche una versione estesa delle opzioni, in cui queste si possono specificare con parole intere invece che con singole lettere, nel qual caso esse iniziano con un `--`. Tutti i comandi GNU ad esempio supportano le due opzioni `--help` che stampa una schermata riassuntiva della sintassi, e `--version` che stampa il numero di versione. Se l'opzione estesa deve indicare un valore questo deve essere specificato in forma di assegnazione con un `=`, ad esempio `--tabsize=80`.

Infine una menzione speciale per due casi particolari; in genere la combinazione `--` viene utilizzata per indicare di aver completato le opzioni,³⁸ in modo tutti gli argomenti che seguono verranno usati direttamente senza essere considerati una opzione anche se cominciano per un `-`. Così se ad esempio avete in file il cui nome inizia per `-` vedrete che non è affatto facile cancellarlo con `rm`, dato che il comando si lamenterà di una opzione sbagliata; tutti i programmi infatti prima esaminano le opzioni, segnalando errori se ne trovano di inesistenti, e poi trattano gli altri argomenti. Per questo scrivendo `--` prima del nome del file l'interpretazione delle opzioni sarà terminata ed il nome (anche se inizia per `-`) sarà preso come argomento. Invece per l'uso del `-` vale la convenzione, per quei comandi che prendono come argomento un file, di considerarlo (a seconda del contesto) come sinonimo dello *standard input* o dello *standard output*.

2.2.2 I comandi per le ricerche sui file

La ricerca di uno specifico file all'interno del filesystem, o dei file con una certa serie di caratteristiche, è una operazione molto comune, e per questo sono stati sviluppati alcuni comandi molto flessibili, che permettono di effettuare le più complesse tipologie di ricerca.

Il primo fra i comandi usati per cercare i file l'abbiamo già incontrato in sez. 2.1.4, ed è `which`, che ci indica a quale file eseguibile corrisponde un certo comando, facendo una ricerca nel *PATH*. Questo però esegue la ricerca solo fra i comandi.

Il comando più veloce per cercare un file qualunque è invece `locate`, che come suggerisce il nome, serve a localizzare nel filesystem tutti i file che contengono nel loro pathname la stringa passata come argomento. Il vantaggio di questo programma è la sua velocità, esso infatti non

³⁸è un'altra delle funzionalità introdotte dalla sezione delle GNU libc che permette di gestire le opzioni a riga di comando.

effettua la ricerca scandendo il contenuto del disco, ma in piccolo database interno che contiene l'elenco di tutti i file presenti nel sistema.

Il comando riconosce l'opzione **-i**, che richiede che venga effettuata una ricerca *case insensitive*, e **-e** che richiede che sia verificata l'effettiva esistenza del file. Il comando inoltre consente l'uso come argomento di espressioni analoghe a quelle usate dalla shell per il *filename globbing*. Le altre opzioni e la descrizione completa del comando è al solito disponibile nella relativa pagina di manuale accessibile con **man locate**.

Il fatto che il comando si affidi ad un database ci fa capire immediatamente anche i suoi limiti: anzitutto la ricerca può essere effettuata solo per nome, ed inoltre è in grado di cercare solo i file già inseriti nel database. Questo viene in genere creato dal comando **updatedb** (vedi sez. 3.2.3) che viene eseguito in genere una volta al giorno (fra i lavori periodici di **cron** che vedremo in sez. 3.3.1), per cui se un file è stato creato da poco non potrete vederlo.

Per superare i limiti di **locate**, si può usare il comando **find**, che non utilizza un database, ma esegue la ricerca direttamente nel filesystem, al costo di una notevole attività su disco, e di tempi di esecuzione decisamente più lunghi. Si può ridurre il carico comunque facendo effettuare la ricerca su sezioni ridotte dell'albero dei file, il comando infatti prende come primo argomento la directory da cui iniziare la ricerca, che verrà eseguita ricorsivamente in tutte le directory sottostanti, se non si specifica nulla la ricerca partirà dalla directory corrente.

Il comando supporta quattro categorie principali di opzioni, descritte da altrettante sezioni della pagina di manuale (accessibile al solito con **man find**). La prima categoria (descritta nella sezione **OPTIONS**) contiene le opzioni vere e proprie, che controllano il comportamento di **find**, la seconda, (descritta nella sezione **TESTS**) contiene le opzioni di ricerca che permettono di selezionare i file in base ad una loro qualunque proprietà (nome, tipo, proprietario, i vari tempi, permessi, ecc.), la terza (descritta nella sezione **ACTIONS**) contiene le opzioni che permettono di specificare una azione da eseguire per ciascun file che corrisponde alla ricerca, la quarta (descritta nella sezione **OPERATORS**) contiene le opzioni che permettono di combinare fra loro diverse selezioni.

Le opzioni generiche, le principali delle quali sono riportate in tab. 2.7, permettono di modificare il comportamento del comando, ad esempio con **-maxdepth** si può limitare la ricerca ai primi livelli di sottodirectory, mentre con **-mindepth** la si può far partire da un certo sottolivello.

Opzione	Significato
-follow	dereferenzia i link simbolici.
-mount	resta nel filesystem corrente e non analizza sottodirectory in altri filesystem.
-maxdepth	seguita da un numero di livelli indica il massimo numero di volte che scende in una sottodirectory.
-mindepth	seguita da un numero di livelli indica quanti livelli di directory ignorare prima di iniziare la ricerca.

Tabella 2.7: Principali opzioni generiche di **find**.

Le maggiori potenzialità di **find** derivano dalla sua capacità di effettuare ricerche con i criteri più svariati, da quelli sul nome del file in varie forme (con **-name**, **-regex**, **-path**), a quelli per gruppo e utente (con **-group** e **-user**), secondo i permessi (con **-perm**), secondo i vari tempi (**-atime**, **-ctime**, **-mtime**) per tipo di file, filesystem su cui è il file, ecc. Un elenco delle principali opzioni di ricerca, con il relativo significato è riportato in tab. 2.8.

Alcune di queste opzioni vanno chiarite, ad esempio con l'opzione **-name** si può effettuare la classica ricerca sul nome del file, con tanto di supporto per le wildcard (che però vanno adeguatamente protette per evitarne l'espansione da parte della shell). La ricerca è effettuata esattamente sul nome del file così come è scritto nella directory che lo contiene, non sul suo pathname, se si vuole ricercare su quest'ultimo occorre usare **-path**.

Per tutte le opzioni che prendono un valore numerico (quelle sui tempi, gli identificatori, il

Opzione	Significato
-amin n	Un file acceduto n minuti fa, le opzioni -cmin e -mmin eseguono lo stesso controllo rispettivamente con i tempi di ultimo cambiamento e ultima modifica.
-atime n	Un file acceduto n giorni fa, le opzioni -ctime e -mtime eseguono lo stesso controllo rispettivamente con i tempi di ultimo cambiamento e ultima modifica.
-anewer file	Un file acceduto più recentemente di file , le opzioni -cnewer e -mnewer eseguono lo stesso controllo rispettivamente con i tempi di ultimo cambiamento e ultima modifica.
-gid n	Il group ID del gruppo proprietario è n .
-group group	Il gruppo proprietario è group .
-links n	Il file ha n hard link.
-name pattern	Il nome del file corrisponde al pattern pattern . Prevede anche -iname per una ricerca case insensitive.
-path pattern	Il pathname del file (comprese quindi le directory a partire dalla radice) corrisponde al pattern pattern . Prevede anche -ipath per una ricerca case insensitive.
-perm mode	I permessi corrispondono a mode .
-size n	La dimensione del file è n .
-type c	Seleziona sul tipo di file, il valore di c corrisponde alla lettera usata da ls e riportata in tab. 1.1.
-uid n	L'user ID del proprietario è n .
-user user	Il proprietario è user .

Tabella 2.8: Principali opzioni di **find** per la ricerca.

numero di link), che è stato indicato in tab. 2.8 con **n**, il comando permette una sintassi molto potente: specificando solo il numero si richiede una corrispondenza esatta, precedendolo con il segno **-** si richiede invece che il valore sia inferiore, mentre precedendolo con un **+** si richiede che sia superiore; è così allora che per esempio, nel caso dei tempi, si può richiedere che un file sia più vecchio o più giovane di un dato tempo. Così ad esempio se si vuole cercare i file modificati negli ultimi 5 minuti si dovrà fare:

```
piccardi@anarres:~/Truelite/documentazione/corso$ find . -mmin -5
.
./shell.tex
```

mentre se si vuol cercare quelli non acceduti da più di quindici giorni si farà:

```
piccardi@anarres:~/Truelite/documentazione/corso$ find . -atime +15
./ringraziamenti.tex
```

Una spiegazione a parte poi deve essere fatta per l'opzione **-perm**, il cui valore **mode** deve essere specificato in ottale, e supporta i due segni **+** e **-** come per gli altri valori numerici. In questo caso però, trattandosi di una maschera di bit, il significato è diverso. Come prima il valore senza segno richiede la corrispondenza esatta, questo però ci renderebbe impossibile selezionare per la presenza di uno o più bit senza curarsi dello stato degli altri (che è in genere il tipo di ricerca più utile). Per questo si possono usare le altre due forme, se si usa il segno **-** allora **mode** specifica la maschera dei bit dei permessi che devono essere presenti sul file (i bit nulli cioè vengono ignorati); se invece si usa **+** la richiesta è ancora più debole ed il file corrisponde purché almeno uno dei bit di **mode** sia attivo. In questo modo con **-mode** si può richiedere una condizione in cui siano attivi un bit e un altro, mentre con **+mode** una in cui siano attivi un bit o un altro.

Come accennato una seconda importante categoria di opzioni è quella relativa alle azioni; è possibile infatti, per ogni file che corrisponde al criterio di ricerca specificato, far eseguire una

certa azione. Se non si specifica nulla l'azione di default è quella di stampare il nome del file, equivalente alla opzione `-print`; ma si possono anche scrivere i nomi su un file qualunque usando l'opzione `-fprint file`, o usare vari formati.

Opzione	Significato
<code>-exec</code>	esegue un comando usando come argomento il nome del file.
<code>-print</code>	stampa il nome del file terminato con un a capo.
<code>-print0</code>	stampa il nome del file terminato con un carattere NUL (il valore 0).
<code>-fprint file</code>	scrive il nome del file sul file <code>file</code> .
<code>-ok</code>	come <code>-exec</code> ma chiede conferma del comando.

Tabella 2.9: Principali opzioni di `find` per specificare azioni.

L'elenco delle opzioni principali è riportato in tab. 2.9, ma quella di gran lunga più importante è `-exec` che permette di eseguire, per ogni file corrispondente alla selezione, un comando. La sintassi dell'opzione è complessa in quanto si deve inserire una riga di comando all'interno di un'altra, e ci sono delle convenzioni usate dal comando per passare i valori. Quando si usa `-exec` tutto quello che segue viene interpretato come una riga di comando fino a che non si incontra un carattere `;`, in detta riga si può fare riferimento al file che corrisponde con la stringa `{}`. Il problema è che tutti questi caratteri vengono interpretati dalla shell, e devono quindi essere adeguatamente protetti; allora se ad esempio si vogliono spostare tutti i file non acceduti da più di 15 giorni in una directory `old`, si potrà usare un comando del tipo:

```
piccardi@anarres:~/Truelite/$ find . -atime +15 -exec mv {\} old \;
```

La potenza del comando `find` è poi ulteriormente aumentata dal fatto che le varie opzioni precedenti possono essere combinate fra di loro con degli operatori logici. Ma se il significato di `-and` o `-or` può sembrare immediato nel caso di criteri di ricerca, diventa meno chiaro quando si ha a che fare con delle azioni. In realtà infatti il comando associa un valore logico ad ogni opzione, e quando si esegue una selezione il valore è automaticamente vero, lo stesso vale per tutte le azioni, tranne `-exec` (e derivate come `-ok`) in cui il valore è vero se il comando ha uno stato di uscita nullo, e falso altrimenti.

Il funzionamento di un operatore come `-and` (che è sottinteso se si specificano più opzioni) è che la seconda opzione (sia questa di ricerca, che una azione) viene eseguita solo se la prima è vera. Viceversa con `-or` la seconda opzione viene eseguita solo se la prima è falsa. Infine `-not` nega il risultato di una opzione.

Nel caso si combinino opzioni di ricerca tutto questo è del tutto influente riguardo il risultato del comando, che è quello che ci si aspetta intuitivamente (entrambe le condizioni di ricerca devono essere soddisfatte per `-and` o solo una per `-or`, o si inverte la selezione con `-not`), ma cambia profondamente quando ci sono di mezzo delle azioni come `-exec`, perché in tal caso l'esecuzione della seconda opzione dipende in maniera essenziale dal risultato della prima (se si chiede di eseguire due comandi ad esempio le cose dipendono dal risultato di quello che si esegue per primo).

Per questo ad esempio specificare con `-and` più comandi (o semplicemente scriverne più di uno, dato che in tal caso il `-and` è sottinteso) non significa affatto che essi saranno eseguiti tutti: lo saranno solo se tutti hanno successo, se uno non ha successo i successivi non saranno eseguiti. Qualora si voglia essere sicuri di eseguire tutti i comandi in una lista si può usare l'operatore `,` nel qual caso saranno eseguiti comunque tutti, ma si avrà un valore finale corrispondente all'ultimo della lista.

Abbiamo allora visto come `find` ci permette di trovare un file in base al nome e alle sue caratteristiche generiche, una ulteriore modalità di ricerca è quella che permette di effettuare

ricerche in base al suo contenuto. Il comando che implementa questa funzionalità è **grep**, insieme ai suoi confratelli evoluti (come **egrep**) che nella loro forma elementare servono a cercare una stringa di caratteri all'interno di uno o più file, ma permettono anche di effettuare ricerche estremamente evolute attraverso l'uso delle *espressioni regolari*.³⁹

L'uso elementare di **grep** è banale, il comando prende come primo argomento la stringa da cercare seguita dal nome del file (o dalla lista di file) in cui effettuare la ricerca. Il comando stampa in uscita ogni riga del file (o dei file, se se ne è indicati più di uno) nella quale ha rilevato una corrispondenza. Ad esempio:

```
piccardi@anarres:~/Truelite/documentazione/corso$ grep Dispense *.tex
Struttura.tex:%% Dispense amministrazione base
baseadm.tex:%% Dispense editor e amministrazione di base
corso.tex:%% Corso Linux : Dispense dei corsi GNU/Linux di Truelite
netadmin.tex:%% Dispense Amministrazione di rete
shell.tex:%% Dispense amministrazione base
struttura.tex:%% Dispense amministrazione base
```

Altre opzioni del comando sono **-i** che permette di effettuare ricerche *case insensitive*, **-r** che effettua la ricerca ricorsivamente, e **-v** che inverte il risultato della ricerca (cioè stampa le righe che non corrispondono alla stringa utilizzata). Di nuovo le opzioni del comando sono moltissime; si sono riportate in tab. 2.10 le più importanti, al solito si rimanda alla pagina di manuale per una descrizione più completa.

Opzione	Significato
-b	stampa la posizione nel file (in byte) in testa a ciascuna riga in output.
-c	non stampa le righe con le corrispondenze, ma solo il numero totale delle stesse.
-E	interpreta la stringa di ricerca come una espressione regolare estesa.
-e	indica esplicitamente la stringa di ricerca (serve a proteggere stringhe di ricerca che iniziano con "-").
-G	interpreta la stringa di ricerca come espressione regolare normale (il default).
-H	stampa il nome del file in testa a ciascuna riga di output (anche se si ricerca su un solo file).
-h	sopprime la stampa del nome del file in testa a ciascuna riga di output (quando sono più di uno).
-i	non distingue fra maiuscole e minuscole.
-m N	sospende la stampa in uscita dopo N corrispondenze.
-n	stampa il numero di riga del file in testa a ciascuna linea in uscita.
-P	interpreta la stringa di ricerca come espressione regolare in stile Perl.
-r	esegue la ricerca in forma ricorsiva, ripetendola anche per tutti i file contenuti nelle sottodirectory di quelle passate in ingresso (può essere specificata anche come -R).
-s	sopprime la stampa degli errori al riguardo di file non leggibili o non esistenti.
-v	inverte la selezione, stampa le linee non corrispondenti.

Tabella 2.10: Principali opzioni del comando **grep**.

Come gli altri comandi Unix anche **grep** legge, qualora non gli sia passato nessun file come argomento, dallo *standard input* e scrive sullo *standard output*; diventa allora evidente la sua

³⁹le *espressioni regolari*, o *regex*, dall'inglese *regular expressions*, sono una specie estensione del sistema del *filename globbing* (che abbiamo illustrato in sez. 2.1.4) in cui, attraverso una serie di operatori, si possono effettuare corrispondenze fra stringhe con un grado di complessità incredibilmente elevato, questo le rende allo stesso tempo uno degli strumenti più potenti ed uno degli argomenti più ostici del mondo Unix.

utilità come filtro per selezionare a piacere, sulla base delle opportune corrispondenze le righe di un file. Si noti come facendo così si possano effettuare ricerche sempre più mirate semplicemente concatenando in successione diverse chiamate al comando.

Come accennato la vera potenza di **grep** sta nel fatto che la ricerca non viene semplicemente eseguita sulla corrispondenza ai caratteri contenuti nella stringa passata come argomento, ma nel fatto che quest'ultima viene interpretata, attraverso l'uso di alcuni caratteri riservati, come un *pattern* all'interno del file, indicato con una sintassi speciale che è quella delle *espressioni regolari*.

La trattazione dettagliata delle espressioni regolari va ben al di là delle possibilità di questo testo,⁴⁰ in cui ci limiteremo a fornire la descrizione delle funzionalità più elementari. Una espressione regolare è in sostanza una stringa di caratteri che identifica un *pattern*, cioè una struttura ordinata di caratteri e stringhe, di cui cercare l'occorrenza in un file eseguendo l'operazione che viene usualmente chiamata *pattern matching*.

In maniera analoga ad una espressione matematica una espressione regolare viene costruita combinando delle espressioni elementari attraverso gli opportuni operatori. Gran parte dei caratteri (tutte le lettere, maiuscole o minuscole, e i numeri) di una espressione regolare non viene interpretata, e la loro presenza richiede semplicemente la presenza di un corrispondente carattere nel contenuto su cui si effettua la ricerca, alcuni caratteri sono invece riservati per svolgere il ruolo di operatori; uno di questi è la barra rovescia “\” con la quale si può richiedere l'interpretazione letterale del carattere successivo (bloccandone l'intepretazione come operatore).

Il carattere “^” viene utilizzato per identificare l'inizio di una riga, mentre il carattere “\$” serve ad identificarne la fine; pertanto si potranno identificare i commenti in un file di configurazione con l'espressione regolare “^#” mentre con l'espressione “^\$” si identificheranno le righe vuote. Il carattere “.” viene invece utilizzato per indicare un carattere qualunque.

Alcune delle espressioni usate nel *filename globbing* si ritrovano anche nelle espressioni regolari, anche se in questo caso assumono un significato leggermente diverso⁴¹ In particolare le parentesi quadre vengono utilizzate come nel *filename globbing* per indicare una lista o un intervallo di caratteri (ad esempio “[aeiou]” indica le vocali e “[a-z]” le minuscole), la differenza col *filename globbing* è che in questo caso il carattere “^” messo all'inizio viene interpretato come inversione della lista seguente (con “[^A-Z]” si indicano tutti i caratteri che non siano una lettera maiuscola). Oltre a liste e intervalli specificati direttamente si possono indicare fra parentesi quadre una serie di *classi* predefinite di caratteri con le espressioni riportate in tab. 2.11.

Classe	Significato
[:alnum:]	lettere (maiuscole e minuscole, indipendenti dalla localizzazione) e numeri.
[:alpha:]	lettere (maiuscole e minuscole, indipendenti dalla localizzazione).
[:cntrl:]	caratteri di controllo.
[:digit:]	numeri.
[:graph:]	caratteri stampabili (esclusi gli spazi).
[:lower:]	minuscole.
[:print:]	caratteri stampabili (caratteri vuoti compresi).
[:punct:]	punteggiatura.
[:space:]	caratteri vuoti verticali ed orizzontali (spazi, tabulatori, ritorni a capo).
[:upper:]	maiuscole.
[:xdigit:]	cifre esadecimali.

Tabella 2.11: Le classi di caratteri utilizzabili nelle espressioni regolari.

Anche i caratteri “*” e “?” assumono un significato simile a quello del *filename globbing*,

⁴⁰un buon testo sull'argomento è [3].

⁴¹questo è spesso causa di confusione ed errori.

ma nel caso delle espressioni regolari questo accade perché essi vanno a far parte di un gruppo particolare di operatori che sono chiamati *operatori di ripetizione*, riportati in tab. 2.12. Gli operatori di ripetizione si applicano all’oggetto che li precede (che sia un carattere singolo o una espressione complessa) e richiedono che l’occorrenza dello stesso avvenga un certo numero di volte, nel caso di “*” questo può essere un numero qualunque (compreso nessuna) mentre con “?” si richiede che sia al più una volta (di nuovo lo zero è compreso).

Operatore	Significato
?	L’espressione precedente può corrispondere zero o una volta.
*	L’espressione precedente può corrispondere da zero ad un qualsiasi numero di volte.
+	L’espressione precedente può corrispondere da uno ad un qualsiasi numero di volte.
{n}	L’espressione precedente deve corrispondere esattamente n volte.
{n,}	L’espressione precedente deve corrispondere n o più volte.
{n,m}	L’espressione precedente deve corrispondere fra n e m volte..

Tabella 2.12: Gli operatori di ripetizione nelle espressioni regolari.

Questo significato è di nuovo leggermente diverso da quello presente nel *filename globbing*, in particolare una espressione come “ab*” nel primo caso seleziona tutti i file il cui nome inizia per “ab” seguito da un numero qualunque di altri caratteri qualsiasi, mentre nel secondo caso corrisponde ad una linea che contenga una “a” seguita da un numero qualunque (compreso zero) di “b”, per cui anche una stringa come “ac” corrisponderebbe. Per riottenere lo stesso significato precedente con una espressione regolare occorrerebbe usare la stringa “ab.*”, in questo modo si richiede la presenza iniziale di entrambe le lettere “a” e “b” seguite da un qualunque numero (indicato da “*”) di qualunque altro carattere (indicato da “.”).

Negli esempi appena mostrati gli operatori di ripetizione sono applicati solo al carattere che li precede, è possibile però applicarli ad una intera espressione regolare mettendola fra parentesi tonde, usando quello che viene chiamato un *raggruppamento*. Ad esempio per trovare in questo testo gli errori di battitura in cui si era scritto due volte la parola *una* si è usato il comando:

```
piccardi@monk:~/Truelite/documentazione/corso$ grep -E '(una ){2,}' *.tex
shell.tex:mentre nel secondo caso corrisponde ad una una linea che contenga una
```

dove appunto si è richiesto che l’occorrenza della stringa “una ” avvenisse almeno due volte di fila.

Un raggruppamento viene a costituire quello che viene chiamato un *subpattern*; e non solo si può richiedere, con l’uso degli operatori di ripetizione la presenza multipla di uno stesso *pattern*, ma si può anche usare l’operatore “|” posto fra due raggruppamenti per richiedere la presenza alternativa di uno di essi. Così si può richiedere la presenza della parola *pluto* o della parola *pippo* nella stessa riga con:

```
piccardi@monk:~/Truelite/documentazione/corso$ grep '\(pluto\)|\(\pippo\)' shell.tex
opzione mentre, non necessitando questa di nessun parametro, \texttt{pippo},
\texttt{pluto} e \texttt{paperino} verranno considerati come argomenti che nel
...
```

In questo caso la sintassi varia a seconda che si usi la sintassi delle espressioni regolari semplici (quelle di default) nelle quali le parentesi tonde e la “|” devono essere protetti⁴² con la barra trasversa “\” o quella delle espressioni regolari estese (quelle attivate con l’opzione -E) dove la protezione non è necessaria.

⁴²all’interno dell’espressione regolare stessa, si noti infatti come si sia espressa quest’ultima fra degli apici per bloccare l’interpretazione degli stessi caratteri da parte della shell.

L'uso dei *subpattern* ha poi un'ulteriore vantaggio, e cioè che tutte le volte che uno di questi viene trovato, il suo valore viene memorizzato in un elenco progressivo (secondo la sua posizione nell'espressione regolare) e detto valore può essere referenziato (e riutilizzato all'interno della stessa espressione) con l'uso delle sequenze speciali “\1”, “\2”, ecc. Così se ad esempio si vogliono trovare tutte le righe in cui compare due volte, in posizione qualsiasi, una stessa identica combinazione di almeno 10 caratteri si potrà utilizzare l'espressione:

```
piccardi@monk:/usr/share/common-licenses$ grep -E '(.{10,}).*\1' GPL
patent must be licensed for everyone's free use or not licensed at all.
running the Program is not restricted, and the output from the Program
conspicuously and appropriately publish on each copy an appropriate
with the Program (or with a work based on the Program) on a volume of
Software Foundation, write to the Free Software Foundation; we sometimes
```

dove fra parentesi si è richiesta la selezione di 10 caratteri consecutivi qualunque e poi si è richiesto che la stessa selezione di caratteri comparisse di seguito sulla stessa linea (con interposti un numero qualunque di altri caratteri).

2.2.3 I comandi per visualizzare il contenuto dei file

Un primo comando che permette di visualizzare il contenuto di un file lo abbiamo già incontrato in sez. 2.1.5, affrontando l'uso di `cat`, ed in tale occasione abbiamo anche citato che se lo scopo è solo quello della visualizzazione del contenuto di un file esistono alternative migliori, che sono quelle che tratteremo adesso.

Il problema maggiore dell'uso di `cat` come visualizzatore è che questo scrive tutto sul terminale, senza possibilità di mostrare il contenuto del file un po' alla volta. Per questo sono stati allora creati tutta una serie di programmi studiati per mostrare il contenuto dei file una *pagina* alla volta (dove per pagina si intende la schermata del terminale), che per questo sono detti *pager*. Ad essi è dedicata anche una variabile di ambiente, `PAGER`, usata dai programmi che necessitano di visualizzare il contenuto di un file, per scegliere quale di questi lanciare.

Il primo programma usato per la visualizzazione è `more`, il quale prende come argomento una lista di file da leggere di cui stampa il contenuto sul terminale una pagina alla volta, attendendo che l'utente gli invii dei comandi da tastiera. Al solito la pagina di manuale riporta l'elenco completo delle opzioni usate per controllare il comportamento del programma, ad esempio con `-num` si può specificare un parametro che indica il numero di linee che devono essere stampate sullo schermo (utile solo quando il comando non riesce a determinarlo da solo) ed i vari comandi. Rimandiamo ad essa per le informazioni complete, qui faremo solo una breve panoramica sui principali comandi che si possono dare durante la visualizzazione, il cui elenco comunque può essere ottenuto direttamente durante l'uso del programma premendo i tasti `? o h`.

Una volta stampata una pagina `more` consente di passare a quella successiva con la pressione dello spazio, mentre l'uso del ritorno a capo permette di avanzare lo scorrimento di una riga alla volta. Si può interrompere la visualizzazione con `q`, mentre con `b` si può tornare una pagina indietro. Se si sono indicati più file con `:n` si può passare alla visualizzazione del successivo mentre con `:p` tornare al precedente. Con il tasto `/` si fa apparire un prompt dove inserire una stringa da ricercare all'interno del file.⁴³ Infine con `v` si può lanciare l'editor impostato con la variabile di ambiente `EDITOR` (gli editor sono trattati in sez. 2.4, quello usato di default è `vi`) per modificare il contenuto del file.

Il comando `more` è stato creato fin dagli albori di Unix, e la sua sintassi risente anche del fatto che i primi terminali erano delle telescriventi, dove lo scorrere avanti ed indietro significa semplicemente ristampare pezzi del file. Dato che ben presto tutti i terminali iniziarono a

⁴³in realtà si può usare una *regular expression*, e compiere quindi anche ricerche molto complesse.

supportare la riscrittura dello schermo, e che tutte le tastiere iniziarono ad avere i tasti di freccia, venne creato **less** come *evoluzione*⁴⁴ di **more**.

Le funzionalità di **less** sono analoghe, e supporta anche tutti i comandi precedentemente illustrati per **more**, ma il comando anche consente degli spostamenti più comodi, potendo navigare il contenuto del file avanti ed indietro con i tasti di freccia, pagina su e giù, ecc. Il comando poi supporta funzionalità avanzate come la possibilità di ridefinire dei *keybinding*, di lanciare dei programmi per pre-processare dei dati (ad esempio decomprimere al volo dei file compressi), ecc. Per i dettagli si faccia al solito riferimento alla pagina di manuale.

Un altro programma di visualizzazione, più utile in caso di file binari, è **od**, (da *Octal Dump*) che permette di stampare il contenuto di un file in forma numerica, usando vari formati: decimale, ottale (il default), esadecimale e pure direttamente in semplice ASCII. La modalità in cui viene stampato il contenuto è controllata dall'opzione **-t**, che prende come parametro una stringa indicante il formato, il cui primo carattere indica il tipo di rappresentazione scelta, secondo quanto riportato in tab. 2.13, mentre, nel caso di forma numerica, si può utilizzare un secondo carattere per indicare la dimensione in byte del numero in questione.

Carattere	Formato
a	caratteri ASCII, coi caratteri non stampabili riportati tramite un nome simbolico
c	carattere ASCII, coi caratteri non stampabili riportati in forma numerica preceduta dalla barra rovescia
d	decimale
f	virgola mobile
o	ottale
u	decimale senza segno
x	esadecimale

Tabella 2.13: I caratteri indicanti il formato per la stampa dell'output del comando **od**.

Una seconda opzione che permette di modificare il formato di stampa di **od** è **-A**, che stabilisce come deve essere stampato il numero progressivo che indica la posizione nel file; l'opzione prende come parametro uno dei caratteri **d**, **o**, **x**, o **n** dove i primi tre hanno lo stesso significato riportato in tab. 2.13 mentre **n** indica che non deve essere stampato nulla.

Altre due opzioni utili sono **-j** che permette di iniziare la stampa a partire da una certa posizione all'interno del file e prende come parametro il numero di byte da saltare, e **-n** che permette di specificare (passandolo come parametro) il numero di byte da stampare (altrimenti il comando stampa tutto il contenuto del file). Per le altre opzioni ed i dettagli di funzionamento del comando si faccia al solito riferimento alla pagina di manuale.

2.2.4 I comandi per suddividere il contenuto dei file

Ma al di là della necessità di leggere il contenuto di un file scorrendolo un poco per volta, si può essere interessati ad effettuare delle selezioni più mirate. La nostra scatola degli attrezzi dei comandi Unix fornisce allora due comandi specializzati, **head** e **tail**, che ci permettono di selezionare (nel caso scrivere sullo *standard output*) rispettivamente l'inizio e la fine del file. Entrambi usano l'opzione **-n** per indicare il numero di linee totali da selezionare (il default è 10), e **-c** per effettuare la selezione in byte invece che in linee. Al solito si faccia riferimento alla pagina di manuale per l'elenco completo e la descrizione dettagliata dei comandi.

In questo caso nostra cassetta degli attrezzi sembrerebbe mancare di un comando ulteriore che ci permetta di selezionare una sezione qualunque del file a partire da una certa riga **N** per finire con un'altra **M**. Ma questo è ancora una volta facilmente ottenibile concatenando i due

⁴⁴si, volevano davvero fare gli spiritosi!

comandi precedenti; basterà tagliare prima la coda del file con **head** e poi la testa con **tail**, costruendo una linea di comando del tipo:⁴⁵

```
head -n M file | tail -n $((M-N))
```

Vale la pena poi menzionare esplicitamente l'opzione **-f** di **tail** che quando usata fa sì che il comando non esca e continui a stampare ogni eventuale altro dato aggiunto in coda al file, permettendo così di seguire la crescita di quest'ultimo. Questa è una opzione molto utile per tenere sotto controllo i file di log, ed in generale tutti i file in cui altri programmi scrivono in coda i loro dati.

Come contraltare di **cat** (che si ricordi serve a concatenare il contenuto dei file) si può usare **split**, che viene usato per *tagliare a fette* un file. Il comando prende come argomento il file da *affettare*, e lo suddivide in tanti file di dimensione uguale che chiama progressivamente **xaa**, **xab**, **xac**, ecc. Se non specifica un argomento al solito **split** legge dallo *standard input*, consentendo così ad esempio di creare un file con un comando e suddividerlo al volo con una *pipe*). Dato che il comando crea le sue *fette* in ordine alfabetico, per ricomporre il file originario basterà usare un comando del tipo **cat x* > file**.

Aggiungendo un secondo argomento dopo il nome del file da suddividere si può specificare un prefisso diverso da **x** come intestazione dei nuovi file creati dal comando. La dimensione dei file viene specificata con l'opzione **-C** se le si vuole in linee o con **-b** se la si vuole in byte, quest'ultima supporta anche un valore del parametro con i suffissi **m** e **k** per indicare rispettivamente megabyte e kilobyte. Infine se due lettere non bastano per indicizzare i file che si generano si può usare l'opzione **-a** per specificarne un numero diverso. Per tutti i dettagli si faccia al solito riferimento alla pagina di manuale.

Se si vuole tagliare un file per colonne invece che per righe si può usare il comando **cut**. Il comando opera sul file passato come argomento (o sullo *standard input*, rendendo di nuovo possibile operazioni complesse e filtri ricorsivi), stampando le colonne selezionate sullo *standard output*. Con l'opzione **-c** si può creare la colonna selezionando i caratteri in base alla loro posizione rispetto all'inizio della riga. L'opzione prende una lista dei caratteri, separata da virgole, e supporta la presenza di intervalli, indicati con un **-**, così se si vuole ottenere la stringa dei permessi dall'output di **ls -l** basterà fare:

```
piccardi@anarres:~/Truelite/documentazione/corso$ ls -l *.tex | cut -c 1-10
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
```

L'utilità del comando è che oltre alle posizioni assolute, permette di effettuare la selezione in termini di *campi* delimitati da un carattere qualunque che può essere specificato con l'opzione **-d** (di default il comando usa come separatore il tabulatore), in tal caso si effettuerà la selezione di quali campi stampare con l'opzione **-f**, che indica la posizione in maniera analoga a **-c**. Così si potrà ad esempio stampare il proprio user ID con:

```
piccardi@anarres:~/Truelite$ cat /etc/passwd | grep piccardi | cut -d: -f 3
1000
```

⁴⁵dove si è usata la *arithmetic expansion* brevemente descritta in sez. 2.1.4.

Come contraltare a `cut`, il comando `paste` permette di concatenare file diversi in colonna. Il comando prende come argomenti i nomi di una serie di file, e produce in uscita un file le cui righe sono l'unione delle righe dei file in ingresso, separate da dei caratteri di tabulazione. Se non si specifica nessun file il comando legge dallo *standard input*, che può essere usato anche all'interno di una sequenza di file indicandolo con `-`.

Quando i file hanno dimensioni diverse il file prodotto sarà esteso alla lunghezza (in righe) del più lungo dei file in ingresso, in cui le righe finali avranno dei campi vuoti in corrispondenza alle righe mancanti nei file più corti. Questo comportamento può essere modificato usando l'opzione `-f` che ferma la generazione di nuove righe non appena si incontra la fine di uno dei file dati in ingresso.

Con l'opzione `-s` invece si può effettuare una *trasposizione* dei file, in cui il contenuto (in righe) di ciascuno, viene messo in colonna su di un'unica riga. Se si usano più file in ingresso saranno generate tante righe quanti sono i file.

Con l'opzione `-d` si possono modificare i caratteri usati per la separazione delle colonne, l'opzione prende come parametro una stringa i cui caratteri saranno usati in sequenza come separatori fra le varie righe, nell'ordine in cui li si sono indicati.

2.2.5 I comandi per filtrare il contenuto dei file

In questa sezione prenderemo brevemente in esame una serie di comandi che permettono di filtrare e manipolare in maniera automatica il contenuto dei file. Questi costituiscono in genere uno strumento estremamente potente, in quanto il loro inserimento all'interno di una concatenazione di comandi permette di eseguire delle operazioni di conversione in maniera rapida ed efficiente.

Il primo comando utilizzabile per manipolare il contenuto di un file è `tr`, il cui nome sta per *TRanslate*. Lo scopo del comando è quello di effettuare delle conversioni dei caratteri. Come tutti i comandi opera esso direttamente su un file qualora questo sia specificato a riga di comando altrimenti utilizza lo *standard input*, scrivendo il risultato sullo *standard output*; in tal modo lo si può utilizzare come una sorta di *filtro*.

Il comando, a parte le opzioni, può prendere uno o due argomenti. Il primo indica un insieme di caratteri di cui eseguire la ricerca nel file, il secondo, quando presente, indica la lista dei caratteri con cui il corrispondente del primo insieme deve essere sostituito. In genere quando si specificano due argomenti questi devono specificare due insiemi almeno della stessa dimensione. Qualora questo non avvenga il secondo insieme viene esteso ripetendone l'ultimo carattere, a meno di non usare l'opzione `-t` che tronca invece il primo insieme alle dimensioni del secondo. Se invece il secondo insieme è più lungo vengono utilizzati solo i caratteri iniziali.

Qualora si specifichi l'opzione `-d` il secondo insieme viene ignorato, e tutti i caratteri del primo insieme vengono cancellati. Se invece si specifica l'opzione `-c` il primo insieme viene considerato come quello dei caratteri che *non* devono corrispondere. Infine usando l'opzione `-s` si possono *strizzare* preventivamente le ripetizioni di un qualunque carattere del primo insieme trasformandole in un carattere singolo (che poi potrà essere sostituito).

Le liste dei caratteri in genere si possono esprimere direttamente con delle stringhe, il comando però supporta anche le classi di caratteri definite in tab. 2.11, e si possono usare gli identificatori ivi riportati. Inoltre l'utilizzo del carattere di *escape* `"\"` permette non solo di proteggere i caratteri speciali, ma di inserire negli insiemi caratteri non stampabili secondo quanto riportato in tab. 2.14.

Per la sintassi completa del comando ed i dettagli riguardo le varie forme che si possono utilizzare per specificare gli insiemi dei caratteri si faccia comunque riferimento alla pagina di manuale, al solito accessibile con `man tr`.

Un secondo insieme di programmi operano sul contenuto dei file di testo, principalmente a scopo di riformattarlo adeguatamente. Il primo di questi è `pr` che serve a suddividere il testo

Espressione	Significato
\NNN	carattere specificato col suo valore numerico ottale.
\\	barra trasversa.
\a	<i>bell</i> (suona sul terminale).
\b	cancella indietro (<i>backspace</i>).
\f	pagina nuova (<i>form feed</i>).
\n	a capo (<i>new line</i>).
\r	ritorno carrello (<i>return</i>).
\t	tabulazione orizzontale.
\v	tabulazione verticale.

Tabella 2.14: Caratteri speciali ad uso del comando `tr`.

in pagine, numerate progressivamente dal programma stesso, in un formato adatto alla stampa (da cui deriva il nome), il programma supporta numerose opzioni volte a configurare in maniera diversa le modalità con cui viene effettuata la suddivisione in pagine ed impostare le informazioni mostrate in testa e coda delle stesse.

Il secondo programma è `fmt` che riformatta il file in uscita paragrafo per paragrafo, permettendo di impostare caratteristiche come indentazioni della prima riga e numero di spazi e mantenere una lunghezza costante della riga. Di nuovo le varie modalità con cui si può compiere la formattazione sono controllate dalle opzioni, per le quali si rimanda alla pagina di manuale.

Il terzo programma è `fold` che si limita invece a troncare le righe troppo lunghe rispetto ad una dimensione fissa delle stesse specificabile con l'opzione `-w`. Questo è il più semplice dei comandi di formattazione, con forse l'eccezione di `n1`, che come indica il nome serve a numerare le linee di ciascun file. Di nuovo si può fare riferimento alle pagine di manuale per le singole opzioni.

Vale poi la pena citare i due programmi `expand` e `unexpand` che servono rispettivamente a convertire tabulazioni in spazi e viceversa. Entrambi prendono le opzioni `-t` che permette di specificare le dimensioni del tabulatore se seguita da un numero come parametro, o le posizioni delle tabulazioni, che devono invece essere specificate come lista separata da virgole. Per i dettagli di nuovo si può fare riferimento alla pagina di manuale.

Abbiamo lasciato per ultimo più importante dei comandi per filtrare il contenuto dei file: `sed`, il cui nome sta per *Stream EDitor*. Il programma serve appunto per eseguire una serie di trasformazioni su un flusso di dati come quelle potrebbero essere fatte da un normale editor su un file, in modo da poter utilizzare il comando in una catena di pipeline che mettano in grado di effettuare le volute modifiche nel passaggio dei dati dallo *standard input* allo *standard output*.

Il comando prende come primo argomento l'*espressione di editing* che indica le operazioni da eseguire, a meno che non si sia specificata quest'ultima nel contenuto del file indicato tramite l'opzione `-f` o direttamente come parametro per l'opzione `-e`. I restanti argomenti, se presenti, indicano i file su cui applicare la suddetta espressione; se non è presente nessun altro argomento lo script viene applicato sullo standard input. Le principali opzioni sono riportate in tab. 2.15.

Opzione	Significato
<code>-e</code>	indica esplicitamente l'espressione che indica i comandi di editing da applicare.
<code>-f</code>	indica un file contenente l'espressione dei comandi di editing.
<code>-n</code>	sopprime la stampa del <i>pattern space</i> che avviene alla fine del processo di ogni riga.
<code>-r</code>	usa le espressioni regolari estese.

Tabella 2.15: Principali opzioni del comando `sed`.

A meno di non utilizzare l'opzione `-n` il comando stampa tutto quanto è stato letto in ingresso (che sia stato modificato o meno) sullo standard output. È così possibile salvare il risultato del

filtraggio reindirizzando l'output del comando. L'opzione `-n` è fornita per permettere (all'interno dei vari comandi) di eseguire una stampa in uscita solo quando lo si ritiene opportuno (ed ottenere pertanto un risultato diverso).

La potenza di `sed` sta nella grande flessibilità dei comandi che possono essere dati tramite l'espressione di editing, alcuni di questi infatti utilizzano le espressioni regolari (le cui basi si sono illustrate in sez. 2.2.2) per individuare dei *pattern* nel file ed operare su di essi compiendo sostituzioni, cancellazioni, ecc.

Le espressioni di editing sono normalmente divise in *indirizzi* e *comandi*. Un indirizzo esprime l'insieme delle righe del file originario a cui si applica il successivo comando. Un singolo indirizzo indica la riga cui si applica il comando, con due indirizzi separati da una virgola si indica un intervallo di righe, mentre quando, come nella maggior parte dei casi, non si specifica nessun indirizzo, il comando si applica a tutte le righe del file. Infine se ad un indirizzo si fa seguire (prima del comando) il carattere `!` la selezione delle righe viene invertita.

In genere un indirizzo viene specificato tramite il numero della relativa riga (ad esempio `"10"` indica la riga 10, `"5,15"` indica le righe dalla quinta alla quindicesima e `"3,7!"` tutte le righe escluse quelle nell'intervallo fra la terza e la settima. Sono però possibili delle selezioni più complesse, come `"$"` che indica l'ultima riga del file, o `"/regexp/"` che seleziona le righe corrispondenti ad una espressione regolare; un elenco delle principali forme con cui si può specificare un indirizzo è riportato in tab. 2.16.

Espressione	Significato
<code>N</code>	selezione la N-sima riga.
<code>N,M</code>	seleziona le righe dalla N alla M.
<code>N~M</code>	selezione le righe a partire dalla N-sima a passi di M.
<code>/regexp/</code>	seleziona le righe che corrispondono all'espressione regolare <code>/regexp/</code> .
<code>\$</code>	seleziona l'ultima riga del file.
<code>N,+M</code>	seleziona M righe a partire dalla N-sima.

Tabella 2.16: Espressioni per specificare un indirizzo al comando `sed`.

Una volta specificato l'*indirizzo* la seconda parte di una espressione di editing è composta da un *comando*. Un comando viene sempre introdotto da un carattere di controllo, seguito da eventuali parametri. I comandi possono essere ripetuti e raggruppati con l'uso delle parentesi graffe, ed in genere vengono scritti uno per riga e possono essere letti da un file con l'opzione `-f`.

Per capire l'azione dei vari comandi occorre capire come `sed` effettua la manipolazione dei dati e dove questi sono mantenuti. Quando una riga viene letta da un file essa viene posta nel cosiddetto *pattern space* dove vengono effettuate tutte le operazioni e la manipolazione del contenuto: è cioè nel *pattern space* che si troveranno le modifiche eseguite al contenuto originale dalle operazioni di `sed`. Il comando prevede inoltre anche la presenza di un altro spazio lo *hold space*, inizialmente vuoto, dove è possibile inviare dati a partire dal *pattern space*, che possono essere ripresi successivamente; questo permette operazioni complesse in cui i dati sono opportunamente spostati e ricombinati all'interno di detti *spazi*.

L'elenco dei comandi più comuni è riportato in tab. 2.17; ma `sed` ne supporta molti altri. Un elenco completo con una descrizione sommaria si può trovare nella pagina di manuale, ma la documentazione completa, fornita anche di parecchi esempi, è disponibile solo nelle pagine *info*, accessibili con `info sed`.

Il più importante ed utilizzato (e l'unico che tratteremo esplicitamente) dei comandi di `sed` è `"s"` che permette di sostituire una sezione di testo nel *pattern space* con un'altra. La sintassi del comando è nella forma `s/ricerca/sostituzione/` ed in questa forma rimpiazza (all'interno del *pattern space*) la prima occorrenza della stringa `ricerca` con la stringa `sostituzione`.

La potenza del comando sta nel fatto che la stringa di ricerca viene specificata come espressione regolare, pertanto diventa possibile fare selezioni estremamente complesse; inoltre si possono

Espressione	Significato
q N	esce con codice di uscita pari a N.
p	stampa il <i>pattern space</i> .
d	cancella il <i>pattern space</i> e passa al ciclo seguente.
s	sostituisce il testo corrispondente ad una espressione regolare con un altro testo.

Tabella 2.17: Principali espressioni di comando per `sed`.

utilizzare i *subpattern* per selezionare pezzi di testo che possono essere riutilizzati nella stringa di sostituzione con le usuali espressioni `\1`, `\2`, ecc. consentendo così manipolazioni molto sofisticate.

Dopo la `/` finale si possono specificare degli ulteriori sottocomandi, ad esempio usando “g” si indica al comando di sostituire tutte le occorrenze della stringa di ricerca, e non solo la prima, con “p” si richiede la stampa del *pattern space* (si usa in genere in combinazione con l’opzione `-n`) mentre specificando un numero N si esegue la sostituzione solo per la N-sima corrispondenza trovata.

2.2.6 Altri comandi dei file

Raccogliamo qui una serie di altri comandi di uso abbastanza comune che non rientrano in nessuna delle precedenti categorie. Il primo di questi è `touch`, che viene usato in quasi tutti gli esempi per creare un file vuoto. In realtà il comando non serve a questo (dato che lo stesso compito si potrebbe fare in molti altri modi) quanto, come dice il nome, a *toccare* un file.

Se il file passato come argomento non esiste infatti il risultato del comando è quello di crearlo vuoto, ma se invece esiste l’effetto del comando è quello di modificare al tempo corrente i tempi di ultimo accesso e ultima modifica (si ricordi quanto illustrato in sez. 1.2.1). Il comando prende varie opzioni e permette di modificare solo il tempo di ultimo accesso, se usato con l’opzione `-a` o solo quello di ultima modifica, se usato con l’opzione `-m`. Le altre opzioni sono al solito sulla pagina di manuale.

Un altro programma molto utile è `sort`, che permette di ordinare il contenuto di un file. Il comando prende come argomento un file e ne stampa il contenuto con le righe in ordine alfabetico. Dato che se non si specifica nessun file il comando opera sullo *standard input*, può essere usato di nuovo in una catena di comandi per riordinare l’uscita di un altro comando. Così se si vuole riordinare un elenco basterà darlo in pasto a `sort`. Le opzioni permettono di controllare le modalità di ordinamento, ad esempio con `-b` si può dire al comando di ignorare gli spazi all’inizio delle righe, con `-r` di invertire l’ordinamento, con `-n` di ordinare le stringhe che contengono numeri sulla base del valore di questi e non di quello alfabetico (per avere 2 prima di 10), con `-f` di non differenziare fra maiuscole e minuscole. Per l’elenco completo si faccia al solito riferimento alla pagina di manuale.

Un altro particolare riordinamento del contenuto di un file viene eseguito dal comando `tac`, che serve a stampare il contenuto di un file alla rovescia, cioè a partire dall’ultima riga verso la prima.⁴⁶ Il comando in realtà divide un file in campi separati da un carattere, che di default è il ritorno a capo per cui i campi vengono a coincidere con le righe; con l’opzione `-s` però si può passare come parametro un diverso separatore, mentre con `-r` si può usare come separatore una espressione regolare. Per i dettagli conviene consultare la pagina *info* con `info tac` dato che la pagina di manuale è molto sintetica.

Un altro comando che permette di filtrare il contenuto di un file è `uniq`, che elimina le linee adiacenti uguali; il comando prende come argomento un nome di file (ma se non viene specificato legge lo *standard input*) e stampa il risultato sullo *standard output*. Al solito le varie

⁴⁶si, l’idea era proprio quella di fare `cat` al contrario...

opzioni permettono di controllare le modalità con cui vengono effettuati confronti: con `-i` si può ignorare la differenza fra maiuscole e minuscole, con `-d` si limita a stampare (senza rimuoverle) le linee duplicate, con `-s` si può specificare il numero di caratteri ad inizio riga da non inserire nel confronto. Altre opzioni, al solito dettagliate nella pagina di manuale, permettono anche selezioni più complesse.

Di nuovo considerata a se stante, l'utilità di un comando come questo può apparire limitata, ma basta pensare alle combinazioni con altri comandi per apprezzarne la funzionalità. Si consideri ad esempio la necessità di riunire elenchi di parole contenuti in più file (supponiamo siano `elenco1.txt`, `elenco2.txt`, ecc.), lo scopo è quello di avere un file con l'elenco completo in cui tutte le parole compaiono una volta sola; questo può essere ottenuto in un batter d'occhio con un comando come:

```
cat elenco*.txt | sort | uniq > elencofinale
```

Una ulteriore serie di comandi sono quelli che possono essere usati per fare dei sommari del contenuto di un file. Il più semplice è `wc`, (da *Word Count*) che viene usato per contare le parole contenute in un file. Il comando prende come argomento una lista di file (se non se ne specificano al solito viene usato lo *standard input*) di cui stampa il numero totale di linee, di parole e byte. In genere il comando stampa tutte queste informazioni insieme al nome del file, se ne sono specificate più di uno. Si può far stampare solo il numero di linee, di parole o di byte con le opzioni `-l`, `-w` e `-c`; l'opzione `-L` stampa la lunghezza della linea più lunga.

Altri comandi sono `cksum` e `md5sum` che stampano delle opportune *checksum* (delle *somme di controllo*,⁴⁷ o *hash* che confrontate permettono di verificare l'integrità di un file). Entrambi prendono come argomenti una lista di file, per ciascuno dei quali sarà stampato a video il risultato del calcolo, per `cksum` dalla lunghezza e dal nome, per `md5sum` solo dal nome.

Al solito se non si specifica nulla i comandi leggono dallo *standard input*. Inoltre `md5sum` supporta un'opzione `-c`, che permette di specificare un solo parametro, che in questo caso sarà un file che contiene una lista di risultati di precedenti invocazioni del programma. Il comando verrà applicato a ciascuno dei file elencati, segnalando eventuali differenze. Diventa così possibile effettuare direttamente un controllo di integrità.

2.3 Altri comandi

Dopo aver trattato i comandi che operano sui file, faremo una panoramica su una serie di altri comandi di varia utilità che non sono direttamente connessi alla gestione dei file, ma che risultano di grande utilità come quelli per la documentazione, per impostare i tempi del sistema, per eseguire manipolazione avanzate sulla redirectione ed in generale tutti i comandi che non hanno direttamente a che fare con la gestione dei file.

2.3.1 I comandi per la documentazione

Benché talvolta sia difficile trovare informazione sulle funzionalità più esoteriche, una delle caratteristiche di un sistema GNU/Linux è quella di essere fornito di una quantità impressionante di documentazione, tanto che una delle risposte più frequenti alle domande di chiarimento è RTFM.⁴⁸

Come accennato in sez. 2.1.4 ciascun comando di norma supporta da suo una opzione `--help` che permette di visualizzarne brevemente la sintassi. Dato che questa informazione è in genere

⁴⁷si chiamano così delle opportune funzioni matematiche che hanno la caratteristica di dare risultati molto diversi anche per piccole differenze nell'input. In particolare `cksum` usa un algoritmo chiamato CRC, che è piuttosto debole, cioè è più facile avere lo stesso risultato, `md5sum` usa un'altro algoritmo, detto MD5, più recente, che è meno soggetto ad errori, ma comporta più calcoli.

⁴⁸sigla che sta, a seconda dell'umore del momento, per *Read The Fine Manual* o *Read The Fucking Manual*.

solo uno stringato riassunto delle opzioni disponibili, la fonte primaria delle informazioni relative ai comandi è nelle *pagine di manuale*, fin qui abbondantemente citate, che si accedono con il comando **man**.

Tutti i comandi prevedono una pagina di manuale che si accede semplicemente con la sintassi **man comando**. In particolare poi gli sviluppatori di Debian hanno come impegno preciso quello di fornire per ogni pacchetto la relativa documentazione. Ma le pagine di manuale non fanno riferimento solo ai comandi, il sistema infatti origina fin dai primi Unix e prevede la documentazione di tutto il sistema.

Per questo le pagine di manuale sono divise in sezioni, il cui numero è quello che compare fra parentesi nella prima riga di ciascuna pagina dopo il nome del comando in maiuscolo. Ciascuna sezione contiene la documentazione relativa ad un certo argomento, secondo la classificazione riportata in tab. 2.18.⁴⁹

Sezione	Significato
(1)	programmi eseguibili o comandi di shell.
(2)	system call (funzioni fornite dal kernel).
(3)	funzioni di libreria.
(4)	documentazione sui file di <code>/dev</code> .
(5)	formati dei file di configurazione.
(6)	giochi.
(7)	varie (convenzioni, informazioni generiche su argomenti).
(8)	comandi di amministrazione.

Tabella 2.18: Sezioni delle pagine di manuale.

Con il comando **man** si richiama la pagina di manuale, dove in genere si trova una documentazione esaustiva e dettagliata della sintassi e delle opzioni di un comando o del formato e del significato delle direttive di un file di configurazione. Il comando supporta una serie di opzioni di formattazione e per inviare l'output su stampante, che al solito sono descritte in dettaglio nella sua pagina di manuale, che come per gli altri comandi si accede con **man man**.

Si tenga presente che il comando **man** richiama la pagina relativa al nome che si è passato come argomento, cercando in sequenza⁵⁰ nelle varie sezioni e restituendo la prima che trova. Per questo se esistono più versioni della stessa pagina in sezioni diverse (come ad esempio per il comando **passwd** e per il file `/etc/passwd`) verrà mostrata solo la prima, se si vuole accedere alla seconda si dovrà richiamarla esplicitamente indicando la sezione con un qualcosa del tipo **man 5 passwd**.

Il sistema delle pagine di manuale (torneremo sulla sua configurazione in sez. 3.1.5) permette però di verificare se esistono più pagine associate ad uno stesso nome con il comando **whatis**, che stampa l'elenco delle pagine ad esso corrispondenti, così ad esempio avremo:

```
piccardi@anarres:~/Truelite/documentazione/corso$ whatis passwd
passwd (1)          - change user password
passwd (5)          - The password file
```

Un'altra funzionalità utile del sistema è fornita dal comando **apropos** che permette di effettuare la ricerca della parola passata come argomento fra le descrizioni brevi dei comandi che compaiono nella intestazione delle pagine di manuale (quelle appena mostrate anche nell'output di **whatis**) per cui potremo eseguire la ricerca:

⁴⁹in realtà qui si sono messe solo le sezioni "classiche", ci possono essere altre sezioni specifiche installate insieme ad alcuni pacchetti come quelle della documentazione del *perl* (un linguaggio di programmazione).

⁵⁰si tenga presente che la sequenza esatta non è detto sia quella di tab. 2.18, in realtà essa si imposta nella configurazione del sistema di gestione delle pagine di manuale, trattata in sez. 3.1.5.

```

piccardi@anarres:~/Truelite/documentazione/corso$ apropos "user password"
chage (1)          - change user password expiry information
passwd (1)         - change user password

```

e si noti come si siano usati i doppi apici per effettuare una ricerca su una stringa contenente uno spazio, dato che altrimenti si sarebbero passate due stringhe al comando.

Una seconda fonte di informazioni è costituita dal sistema di *help on line* fornito dal comando **info**. In questo caso si tratta di una forma alternativa di strutturazione delle informazioni che usa un formato diverso e che aggiunge, rispetto alle pagine di manuale, delle caratteristiche più avanzate e molto comode come: la possibilità di navigare usando link ad altre pagine, una organizzazione gerarchica strutturata ad albero con nodi e sezioni, la possibilità di consultare indici, ecc. Grazie a questa struttura **info** permette anche di accedere a documenti più complessi di una semplice pagina di manuale, e vi si trovano una grande quantità di manuali di grandissima importanza, come tutta la documentazione dei sistemi di sviluppo, dei linguaggi, delle librerie, ecc.

Data le sue caratteristiche evolute *info* è il formato raccomandato dal progetto GNU, che lo considera alternativo rispetto alle pagine di manuale. Molti però continuano ad utilizzare quest'ultime per cui alla fine i due sistemi si trovano a convivere. In genere il contenuto delle pagine di manuale è direttamente accessibile anche con il comando **info comando**; se però ci si limita a richiamare il comando **info** ci verrà mostrata la radice del sistema dove sono elencati tutti i documenti installati, e sarà possibile iniziare la navigazione spostandosi con le frecce, seguire i link premendo invio, tornare al livello precedente premendo **u**, passare al nodo successivo con **n**, tornare al precedente con **p**, andare all'indice con **i**, effettuare una ricerca con **/**, ed infine visualizzare i vari comandi disponibili con **?**.

Infine si tenga presente che in genere i singoli pacchetti dei vari programmi vengono distribuiti la documentazione prodotta direttamente dagli autori che di norma, secondo il FHS, si trova nella directory **/usr/share/doc/nomeprogramma**. A questa poi si aggiunge il grande patrimonio degli HOWTO, una serie di documenti sul *come fare* a riguardo di un certo argomento, disponibili in diversi formati (dal testo puro, all'HTML, al PDF) raccolti dal Linux Documentation Project, e di norma installati dalle varie distribuzioni insieme all'altra documentazione in **/usr/share/doc/HOWTO/**.

Infine quando non si riescono a trovare informazioni nel sistema si può ricorrere direttamente ad internet. Una delle risorse più usate è quella degli HOWTO del Linux Documentation Project, che sono continuamente aggiornati sul sito del progetto, <http://www.tldp.org>. Qui sono disponibili anche delle guide, veri e propri libri su vari argomenti come l'amministrazione di sistema, il kernel, ecc. La traduzione italiana di questi documenti si può trovare sul sito del PLUTO (gruppo storico di utenti Linux italiani) nella sezione dedicata all'*Italian Linux Documentation Project* all'indirizzo <http://ildp.pluto.it/>.

Una grande quantità di informazioni, e l'aiuto di altri utenti, può essere trovato sui gruppi di discussione di *usenet*, una rete internazionale di distribuzione di notizie in cui chiunque (usando il relativo client) può lasciare messaggi e ricevere risposte (la filosofia di funzionamento del sistema è ripresa da quella delle bacheche pubbliche delle università).

In tal caso il proprio provider deve aver fornito un accesso ad un *news server* che riceve i gruppi dedicati alle discussioni su Linux, che nella gerarchia internazionale⁵¹ sono quelli che si trovano sotto **comp.os.linux**, mentre quelli della gerarchia italiana stanno sotto **it.comp.os.linux**. In tal caso è comunque opportuno leggere in anticipo le FAQ (*Frequently Asked Question*) che

⁵¹tanto per cambiare, dato che anche questo servizio è nato su Unix, i gruppi di discussioni sono organizzati per temi, e strutturati ad albero in una gerarchia per cui si va da un tema più generale (**comp** per i computer) a temi sempre più specifici (**comp.os** per i sistemi operativi, **comp.lang** per i linguaggi di programmazione, ecc.).

raccolgono una serie di risposte alle domande più ricorrenti, onde evitare di ripetere domande banali che non sarebbero accolte molto amichevolmente.

Infine l'utilizzo dei motori di ricerca è un ottimo sistema per recuperare le informazioni pubblicate su internet. Fra queste troverete le versioni pubblicate degli HOWTO e delle FAQ (anche se queste vengono spesso distribuite come documentazione nel sistema, non è detto che siano complete o aggiornate) ma anche altre risorse come le annotazioni ed i log lasciati in innumerevoli pagine web e gli archivi delle liste di discussione dei vari progetti in cui qualcuno può aver già avuto il vostro problema e trovato una risposta.

2.3.2 I comandi per la gestione dei tempi

Prima di accennare ai principali comandi per la gestione di tempo e date in GNU/Linux occorre una breve introduzione sulla gestione del tempo nei sistemi Unix. Storicamente i sistemi unix-like hanno sempre mantenuto due distinti tipi di tempo all'interno del sistema: essi sono rispettivamente chiamati *calendar time* e *process time*, secondo le definizioni:

calendar time : detto anche *tempo di calendario*. È il numero di secondi dalla mezzanotte del primo gennaio 1970, in *tempo universale coordinato* (o UTC), data che viene usualmente indicata con 00:00:00 Jan, 1 1970 (UTC) e chiamata *the Epoch*. Questo tempo viene anche chiamato anche GMT (*Greenwich Mean Time*) dato che l'UTC corrisponde all'ora locale di Greenwich. È il tempo su cui viene mantenuto l'orologio del kernel, e viene usato ad esempio per indicare le date di modifica dei file o quelle di avvio dei processi.

process time : detto talvolta *tempo di processore*. È il tempo usato internamente dal kernel per le sue temporizzazioni. In genere è legato alle interruzioni del timer, e viene impiegato per tutti i calcoli dello scheduler, è pertanto il tempo in cui viene misurato il tempo di esecuzione dei processi.

In genere il tempo a cui si fa riferimento è sempre il *calendar time*, il *process time* viene usato soltanto dai comandi che riportano le proprietà specifiche dei processi (come `ps` o `top`) relative ai tempi di esecuzione degli stessi. Oltre a questi due, già trattati in sez. 1.3.1, l'unico altro comando che usa il *process time* è `time`, che prende come argomento una riga di comando da eseguire, e stampa a video, alla terminazione dell'esecuzione di quest'ultima, tre valori di tempo espressi in *process time*, che sono il tempo totale impiegato per l'esecuzione del programma (con la sigla **real**), il tempo passato nell'esecuzione di codice in user space (con la sigla **user**), ed il tempo passato nell'esecuzione di codice dentro il kernel, cioè all'interno delle system call invocate dal processo (con la sigla **sys**), ad esempio:

```
piccardi@monk:~/Truelite/documentazione$ time ls
CVS  README  corso  internet-server  ldap  lucidi  samba

real    0m0.030s
user    0m0.000s
sys     0m0.010s
```

e si noti come il tempo *reale* è sempre maggiore degli altri due in quanto tiene conto anche del tempo in cui il processo è stato in stato di *sleep* in attesa di I/O.

Il comando prende varie opzioni, le principali delle quali sono `-o` che permette di specificare un file su cui scrivere i risultati al posto dello *standard output*, e `-f` che permette di specificare un formato per i tempi. La descrizione completa del comando (comprese le stringhe usate per l'opzione `-f`) si trova al solito nella pagina di manuale.

Tutti gli altri comandi relativi ai tempi hanno a che fare con la gestione del *calendar time*. Qui si aggiungono ulteriori complicazioni; la prima è che esistono due orologi, quello di sistema,

usato dal kernel per tutte le sue operazioni, ed aggiornato via software dal kernel stesso, e l'orologio hardware che funziona in maniera del tutto indipendente e che rimane in funzione anche quando la macchina è spenta.⁵² Di solito il kernel lo legge all'avvio per impostare il valore iniziale dell'orologio di sistema e non lo considera più.

La seconda complicazione è che il kernel non conosce assolutamente niente dei fusi orari. Per lui il tempo è assoluto, e fa sempre riferimento al tempo standard universale (l'UTC appunto); tutta la gestione dei fusi orari è demandata alle applicazioni in user space che, tutte le volte che leggono un tempo, devono essere in grado di convertirlo nell'ora locale.⁵³

Se ci si pensa questa è la scelta più logica: non solo si evita di inserire una serie di informazioni complesse all'interno del kernel, ma facendo così i tempi di sistema sono sempre coerenti, e non dipendono dal fuso orario in cui ci si trova (nel senso che i file modificati un'ora fa risulteranno sempre con l'ora giusta qualunque cambiamento sia stato fatto nel fuso orario). Purtroppo altri sistemi operativi usano l'ora locale per l'orologio di sistema che coincide con l'orologio hardware, con tutti i problemi che questo comporta quando si deve cambiare fuso orario (con file che possono anche risultare essere stati creati nel futuro), e soprattutto di incompatibilità dei tempi in caso di dual boot.

Il comando che permette di leggere ed impostare l'orologio di sistema è **date**. Se usato senza argomenti questo si limita a stampare data ed ora corrente nel formato standard:

```
piccardi@monk:~/Truelite/documentazione/corso$ date
Fri Sep 19 12:45:42 CEST 2003
```

il comando prende come argomento o il tempo da impostare (operazione privilegiata che può eseguire solo l'amministratore) o un formato di stampa per il tempo che modifica l'output del comando facendogli stampare solo le parti di data volute.

Se si vuole impostare la data questa deve essere specificata con una stringa nella forma **MMDDhhmm** **[CCYY]** **[.ss]**, dove i termini fra parentesi quadra sono opzionali. Con **MM** si indica il mese (in numero), con **DD** il giorno, con **hh** l'ora, con **mm** il minuto, mentre l'anno si indica o con le cifre finali **YY** o per intero con tanto di secoli come **CCYY**, infine i secondi **ss** devono essere preceduti da un punto; ad esempio si può provare ad impostare l'orologio con:

```
piccardi@monk:~/Truelite/documentazione/corso$ date 09191854
date: cannot set date: Operation not permitted
Fri Sep 19 18:54:00 CEST 2003
```

e l'operazione fallisce, dato che non si è l'amministratore, ma il tempo che si voleva impostare viene comunque stampato a video.

Se invece l'argomento che si passa inizia con un **+** esso viene interpretato come una stringa di formattazione per il risultato del comando. La stringa usa il carattere **%** per indicare una direttiva di formattazione che verrà sostituita dall'opportuno valore, tutti gli altri caratteri resteranno immutati. Le principali direttive sono riportate in tab. 2.19, l'elenco completo è nella pagina di manuale.

Il comando prende inoltre varie opzioni, le principali delle quali sono **-d** che permette di specificare una data da stampare al posto del tempo corrente, e **-s** che permette, con lo stesso formato, di specificare l'impostazione dell'orologio di sistema. L'utilità di queste due opzioni è che la stringa che prendono come parametro può essere nelle forma più varie, e non solo riconosce tutti i formati standard che si ottengono con le direttive di tab. 2.19, ma anche descrizioni verbali (solo in inglese, però) come **8 day ago**, o **1 month**. La descrizione di queste stringhe può essere

⁵²è l'orologio che si trova sulla scheda madre, ed è alimentato dalla batteria che si trova su di essa.

⁵³il tutto è fatto attraverso delle opportune funzioni di libreria, che permettono di eseguire il compito in maniera trasparente.

Direttiva	Significato
%%	il carattere %.
%a	il nome del giorno della settimana abbreviato, secondo la localizzazione.
%A	il nome del giorno della settimana, secondo la localizzazione.
%b	il nome del mese abbreviato, secondo la localizzazione.
%B	il nome del mese, secondo la localizzazione.
%c	data e orario, secondo la localizzazione.
%d	giorno del mese, nella forma (01..31).
%D	data, nella forma mm/dd/yy).
%H	ora del giorno, nella forma (0..23).
%I	ora del giorno, nella forma (1..12).
%m	mese, nella forma (01..12).
%M	minuto, nella forma (00..59).
%r	orario nella forma (hh:mm:ss [AP]M) su 12 ore.
%T	orario nella forma (hh:mm:ss) su 24 ore.
%S	numero di secondi.
%w	giorno della settimana, nella forma (0..6) a partire dalla domenica.
%x	data nella rappresentazione secondo la localizzazione.
%y	anno abbreviato alle ultime due cifre.
%Y	anno completo.
%Z	nome convenzionale del fuso orario.

Tabella 2.19: Direttive di formattazione per il comando `date`.

trovata solo nelle pagine info del comando, accessibili con `info date`, dove si trova anche la trattazione completa tutte le funzionalità.

Il secondo comando che riguarda la gestione del tempo è `hwclock`, che permette di impostare l'orologio hardware. Il comando non prende argomenti, e lo si utilizza attraverso le opzioni. Se non si specifica nessuna opzione il comando si limita leggere il valore dell'orologio hardware e a stamparlo sullo *standard output*, con un risultato del tipo:

```
monk:/home/piccardi/Truelite/documentazione/corso# hwclock
Tue Sep 23 15:36:58 2003  -0.763201 seconds
```

(risultato che si ottiene anche con l'opzione `-r` o `--show`).

Il comando poi permette di impostare l'orologio hardware con l'opzione `--set`, cui deve seguire un `--date` che specifichi la data da usare nel formato con cui la si specifica anche per `date`. Si può però usare l'opzione `-w` (o `--systohc`) per impostare l'ora direttamente al tempo segnato dall'orologio di sistema. Viceversa l'opzione `-s` (o `--hctosys`) imposta l'orologio di sistema al tempo di quello hardware.

2.3.3 I comandi di ausilio per la redirectione

Nonostante la grande flessibilità degli operatori di redirectione della shell, esistono situazioni in cui il loro uso non è sufficiente a fornire le funzionalità necessarie, per questo esistono dei comandi che permettono di estendere l'uso della redirectione.

Ad esempio uno dei fraintendimenti più comuni riguardo l'uso della concatenazione dei comandi è quello in cui si pensa di usare la *pipe* per passare come argomenti ad un comando successivo l'output di un comando precedente. Questo non è ovviamente possibile perché l'uso di una *pipe* consente solo di passare lo *standard output* di un comando sullo *standard input* del successivo, e non ha nulla a che fare con gli argomenti di quest'ultimo, che provengono dalla linea di comando.

È però possibile fornire questa funzionalità con l'uso del comando `xargs`, il cui compito è replicare il comportamento della shell, che legge la linea di comando e ne estrae gli argomenti. Il comando cioè effettua la lettura *standard input* e ne ricava una lista di argomenti da passare

ad un successivo comando, in questo modo diventa possibile ricevere gli argomenti dall'output di un comando precedente tramite una *pipe*.

Il comando **xargs** prende come argomenti un comando da eseguire e le eventuali opzioni o argomenti iniziali, e per ogni riga ricevuta sullo *standard input* esegue il comando passandogli gli ulteriori argomenti, separati da spazi, così come letti dallo *standard input*. È possibile proteggere la presenza di spazi all'interno degli argomenti così come si fa con la shell, usando le virgolette o la barra trasversa. Se non si passa nessun argomento viene usato **echo** come comando di default.

In questo modo diventa ad esempio possibile applicare un comando qualunque ad una lista di file usando semplicemente **cat** e **xargs**. Se cioè si vogliono cancellare tutti i file contenuti nel file *lista*, si potrà usare un comando del tipo:

```
cat lista | xargs rm -f
```

ovviamente nel far questo occorrerà stare molto attenti, è sempre consigliabile infatti, quando si usa **xargs**, lanciare il comando senza argomenti per verificare che non si stiano facendo degli errori.

Il comando permette di impostare, con l'opzione **-e** una stringa da usare come marcatore per la fine del file, in modo da ignorare tutto quanto sarà letto dopo; con l'opzione **-t** inoltre permette di stampare a video il comando eseguito, e con **-p** di richiedere conferma dell'esecuzione sul terminale. Infine l'opzione **-0** richiede che le linee sullo *standard input* siano terminate da zeri, e interpreta letteralmente i vari caratteri (in modo da non interpretare spazi, virgolette, ecc.). L'elenco completo delle opzioni è riportato nella pagina di manuale, accessibile con **man xargs**.

Un secondo problema è quello che si ha quando si reindirige lo *standard output* su un file e si perde così la possibilità di esaminarlo sullo schermo. Anche in questo caso è possibile avere questa funzionalità usando un comando apposito **tee**. Il comando, come per la reindirizione, è in grado anche di eseguire la scrittura dei file in *append* usando l'opzione **-a**, al solito tutte le opzioni ed i dettagli di funzionamento si trovano nella pagina di manuale.

Un ultimo comando usato come ausilio per la reindirizione è **yes**, che ha il semplice compito (se invocato senza argomenti) di scrivere continuamente sullo *standard input* una serie di **y**. Anche in questo caso l'utilità di fare un comando per un compito così specifico diventa evidente solo considerando la capacità della shell di concatenare i comandi, per cui si può usare **yes** per *pilotare* automaticamente lo *standard input* di un comando che richiede conferme (come potrebbe essere **rm -i**). Il comando non ha nessuna opzione specifica e prende come argomento una stringa, che verrà usata al posto di **y**.

2.4 Gli editor di testo

Si è preferito mantenere in una sezione separata la trattazione di una classe di programmi, quella degli *editor* di testo, che vengono a costituire il principale strumento usato da tutti gli amministratori di sistema. Lo scopo di questa sezione è quello di mettere il lettore in grado di cavarsela con i principali editor disponibili in tutte le distribuzioni, poi con il tempo e l'esperienza ognuno finirà con l'adottarne uno come preferito.

2.4.1 Introduzione

Un'altra delle caratteristiche fondamentali di un sistema unix-like è, per le ragioni che tratteremo in dettaglio in sez.3.1.1, quella di mantenere le configurazioni dei programmi all'interno di semplici file di testo. Per questo motivo lo strumento più usati dai professionisti nell'amministrazione di sistema è quello dell'*editor* di testi.

Infatti nonostante stia crescendo la disponibilità di strumenti che permettono le più comuni operazioni di amministrazione tramite una interfaccia grafica, questi in genere mettono a

disposizione solo un limitato numero di opzioni, e non danno mai il completo controllo sul comportamento di un programma, che si può ottenere soltanto operando direttamente sui file di configurazione. Per questo motivo, l'*editor*, cioè un programma che permetta di leggere file di testo e modificarne il contenuto, diventa il principale strumento dell'amministrazione di sistema.

Inoltre quando un qualche problema sul disco, o il classico `rm -fR` dato un po' troppo allegramente da root avrà danneggiato qualche file essenziale o bloccato il sistema all'avvio, sarà sempre possibile usare una distribuzione su dischetto per rimettere le cose a posto, ma lì gli strumenti grafici non saranno disponibili. E quand'anche si usasse una delle tante distribuzioni su CD che sono in grado di fornire un desktop completo,⁵⁴ le configurazioni sul sistema danneggiato andrebbero comunque effettuate a mano.

Infine, anche se la diffusione della banda larga riduce il problema, usare una interfaccia grafica da remoto resta sempre (considerazioni di sicurezza a parte) estremamente lento (e sostanzialmente impossibile senza una buona ADSL), mentre con la riga di comando si può usare un terminale remoto e fare tutto quello che si vuole anche con la banda fornita da un semplicissimo modem analogico.

Dato che l'editor di testi ha sempre avuto questo ruolo fondamentale, è stato forse il primo programma applicativo sviluppato sotto Unix, e come per molte altre applicazioni di uso generale ne esistono molte versioni, con i nomi più pittoreschi, che pur svolgendo lo stesso compito di base, la modifica dei file di testo, vanno da quelli dotati solo delle funzionalità più elementari, come quelle di `ed`, uno dei primi editor che permette di operare solo su una linea alla volta⁵⁵ ai più complessi e sofisticati, come `emacs` che ha una quantità infinita di funzionalità diverse e viene paragonato da alcuni esagerati ad un secondo sistema operativo.

Nel prosieguo di questa sezione daremo una breve panoramica sull'uso dei più comuni editor di testo, ma restando nell'ottica dell'amministrazione di sistema tratteremo esclusivamente di editor accessibili da console, e quindi utilizzabili anche attraverso connessioni remote con dei semplici modem, 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`, i più blasonati fra gli editor di testi).

2.4.2 Un editor evoluto: emacs o xemacs

Per molti `emacs`⁵⁶ è l'editor. Sicuramente è il più potente; dentro `emacs` si può davvero fare di tutto: navigare fra i file e in internet, leggere la posta e le news di *Usenet*, programmare (con evidenziazione della sintassi e scorciatoie ai costrutti principali di qualunque linguaggio), fare debug, scrivere dispense come queste, giocare (a tetrax o a qualche avventura testuale), ed anche farsi psicanalizzare dal "*doctor*".

Qualunque cosa sia possibile fare con del testo con `emacs` si fa, infatti l'editor è programmabile,⁵⁷ e per un qualunque compito specifico sono state create delle estensioni più varie in grado di eseguire operazioni complesse, gestire automatismi, abbellire la visualizzazione, 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, e la preferenza personale nei suoi confronti, ne parleremo, considerato

⁵⁴ ovviamente, anche se questo fosse dotato degli strumenti grafici di amministrazione grafici necessari, questi opererebbero sul sistema corrente, non su quello danneggiato.

⁵⁵ quello che per questo viene chiamato un *editor di linea*, che origina dai tempi dei primi Unix quando i terminali erano delle vere telescriventi, ma i cui comandi si ritrovano ancora in altri editor e pure in programmi come `sed`.

⁵⁶ nome che in teoria vorrebbe dire *Extensible MACro System*, ma è stato ribattezzato in *Eight Megs And Constantly Swapping*, o *Escape Meta Alt Control Shift*.

⁵⁷ in un dialetto specifico del lisp, un linguaggio funzionale estremamente potente che consente una enorme espandibilità del programma.

poi che molti altri editor ne hanno copiato la sintassi e le modalità d'uso, o forniscono modalità di comando *compatibili*.

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

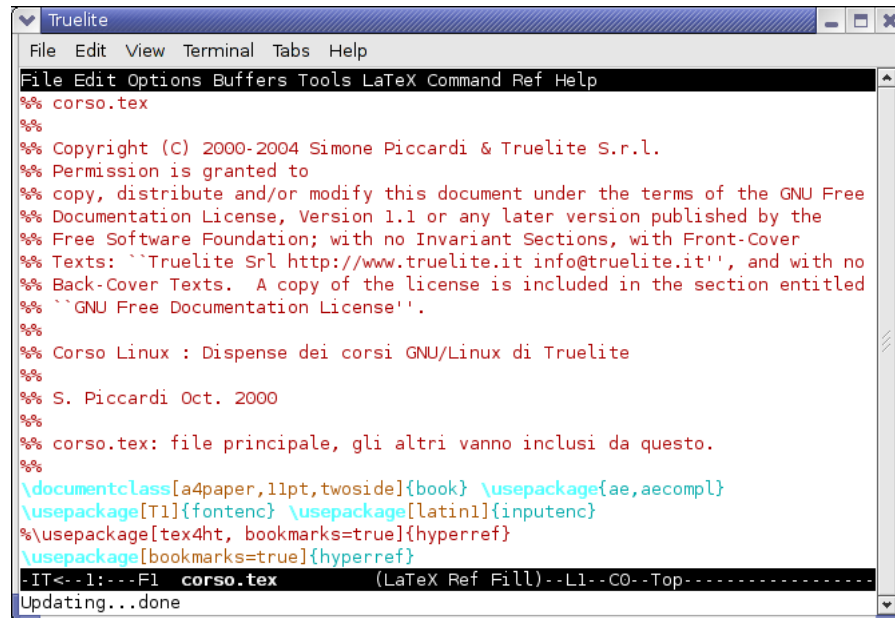


Figura 2.1: Schermata di avvio dell'editor **emacs**.

Come per tutti gli editor una sessione di **emacs** inizia invocando il comando seguito dal nome del file da modificare; in tal caso si otterrà una finestra come quella mostrata in fig. 2.1. Nella prima riga si trova il *menù* dei comandi, ad esso segue la sezione principale, dove compare il testo del file ed in cui ci si muove con le frecce, terminato da una *barra di stato* in cui compaiono varie informazioni (il nome del file, se sono state fatte modifiche, ecc.). Nella la riga finale viene tenuto il cosiddetto *minibuffer*, in cui compaiono brevi messaggi (come nell'esempio, che riporta la scritta (Updating...done)) ed in cui si scrivono gli argomenti dei comandi più complessi.

Uno dei concetti fondamentali di **emacs** è quello di *buffer*, qualunque porzione di testo venga utilizzata da **emacs**, con l'eccezione del *menù* e della *barra di stato* è mantenuta in un *buffer*, sia che si tratti di un file che si è aperto per lavorarci sopra, che del testo generato automaticamente nelle operazioni dell'editor (come le liste dei file da scegliere che compaiono quando si usa l'autocompletamento). Anche il *minibuffer* è un *buffer*, con la peculiarità di essere posizionato sull'ultima riga ed essere utilizzato per l'interazione con i comandi.

In genere un *buffer* viene visto su una *finestra* che ne mostra il contenuto ed è terminata da una *barra di stato* (nell'esempio ce n'è solo una), ma all'interno di una stessa istanza dell'editor possono anche esserci più *finestre* che possono essere aperte sullo stesso *buffer* o su *buffer* diversi (questo vuol dire che ad esempio la sezione centrale di fig. 2.1 può venire divisa in due). Questo permette, anche operando in console, di lavorare su due o più *finestre*, dato che ciascuna può a sua volta essere suddivisa a piacere, con il solo problema che se se ne creano troppe non si vedrà altro che barre di stato.

All'interno di una *finestra* ci si può spostare con le frecce e gli altri tasti di spostamento,⁵⁸ e scrivere del testo, questo verrà sempre inserito nella posizione in cui si trova il cursore (nell'e-

⁵⁸**emacs** prevede una grande quantità di combinazioni di tasti per andare avanti ed indietro, spesso tutt'altro che intuitive per chi proviene da altri sistemi, questo è una conseguenza che il programma è nato quando ancora le tastiere non avevano né frecce né altri tasti di funzione.

sempio all'inizio della prima riga in alto a sinistra). In genere il testo viene inserito spostando quanto già presente nel file, a meno di non porsi nella modalità di sovrascrittura premendo il tasto INS (verrà notificato nella barra di stato) da cui si esce ripremendo lo stesso tasto.

I comandi invece vengono dati con delle combinazioni che prevedono la pressione contemporanea di un *tasto modificatore*⁵⁹ e di una lettera. Molti comandi inoltre prevedono l'uso di due combinazioni di tasti eseguite in successione. Data la complessità e la grande quantità di comandi esistono due modificatori per i comandi, il primo è il classico *control* e ed secondo è *alt* il cui uso può essere simulato, per le tastiere che non ce l'hanno, premendo prima il tasto di *escape*, (ESC) e poi la lettera relativa.

Come già visto in sez. 1.3.4, dove la si è usata senza formalizzarne la definizione, per indicare la pressione contemporanea di un modificatore con un tasto qualunque X utilizzeremo una notazione del tipo C-X per l'uso di *control* e M-X per l'uso di *alt* (l'uso di M- deriva dal fatto che il secondo tasto modificatore viene chiamato “*meta*” in tutta la documentazione di **emacs**).

Operazione	Combinazione di tasti
aprire un file	C-x C-f seguito dal nome del file nel minibuffer (supporta il completamento automatico).
salvare un file	C-x C-s.
salvare con nome	C-x C-w seguito dal nome del file nel minibuffer (supporta il completamento automatico).
uscire	C-x C-c, se ci sono modifiche chiede se salvarle, scartarle o cancellare l'operazione.
annullare	C-_, C-/ o C-x u, ripetendo si torna ulteriormente indietro nell'annullamento.
seleziona	C-x spazio marca l'inizio di una regione e si porta in modalità selezione, poi basta posizionarsi nel punto finale.
taglia	C-w una volta selezionata una regione.
incolla	C-y.
cancella	C-d in avanti e il tasto di backspace all'indietro.
ricerca	C-s seguito dal testo nel minibuffer esegue un 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	(un comando) C-g o tre volte ESC.

Tabella 2.20: I principali comandi di **emacs**.

In tab. 2.20 si è riportata la lista dei comandi principali che sono attivi in qualunque modalità⁶⁰ si usi **emacs**; il programma ne mette a disposizione molti altri sia generici che specifici per il tipo di file su cui si sta lavorando.⁶¹

Molti di questi comandi, ad esempio tutti quelli che richiedono l'immissione di un nome di file, usano il minibuffer per ricevere i relativi dati, e stampare messaggi; inoltre in certi casi (come quello della specificazione di un file) la selezione supporta il meccanismo dell'autocompletamento (usando il tasto di tabulazione), e la capacità, qualora i completamenti siano multipli, di creare automaticamente una lista di selezione in una nuova finestra. In questo caso per spostarsi da una finestra all'altra occorrerà usare il comando C-x o (che ripetuto ritorna alla precedente), mentre per eliminare tutte le finestre presenti tranne quella dove è posizionato il cursore si potrà usare il comando C-x 1.

⁵⁹si chiamano così i tasti che hanno un effetto solo quando sono premuti in contemporanea ad altri tasti, come il tasto per la maiuscole, il *control*, ecc..

⁶⁰essendo **emacs** un editor programmabile esso può essere usato in modalità diverse a seconda del tipo di file su cui si sta lavorando, prevedendo in ciascun caso diverse serie di combinazioni di tasti e diversi comandi di manipolazione.

⁶¹ad esempio scrivendo queste dispense con Latex si hanno tutta una serie di comandi per mettere le parole negli indici, creare le intestazioni delle sezioni o delle tabelle, creare riferimenti a tabelle e figure presenti, ecc.

2.4.3 Un editor di base, vi

Uno dei editor più diffusi, presente fin dagli albori dei sistemi Unix è **vi**.⁶² 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 *modalità di operazioni* corrente in cui si trova il programma.

Questa caratteristica lo rende senz'altro il meno intuitivo e più difficile da usare per il novizio. Ma i fan(atici) tendono invece a considerarla una caratteristica 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, specie se capita di avere a che fare con una installazione che non ha attivato l'uso delle frecce.

Al contrario di **emacs** (di cui è il principale concorrente) **vi** si usa soltanto per manipolare file di testo, e non fornisce pertanto nessuna delle funzionalità più evolute di **emacs** (come la possibilità di fare debug dei programmi all'interno dell'editor facendo riferimento diretto al codice su cui si sta lavorando) ma resta comunque un editor molto potente.

Il principale vantaggio di **vi** è che essendo molto leggero e diffuso fin dalle prime versioni di Unix lo si trova installato praticamente su qualunque sistema e molto spesso è l'editor di default. Inoltre anche se le funzionalità del programma originale sono veramente minime, esistono alcune versioni più moderne, come **vim**, hanno introdotto alcune capacità avanzate, 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, dato che con le funzionalità sono di pari passo aumentate anche le dimensioni.

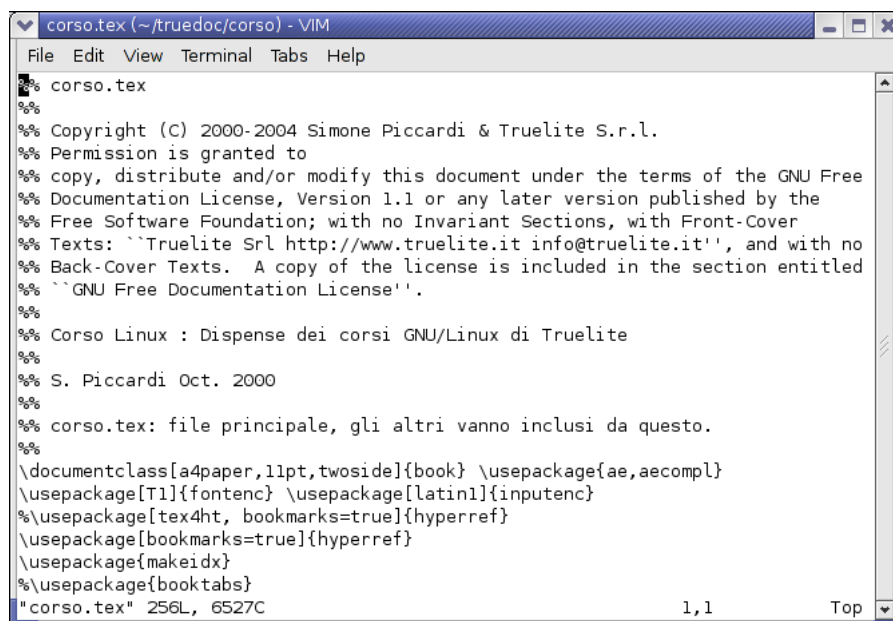


Figura 2.2: Schermata di avvio dell'editor vi.

Al solito **vi** si invoca passando come argomento il nome del file da modificare, il comando apre il file in modalità comando in una finestra unica (mostrata in fig. 2.2), dove compare il testo ed in cui ci si muove con le frecce,⁶³ lasciando libera l'ultima linea, usata per dare i comandi o ricevere le informazioni (nel caso il nome del file, la sua lunghezza in righe e caratteri e la posizione del cursore sulla prima colonna della prima riga).

⁶²purtroppo i perché di questo nome si sono perse nei meandri del tempo, forse erano due lettere rimaste libere scelte a caso ...

⁶³le versioni installate in tutte le distribuzioni di Linux se non altro supportano nativamente l'uso delle frecce.

Tutti i comandi di `vi` sono eseguiti con pressioni di singoli tasti, ma la possibilità di dare dei comandi dipende dalla modalità in cui si trova l'editor al momento, se in fatti si è in modalità di inserimento i tasti verranno utilizzati per scrivere e non interpretati come comandi. Quando si avvia il programma questo si pone in modalità normale, e in questo caso i tasti vengono interpretati come comandi.

In questa fase molti dei comandi sono dati direttamente con la pressione di uno o più lettere, ma sullo schermo non viene visualizzato nulla (se non l'effetto del comando); tutti i comandi più complessi (ad esempio quelli che richiedono una scrittura di una espressione di ricerca o di un nome di file) devono essere invece dati dalla riga di comando, che si attiva premendo `:`, in questo caso il cursore si sposta dal testo e si pone sulla riga finale dove si potrà inserire il resto dell'espressione.

La pressione dei tasti corrispondenti ai relativi comandi consente di cambiare modalità; ad esempio premendo `i` si passa in modalità di inserimento, e si può iniziare a scrivere in corrispondenza del cursore. Ma ci sono altri comandi che permettono di entrare in modalità inserimento, come `a` che vi entra ma riportandosi ad inizio riga o `A` che invece va a fine riga ed anche `o` che lo fa andando a capo creando una nuova riga.

È allora abbastanza frequente che un utente alle prime armi, una volta aperto un file, tenti di scrivervi e non veda nulla finché, premuto uno di questi tasti, non entra in modalità di inserimento. Il problema in genere è che una volta entrati in modalità di inserimento l'unico modo per uscirne è quello di premere il tasto di *escape* `ESC`, non sapendolo ci si troverà bloccati ed incapaci di uscire dal programma.

Una modalità simile a quella di inserimento è quella di rimpiazzo, in cui si entra con premendo `R`, in questo di nuovo tutti tasti assumeranno il loro valore letterale, ma quanto si scrive invece di essere inserito in corrispondenza al cursore, spostando il testo già presente, sovrascriverà quest'ultimo. Anche da questa modalità si può uscire premendo il tasto di *escape* `ESC`.

Operazione	Combinazione di tasti
aprire un file	<code>:ex file.</code>
salvare un file	<code>:w.</code>
salvare con nome	<code>:w nomefile.</code>
uscire	<code>:q</code> , ma se ci sono modifiche non esce; nel caso <code>:qw</code> le salva, <code>:q!</code> le scarta.
annullare	<code>u</code> solo sull'ultima modifica.
seleziona	<code>v</code> , e poi ci si sposta con i tasti di freccia, ma è una estensione esclusiva di <code>vim</code> .
taglia	<code>d</code> , ma vale con questo significato solo con una selezione fatta con <code>vim</code> .
incolla	<code>p</code> inserisce l'ultimo oggetto cancellato.
cancella	<code>d</code> seguito da una altra lettera che specifica cosa cancellare; <code>dw</code> una parola, <code>dd</code> una riga, <code>x</code> cancella un carattere.
ricerca	<code>/</code> seguito da un testo o una espressione regolare.
annulla	<code>ESC</code> annulla un comando.

Tabella 2.21: I principali comandi di `vi`.

In tab. 2.21 si è riportata la lista dei comandi principali usati per i compiti più comuni, si tenga presente però che essi possono essere specificati solo dalla modalità normale, se ci si è posti in modalità di inserimento prima bisognerà uscirne con `ESC`. Un elenco degli altri comandi può essere ottenuto con la documentazione interna, accessibile con `:h`.

Come accennato `vi` supporta, per le tastiere che non hanno frecce, lo spostamento nel file con i tasti `h`, `j`, `k`, `l` che effettuano rispettivamente lo spostamento a sinistra, basso, alto e destra. Inoltre una caratteristica dell'editor è che qualunque comando dato in modalità normale può essere moltiplicato se lo si fa precedere da un numero, per cui per abbassarsi di 10 righe si può scrivere qualcosa tipo `10j`.

Una delle caratteristiche che può lasciare più interdetti con **vi** è che non esiste il classico *taglia e incolla* in cui si seleziona una sezione qualunque del file, a meno che non usiate **vim** che definisce una modalità *visual* (in cui si entra con **v**) che implementa questa funzionalità. È possibile però cancellare caratteri singoli (con **x** per quello corrente, con **X** il precedente), parole (con **dw**) e righe (con **dd**), ed usare i moltiplicatori appena illustrati per cancellazioni multiple; si può poi incollare quanto appena cancellato (anche più volte se si usano i moltiplicatori) con **p**.

L'apertura di nuovi file deve essere fatta sulla riga di comando, e si esegue con **:ex**, altri comandi che si possono utilizzare sono **:w** che salva il file e **:x** che salva ed esce, ed è equivalente al normale comando di uscita con salvataggio che è **:qw**; l'uscita invece si può forzare con **:q!**, per questi due comandi esistono anche delle versioni da dare in modalità normale e cioè rispettivamente **ZZ** e **ZQ**.

Un'altra caratteristica interessante della linea di comando di **vi** è che è possibile dare dei comandi di shell senza uscire dall'editor; una volta entrati sulla linea infatti si può usare il carattere **!** come *shell escape*, si potrà cioè usare una sintassi del tipo **“:!ls”** per ottenere la lista dei file. Un'altra funzionalità presente sulla linea di comando è quella di sostituzione, che ha la stessa sintassi delle espressioni usate da **sed**, potremo cioè effettuare una sostituzione di una stringa con il comando **“:s/ric/sost/g”**, ma potremo anche usare le espressioni regolari con tutte le funzionalità che si sono descritte in sez. 2.2.5.

2.4.4 Gli altri editor

Un altro editor leggero e potente è **joe**, gli affezionati del DOS noteranno che usa la sintassi di *wordstar*, un word processor testuale di quell'epoca. Offre le tutte le normali funzionalità di un editor di testi ha un help in linea per i vari comandi che si attiva con **C-k h**, che lo rende piuttosto facile da usare. Inoltre di solito il comando si può usare in modalità di emulazione usando le sintassi di altri editor, ad esempio invocandolo come **jmacs** userà la sintassi dei comandi di **emacs**.

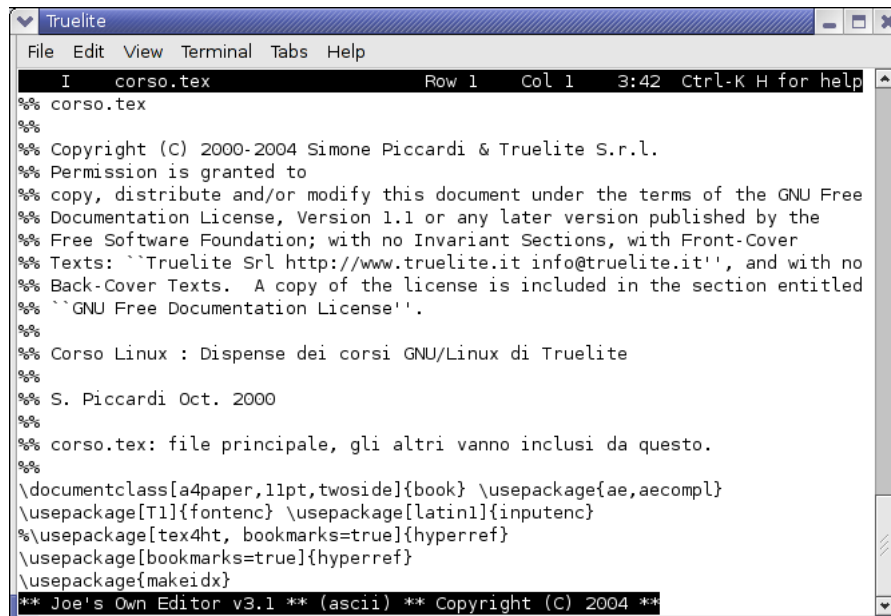


Figura 2.3: Schermata di avvio dell'editor **joe**.

Il comando prende al solito come argomento il nome del file che viene aperto e visualizzato nella finestra principale, mostrata in fig. 2.3. La finestra presenta una riga di stato in testa

contenente i soliti dati (file, posizione, lunghezza ed il suggerimento per ottenere la schermata di aiuto), seguita dalla sezione principale in cui compare il testo del 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 necessaria.

Operazione	Combinazione di tasti
aprire un file	C-k e seguito dal nome del file.
salvare un file	C-k x.
salvare con nome	C-k d seguito dal nome del file.
uscire	C-k x esce e salva, C-c esce e se ci sono modifiche chiede d eventualmente scarta.
annullare	C-_, mentre con C-^ si ripete l'ultima operazione.
seleziona	C-k b all'inizio e poi spostarsi con le frecce e dare C-k k alla fine.
taglia	C-k m muove la regione selezionata sulla posizione attuale.
incolla	C-k c copia la regione selezionata sulla posizione corrente.
cancella	C-k y in avanti e <i>backspace</i> indietro.
ricerca	C-k f seguito da un testo esegue la prima ricerca, C-L cerca l'occorrenza successiva.
annulla	(un comando) C-c.

Tabella 2.22: I principali comandi di joe.

In tab. 2.22 sono riportati i principali comandi per le funzionalità di base. Anche in questo caso il comando non supporta il concetto classico del *taglia e incolla*, ma una volta selezionata una sezione di testo consente solo o di spostarla o di copiarla.

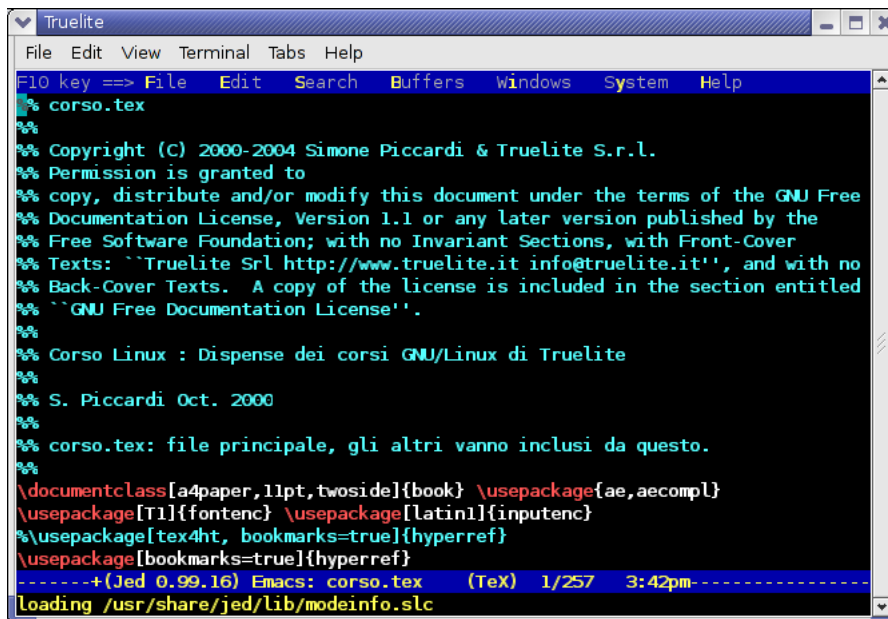


Figura 2.4: Schermata di avvio dell'editor jed.

Un'altro editor per la console è *jed*, scritto come reimplementazione di *emacs* in C, che è molto più leggero di quest'ultimo (ma anche molto meno potente). Supporta anch'esso però un linguaggio di programmazione interno, ed è pertanto in grado di fornire parecchie funzionalità avanzate, e utilizzare diverse sintassi per i comandi a seconda del tipo di file. Fornisce anche un'ampia capacità di evidenziazione della sintassi in console che con *emacs* è disponibile solo nell'ultima versione.

Essendo **jed** in sostanza un clone di **emacs** non staremo a ripeterne i comandi principali, che sono gli stessi di tab. 2.20, anche la composizione della finestra (una cui immagine è mostrata in fig. 2.4) è analoga a quella di **emacs** per cui basta rifarsi quanto detto in precedenza; il vantaggio di **jed** è che l'utilizzo del menù è molto più intuitivo e avviene direttamente in modalità semi-grafica anziché all'interno di finestre secondarie. Essendo un editor leggero ma molto potente, si trova su varie distribuzioni su dischetto.

Un altro editor abbastanza diffuso è **pico**; questo è derivato da **pine**, un client per leggere la posta elettronica in console, separando da esso l'editor interno usato per scrivere le email. Data la maggiore utilizzabilità rispetto a **vi** esso si è diffuso parecchio, il problema è che siccome **pine** non è software libero, non lo è neanche **pico**; per questo motivo è stato realizzato un clone completamente libero chiamato **nano** identico dal punto di vista dei comandi principali, ma più leggero (tanto da essere usato nei dischi di avvio di Debian) e con qualche capacità in più.

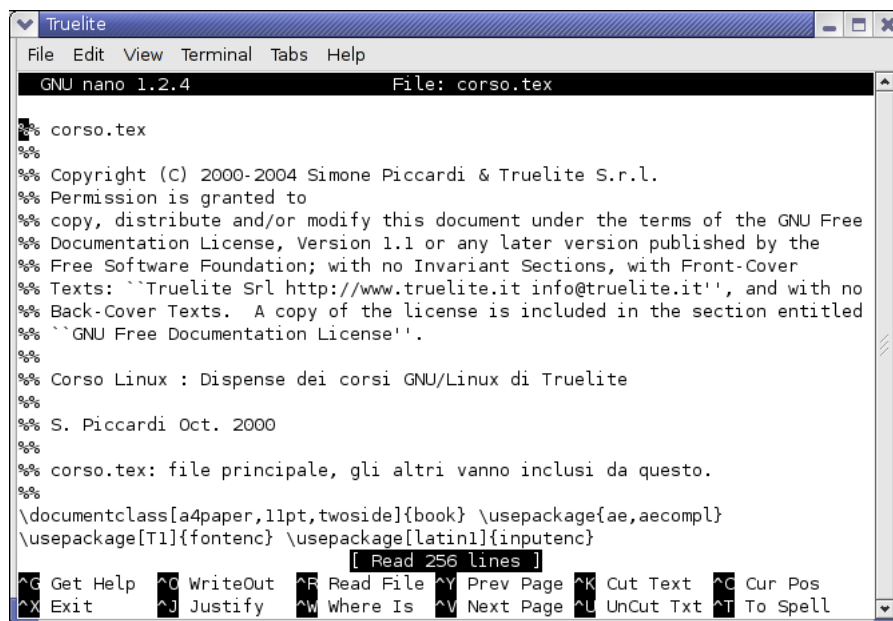


Figura 2.5: Schermata di avvio dell'editor **nano**.

Il programma si invoca al solito passando come argomento il nome del file da modificare, che viene aperto nella finestra mostrata in fig. 2.5, in cui si ha una riga di stato all'inizio, e due righe di aiuto in basso che riportano i comandi disponibili al momento, sulle quali vengono fatte pure le eventuali richieste di immissione o richieste scelte sulle azioni da compiere.

Un altro editor molto usato è **mc**; anche questo è un sottoinsieme di un'altro programma **mc** (nome che sta per *Midnight Commander*, un file manager semigrafico utilizzabile in console che nasce come clone del *Norton Commander*). Anche questo è un editor che mantiene nella prima riga un riassunto dei comandi principali, per cui non staremo a descriverne i dettagli.

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* (**gedit**) e *KDE* (**kate**) che non sono spiegati qui in quanto l'interfaccia grafica è in grado di dare accesso alle funzionalità base con i soliti menù.

Fra gli editor grafici vale però la pena segnalare uno dei più evoluti: **nedit** che dispone di un linguaggio di programmazione interna che ne permette una grande espandibilità ed è dotato di moltissime funzionalità (forse il più vicino ad **emacs**); per questo è uno degli editor più potenti, pur restando anche facile da usare. Esistono infine anche versioni grafiche di alcuni degli editor precedenti (come **gvim** per **vi**), mentre come già accennato sia **emacs** che **xemacs** usati dall'interfaccia grafica mettono a disposizione menù e utilizzo del mouse.

Capitolo 3

La configurazione dei servizi di base

3.1 I file di configurazione

In questa sezione tratteremo la gestione generica dei file di configurazione all'interno di un sistema GNU/Linux, introducendo alcuni concetti validi in generale per qualunque file di configurazione. Descriveremo poi direttamente i file di configurazione di alcuni servizi di base, come quelli che controllano il comportamento delle librerie dinamiche e quelli usati per il login, ed alcuni servizi di base. Si tenga comunque presente che alcuni file di configurazione (in particolare `fstab` e `mtab`) sono già stati descritti in precedenza (vedi sez. 1.2.4).

3.1.1 Una panoramica generale

A differenza di Windows che tiene tutte le configurazioni in un unico file binario, il *registro*, le sole due caratteristiche comuni che potete trovare nei file di configurazione su un sistema GNU/Linux sono che essi sono mantenuti, come illustrato in sez. 1.2.3, nella directory `/etc/` e che sono file di testo. Questo vale in generale per tutti i file di configurazione, e non è limitato a quelli che tratteremo nel prosieguo di questa sezione.

La ragione di questa frammentazione dei file di configurazione deriva dell'architettura del sistema (illustrata in sez. 1.1), per cui tutti i servizi sono forniti da opportuni programmi che non è affatto detto siano sempre gli stessi anche per uno stesso servizio.¹ Ciò comporta che i formati dei file di configurazione possano essere anche i più vari, dipendendo ciascuno dalla sintassi adottata dal relativo programma, per cui, anche se esistono delle convenzioni generali come ignorare le righe vuote o considerare il carattere `#` l'inizio di un commento, non è detto che esse vengano sempre rispettate.

Se da una parte tutto questo può spaventare, vista la lunghezza dell'elenco che si 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
...
```

¹ad esempio esistono diversi programmi che gestiscono l'invio e la ricezione della posta elettronica, ma chiaramente se ne installerà uno solo, e si userà il file di configurazione di questo.

dall'altra ha invece il grande vantaggio che le modifiche ad un singolo file di configurazione non hanno alcun modo di influenzare quelli di altri programmi. Il secondo enorme vantaggio è che essendo i file di configurazione dei file di testo è possibile effettuare ricerche ed operazioni complesse con i soliti comandi di shell abbondantemente trattati in sez. 2.2, ed eseguire le operazioni di configurazione con un editor qualunque.

Una seconda cosa di cui bisogna tenere conto è che Linux è multiutente, per cui è molto spesso possibile per ciascun utente scegliere le impostazioni che si ritengono più appropriate per un programma mettendo un ulteriore 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 le impostazioni generali di tutto il sistema.

È da tenere presente infine che per molti file di configurazione viene installata anche una pagina di manuale che ne spiega il formato, accessibile usualmente con `man nomefile`, o nel caso di omonimia con un comando o una funzione di sistema, con `man 5 nomefile`.² In seguito, per quelli che prenderemo in esame faremo una descrizione generale, trattando solo le caratteristiche principali, per questo varrà sempre la pena controllare la documentazione, che contiene tutti i dettagli.

3.1.2 La gestione delle librerie condivise

Una delle funzionalità più importanti per l'esecuzione dei programmi in un qualunque sistema è quello della gestione delle librerie condivise, quelle che in Windows vengono chiamate DLL (da *Dinamically Linked Library*) e che nei sistemi unix-like sono chiamate *shared object*. Questo è il meccanismo che permette di inserire tutto il codice comune usato dai programmi all'interno di opportune librerie,³ in modo che non sia necessario reinserirlo tutte le volte all'interno di ciascun programma.

Però per far sì che detto codice possa essere utilizzato dai singoli programmi occorre una apposita infrastruttura. Come brevemente accennato in sez. 2.1.6 uno dei compiti fondamentali del sistema, quello di mettere in esecuzione i programmi, viene realizzato attraverso il *link-loader*. Questo non è un vero programma a se stante, quanto una sezione di codice che viene sempre eseguita preventivamente tutte le volte che si deve lanciare un nuovo programma⁴ che permette di identificare le librerie condivise che contengono le funzioni necessarie e di caricarle automaticamente in memoria, in maniera trasparente all'utente.

Per verificare quali librerie dinamiche sono necessarie per l'esecuzione di un programma si può usare il comando `ldd`, che stampa sullo standard output i nomi delle librerie condivise di cui esso ha bisogno. Il comando prende come argomento il pathname assoluto del programma da analizzare e stampa a video il risultato, ad esempio:

```
piccardi@monk:~/Truelite/documentazione/corso$ ldd /bin/ls
librt.so.1 => /lib/librt.so.1 (0x40023000)
libacl.so.1 => /lib/libacl.so.1 (0x40035000)
libc.so.6 => /lib/libc.so.6 (0x4003c000)
libpthread.so.0 => /lib/libpthread.so.0 (0x4016a000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
libattr.so.1 => /lib/libattr.so.1 (0x401ba000)
```

²si ricordi quanto detto in sez. 2.3.1.

³e abbiamo visto in sez. 1.2.3 come ci siano delle directory specifiche previste dal *Filesystem Hierarchy Standard* che devono contenere i relativi file.

⁴a meno che questo non sia stato compilato staticamente, cioè in maniera da includere al suo interno tutto il codice che altrimenti sarebbe stato disponibile attraverso delle librerie condivise.

ci fa vedere le librerie necessarie all'uso di `ls`. L'opzione più usata è `-v` che permette di avere una descrizione più dettagliata con le informazioni relative alle versioni, le altre opzioni al solito sono descritte nella pagina di manuale del comando.

Si noti come le librerie siano file che terminano con l'estensione `.so` (che sta appunto per *shared object*) seguita da un numero che è detto *major version*. Questa è una delle caratteristiche più utili in un sistema unix-like; le librerie cioè sono organizzate sempre con due numeri di versione, *major* e *minor*, ed una convenzione vuole che le interfacce pubbliche delle librerie non debbano mai cambiare fintanto che non cambia la *major version*.⁵ Con questo si ottengono due risultati di grande rilevanza, il primo è che si può cambiare tranquillamente la *minor version* di una libreria senza che i programmi che la usano ne abbiano a risentire,⁶ il secondo è che se si ha bisogno di una versione vecchia delle librerie non c'è nessun problema, basta installare anche quella, e sarà tranquillamente usabile (attraverso la diversa *major version*) senza nessun conflitto.⁷

Dato che il *link-loader* deve essere in grado di determinare quali sono le librerie che contengono le funzioni richieste da un programma tutte le volte che questo viene eseguito, per effettuare la ricerca in maniera efficiente viene utilizzato un apposito file di *cache* in cui sono state indicizzate tutte le informazioni relative alle funzioni presenti ed alle librerie in cui esse sono contenute. Questo file si chiama `/etc/ld.so.cache`, e viene generato (di norma tutte le volte che si installa una nuova libreria) con il comando `ldconfig`.

Il comando `ldconfig` permette sia di ricostruire la cache che di ricreare i link alle ultima versione dei file delle librerie; infatti queste vengono sempre cercate per numero di *major version*, ma la libreria installata sarà comunque contenuta in un file con una sua specifica *minor version*, perciò quello che si fa è creare un link simbolico alla versione effettiva, per cui ad esempio avremo che:

```
piccardi@monk:/lib$ ls -l librt*
-rw-r--r-- 1 root root 26884 Oct 13 21:40 librt-2.3.2.so
lrwxrwxrwx 1 root root    14 Oct 15 17:49 librt.so.1 -> librt-2.3.2.so
```

un cui si può notare come una certa versione specifica di una libreria (la 2.3.2) venga rimappata come prima *major version*.

Se invocato con l'opzione `-v` il comando `ldconfig` stampa l'elenco di tutte le directory esaminate e delle librerie usate nella ricostruzione, mentre con `-N` e `-X` si può bloccare rispettivamente la ricostruzione della cache e dei link.

In sez. 1.2.1 abbiamo visto come secondo il *Filesystem Hyerarchy Standard* le librerie possono essere mantenute in diverse directory; ma di default il comando `ldconfig` esamina soltanto le directory `/lib` e `/usr/lib`, se ci sono altre librerie condivise in altre directory queste possono essere specificate con un opportuno file di configurazione, `/etc/ld.so.conf`, che contiene la lista delle altre directory che contengono le librerie condivise, oltre alle canoniche `/lib` e `/usr/lib`.

Pertanto se ad esempio si installa una nuova libreria dai sorgenti in `/usr/local/lib`, che di norma non compare in `/etc/ld.so.conf`, sarà necessario aggiungerla in questo file e poi eseguire il programma `ldconfig` per aggiornare i link alle librerie condivise disponibili e ricreare la cache, in modo che il linker dinamico possa utilizzarle. Un esempio di questo file, così come viene installato su una Debian Sid, è il seguente:

```
/usr/local/lib
/usr/X11R6/lib
```

⁵ al solito è una convenzione, anche se quasi universalmente rispettata; ogni tanto qualche programmatore anche non più alle prime armi la viola, con il risultato che programmi che fino ad allora funzionavano perfettamente si trovano a riportare errori o a terminare improvvisamente.

⁶ a meno che al solito un programmatore non troppo furbo non abbia usato una qualche funzione interna che non fa parte della interfaccia pubblica, nel qual caso può di nuovo succedere di tutto.

⁷ questo è il motivo per cui il problema dei conflitti di versione delle librerie tristemente noto su Windows, dove questa distinzione non esiste, come *DLL hell*, è sostanzialmente assente su GNU/Linux.

Il default di `ldconfig` prevede l'uso di questo file, ma usando l'opzione `-f` si può specificare un qualunque altro file al suo posto, mentre con `-n` si può passare direttamente sulla linea di comando una lista di directory dove effettuare la ricerca. Per le altre opzioni e la documentazione completa si consulti al solito la pagina di manuale disponibile con `man ldconfig`.

Infine, nei casi in cui si vogliano utilizzare solo in forma temporanea delle librerie condivise, si può ricorrere alla variabile di ambiente `LD_LIBRARY_PATH`, in cui passare una lista⁸ di ulteriori directory in cui verrà effettuata la ricerca di altre librerie (questa volta senza usare l'indicizzazione, per cui il sistema è più lento).

In genere si usa questa variabile quando si sviluppano delle librerie o se si vuole usare qualche pacchetto sperimentale oppure una versione alternativa delle librerie di sistema. Infatti le librerie contenute nelle directory specificate tramite `LD_LIBRARY_PATH` hanno la precedenza e vengono utilizzate per prime; in questo modo si può far uso di una libreria sperimentale senza conseguenze per gli altri programmi⁹ del sistema che continueranno ad usare la versione abituale.

3.1.3 Il *Name Service Switch*

Una delle tante funzionalità provviste dalle librerie standard del sistema è fornire una serie di funzioni che permettono ai programmi di ottenere alcune informazioni relative alla gestione del sistema, come i nomi degli utenti, le loro password, i nomi dei gruppi, delle macchine ecc.

Tradizionalmente, con l'eccezione per i nomi delle macchine che possono essere forniti anche attraverso l'uso del DNS (vedi sez. 7.4), queste informazioni sono memorizzate in opportuni file di configurazione mantenuti sotto `/etc`.¹⁰ I sistemi moderni però permettono di mantenere queste informazioni anche in maniera diversa, e di centralizzarle per interi gruppi di macchine tramite opportuni servizi.¹¹

Con l'introduzione di queste estensioni si presentava però il problema di come indicare alle varie funzioni di libreria dove prendere le informazioni; all'inizio questo veniva fatto introducendo tutta la casistica possibile nell'implementazione delle funzioni stesse, con degli ovvi problemi di estendibilità e compatibilità. Per risolvere il problema venne creata una apposita interfaccia, il *Name Service Switch*,¹² che permettesse di demandare a delle librerie esterne, configurabili in maniera indipendente, le modalità con cui queste informazioni vengono ottenute.

Il grande vantaggio del *Name Service Switch* è che diventa possibile definire in maniera modulare ed estendibile sia delle classi di informazioni (cosicché qualora si debba fornire qualche nuovo servizio si ha l'infrastruttura già pronta) che il supporto (file, database o servizi di rete) su cui queste informazioni sono mantenute; è possibile inoltre specificare al sistema anche in quale ordine utilizzare le varie fonti, permettendo configurazioni ibride.

Le modalità di funzionamento del *Name Service Switch* vengono gestite attraverso il suo file di configurazione che è `/etc/nsswitch.conf/etc/nsswitch.conf`; un esempio di questo file, come installato su una Debian Sid, è il seguente:

```
# /etc/nsswitch.conf
#
# Example configuration of GNU Name Service Switch functionality.
# If you have the 'glibc-doc' and 'info' packages installed, try:
# info libc "Name Service Switch" for information about this file.
```

⁸nella stessa forma usata per `PATH`, cioè con le directory separate da dei ":".

⁹o per l'intero sistema, dato che se si usasse una versione non funzionante di una libreria fondamentale come la `glibc`, smetterebbero di funzionare praticamente tutti i programmi.

¹⁰ne tratteremo alcuni in seguito, ad esempio quelli relativi alla gestione di utenti e gruppi in sez. 4.3 e quelli relativi alla rete in sez. 7.4.

¹¹gli esempi più comuni sono il NIS (*Network Information Service* ed LDAP (*Lightweight Directory Access Protocol*) due servizi di rete con cui si possono centralizzare le informazioni relative agli utenti.

¹²il sistema è stato introdotto la prima volta nelle librerie standard di Solaris, le librerie standard GNU hanno ripreso lo stesso schema.

```

passwd:      compat
group:       compat
shadow:      compat

hosts:       files dns
networks:    files

protocols:   db files
services:    db files
ethers:      db files
rpc:         db files

netgroup:    nis

```

Il formato del file è sempre lo stesso, le linee vuote o che iniziano per # vengono ignorate, le altre linee indicano una opzione. La pagina di manuale contiene una lista completa delle opzioni disponibili. La prima colonna, terminata da un “:” indica una classe di informazioni, di seguito vengono elencate, separate da spazi, le modalità con cui queste informazioni vengono fornite; la ricerca dei dati sulle varie fonti sarà eseguita nell’ordine in cui queste sono indicate. L’elenco delle classi di informazioni disponibili è riportato in tab. 3.1.

Classe	Tipo di corrispondenza
shadow	corrispondenze fra username, <i>shadow password</i> ed altre informazioni sulla gestione delle password (vedi sez. 4.3.3).
passwd	corrispondenze fra username, identificatori di utente e gruppo principale, shell di default, ecc. (vedi sez. 4.3.3).
group	corrispondenze fra nome del gruppo e proprietà dello stesso (vedi sez. 4.3.3).
aliases	alias per la posta elettronica.
ethers	corrispondenze fra numero IP e <i>MAC address</i> della scheda di rete (vedi sez. 7.5.4).
hosts	corrispondenze fra nome a dominio e numero IP (vedi sez. 7.4.3).
netgroup	corrispondenze gruppo di rete e macchine che lo compongono.
networks	corrispondenze fra nome di una rete e suo indirizzo IP (vedi sez. 7.4.4).
protocols	corrispondenze fra nome di un protocollo e relativo numero identificativo (vedi sez. 7.4.4).
rpc	corrispondenze fra nome di un servizio RPC e relativo numero identificativo (vedi sez. 7.5.5).
services	corrispondenze fra nome di un servizio e numero di porta (vedi sez. 7.4.4).

Tabella 3.1: Le diverse classi di corrispondenze definite all’interno del *Name Service Switch*.

Per ciascuna nuova modalità deve esistere una opportuna libreria che garantisce l’accesso alle informazioni con quella modalità; esse si trovano tutte in `/lib/` e devono avere il nome `libnss_NAME.so.X`, dove X fa riferimento alla versione delle `glibc`,¹³ mentre NAME indica il tipo di supporto per quell’informazione (nel caso dell’esempio sarà `files`, `db`, ecc.).

Di norma non c’è niente da cambiare in questo file a meno che non si aggiunga un ulteriore supporto, come un sistema di autenticazione basato su qualche altro meccanismo (ad esempio su LDAP), nel qual caso andrà installato l’apposito pacchetto (che per LDAP è `libnss-ldap`) ed inserita la relativa parola chiave (nel caso `ldap`) nella adeguata posizione all’interno delle colonne che specificano dove sono disponibili i servizi.

¹³vale 1 per le `glibc` 2.0 e 2 per le `glibc` 2.1.

3.1.4 I file usati dalla procedura di *login*

Come accennato in sez. 1.3.4 la procedura di login da terminale è gestita dai due programmi `getty` e `login`. Questi usano una serie di file di configurazione che provvedono al controllo di alcune funzionalità, che poi sono spesso riutilizzati anche dagli altri programmi che eseguono la procedura di collegamento al sistema.¹⁴

In quella occasione abbiamo accennato che il messaggio di presenza stampato sui terminali viene letto da `/etc/issue`, che è un semplice file il cui testo viene stampato sul terminale prima della stringa `login:` . Il contenuto del file viene stampato integralmente, ma è possibile inserire delle direttive, attraverso l'uso del carattere `\`, che permettono di stampare alcune informazioni dinamiche; ad esempio con `\s` si inserisce automaticamente il nome del sistema operativo, con `\l` il nome del terminale, con `\n` il nome della macchina. I principali valori sono riportati in tab. 3.2, l'elenco completo è riportato nella pagina di manuale di `getty`.

Opzione	Significato
<code>\d</code>	data.
<code>\l</code>	nome del terminale.
<code>\m</code>	architettura (i486, ppc, ecc.).
<code>\n</code>	<i>hostname</i> .
<code>\r</code>	versione del kernel.
<code>\s</code>	nome del sistema (Debian, RedHat, ecc).
<code>\t</code>	ora.

Tabella 3.2: Principali caratteri di estensione per `/etc/issue`.

Analogo a `issue` è il file `/etc/issue.net` che viene usato al suo posto da `telnet` per i collegamenti effettuati via rete; se invece si usa `ssh` (vedi sez. 8.3) detti file vengono ignorati. Una volta completato il login viene invece mostrato un messaggio di benvenuto, che è mantenuto nel file `/etc/motd`; il nome del file sta per *message of the day*, e scrivere un messaggio in questo file è una modalità veloce per mandare un avviso a tutti gli utenti che entrano nel sistema.

Uno dei file che controllava¹⁵ una serie di funzionalità relative alla procedura di *login*, è `/etc/login.defs` (che in realtà è il file di configurazione del sistema di gestione delle *shadow password*, trattate in sez. 4.3.3). Al solito sono considerati commenti le righe che iniziano per `#` e ignorate le righe vuote. Il file contiene una serie di specificazioni di parametri, fatte nella forma:

NOME **valore**

quelle che concernono la procedura di *login* sono ad esempio `LOGIN_RETRIES` che imposta quante volte può essere ritentata la procedura di login, e `LOGIN_TIMEOUT` che indica il numero di secondi in cui il programma aspetta l'immissione della password prima di cancellare la procedura, mentre `MOTD_FILE` permette di specificare un altro file al posto di `/etc/motd`.

Un altro file di controllo per la procedura di *login* è `/etc/securetty`. Questo file contiene la lista delle console da cui si può collegare l'amministratore di sistema (cioè l'utente `root`). Viene usato dal programma `login`, che legge da esso i nomi dei dispositivi di terminale (le `tty`) dai quali è consentito l'accesso. Un esempio di questo file è il seguente:

```
# Standard consoles
tty1
tty2
tty3
tty4
```

¹⁴in genere sono gestiti, come vedremo in sez. 4.3.4, attraverso dei moduli di PAM.

¹⁵con l'uso di PAM (vedi sez. 4.3.4) queste funzionalità sono state inserite all'interno di questa libreria e gestite dai relativi file di configurazione.

```

tty5
tty6
# Same as above, but these only occur with devfs devices
vc/1
vc/2
vc/3
vc/4
vc/5
vc/6

```

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

Dato che le console virtuali non sono indicate in questo file non è normalmente possibile eseguire un `telnet` da remoto per collegarsi come `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.¹⁶

Infine quando si vuole bloccare temporaneamente l'accesso al sistema degli utenti normali, l'amministratore può creare il file `/etc/nologin`; questo non è propriamente un file di configurazione, dato che il suo contenuto è ininfluenza, la sua presenza serve solo a dire alla procedura di collegamento al sistema che solo l'amministratore può essere entrare. La sua rimozione ripristinerà l'accesso per tutti gli utenti.

3.1.5 La configurazione del sistema delle pagine di manuale

Come accennavamo in sez. 2.3.1 il comando `man` supporta anche la possibilità di generare documentazione stampabile. Questo avviene perché in realtà le pagine di manuale non sono scritte direttamente nel formato in cui le vediamo su video, ma in uno speciale linguaggio di formattazione, chiamato *troff*, che permette la creazione di sezioni, indici, ecc. e la generazione di vari tipi di formati in di uscita, fra cui quello del testo mostrato sul terminale è uno fra i tanti.

Tutte le volte che si richiama il comando `man` per visualizzare una nuova pagina di manuale il comando dovrà recuperare uno di questi file generando un testo in formato opportuno per la visualizzazione. Per evitare di ripetere il procedimento della generazione ad una successiva invocazione tutti questi file vengono salvati, usualmente sotto `/var/cache/man/`, in una directory `catN`, dove `N` corrisponde alla sezione cui appartiene la pagina, così da poterli recuperare velocemente.

Un secondo aspetto del sistema delle pagine di manuale è quello dell'indicizzazione dei contenuti usata dai comandi `whatis` e `apropos`; questa infatti non è diretta (come si può verificare tutte le volte che si installa un nuovo pacchetto e non lo si trova nei risultati di detti comandi), ma viene effettuata normalmente tramite il programma `mandb`, che costruisce gli opportuni file con gli indici usati dai precedenti.

In genere il programma viene invocato periodicamente nelle operazioni giornaliere eseguite da `cron` (vedi sez. 3.3.1), ma lo si può anche invocare direttamente; le opzioni principali sono `-c` che gli fa ricreare tutti gli indici da zero (il default è aggiornare quelli esistenti) e `-p` che non fa eseguire la ripulitura degli indici dei dati della pagine non più presenti; per la descrizione completa delle altre opzioni ed i dettagli sul comando al solito si faccia riferimento alla pagina di manuale.

Per stabilire in quale directory si trovano gli originali delle pagine, in che ordine cercarle, dove devono essere memorizzate le pagine riformattate per la visualizzazione, dove mantenere gli indici delle pagine presenti e delle parole su cui effettuare le ricerche, il sistema usa il file di configurazione `/etc/manpath.config`.

¹⁶in generale non è comunque una buona idea neanche quella di usare `telnet`, quindi è meglio se proprio lasciate perdere la cosa.

Il primo aspetto della configurazione è quello della definizione delle directory in cui si trovano le pagine di manuale; come per gli eseguibili infatti anch'esse possono essere installate in diverse sezioni dell'albero,¹⁷ una lista di tutte queste directory si può stampare con il comando `manpath`, ed in genere si otterrà qualcosa del tipo:

```
[piccardi@gont corso]$ manpath
/usr/local/man:/usr/share/man:/usr/X11R6/man
```

a meno di non aver definito in maniera diretta la variabile di ambiente `MANPATH` (nel qual caso sarà semplicemente stampato il suo valore) che permette di soprassedere il valore preimpostato dal sistema.

La configurazione delle directory in cui cercare le pagine di manuale è una di quelle mantenute in `manpath.config`. La direttiva `MANDATORY_MANPATH` serve per indicare tutte le directory che si vuole siano sempre aggiunte (se esistono) alla lista di quelle in cui si cercano le pagine di manuale; la direttiva prende come parametro il pathname alla directory contenente gli originali, e deve essere ripetuta per tutte le directory che si vuole siano aggiunte alla lista.

La direttiva `MANPATH_MAP` indica invece la corrispondenza fra le directory dei comandi e le relative pagine di manuale; serve per aggiornare dinamicamente la lista delle directory in cui cercare le pagine di manuale sulla base delle directory presenti nel `PATH` (vedi sez. 2.1.4) dell'utente. La direttiva richiede due parametri; il primo indica una directory contenente i comandi, il secondo la directory delle corrispondenti pagine di manuale. Se la prima compare nel `PATH` dell'utente la seconda è aggiunta al `MANPATH` in maniera implicita.

Infine la direttiva `MANDB_MAP` serve ad indicare a `mandb` dove devono essere messi i file degli indici, e dove devono essere create le pagine temporanee. Un estratto del file `manpath.config`, così come installato su una Debian Sid, è il seguente:

```
MANDATORY_MANPATH      /usr/man
MANDATORY_MANPATH      /usr/share/man
MANDATORY_MANPATH      /usr/X11R6/man
MANDATORY_MANPATH      /usr/local/man
...
MANPATH_MAP             /bin                /usr/share/man
MANPATH_MAP             /usr/bin          /usr/share/man
MANPATH_MAP             /sbin            /usr/share/man
MANPATH_MAP             /usr/sbin        /usr/share/man
MANPATH_MAP             /usr/local/bin    /usr/local/man
...
MANDB_MAP               /usr/man                /var/cache/man/fsstnd
MANDB_MAP               /usr/share/man    /var/cache/man
MANDB_MAP               /usr/local/man     /var/cache/man/oldlocal
MANDB_MAP               /usr/local/share/man /var/cache/man/local
MANDB_MAP               /usr/X11R6/man     /var/cache/man/X11R6
MANDB_MAP               /opt/man           /var/cache/man/opt
...
SECTION                1 n 1 8 3 2 3pm 3perl 5 4 9 6 7
```

L'ultima direttiva, `SECTION`, indica l'ordine di ricerca delle pagine di manuale nelle varie sezioni, si noti come in questo caso ci siano altre sezioni oltre a quelle classiche illustrate in tab. 2.18.

¹⁷si pensi al caso in cui si siano installate diverse versioni di uno stesso pacchetto, ad esempio una versione più recente a mano sotto `/usr/local`, in tal caso le pagine di manuale saranno sotto `/usr/local/man` ed il sistema deve essere in grado di vederle, ed in un certo ordine rispetto alle altre.

3.2 Altri file

Raccogliamo in questa sezione le informazioni relative ad una serie di altri file di configurazione relativi a funzionalità del sistema che non sono facilmente raggruppabili sotto un denominatore comune, o che hanno a che fare con la configurazione di comandi di base.

3.2.1 Il file `rc.local` e la directory `rc.boot`

Nella trattazione della procedura di avvio del sistema in sez. 5.3.4 vedremo che per far eseguire un proprio script nella sequenza di avvio basterà creare un opportuno link simbolico e come lo si potrà inserire in un punto preciso della sequenza, compreso in fondo alla stessa; in alcune distribuzioni però si è soliti usare un file specifico per far eseguire dei comandi dopo che tutta la procedura di avvio è stata completata.

In genere questo file è `rc.local` e si trova fra gli script di avvio (in `/etc/rc.d` o anche direttamente in `/etc/`). Esso viene eseguito alla fine della procedura di avvio, ed 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.

Una modalità alternativa (supportata ad esempio da Debian), è quella dell'uso della directory `/etc/rc.boot` nella quale si possono mettere gli script che si vuole siano eseguiti alla fine della procedura di inizializzazione. In tal caso occorre tenere presente che i file dovranno essere eseguibili, e il loro nome non dovrà contenere caratteri speciali.¹⁸

3.2.2 La directory `/etc/skel` ed il file `/etc/shells`

Altri file non classificabili in una specifica categoria, anche se connessi alla gestione degli utenti, sono `/etc/shells` ed il contenuto della directory `/etc/skel`. Come vedremo meglio in sez. 4.3.2 trattando i comandi per la gestione degli utenti, è possibile creare uno *scheletro* del contenuto della home directory di ogni nuovo utente in modo che questo sia automaticamente disponibile tutte le volte che se ne crea uno.

La directory `/etc/skel`, come il nome stesso suggerisce, è quella che contiene questo scheletro. Tutti i file e le directory che si vuole siano creati nella home directory dei nuovi utenti (ad esempio una opportuna copia di `.bashrc`, `.bash_profile` e di altri eventuali file di configurazione) possono essere messi in questa directory, e saranno automaticamente copiati nella relativa home alla creazione di ogni nuovo utente.

Il file `/etc/shells` invece è quello che contiene la lista delle shell valide che l'utente può selezionare con il comando `chsh` (vedi sez. 4.3.2). Il file utilizza il solito formato, le righe inizianti per `#` sono considerate commenti, e le righe vuote sono ignorate. Ogni riga non vuota contiene un solo campo che specifica il pathname completo del programma che può essere usato come shell.

Un utente che voglia cambiare la propria shell di default potrà usare solo una shell fra quelle che sono indicate in questo file; in questo modo l'amministratore può lasciare all'utente la libertà di modificare la propria shell di login, restringendola però alle shell autorizzate.

Il file viene anche usato da alcuni servizi per verificare se un utente è un utente normale, sulla base della presenza della sua shell di login (tratteremo l'argomento in sez. 4.3.3) in questo file; ad esempio vari server FTP rifiutano l'accesso ad username corrispondenti ad utenti che non hanno una shell valida fra quelle elencate in `/etc/shells`.

¹⁸infatti Debian esegue gli script usando lo speciale comando `run-parts` che, come vedremo in sez. 3.3.1, lancia tutti gli script presenti in una directory posto che il loro nome sia nel formato adatto.

Si tenga conto comunque che un utente può comunque installare nella propria home directory un'altra shell ed usare quella al posto della shell di login, semplicemente lanciandola come un qualunque altro programma; non potrà però cambiare quella con cui entra nel sistema al login.

3.2.3 Il file `/etc/updatedb.conf`

Abbiamo visto in sez. 2.2.2 come con `locate` si possa ricercare la presenza di un file utilizzando un database di tutti quelli presenti creato attraverso il comando `updatedb`; quest'ultimo prende una serie di opzioni (in forma estesa, si consulti la pagina di manuale) che gli permettono di definire quali directory (ad esempio non ha molto senso indicizzare `/tmp`) e quali filesystem (non ha senso indicizzare un CDROM) inserire nel database dei file e quali no.

Oltre alle opzioni il comando utilizza delle omonime¹⁹ variabili di ambiente per ottenere le stesse impostazioni. Dato che normalmente il comando viene lanciato da uno degli script periodici eseguiti da `cron` (vedi sez. 3.3.1), quello che viene fatto è di impostare queste variabili di ambiente all'interno di un file, `/etc/updatedb.conf`, che serve come una specie di file di configurazione del comando. Il contenuto tipico di questo file (così come ottenuto dalla versione installata su una Debian Sid) è il seguente:

```
# This file sets environment variables which are used by updatedb

# filesystems which are pruned from updatedb database
PRUNEFS="NFS nfs afs proc smbfs autofs iso9660 ncpfs coda devpts ftpfs devfs mfs sysfs"
export PRUNEFS
# paths which are pruned from updatedb database
PRUNEPATHS="/tmp /usr/tmp /var/tmp /afs /amd /alex /var/spool /sfs"
export PRUNEPATHS
# netpaths which are added
NETPATHS=""
export NETPATHS
# run find as this user
LOCALUSER="nobody"
export LOCALUSER
# cron.daily/find: run at this priority -- higher number means lower priority
# (this is relative to the default which cron sets, which is usually +5)
NICE=10
export NICE
```

Trattandosi di una sezione di uno script (che sarà utilizzato con un `source` dallo script che esegue la ricostruzione del database) la forma è quella dell'assegnazione delle variabili di ambiente che poi saranno usate da `updatedb`. Allora `PRUNEFS` indicherà i filesystem da non indicizzare, come quelli di rete, i filesystem virtuali come `/proc` o quelli di dispositivi estraibili; `PRUNEPATHS` indicherà le directory da non indicizzare (quelle dei file temporanei, gli *spool* delle code di stampa e della posta, ecc.) e `LOCALUSER` indica l'utente con i permessi del quale sarà effettuata l'indicizzazione, nel caso `nobody`,²⁰ in modo da evitare che siano indicizzati file che gli utenti hanno protetto da lettura.

3.3 I servizi di base

In questa sezione prenderemo in esame la configurazione di alcuni servizi di base del sistema, come quelli per l'esecuzione periodica dei comandi, il sistema di gestione dei log, ecc. In genere questi sono realizzati da degli appositi programmi detti *demoni* (si ricordi quanto detto in

¹⁹rispetto al nome delle opzioni le variabili sono scritte in maiuscolo anziché miniscopo, e senza il `--` iniziale.

²⁰un utente di servizio che non ha nessun privilegio nel filesystem, utilizzato come titolare dei programmi che devono avere accesso solo alle informazioni pubbliche.

sez. 1.3.4) che lavorano in background ed il cui comportamento è controllato dai relativi file di configurazione.

3.3.1 Il servizio *cron*

Una funzionalità importante presente in qualunque sistema operativo è quella dell'esecuzione di programmi su base periodica, essenziale per compiere tutte quelle operazioni che devono essere eseguite a periodi fissi, come la creazione del database dei file e l'indicizzazione delle pagine di manuale, trattati rispettivamente sez. 3.2.3 e 3.1.5. Questa funzionalità viene gestita dal servizio chiamato *cron*, implementato attraverso l'uso del demone *crond*. Il demone ha il compito di svegliarsi ogni minuto ed eseguire ogni programma che è stato programmato per quel momento.

Il file di configurazione principale di *crond* è */etc/crontab* che contiene l'elenco delle operazioni periodiche generali da eseguire nel sistema. Il primo file controllato da *crond* per decidere se c'è da eseguire una operazione è questo. In genere si deve intervenire su questo file solo quando si vuole cambiare uno degli orari a cui le operazioni vengono eseguite o per inserire una nuova operazione periodica.

Il formato del file segue la solita regola di ignorare righe vuote ed inizianti per *#*, ogni riga deve contenere o una assegnazione di una variabile di ambiente (nello stesso formato usato negli script di shell) o la specificazione di una azione periodica. L'azione viene specificata da una serie di campi separati da spazi o tabulatori, i primi cinque indicano la periodicità con cui il comando indicato nell'ultimo campo viene eseguito, il sesto campo indica l'utente per conto del quale eseguire il comando scritto nel seguito della riga.

I cinque campi della periodicità indicano rispettivamente: minuto (da 0 a 60), ora (da 0 a 23), giorno del mese (da 1 a 31), mese dell'anno (da 1 a 12) e giorno della settimana (da 0 a 7, dove sia 0 che 7 indicano la domenica); per quest'ultimo campo sono accettati anche valori tipo Mon, Thu, etc. L'utilizzo del carattere *** vale da jolly e lo si usa per indicare un valore qualsiasi. Se il tempo corrente corrisponde a tutti i valori specificati nei cinque campi il comando viene eseguito.

Il demone *crond* di GNU/Linux²¹ supporta poi alcune estensioni non presenti in altri sistemi unix-like: si può usare una lista (separata da virgole) per indicare più valori, un intervallo, specificando gli estremi separati con un *–*. Infine il programma supporta la possibilità di specificare periodi personalizzati attraverso l'uso del carattere jolly abbinato con il carattere */* seguito dal divisore da applicare al valore generico del campo; così ad esempio usando **/2* nel primo campo si intenderà chiedere la ripetizione del lavoro ogni due minuti.

Si tenga presente che siccome il programma specificato nella parte conclusiva della linea viene eseguito direttamente dal demone *crond*, non saranno necessariamente definite le usuali variabili di ambiente presenti normalmente quando si esegue una shell. In particolare è da tenere presente che il demone si limita a definire solo alcune di esse ed in particolare per quanto riguarda *PATH* questa comprenderà soltanto le directory */usr/bin* e */bin*. Per questo motivo se necessita la presenza di altri valori o altre variabili occorrerà inserire in testa al file le relative definizioni. Per ulteriori dettagli si faccia riferimento alla pagina di manuale del file, accessibile con *man 5 crontab*.

Un esempio del contenuto del file */etc/crontab*, preso dal file installato di default su una Debian Sarge, 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.
```

²¹si tratta del programma *vixie-cron*, creato da Paul Vixie, noto anche per essere l'autore di *bind*, il server DNS (vedi sez. 9) più usato nel mondo.

```

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 6:25,²² tutte le domeniche alle 6:27 e tutti i primi giorni del mese alle 6:52.

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**²³ dagli script presenti nelle directory elencate nell'esempio precedente, per cui se non si hanno esigenze particolari non è il caso di intervenire su questo file ma di aggiungere il proprio script direttamente in `/etc/cron.daily/`, `/etc/cron.weekly/`, `/etc/cron.monthly/`. Occorre comunque fare attenzione, perché **run-parts** non esegue gli script il cui nome contiene caratteri diversi da lettere maiuscole e minuscole, numeri, il carattere di sottolineatura ed il “-” (per i dettagli si consulti la relativa pagina di manuale).

Oltre ai compiti generali appena illustrati il servizio *cron* è in grado di eseguire anche operazioni richieste da un singolo utente; in questo caso questi dovrà creare un suo *crontab* personale tramite il comando **crontab**.²⁴ Se lo si chiama con l'opzione **-e** questo invocherà l'editor predefinito (di norma è **vi**, a meno di non aver impostato diversamente la variabile **EDITOR**). Il formato di questo file è identico a quello di `/etc/crontab` con l'eccezione del sesto campo, che nel caso non è necessario e non deve essere specificato, dato che l'utente è già ben definito.

Se lanciato senza opzioni il comando prende come argomento il file da usare come tabella, di cui legge il contenuto che deve essere nella forma in cui lo si scriverebbe con l'editor; usando “-” si legge direttamente dallo standard input. Si può poi vedere la lista delle operazioni programmate usando l'opzione **-l** che scrive sullo standard output il contenuto del *crontab* dell'utente.²⁵ Con **-r** invece si esegue la rimozione completa della tabella corrente. Infine con **-u** si potrà modificare (solo da parte dell'amministratore) la tabella dell'utente specificato come parametro.

Si tenga presente che l'amministratore può riservare l'uso del servizio *cron* solo ad alcuni utenti, creando il file `/etc/cron.allow`, nel quale si devono elencare gli username degli utenti che lo possono usare. Se il file non esiste è possibile vietare l'uso del servizio solo ad alcuni utenti con la creazione del file `/etc/cron.deny` che di nuovo dovrà contenere la lista degli username di coloro che non possono usarlo. In assenza di detti file chiunque può usare il servizio.

3.3.2 Il servizio *at*

Abbiamo visto nella sezione precedente come si può programmare l'esecuzione periodica di processi con **crontab**, esistono però anche necessità di avere una esecuzione differita dei programmi, senza che questa debba essere periodica. Per provvedere a questo compito esiste un altro servizio, detto *at*, per il nome dal comando che permette di richiedere l'esecuzione di un programma ad

²²la prima riga si legge appunto come minuto 25, ore 6, essendo le altre indicazioni tutti * si intende che giorno, mese e giorno della settimana sono qualsiasi, allo stesso modo si interpretano le altre due righe.

²³nel file di **crontab** mostrato nell'esempio viene fatto un controllo preventivo della presenza del comando **anacron** (installabile dal pacchetto omonimo), questo è un sostituto di **crond** per le macchine che non è previsto siano sempre accese, il quale si incarica anche di eseguire le operazioni che non sono state effettuate perché la macchina era spenta, se è installato le operazioni periodiche vengono eseguite da lui e non è necessario l'intervento di **crond**; l'installazione di default presuppone che la macchina non ne abbia bisogno, per cui il comando non sarà presente e verrà eseguito invece **run-parts**.

²⁴i file vengono mantenuti in `/var/spool/cron/crontabs`.

²⁵la versione usata da Debian in realtà non stampa le righe di commento iniziale che avvisano di non modificare direttamente il contenuto del *crontab*, in modo da poter riutilizzare l'output del comando direttamente come input per un successivo **crontab -**.

un tempo successivo. Il servizio comporta diverse modalità di gestione della esecuzione differita, e vari programmi per la gestione della stessa.

Il servizio è realizzato dal demone `atd`, che come `crond` viene lanciato dagli script di avvio. Il demone controlla la presenza nelle code di programmi in attesa di essere eseguiti, e li manda in esecuzione al tempo stabilito. Il demone può essere anche lanciato manualmente, e prevede come opzioni `-l` che permette di impostare manualmente un limite massimo per il carico della macchina, oltre il quale non saranno i programmi programmati (di default vale il valore preimpostato di 1.5, ha senso cambiarlo solo in caso di macchine multiprocessore), `-b` che permette di impostare un intervallo di tempo minimo (il default è 60 secondi) fra la messa in esecuzione di due successivi programmi. Al solito per l'elenco completo si può fare riferimento alla pagina di manuale.

Il comando di base, che permette di programmare l'esecuzione di un programma ad un dato momento, è `at`. Il comando vuole come argomento data ed ora in cui il programma specificato deve essere messo in esecuzione; il nome di quest'ultimo viene letto dallo standard input o può essere specificato direttamente dalla riga di comando con l'opzione `-f`. La directory di lavoro e l'ambiente del comando che verrà lanciato sono quelli presenti al momento in cui si invoca `at`.

Il comando supporta una grande varietà di formati per le modalità in cui si indica la data, fra cui tutti quelli del comando `date`, più altre estensioni che permettono di usare anche specificazioni in forma più *discorsiva* come `at 1pm tomorrow`. Una descrizione completa di queste si trova nel file `/usr/share/doc/at/timespec`.

Una alternativa ad `at` è l'uso del comando `batch` che permette di programmare una esecuzione differita non in base ad un orario ma in base al carico della macchina. Il comando cioè sarà posto in esecuzione solo quando il carico medio della macchina scende sotto un certo valore.

Una volta programmata l'esecuzione di un programma questo viene messo, per ciascun utente, in una opportuna coda. Gli utenti possono vedere la lista dei propri programmi in coda con il comando `atq`, mentre l'amministratore può usare lo stesso comando per vedere la lista di tutti quanti. Il comando stampa il nome di ciascun programma, l'orario per cui è stata programmata l'esecuzione, ed un numero identificativo ad esso associato.

Un programma può essere rimosso dalla coda con il comando `atrm`, che prende come argomento l'identificativo dello stesso ottenibile con `atq`. Di nuovo un utente normale può operare solo sui propri programmi, mentre l'amministratore può operare su tutti quanti.

Infine come per l'uso di `cron` è previsto un controllo degli accessi al servizio attraverso i due file `/etc/at.allow` e `/etc/at.deny`, il cui formato e significato sono identici a quello degli analoghi `cron.allow` e `cron.deny`.

3.3.3 Il servizio *syslog*

Il servizio di *syslog* è il servizio usato dai demoni che girano in background (e dallo stesso kernel) per inviare dei messaggi. Dato che detti programmi non sono associati ad un terminale non è loro possibile scrivere messaggi di avviso o di errore; per questo esiste un demone apposito, `syslogd`, che si occupa di fornire una sorta di servizio di *segreteria telefonica*, dove i vari programmi che devono dire qualcosa possono scrivere i loro messaggi.

Di norma il servizio è attivato automaticamente dagli script di avvio, ed è possibile attivarlo e disattivarlo direttamente con l'uso diretto degli stessi, ma in caso di necessità lo si può anche lanciare direttamente (ad esempio per poter utilizzare l'opzione `-d` che attiva la modalità di debug). Se si vuole abilitare la ricezione di messaggi via rete invece occorre utilizzare l'opzione `-r`. La descrizione completa del comando e di tutte le opzioni è disponibile nella pagina di manuale, al solito accessibile con `man syslogd`.

Il file di configurazione per il demone `syslogd` è `/etc/syslog.conf`. Il formato del file è sempre lo stesso, ogni linea definisce una regola di registrazione, le linee vuote o che iniziano per `#` vengono ignorate. Ogni regola è costituita da due campi separati da spazi o tabulatori; il

Servizio	Significato
auth	servizio identico a <code>authpriv</code> , deprecato.
authpriv	messaggi relativi ad autenticazione e sicurezza.
cron	messaggi dei demoni (<code>at</code> e <code>cron</code>) .
daemon	demoni di sistema che non hanno una categoria di servizio a se stante.
ftp	servizio FTP (<i>File Transfere Protocol</i>).
kern	messaggi del kernel.
lpr	messaggi dai servizi di stampa.
mail	messaggi dai demoni di gestione della posta elettronica.
mark	uso interno.
news	messaggi del servizio di gestione di USENET (la rete dei gruppi di discussione).
security	sinonimo di <code>auth</code> .
syslog	messaggi interni generati da <code>syslogd</code> .
user	messaggi generici a livello utente.
uucp	messaggi del sistema UUCP (<i>Unix to Unix CoPy</i> , un meccanismo di comunicazione precedente internet).

Tabella 3.3: I servizi standard in cui sono classificati i messaggi del *syslog*.

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. Per una descrizione completa di tutti i dettagli si faccia al solito riferimento alla pagina di manuale.

Il *servizio* identifica una categoria di servizi di sistema per conto dei quali si vuole registrare il messaggio, e viene specificato tramite una delle parole chiave riportate in tab. 3.3, dove sono elencati i servizi standard predefiniti. Oltre a questo ci sono poi una serie servizi ausiliari, identificati dalle parole chiave da `local0` a `local7` che sono lasciati a disposizione dell'utente per un uso non specifico.

I valori delle *priorità* invece sono indicati in tab. 3.4 in ordine crescente, dalla più bassa alla più alta. Questi identificano l'importanza del messaggio, tutti i messaggi di priorità superiore od uguale a quella indicata nella regola verranno registrati.

Priorità	Significato
debug	messaggio di debug.
info	messaggio informativo.
notice	situazione normale, ma significativa.
warning	avvertimento.
warn	sinonimo di <code>warning</code> , deprecato.
err	condizione di errore.
error	sinonimo di <code>err</code> , deprecato.
crit	condizione critica.
alert	si deve intervenire immediatamente.
emerg	il sistema è inusabile.
panic	sinonimo di <code>emerg</code> , deprecato.

Tabella 3.4: Le varie priorità dei messaggi del servizio di *syslog*.

Oltre a queste parole chiave `syslogd` riconosce alcune estensioni, un asterisco “*” seleziona o tutti 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 certo servizio.

Si possono infine associare più selettori ad una stessa azione separandoli con il “;” mentre ulteriori estensione di questa sintassi sono date dal segno “=” che permette di registrare solo una specifica priorità, e dal segno “!” che permette di escludere una specifica priorità.

L'*azione* usata come secondo campo è un termine astratto per descrivere come si vuole che siano registrati i messaggi, il caso più comune è scriverli in un file, che in questo caso esso dovrà

essere specificato dal pathname assoluto, si può inserire un “-” opzionale davanti al nome per impedire che il contenuto del file venga sincronizzato ad ogni messaggio, lasciando spazio per la bufferizzazione degli stessi.

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 indica che si sta facendo riferimento ad una fifo (vedi sez. 1.2.1), oppure si può mandare l’output su una console specificando come file quello di un dispositivo a terminale (ad esempio `/dev/tty10`). Si possono anche mandare i messaggi a liste di utenti, identificati per username e separate da virgole, e questi li riceveranno sul terminale su cui sono collegati, infine il carattere “*” fa sì che i messaggi siano inviati a chiunque sia collegato.

Una delle caratteristiche più utili del **syslog** è che si possono mandare tutti i messaggi ad una macchina remota. Questo si fa usando il carattere “@” seguito dall’*hostname* della destinazione. Se su quella macchina è stato predisposto un **syslogd** abilitato all’ascolto via rete questo riceverà tutti i messaggi. Si può realizzare così una macchina dedicata solo a questo servizio, in modo da proteggere i file di log, che spesso possono contenere informazioni preziose utili in caso di intrusione (ovviamente detta macchina deve essere molto ben protetta).

Un esempio per il file `/etc/syslog.conf` è il seguente, che si è estratto dal file di riferimento che viene installato di default su una Debian Sid:

```
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                   *
```

In questo caso si noti come tutti i messaggi relativi ai *servizi* **auth** e **authpriv**, **daemon**, **kern**, **lpr**, **mail**, **user** e **uucp** vengano salvati su file a parte, mentre sul file **syslog** vengono mandati tutti i messaggi eccettuati quelli relativi ad **auth** e **authpriv**. Inoltre sono trattati a parte i messaggi relativi al sistema di posta, dove per ciascuna priorità viene usato un file a parte.

Vengono poi definiti alcuni file per i messaggi generici come **messages** in cui si richiede l’invio dei messaggi solo delle priorità **info**, **notice** e **warn** escludendo una serie di *servizi* i cui

messaggi sono archiviati altrove. Infine si noti come le emergenze (ad esempio i messaggi di uscita di **shutdown**) vengano stampati su tutti i terminali attivi.

3.3.4 Il sistema di rotazione dei file di log

I file di log, la cui produzione è governata dal demone **syslogd** appena trattato, sono una delle caratteristiche più utili di un sistema unix-like in quanto vi sono registrati i messaggi (errori, avvertimenti, notifiche, etc.) provenienti dai vari servizi, e per questo sono di importanza fondamentale per capire le ragioni di eventuali malfunzionamenti.

Il problema con i file di log è che essi tendono a crescere di dimensione, per cui si rischia che finiscano fuori controllo, riempiendo la partizione su cui risiede la directory **/var/log/** in cui normalmente vengono memorizzati. Per risolvere questo problema esiste il programma **logrotate** che, lanciato su base periodica, (al solito da **crond**) gestisce la *rotazione* dei file di log specificati, con tanto di eventuale compressione, rimozione delle versioni troppo vecchie, ed e-mail di avviso all'amministratore.

Il comando deve essere lanciato specificando come argomento un file di configurazione che specifica le modalità di rotazione. Se si usa l'opzione **-v** il comando stamperà una serie di messaggi durante le operazioni di rotazione. L'opzione **-d** permette di eseguire il comando in modalità di *debug* (ed implica **-v**), mentre **-f** forza il comando ad eseguire una rotazione anche quando questa non è in programma. Al solito per l'elenco completo delle opzioni si può fare riferimento alla pagina di manuale.

Il formato del file di configurazione prevede la presenza di una direttiva per riga. Queste possono essere specificate direttamente nel file e vengono prese come impostazioni globali da applicare in maniera generica a tutti i file. È poi possibile indicare delle opzioni specifiche per un singolo file scrivendo al posto di una direttiva il pathname assoluto di quest'ultimo seguito da un blocco delimitato da parentesi graffe, contenente le direttive da applicargli, specificate sempre una per riga. Infine come per gli altri file di configurazione le linee vuote o che iniziano per **#** vengono ignorate.

Una direttiva particolare è **include** che permette di includere altri file all'interno della configurazione. Il suo uso più interessante è però quello in cui viene usata per includere una directory; in tal caso infatti vengono inclusi tutti i file in essa contenuti²⁶. Questo consente ad ogni servizio che gestisce autonomamente i suoi file di log, di usare un suo file di configurazione con delle impostazioni personalizzate semplicemente scrivendolo in una directory comune.

La rotazione può essere specificata sia per intervalli temporali (con le direttive **daily**, **weekly** e **monthly**) che sulla base del superamento di una certa dimensione (con la direttiva **size**); quando il comando viene invocato viene controllato se la condizione è soddisfatta e solo in questo caso (a meno di non aver specificato l'opzione **-f**) la rotazione viene eseguita. Le altre principali direttive sono riportate in tab. 3.5, la descrizione completa di tutte le direttive e del loro significato si trova nella pagina di manuale del comando, accessibile con **man logrotate**.

Benché sia possibile lanciare il comando a mano, gran parte delle distribuzioni invocano **logrotate** fra le operazioni giornaliere eseguiti da **crond**. In quasi tutti i casi negli script viene usato come standard il file di configurazione **/etc/logrotate.conf** mentre ulteriori file di configurazione di singoli servizi sono mantenuti nella directory **/etc/logrotate.d/**.

In genere si mettono in **logrotate.conf** solo alcune opzioni generali, le opzioni specifiche per ciascun servizio vengono messe nella directory **/etc/logrotate.d/**. In questo modo si fa sì che ogni pacchetto che ha bisogno di produrre dei log e ruotarli, inserisca in questa directory un opportuno file di configurazione per **logrotate** quando viene installato. Il contenuto della directory viene poi incluso dall'apposita direttiva presente in **logrotate.conf** così che non ci sia necessità di modificare quest'ultimo ogni volta che si installa un altro pacchetto.

²⁶a parte quelli con alcuni caratteri o con certe estensioni che li identificano come copie, definibili con la direttiva **tabooext** (vedi tab. 3.5).

Direttiva	Significato
daily	effettua una rotazione giornaliera.
weekly	effettua una rotazione settimanale, un file viene ruotato se è passata più di una settimana dall'ultima rotazione.
monthly	effettua una rotazione mensile, i file vengono ruotati la prima volta che il comando viene eseguito in un certo mese.
size	effettua la rotazione al superamento di una certa dimensione da parte del file di log (invece che su un periodo temporale); richiede che si specifichi la dimensione come parametro (supporta i suffissi k , M e G).
rotate	specifica il numero di copie (da passare come parametro) dei file di log che devono essere mantenute in una successione di rotazioni.
include	legge tutti il file passato come parametro, o se trattasi di directory, tutti i file presenti all'interno di questa che non abbiano nel nome una delle estensioni vietate (tramite la direttiva tabooext).
create	crea un nuovo file di log vuoto immediatamente dopo aver eseguito la rotazione del precedente assegnandogli un insieme di permessi, un proprietario ed un gruppo proprietario, che devono essere specificati come parametri.
tabooext	permette di definire una lista di estensioni e caratteri vietati nei nomi dei file da includere con la direttiva include ; questi devono essere passati come lista separata da virgole. Un carattere + indica di aggiungere la lista ai caratteri già esclusi, di default questi sono .rpmorig , .rpmsave , .dpkg-dist , .dpkg-old , .dpkg-new , .disabled , .v , .swp , .rpmnew , e ~ .
compress	comprime le vecchie versioni dei file di log usando gzip .
missingok	se il file di log è assente non dà errori.

Tabella 3.5: Le principali direttive usate nel file di configurazione di **logrotate**.

Un esempio del contenuto di **/etc/logrotate.conf** è il seguente, i commenti spiegano in maniera molto chiara il significato delle varie opzioni, l'esempio è estratto dalla versione del file installata su una Debian:

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

# keep 4 weeks worth of backlogs
rotate 4

# 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
```

In questo caso prima si specifica di ruotare i file settimanalmente e di mantenere quattro copie degli stessi; gli eventuali errori verranno notificati via e-mail all'amministratore. Poi si dice di leggere tutto il contenuto della directory `/etc/logrotate.d`, infine si cambiano le impostazioni di default per i due file `/var/log/wtmp` e `/var/log/utmp` che non fanno riferimento a nessun pacchetto (contengono i dati delle sessioni degli utenti) che vengono ruotati mensilmente invece che settimanalmente.

3.4 L'X Window System

In questa sezione tratteremo la configurazione dell'interfaccia grafica nei sistemi Unix, in particolare vedremo le caratteristiche principali del sistema *X Window System* (spesso chiamato per brevità *X11*, dalla versione finale del protocollo), e di come gestirlo e configurarlo.

3.4.1 Introduzione a X Window System

Benché in realtà esistano diverse alternative per la realizzazione di applicazioni in ambiente grafico, sia attraverso l'uso diretto del *framebuffer* o dell'hardware video, sia attraverso opportuni *toolkit* grafici che attraverso sistemi a finestre completi come *Fresco* (<http://www.fresco.org/>), a tutt'oggi l'ambiente grafico più utilizzato in un sistema GNU/Linux (ed in generale in un qualunque sistema Unix) è *X Window System*.

Il fatto che esistano delle alternative complete per l'uso dell'interfaccia grafica ci mette immediatamente di fronte ad una delle principali differenze che c'è fra un sistema unix-like come GNU/Linux ed altri sistemi operativi: al contrario di altri sistemi come Windows o MacOS in un sistema GNU/Linux l'interfaccia grafica non fa parte del kernel, e non ha nessuna caratteristica privilegiata, ma viene eseguita dal kernel in user space come un qualunque altro programma. Tutto il sistema grafico in sostanza non è che un altro strato che viene posto sopra il kernel, che può essere tranquillamente sostituito da uno strato diverso, e che può anche essere eliminato tutte le volte che non serve (ad esempio sui server).

L'interfaccia grafica *X Window System* nasce a metà degli anni '80 con il progetto *Athena* all'MIT, con l'intento di fornire un ambiente grafico per i sistemi UNIX, e costituisce probabilmente la prima interfaccia grafica mai realizzata su un sistema operativo. La caratteristica principale del sistema, che a tutt'oggi lo distingue dalle più diffuse interfacce grafiche presenti in altri sistemi operativi, è che, oltre a fornire alle usuali funzionalità che permettono disegnare finestre ed altri elementi grafici su uno schermo, il sistema è completamente trasparente rispetto alla rete. Questo significa che *X Window* è in grado di operare allo stesso modo sia in locale, interfacciandosi direttamente all'hardware grafico della macchina su cui si sta operando, sia in rete, inviando attraverso di essa le informazioni per disegnare la grafica di una applicazione su una macchina remota.

L'architettura di *X Window* infatti definisce una relazione *client-server*, illustrata in fig. 3.1 fra una applicazione ed il suo *display*, fornendo una serie di funzioni di libreria (tramite le *Xlib*) che separano completamente l'applicazione (detta per questo *X-client*) dal programma che si incaricherà di disegnarne le finestre (l'*X-server*). È questa separazione che consente di definire un protocollo di comunicazione (l'*X protocol*), fra client e server, che rende l'intera interfaccia trasparente rispetto alla rete.

Si tenga presente però che in questa architettura in genere client e server hanno posizione invertita rispetto al solito. Infatti in genere è il client che viene eseguito localmente per interrogare un server remoto che fornisce delle risposte; in questo caso sono dei client remoti (le

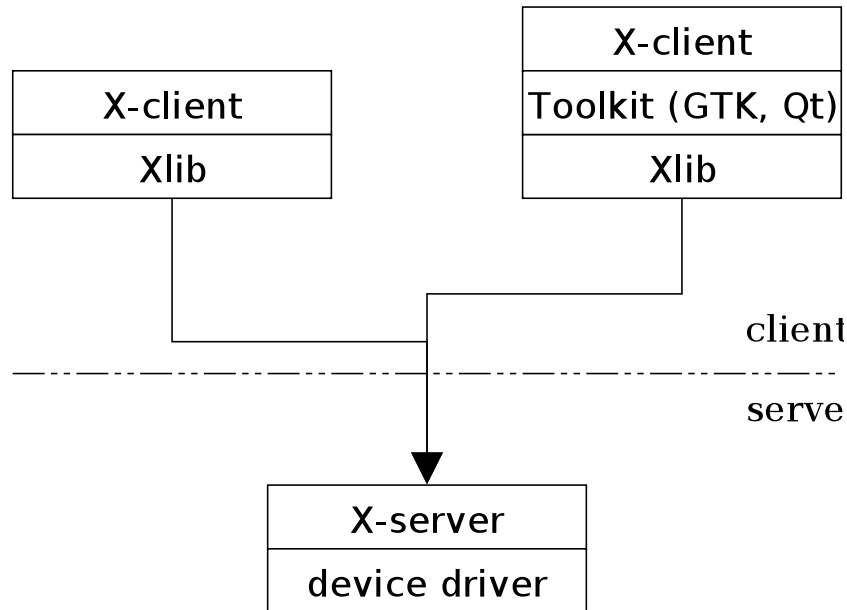


Figura 3.1: Schema di funzionamento del protocollo X11

applicazioni) che inviano le informazioni al server locale (l'*X-server*) perché questo ne disegni le finestre.

Nel caso di GNU/Linux la versione più diffusa dell'*X Window System* è XFree86, disponibile su <http://www.xfree86.org/>, che fornisce anche numerose estensioni (come il rendering dei font truetype, e varie accelerazioni grafiche) attraverso una architettura modulare. In realtà essendo un protocollo standardizzato il cui codice di riferimento è sempre stato rilasciato con licenza libera,²⁷ ne esistono anche varie versioni.

A causa di un recente cambiamento di licenza che lo rende incompatibile con la GPL, è molto probabile che XFree86 resti l'implementazione più diffusa ancora per poco. Infatti con la conversione dell'*X-Consortium*, il titolare dei diritti dell'implementazione originale di *X Window*, da consorzio di industrie a fondazione aperta, si è ripartiti dall'ultima versione di XFree86 con la licenza originale ed è già stata rilasciata una prima versione del server *X.org* (si veda <http://www.x.org/>), che mantiene la precedente licenza e che è già previsto essere il sostituto di XFree86 nelle nuove versioni delle principali distribuzioni. Per il momento però la gran parte delle distribuzioni continua ad usare la vecchia versione di XFree86, e noi faremo riferimento, almeno per i nomi dei file, a quella.

3.4.2 La configurazione del server X

Il primo passo per utilizzare l'interfaccia grafica su GNU/Linux è configurare il server X sulla propria macchina, perché questo possa disegnare le finestre e ricevere i dati da tastiera e mouse (oltre che dai client). Fino alla versione 3 di XFree86 il primo passo era quello di identificare la propria scheda grafica e installare il pacchetto con il relativo server X in grado di parlare con essa. A partire dalla versione 4 il supporto delle varie schede è stato modularizzato, ed il pacchetto che contiene il server è uno solo. Noi faremo sempre riferimento a questa ultima versione.

La configurazione del server in XFree86 è governata dal file **XF86Config**, il formato di questo file è stato modificato a partire dalla versione 4, per cui nelle distribuzioni più recenti si trova

²⁷la cosiddetta *licenza X11*, creata appunto per questo progetto e poi utilizzata in seguito anche per molti altri, che non essendo *copyleft* consente anche l'esistenza di versioni proprietarie.

al suo posto il file `XF86Config-4`. Il server X in genere cerca prima quest'ultimo e se non lo trova ricorre al precedente. Tutte le distribuzioni mantengono i vari file di configurazione dell'*X Window System* in una directory separata, sotto `/etc/X11`.

Le informazioni necessarie alla configurazione del server X sono quelle che riguardano le varie componenti utilizzate dallo stesso, e cioè mouse, tastiera, scheda video e monitor. Si tenga presente che fin dall'inizio XFree86 ha sempre supportato la presenza di schermi multipli, e che nelle versioni più recenti sono state inserite pure delle estensioni che permettono di “*spalmare*” un desktop su più schermi.

Configurare il server X significa allora creare un opportuno file `/etc/X11/XF86Config-4` che lo metta in grado di utilizzare l'hardware; in generale questo viene fatto automaticamente in fase di installazione (specie per le nuove distribuzioni su live-CD che usano l'autorilevamento dell'hardware) o attraverso un opportuno programma di configurazione che chiede i parametri.

Ogni distribuzione ha creato un suo programma dedicato, ma con il pacchetto XFree86 vengono forniti direttamente anche due programmi per la configurazione, `xf86cfg` che opera ad interfaccia grafica e `xf86config` che opera linea di comando; questi programmi, una volta inseriti i dati dell'hardware, permettono di generare automaticamente il file di configurazione. Infine si può generare uno scheletro del file direttamente con il comando `X -configure`.

Il file è diviso in varie sezioni ciascuna dotata di un suo nome che la identifica, introdotta dalla parola chiave **Section** e conclusa dalla parola chiave **EndSection**, all'interno di queste sezioni dovranno poi essere specificate le opzioni ed i valori delle configurazioni (dipendenti dal tipo di sezione), con una sintassi generica del tipo:

```
Section "SectionName"
    SectionEntry
    ...
EndSection
```

I nomi delle principali sezioni utilizzate comunemente sono riportati in tab. 3.6, l'elenco completo può essere trovato nella pagina di manuale accessibile al solito con `man XF86Config-4`. Le sezioni hanno poi una loro gerarchia, ed alcune di esse devono essere necessariamente specificate.

Nome	Contenuto
Files	Indica i pathname di file o directory contenenti informazioni usate dal server come le directory dei font o i file con le descrizioni dei colori.
ServerFlags	Indica le opzioni generali che controllano globalmente il comportamento del server, di norma non viene utilizzato e valgono i valori di default delle stesse.
Module	Indica i moduli aggiuntivi che devono essere caricati dinamicamente (e che forniscono estensioni come le accelerazioni 3D e il rendering dei font truetype).
InputDevice	Contiene la descrizione di un dispositivo di input (mouse, tastiera, tavoletta grafica, ecc.) ne deve essere specificata una per singolo dispositivo.
Device	Contiene le informazioni di configurazione di una scheda grafica, ne deve essere specificata una per ogni scheda disponibile.
Monitor	Contiene le informazioni di descrizione di un monitor, ne deve essere specificata una per ogni monitor disponibile.
Modes	Contiene la descrizione di una modalità video (corrisponde a specificare risoluzione, frequenza di refresh e profondità di colore).
Screen	Contiene la definizione di uno <i>schermo</i> , combinando una scheda video ed un monitor.
ServerLayout	Contiene la configurazione generale del server.
DRI	Contiene le specifiche per il Direct Rendering Interface un'interfaccia diretta alle accelerazioni hardware delle schede grafiche.

Tabella 3.6: I nomi delle varie sezioni del file `XF86Config-4`.

Al livello più alto c'è la sezione **ServerLayout** che serve a collegare insieme i dispositivi di ingresso e di uscita che vengono usati dal server in una sessione. Un singolo dispositivo di ingresso deve essere descritto in una sezione **InputDevice**, mentre i dispositivi di uscita devono essere collegati in una sezione **Screen**, che comprende una sezione **Device** che descrive una scheda video ed una sezione **Monitor** che descrive il monitor ad essa collegato. Le altre sezioni sono invece indipendenti, e non dipendono dalle altre.

Ciascuna sezione comprende una serie di direttive, proprie della sezione, che vanno espresse su una singola riga. Queste sono introdotte da una parola chiave seguita da uno o più valori, che hanno un formato comune per tutte le sezioni e possono essere di tre tipi diversi: o dei valori numerici interi, espressi direttamente col numero o con un prefisso **0x** qualora li si scrivano in esadecimale, o dei valori numerici in virgola mobile, anch'essi scritti direttamente, o delle stringhe, che devono essere espresse inserendole fra virgolette.

Inoltre per tutte le sezioni esiste la direttiva generica **Options** che specifica il valore di una opzione; questa richiede sempre due parametri di tipo stringa, il primo dei quali ne specifica il nome ed il secondo il valore. La stringa che specifica quest'ultimo viene poi interpretata ed il suo contenuto viene classificato in altrettanti tipo di valore, i numeri (reali o interi, secondo il formato visto per i parametri generici) verranno interpretati come tali, le parole chiave **yes/no**, **on/off**, **true/false** esprimeranno invece dei valori logici, mentre apponendo ad un numero uno dei suffissi **Hz**, **kHz**, **MHz**, ecc. si potranno specificare delle frequenze; il resto verrà interpretato come stringa (l'elenco completo dei tipi di valori è comunque riportato nella pagina di manuale). Inoltre usando il suffisso **No** nello scrivere il nome di una opzione si specificherà per essa direttamente un valore logico pari a **No**, **off** o **false**.

Tratteremo qui solo le principali sezioni, indicando le principali direttive previste per ciascuna di esse; al solito per l'elenco completo si può fare riferimento alla pagina di manuale. La sezione **Files** contiene la definizione di una serie di pathname che riguardano la posizione di file e directory utilizzate dal server. Le direttive possibili sono le seguenti:

FontPath specifica una lista di directory, separate da virgole, in cui si trovano i font usati dal server. Si può usare la direttiva più volte, e le relative directory verranno unite insieme. La direttiva richiede o un pathname assoluto ad una directory, o l'identificatore di un *font server*,²⁸ nella forma:

`<trans>/<hostname>:<port-number>`

dove **trans** indica la modalità di connessione (e vale **unix** per i socket locali e **tcp** per un collegamento remoto) **hostname** indica il nome della macchina cui si collega e **port** la porta. Un esempio tipico è:

`FontPath "unix/:7100"`

che indica il fontserver sulla macchina locale.

RgbPath indica il pathname del file contenente il database dei colori RGB, qualora non specificato viene utilizzato il valore di default che è `/usr/X11R6/lib/X11/rgb`.

ModulePath indica il pathname delle directory in cui verranno cercati i moduli binari che forniscono le estensioni del server, come per la direttiva **FontPath** si possono specificare più directory usando più istanze della direttiva. Qualora non sia specificato nulla viene usata di default la directory `/usr/X11R6/lib/modules/`.

²⁸il font server è un servizio fornito dal programma **xfs** che permette di mantenere i font su una singola macchina, e distribuirli via rete ad altre, in modo da diminuire lo spazio disco necessario all'installazione del server o consentire l'utilizzo di font ai cosiddetti *dumb terminal* non dotati di disco; veniva utilizzato, prima del supporto diretto del rendering dei font truetype per generare automaticamente da questi ultimi dei font compatibili con il server X allora in uso.

La sezione **ServerFlags** permette di impostare alcune opzioni generali relative al server, essa contiene solo direttive di tipo **Options**, la sezione **Module** è usata per specificare quali moduli di estensione del server devono essere caricate; questa in genere prevede solo la direttiva **Load** seguita dal nome del modulo, questo è lo stesso del file corrispondente che si deve trovare all'interno del path indicato **ModulePath**, tolto il **lib** iniziale e l'estensione. L'unica formulazione alternativa è usare una sintassi del tipo:

```
SubSection "extmod"
    Option "omit XFree86-DGA"
EndSubSection
```

dove il nome del modulo è lo stesso e la direttiva **Option** viene utilizzata per passare dei parametri al modulo.

La sezione **InputDevice** serve a specificare le caratteristiche di un singolo dispositivo di input, e normalmente se ne hanno almeno due, una per la tastiera e l'altra per il mouse, ma se ne possono specificare anche di più, in caso di presenza di più tastiere (caso non molto comune) o di più mouse (caso comune invece con i portatili, cui in genere si usa un mouse USB da affiancare al touchpad, o con l'uso di tavolette grafiche). Ogni sezione viene specificata nel formato:

```
Section "InputDevice"
    Identifier "nome"
    Driver      "dispositivo"
    Option      "... "
    ...
EndSection
```

dove le direttive **Identifier**, che indica un nome cui fare riferimento al dispositivo nelle altre sezioni e **Driver**, che ne specifica il tipo, devono sempre essere specificate. In particolare **Driver** permette di indicare se il dispositivo in questione viene usato per il puntamento (con il valore **mouse**) o per l'immissione di dati (con il valore **keyboard**) facendo riferimento al caso più comune di entrambe le tipologie. Le restanti direttive sono tutte da specificare nella forma di opzioni, le principali delle quali sono:

CorePointer	di valore booleano, che se viene impostata indica che il relativo dispositivo viene considerato come il mouse principale.
CoreKeyboard	di valore booleano, che se viene impostata indica che il relativo dispositivo viene considerato come la tastiera principale.
Device	indica il file di dispositivo da utilizzare per accedere al relativo dispositivo.
SendCoreEvents	invia gli eventi relativi ad un altro dispositivo come se fossero generati dal dispositivo principale (lo si usa per abbinare gli eventi di un eventuale mouse USB a quelli della touchpad sui portatili).
XkbModel	indica, nel caso di una tastiera, il modello (ad esempio pc105 per una tastiera da PC a 105 tasti).
XkbLayout	indica, nel caso di una tastiera, la disposizione dei tasti, e corrisponde in genere al nome della relativa localizzazione (con valori come us , it , fr , de , ecc.)
Protocol	indica nel caso di un mouse, il protocollo utilizzato (i più comuni sono PS/2 per i mouse ordinari e ImPS/2 per quelli dotati di rotellina).

La sezione **Device** serve a specificare le caratteristiche di una scheda grafica, e ne deve esistere almeno una (ma se ne possono avere diverse in caso di presenza di più schede grafiche). Il suo formato è identico a quello visto in precedenza per **InputDevice**, e prevede le due direttive obbligatorie **Identifier** e **Driver**, dove quest'ultima indica il nome del modulo associato al driver che gestisce la relativa scheda grafica.

Il resto della sezione prevede come nel caso precedente solo delle direttive **Option**, buona parte delle quali sono specifiche del driver della scheda grafica utilizzato e sono descritte nella relativa pagina di manuale, che si accede nella sezione 4 usando il nome del driver stesso (ad esempio per una scheda Matrox che si utilizza con il driver **mga** si potranno ottenere le informazioni con **man mga** o **man 4 mga**). Alcune opzioni generiche, valide per tutti i driver, sono tuttavia specificate direttamente nella pagina di manuale di **XF86Config-4**.

La sezione **Monitor** serve ad indicare le caratteristiche di un monitor, e ne deve essere presente almeno una. Il formato generale della sezione è del tipo:

```
Section "InputDevice"
    Identifier "nome"
    direttiva
    ...
EndSection
```

dove come nei casi precedenti **Identifier** specifica una stringa identificativa del monitor (che può essere qualunque) ed è obbligatoria; le restanti direttive servono a specificare le caratteristiche del monitor, le principali sono:

VendorName	stringa che identifica la marca del monitor.
ModelName	stringa che identifica il modello del monitor.
HorizSync	indica la frequenza (o l'intervallo di frequenze) di sincronia orizzontale ²⁹ supportato dal monitor. Può essere indicato come singola lista di valori separati da virgole come intervallo separato da un meno, e se non specificata con uno degli indicatori illustrati in precedenza è supposto essere espressa in kHz.
VertRefresh	indica la frequenza (o l'intervallo di frequenze) di aggiornamento verticale ³⁰ supportato dal monitor. Come per HorizSync può essere indicato sia come singola lista di valori che come intervallo, espresso di default in Hz. Questo valore viene usato dal server X per determinare se un modo video è compatibile con il monitor.
Mode	è una direttiva speciale da scriversi su più linee (chiusa da una direttiva EndMode) che permette di specificare una modalità video cui viene dato il nome passato come argomento, mentre i valori da usare in termini di risoluzione e frequenze di aggiornamento vengono specificati nelle righe successive (per la sintassi si consulti la pagina di manuale). In genere non è necessario utilizzarla in quanto vengono riconosciuti come predefiniti una serie di modalità video in standard VESA, con nomi del tipo di 1024x768, 1240x1024, ecc. da utilizzare in una successiva sezione Screen .

In genere i valori per **HorizSync** e **VertRefresh** sono critici per l'uso del monitor, in quanto sbagliandoli è possibile mandare fuori sincronia lo stesso con il risultato di non vedere più nulla.

²⁹la frequenza di sincronia orizzontale è la frequenza a cui il monitor può spostare il fascio che disegna una riga orizzontale.

³⁰è la frequenza con cui può essere ridisegnato l'intero schermo.

I monitor recenti supportano comunque un meccanismo di notifica alle applicazione di questi ed altri dati, detto EDID, che evita di doversi andare a cercare i rispettivi valori su una qualche oscura pagina del manuale (sempre che si siano scomodati a scriverceli). Questi dati possono essere letti tramite il programma `get-edid` il cui output può essere trasformato direttamente nella corretta sezione **Monitor** pronta per l'inclusione in **XF86Config-4** mandandolo in *pipe* al comando `parse-edid`.³¹

Una volta definite le schede video ed i monitor presenti, questi vanno collegati fra loro oltre che con il cavo tramite una apposita sezione **Screen** che identifica l'insieme dei due, e definisce le caratteristiche dello "schermo" che verrà utilizzato dal server X. Il formato generale di questa sezione è il seguente:

```
Section "Screen"
    Identifier "name"
    Device      "device id"
    Monitor     "monitor id"
    direttive
    ...
    SubSection "Display"
        voci
        ...
    EndSubSection
    ...
EndSection
```

di nuovo la direttiva **Identifier** serve a fornire una stringa che identifica lo schermo in questione, ed è obbligatoria come le due **Device** e **Monitor** che servono a specificare la scheda video ed il monitor collegati allo schermo, che prendono come argomento l'identificatore utilizzato in una delle sezioni omonime. Viene spesso utilizzata anche la direttiva **DefaultDepth** che indica la profondità di colore (in numero di bit) da utilizzare di default per lo schermo quando si hanno più sottosezioni di tipo **Display**.

È inoltre obbligatoria la presenza di almeno una sottosezione **Display** che specifichi la profondità di colore (tramite la direttiva **Depth**, obbligatoriamente presente) e una (o più) varie modalità video (con la direttiva **Modes** che prende un elenco di nomi (separati da spazi) di quelli predefiniti nello standard VESA o indicati da una precedente direttiva **Mode** inserita nella corrispondente sezione **Monitor**. Uno schermo infatti può essere utilizzato a diverse profondità di colore, diverse risoluzioni e diverse frequenze di aggiornamento a seconda delle caratteristiche della scheda video e del monitor, ed qualora si specifichino più modi, è pure possibile passare al volo dall'uno all'altro una volta avviato il server con la combinazione di tasti **Ctrl-Alt-+** e **Ctrl-Alt--**.

Infine la sezione **ServerLayout** serve ad ottenere una configurazione completa collegando uno o più schermi (definiti in altrettante sezioni **Screen**) con uno o più dispositivi di input (definiti in altrettante sezioni **InputDevice**). Di nuovo ne deve essere presente almeno una, nella forma generica:

```
Section "ServerLayout"
    Identifier      "nome"
    Screen          "screen id"
    ...
    InputDevice     "input device id"
    ...
```

³¹ nel caso di Debian per poter utilizzare questo comandi occorrerà installare il pacchetto `read-edid`.


```

    opzioni
    ...
EndSection

```

di nuovo **Identifier** indica una stringa di identificazione, mentre **Screen** e **InputDevice** servono a referenziare quali schermi e dispositivi di input inserire nel layout, usando l'identificatore specificato nelle omonime sezioni.

3.4.3 L'avvio del server

Le modalità per attivare una sessione grafica sono sostanzialmente due, quella che prevede l'avvio del server X e di tutto l'ambiente grafico da console, e quella che prevede direttamente il login dall'ambiente grafico, che viene lanciato direttamente nella procedura di avvio (tramite gli opportuni script) insieme al programma di gestione del login.

Nel caso dell'avvio da console il programma utilizzato per lanciare una sessione è **startx**, questo in realtà non è altro che una interfaccia semplificata per chiamare un altro programma, **xinit**. È quest'ultimo che avvia il server X e poi esegue un eventuale programma client passato come argomento, ad esempio un terminale da cui eseguire gli altri comandi. All'uscita del suddetto programma **xinit** si incarica anche di terminare il server X; si tenga presente però che questo avviene anche se gli eventuali altri programmi lanciati dal primo client eseguito direttamente da **xinit** sono ancora attivi.

Per questo usualmente non si specifica nessun programma, poiché in tal caso **xinit** cerca se nella home dell'utente esiste un file **.xinitrc** e lo esegue come se fosse uno script di shell, terminando il tutto solo alla fine di questo. Questo consente di utilizzare il contenuto di **.xinitrc** per inizializzare il proprio ambiente grafico, chiamando al suo interno i programmi che si intendono lanciare. In questo però occorre considerare che i programmi indicati in **.xinitrc** vengono eseguiti uno dopo l'altro, per cui se si vuole che vengano eseguiti in maniera concorrente occorrerà lanciarli in background, tranne l'ultimo alla cui uscita terminerà anche la sessione grafica. Per questo motivo in genere si lancia sempre per ultimo un *window manager* o un *session manager* (vedi sez. 3.4.4).

Come accennato usualmente invece di usare direttamente **xinit** si lancia la sessione usando **startx**, quest'ultimo infatti permette di definire opportunamente i parametri necessari all'esecuzione del secondo, integrandosi con la distribuzione in modo da utilizzare gli eventuali *window manager* o *session manager* scelti come default. Altrimenti il secondo, in assenza di uno **.xinitrc** dell'utente, lancerebbe semplicemente **xterm**.³²

La seconda modalità di avvio del server X è attraverso un *login manager* grafico (in realtà nella nomenclatura di X questo viene chiamato *X Display Manger*). Quest'ultimo è semplicemente un programma che lancia il server e poi si incarica di mostrare all'utente una finestra di login (grafica) su cui questo può autenticarsi per entrare nel sistema ed eseguire una sessione. Un vero *login manager* deve però essere in grado di fare anche qualcosa di più, e gestire cioè la stessa procedura di autenticazione e di avvio di una sessione pure da remoto, sfruttando la trasparenza sulla rete del protocollo X11, attraverso il supporto di un apposito protocollo, l'**XDMCP** (*X Display Manager Control Protocol*).

Questo consente di presentare anche su macchine remote, che a questo punto assumeranno il compito di fare da semplici *terminali* per la sessione, la stessa finestra di login. In questo modo è possibile centralizzare le applicazioni grafiche su un unico server su cui esse saranno eseguite, ed riutilizzare macchine dotate anche di scarse risorse hardware, che non sarebbero in grado di sopportare il carico di tutta l'interfaccia grafica, solo per il disegno delle finestre attraverso l'installazione del solo X server..

³²il programma **xterm** è probabilmente il più semplice dei programmi ad interfaccia grafica, e si limita semplicemente a riprodurre dentro una finestra un terminale del tutto analogo a quello che si trova sulla console.

Nella distribuzione classica di X Window, quella distribuita direttamente con l'implementazione di riferimento dell'X-server, il programma che implementa il *login manager* è **xdm**. Il programma ha una interfaccia molto spartana, ma è ampiamente configurabile in quanto il suo comportamento viene governato da una serie di script di shell che eseguono i vari compiti, mentre la configurazione è effettuata tramite le delle opportune *risorse* (la trattazione di queste ultime è in sez. 3.4.4) che vengono specificate nel file di configurazione `/etc/X11/xdm/xdm-config`. Il programma però onora anche una serie di opzioni, da indicare nel file `/etc/X11/xdm/xdm.options`, che assumono i valori descritti nell'omonima pagina di manuale, e controllano il comportamento del programma.

Come accennato il funzionamento di **xdm** prevede l'uso di una serie di script e programmi, ad esempio la finestra di login in cui inserire username e password viene gestita dal programma **xlogin**, una volta terminato con successo il login **xdm** esegue lo script **Xstartup** (situato in `/etc/X11/xdm`) per inizializzare la sessione, e se questo ha successo il successivo **Xsession**, che tipicamente usa il file `.xsession` nella home directory dell'utente come analogo di `.xinit` per lanciare la sessione grafica dell'utente.

I file `.xinit` e `.xsession` hanno lo stesso scopo e significato, tanto che in Debian la configurazione di default fa sì che sia usato quest'ultimo anche quando si lancia l'ambiente grafico da console con **startx**. Entrambi contengono cioè i programmi da lanciare per gestire la sessione grafica, e come nel caso di **startx** quando il programma che gestisce la sessione termina, **xdm** esegue un reset dell'X-server con lo script **Xreset** e ripresenta la schermata di login iniziale.

Benché si sia trattato più in dettaglio **xdm**, dato il suo legame diretto con il sistema di X Window, nelle distribuzioni GNU/Linux di oggi i *login manager* più utilizzati sono altri, e principalmente **gdm** (lo *Gnome Display Manager*) che fa parte del desktop Gnome e **kdm** (il *KDE Display Manager* che fa parte del desktop KDE. Entrambi offrono una grafica molto più accattivante di **xdm** ed un sistema di configurazione molto più avanzato (e semplificato).

Infatti benché entrambi non necessitino della presenza dei rispettivi ambienti desktop e possano essere usati indipendentemente, vengono solitamente installati in corrispondenza a questi. Il grande vantaggio di entrambi è che, pur essendo controllati da un file di configurazione testuale (rispettivamente `/etc/gdm/gdm.conf` per **gdm** e `/etc/kde3/kdm/kdmrc` per **kdm**), è possibile controllarli e modificarne l'impostazione con un apposito programma di configurazione ad interfaccia grafica (eseguibile anche al loro interno), che permette di attivare e controllare le loro funzionalità senza dover modificare a mano il file di configurazione.

3.4.4 L'uso di X Window dal lato client

Come accennato qualunque programma che usi l'interfaccia grafica tramite le *Xlib* deve agire come client per un server X che si incaricherà di disegnarne le gli elementi grafici sullo schermo. Una delle caratteristiche di X Window è quella di avere definito un meccanismo per memorizzare in maniera generica le preferenze relative all'ambiente grafico di ciascuna applicazione, attraverso quelle che vengono chiamate *risorse*.

Il valore di ciascuna risorsa è specificato da una stringa, ma queste ultime sono organizzate in una gerarchia ad albero³³ in cui ogni nodo è identificato dal nome della risorsa, in cui i singoli nodi della gerarchia sono separati da un "." e che usa come radice il nome del programma stesso (spesso scritto con la prima lettera maiuscola).

In genere ciascuna applicazione mantiene le impostazioni generali per le sue risorse in un

³³le risorse si chiamano così in quanto corrispondono effettivamente a delle strutture dati, che a loro volta possono contenere altre strutture con altri dati; i relativi campi identificano un possibile valore (o una ulteriore risorsa); ciascuna struttura viene associata al singolo programma, e ne controlla vari aspetti come il colore delle finestre, dello sfondo, il titolo della finestra, ecc., il valore è associato ad un certo campo della risorsa stessa, con un certo nome, per cui viene naturale accedere all'insieme di strutture e sottostrutture attraverso una gerarchia ad albero.

file apposito che in fase di installazione viene posto in una directory comune (di default viene usata `/usr/X11R6/lib/X11/app-defaults/`) con il nome stesso della risorsa; i singoli utenti però possono cambiare a piacere questi valori tramite l'uso del file `.Xresources` nella propria home directory, nel quale specificare i valori che all'avvio del server andranno a soprassedere quelli di default. Se poi si vogliono cambiare gli stessi valori una volta avviato il server si può usare il comando `xrdb` che prende come argomento il file in cui sono stati scritti i nuovi valori che saranno caricati immediatamente. Un esempio di un file di configurazione delle risorse può essere il seguente:

```
...
emacs*Background: DarkSlateGray
emacs*Foreground: Wheat
emacs*pointerColor: LightSteelBlue
emacs*cursorColor: LightSteelBlue
emacs*bitmapIcon: on
!emacs*font: fixed
emacs*font: -misc-fixed-***-15-*75-75-***-iso8859-*
emacs.geometry: 80x28
emacs*BorderColor: DarkSlateGray
emacs*fringe.Background: DarkSlateGray
...
```

in cui si modificano le impostazioni dei colori di sfondo e primo piano, si specifica la geometria iniziale ecc. Si noti come la sintassi preveda una riga in cui si specifica il nome della risorsa terminato da “:”, cui segue la stringa che assegna il valore. Si noti anche come il nome della risorsa possa contenere il carattere jolly “*” che assume lo stesso significato che avrebbe se si specificasse un pathname.³⁴ Infine si tenga presente che il carattere “!” serve ad introdurre una riga di commento.

Se le risorse permettono di controllare i vari parametri interni delle singole applicazioni, quello che contraddistingue una interfaccia grafica è la capacità di poter eseguire vari programmi in contemporanea, ciascuno nella sua finestra. Come però ci si potrà rendere conto ad esempio lanciando `xinit` a mano con una serie di programmi qualsiasi, questi ultimi, seguendo la solita filosofia per cui ognuno si cura di eseguire un suo compito ben preciso, si limitano solo a disegnare il contenuto delle rispettive finestre.

Il posizionamento delle stesse, il loro spostamento, il passaggio dall'una all'altra, ecc. devono essere invece gestiti a parte e non sono compito dei singoli programmi. Per questo per poter usare davvero un'interfaccia grafica è necessario avere almeno un'altro programma che esegua questi compiti specifici: quello che viene chiamato un *window manager*, e che assume un po' quello che è il ruolo della shell nella riga di comando (lanciare altri comandi e passare dall'uno all'altro).

Senza un *window manager* infatti si avrebbe la grafica ma non la possibilità di spostare, nascondere, ridimensionare le finestre, o di passare dall'una all'altra. È il *window manager* che si cura di tutti questi compiti, del disegno degli elementi grafici di contorno, e della gestione il mouse per gestire lo spostamento, la ridimensionamento e la selezione delle finestre. Un compito fondamentale infatti, quando si hanno più finestre da gestire, è appunto quello della selezione di quella che viene *messa a fuoco*, quella cioè cui verranno inviati tutti gli eventi in ingresso (ad esempio quello che si scrive sulla tastiera), che è un po' l'equivalente del processo in *foreground* visto in sez. 1.3.4.

Per usare con profitto una sessione grafica tutto quello che serve è un *window manager*, molti di questi infatti gestiscono anche la presenza di un menù o di una barra direttamente sullo schermo (o eventuali altri elementi di contorno), usando i quali è possibile lanciare altre

³⁴cioè applica il valore a tutte le risorse che corrispondono alla stringa in cui nella gerarchia l'asterisco è sostituito da un nome qualunque.

applicazioni o avere informazioni sul sistema. I più comuni, tuttora in ampio uso da parte di coloro che sono abituati alla loro interfaccia e non necessitano di funzionalità più avanzate, sono *Window Maker*, *ICEwm*, *blackbox*, ecc.

Le interfacce grafiche moderne vanno però al di là della semplice gestione delle finestre; per definire una infrastruttura che consenta operazioni come il cosiddetto *drag and drop*, in cui si possono passare elementi o dati da una finestra ad un'altra (e cioè da una applicazione ad un'altra). Per questo un *window manager* non basta, il passo successivo è allora quello dell'uso di un *desktop environment*, cioè di una infrastruttura (completa di meccanismi di intercomunicazione) che definisca non solo la gestione delle finestre, ma tutto un insieme di elementi (come oggetti grafici comuni e meccanismi di intercomunicazione) che porta appunto un ambiente grafico integrato analogo a quello che si trova su altri sistemi operativi come MacOS o Windows.

3.5 Il sistema di stampa

Tratteremo in questa sezione la configurazione delle stampanti sotto GNU/Linux; dopo una introduzione generale sulla gestione generica della stampa vedremo in particolare la configurazione secondo il sistema tradizionale del *Line Printing Daemon* di BSD e la nuova architettura del *Common Unix Printing System*.

3.5.1 Introduzione generale

Mantenendo la filosofia progettuale di Unix per cui *tutto è un file*, è facile immaginarsi che la prima e più immediata modalità di stampa sia quella di inviare il file da stampare direttamente al dispositivo associato alla stampante. Questo è senz'altro possibile, ma si scontra con una serie di problemi che lo rendono altamente sconsigliabile; il primo è che non esiste un dispositivo univoco che identifichi una stampante, questa può essere infatti agganciata ai dispositivi più vari (e molte stampanti ne supportano più di uno), come la parallela, la seriale, una porta USB, ecc. In ciascuno di questi casi il file andrebbe inviato sul relativo dispositivo (nel caso più comune, di stampante su porta parallela, `/dev/lp0`).

Tutto questo non costituirebbe di per sé un grosso problema, basterebbe identificare il dispositivo specifico e identificarlo con un link simbolico (un po' come si fa per il CDROM). Il problema più importante è invece quello relativo al contenuto del file da stampare; un tempo quando si stampavano solo file di testo senza formattazioni il problema non c'era, oggi però si stampano file con contenuti grafici e con formattazioni complesse e font diversi. Per questo in genere le stampanti moderne prevedono un loro linguaggio di stampa come il PostScript (alcune ne supportano anche più di uno), che permette di ottenere i risultati voluti. Questo significa che anche un semplice file di testo debba essere opportunamente *trattato* prima di essere inviato alla stampante. Inoltre se, come accade spesso, i programmi generano le stampe in un formato (principalmente il PostScript) e la stampante non lo capisce, questo dovrà essere opportunamente tradotto.

Il terzo problema è quello di essere in presenza di un sistema operativo multiutente e multitasking; il che comporta che con l'accesso diretto si possono avere più processi che scrivono in contemporanea sul dispositivo della stampante, coi relativi dati che si mescolano fra di loro dando luogo risultati tutt'altro che piacevoli nella stampa finale.

Per questa serie di motivi in genere il dispositivo della stampante non è mai accessibile direttamente ai singoli utenti, ma viene gestito attraverso un opportuno *sistema di stampa* che si incarica di risolvere questi problemi. In generale quello che succede è che l'accesso alla stampante è regolato attraverso un opportuno demone, che è l'unico che ci scrive, le stampe vengono richieste inviando i file da stampare al suddetto demone, che si incarica di eseguire

tutte le operazioni necessarie (comprese le eventuali conversioni di formato) e di garantire un accesso corretto senza sovrapposizioni.

Una delle caratteristiche fondamentali dei sistemi di stampa in ambiente unix-like è quella delle *code*, le stampe cioè non vengono mai eseguite direttamente, ma immesse, tramite un opportuno comando, su una coda di stampa dalla quale poi verranno prelevate (in genere dal demone di cui sopra) ed opportunamente inviate sulla stampante secondo lo schema illustrato in fig. 3.2.

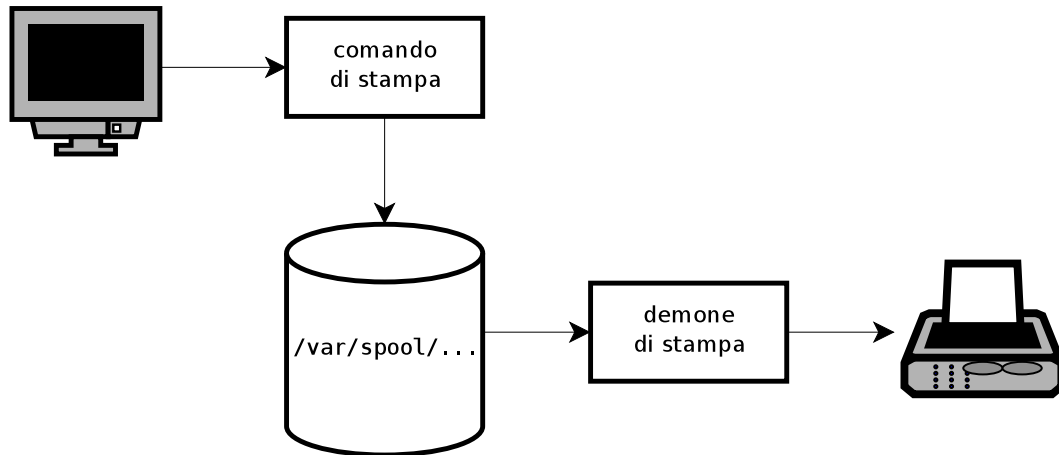


Figura 3.2: Schema generico del funzionamento del sistema di stampa.

Il vantaggio di questo sistema è che la gestione della coda può essere eseguita indipendentemente dalla gestione della stampante, ad esempio diventa possibile inviare la stampa su una coda remota se è previsto un opportuno meccanismo di comunicazione. Lo stesso concetto di stampa remota può essere realizzato in maniera alternativa usando una coda locale ed usando direttamente il demone di stampa per inviare i file ad un altro demone remoto.

Inoltre con lo schema di fig. 3.2 nell'eseguire l'invio dei file dalla coda di stampa alla stampante il demone di stampa può eseguire tutte le operazioni necessarie (realizzate in genere attraverso quelli che si chiamano *filtri di stampa*) per convertire il contenuto del file da stampare in un formato che la stampante è in grado di comprendere.

Infine l'uso di questa architettura permette anche di utilizzare più code di stampa per la stessa stampante; questo consente ad esempio di utilizzare code diverse (con filtri diversi) a seconda del tipo di file che si vuole stampare, o con diversi privilegi o priorità.

3.5.2 Il sistema di stampa in stile BSD

Uno dei sistemi di stampa *storici* in ambiente Unix è quello sviluppato per BSD, e tutt'ora distribuito insieme a questo sistema operativo. È presente un porting anche per GNU/Linux, (nel caso di Debian si può installare con il pacchetto `lpr`) ma il suo funzionamento è difettoso ed il numero di stampanti supportate è limitato. Per questo esso è in genere sostituito da una implementazione alternativa, LPRng, sviluppata indipendentemente che ha la stessa interfaccia e le stesse funzionalità ma supporta anche molte estensioni (in Debian è fornita dal pacchetto `lprng`).

Noi faremo riferimento solo a LPRng, che uno dei sistemi di stampa di uso più comune nelle distribuzioni GNU/Linux (l'altro è CUPS che tratteremo in sez. 3.5.4), ignorando completamente l'implementazione originale di BSD, facendo riferimento al tutto con il nome del protocollo LPD (acronimo di *Line Printer Daemon*) che è pure standardizzato nell'RFC 1179.

Come il nome stesso suggerisce, il cuore del sistema di stampa LPD è l'omonimo demone di stampa `lpd` che gestisce l'invio dei file dalla coda di stampa alla stampante. Il protocollo LPD prevede inoltre che il demone possa essere contattato anche via rete (da un altro demone o direttamente da un client) per eseguire la stampa in remoto, ne tratteremo la configurazione in sez. 3.5.3. Oltre a `lpd` il sistema di stampa in stile BSD prevede una serie di altri programmi per l'immissione delle stampe nelle code e per la gestione di queste ultime.

Opzione	Significato
-#	specifica il numero di copie da stampare, da passare come argomento.
-C	specifica la priorità della <i>stampa</i> , identificata da una lettera maiuscola da "A" (più bassa) a "Z" (più alta), da passare come parametro.
-J	associa alla stampa il nome passato come parametro, in modo che questo venga stampato nell'intestazione (quando questa è prevista).
-m	richiede l'invio di una email all'indirizzo passato come parametro qualora la stampa fallisca.
-U	permette di specificare (come parametro) l'utente per conto del quale eseguire la stampa (utilizzabile solo dall'amministratore).

Tabella 3.7: Le principali opzioni del comando `lpr`.

Per richiedere la stampa di un file si deve utilizzare il comando `lpr`, che prende come argomenti i file da stampare (se non si specifica nulla viene usato lo standard input), una volta messa in coda una serie di stampe il comando riporta sullo standard output un numero identificativo (che potrà essere usato con gli altri comandi per fare riferimento alle stesse) per ciascuna di esse.³⁵

L'opzione principale di `lpr` è `-P` che permette di indicare su quale coda di stampa inviare il file da stampare; se non la si specifica viene usato, se definito, il valore di una delle variabili `PRINTER`, `LPDEST`, `NPRINTER` o `NGPRINTER`, oppure la stampante di default, che è la prima definita in `/etc/printcap`, (vedi sez. 3.5.3). Un'altra opzione utile è `-h` che inibisce la stampa di eventuali pagine iniziali per le code in cui questa è prevista e `-#` che richiede la stampa di un certo numero di copie; le altre opzioni principali sono riportate in tab. 3.7, per l'elenco completo al solito si faccia riferimento alla pagina di manuale.

Per la gestione delle stampe sono disponibili una serie di comandi per controllare lo stato della coda ed eseguirvi sopra delle operazioni, il primo di questi è `lpq`, che permette di esaminare lo stato della coda. Se invocato senza argomenti esso stampa la lista delle stampe presenti, con il formato:

```
parker:/root# lpq
Printer: rlp@parker 'Remote printer entry' (dest epson@davis)
Queue: no printable jobs in queue
Server: no server active
Status: job 'root@parker+648' saved at 16:48:21.800
Rank  Owner/ID              Class Job Files          Size Time
done  root@parker+648        A    648 /root/iptables      2748 16:48:21
epson is ready and printing
root: active                [job 543 localhost]
                               /root/iptables      3072 bytes
```

altrimenti si può specificare come argomento il numero identificativo della stampa (quello riportato nella colonna Job) ed il comando riporterà solo le informazioni ad essa relative.

Di default il comando esegue la ricerca delle stampe sulla coda di default, a meno di non usare l'opzione `-P` per selezionare una coda specifica. Si può usare l'opzione `-a` per fare eseguire

³⁵le stampe sono spesso chiamate, seguendo la nomenclatura inglese, *job*, in particolare i comandi fanno riferimento a questi identificativi come ai *job ID* e riportano il numero di stampe correnti come numero di *job*.

la ricerca su tutte le code; per le altre opzioni e per una descrizione dettagliata si può fare riferimento alla pagina di manuale.

Una volta creata una stampa, la si potrà eliminare dalla coda con il comando `lprm`; questo prende come argomenti la lista delle stampe da rimuovere che possono essere indicati sia tramite il loro identificativo (quello stampato da `lpr` o riportato da `lpq`), che tramite il nome dell'utente ad esse associato. Entrambe le forme supportano l'utilizzo dei caratteri jolly del *filename globbing* (vedi sez. 2.1.4). L'uso della parola chiave `all` seleziona tutte le stampe presenti,³⁶ mentre se non si specifica nulla viene selezionata la prima che può essere rimossa. Di nuovo il comando opera sulla coda di default a meno di non usare l'opzione `-P` per specificarne un'altra o l'opzione `-a` per indicare di operare su tutte le code.

L'uso di `lprm` serve a richiedere a `lpd` la cancellazione delle stampe selezionate, ma un utente potrà rimuovere solo quelle per i quali ha il permesso (per la gestione dei permessi si veda sez. 3.5.3); con l'opzione `-U` si può richiedere (se si è `root`) di operare a nome di un altro utente. Per le altre opzioni ed i dettagli di funzionamento del programma si faccia riferimento alla pagina di manuale.

Chiude la lista dei programmi ausiliari `lpc`, che serve per controllare lo stato del sistema di stampa. Il programma prende come argomenti un comando, seguito da eventuali parametri per lo stesso. Come i precedenti esso opera sulla coda di default a meno di non usare l'opzione `-P` per specificarne un'altra o l'opzione `-a` per indicarle tutte.

Se non si specificano argomenti il programma si porta su una linea di comando interna, da cui questi possono essere specificati. Il primo comando è `help` che stampa la lista di tutti quelli disponibili. Con `status` si ha la stampa dello stato della coda e relativa stampante associata. Si può fermare la stampa con l'uso del comando `stop` e farla ripartire con `start`. Si può invece bloccare l'immissione di stampe su una coda con `disable` e riabilitarla con `enable`. L'elenco dei principali comandi è riportato in tab. 3.8, al solito elenco completo ed i dettagli sono nella pagina di manuale.

Un esempio di risultato di questo comando è il seguente:

```
parker:/var/log# lpq status
Printer: rlp@parker 'Remote printer entry' (dest epson@davis)
Queue: no printable jobs in queue
Server: no server active
Status: job 'root@parker+648' saved at 16:48:21.800
Rank   Owner/ID                      Class Job Files          Size Time
epson is ready
no entries
```

3.5.3 La configurazione della stampa con LPRng

Come accennato il principale programma di un sistema di stampa in stile BSD è il demone `lpd` che gestisce l'invio dei file dalla coda di stampa alla stampante; il demone però è in grado anche di gestire la eventuale ricezione di richieste di stampa via rete (da un altro demone o direttamente da un client), e permette perciò di trasformare una qualunque stampante presente sulla propria macchina in una stampante di rete.

Pertanto la configurazione della stampante su una macchina che usa questo sistema consiste nel configurare opportunamente `lpd`. Nel nostro caso esamineremo la configurazione soffermandoci specificamente sul caso particolare di LPRng, in cui è prevista la presenza di una serie di file di configurazione per i vari compiti eseguiti dal demone.

³⁶questa è una caratteristica della versione di `lprm` del pacchetto `lpr-ng`, il pacchetto originale del demone di stampa di BSD non supporta questa opzione, al suo posto invece si deve usare semplicemente l'argomento `"-"`, che seleziona tutte le stampe di un utente, o del sistema, se usato dall'amministratore.

Comando	Significato
disable	disabilita l'immissione di stampe su una coda di stampa; prende come argomento il nome della coda o all per indicare tutte le code.
down	disabilita sia l'immissione di nuove stampe in coda che la stampa di quelle presenti; prende come argomento il nome della coda o all per indicare tutte le code.
enable	abilita l'immissione di nuove stampe su una coda di stampa; prende come argomento il nome della coda o all per indicare tutte le code.
exit	esce dal programma.
hold	sospende la stampa (su una stampante o per una singola stampa); prende come argomento il nome della stampante ed eventualmente l'identificatore del job.
holdall	sospende automaticamente tutte le nuove stampe, che vengono accodate ma non stampate; prende come argomento il nome della coda o all per indicare tutte le code.
move	sposta una stampa da una coda ad un'altra; prende come argomenti la coda di origine, l'identificativo della stampa e la coda di destinazione.
release	sblocca la sospensione della stampa per una coda o una stampa specifica; prende come argomento il nome della coda ed eventualmente l'identificatore della stampa.
start	avvia una stampante, è utilizzato principalmente quando un demone termina lasciando dei job sulla coda per riavviarlo; prende come argomento il nome della coda o all per indicare tutte le code.
status	riporta lo stato di una coda e relativa stampante; prende come argomento il nome della coda o all per indicare tutte le code.
stop	blocca ogni ulteriore stampa dopo il completamento della stampa corrente; prende come argomento il nome della coda o all per indicare tutte le code.
topq	sposta le stampe selezionate in cima alla coda di stampa; prende come argomento il nome della coda seguito dalla lista degli identificativi delle stampe.
up	abilita sia l'immissione nella coda che la stampa; prende come argomento il nome della coda o all per indicare tutte le code.

Tabella 3.8: I principali comandi di `lpc`.

Il primo file di configurazione, usato da tutti i demoni di stampa in stile BSD, è `/etc/printcap`. Il file contiene la definizione delle code di stampa con la assegnazione delle stampanti (e degli eventuali filtri) ad esse associate. Il file è composto da una serie di blocchi, uno per ciascuna coda che si intende utilizzare. Ciascun blocco è composto da una serie di campi separati dal carattere “:”, un blocco può essere scritto su più righe terminando ciascuna di esse con la barra rovescia “\” come si fa per la shell;³⁷ si tenga conto che l'ultimo campo deve essere anch'esso terminato col carattere “:”. Al solito righe vuote e tutto quello che segue il carattere `#` viene ignorato.

Il primo campo indica il nome della coda di stampa, esso può riportare più nomi (per una migliore identificazione della stampante associata) separati con la barra verticale “|”. Tutti gli altri campi, che controllano le caratteristiche della coda e della stampante associata, possono essere specificati in ordine qualsiasi, ciascun campo inizia sempre per un identificativo di due caratteri, seguito da una assegnazione con un “=” se il suo valore è una stringa o con un “#” se il valore è numerico; un terzo tipo di campi ha valore logico, nel qual caso si usa semplicemente la presenza del rispettivo identificativo.

Di questi campi è fondamentale specificare sempre `sd`, che come il nome fa supporre, indica la *spool directory*, cioè la directory su cui verranno appoggiati temporaneamente i file da stampare; se la stampante è locale deve essere specificato il campo `lp` che indica quale dispositivo usare; se invece la stampante è remota si deve specificare un campo `rm` (da *remote machine*) per indicare

³⁷in realtà con LPRng questo non è necessario, ma è meglio farlo per compatibilità.

l'hostname del computer (o l'indirizzo IP) su cui essa si trova ed il campo **rp** (da *remote printer*) per indicare il nome (cioè la coda di stampa) con cui essa è identificata dal demone LPD presente su quel sistema. I principali campi del file sono riportati in tab. 3.9, al solito per l'elenco completo ed i dettagli si può fare riferimento alla pagina di manuale.

Campo	Significato
if	<i>input filter</i> , indica il filtro da applicare a tutti i file immessi sulla coda prima di inviarli alla stampante, è quello che permette le conversioni di formato, prende come valore il pathname al programma da usare come filtro, che deve leggere i dati dallo standard input e rimettere il risultato del filtraggio sullo standard output.
lf	<i>log file</i> , file su cui vengono registrati tutti gli errori del sistema, prende come valore un pathname.
af	account file , file su cui viene registrato il rendiconto delle pagine stampate, prende come valore un pathname.
lp	<i>line printer</i> , file di dispositivo su cui è situata la stampante (o <i>pipe</i> cui può essere inviato il file da stampare), prende come valore il relativo pathname.
mx	dimensione massima della pagina da stampare, da specificare in byte come valore numerico.
pl	<i>page lenght</i> , lunghezza della pagina in righe, da specificare come valore numerico.
pw	<i>page width</i> , larghezza della pagina in caratteri, da specificare come valore numerico.
rm	<i>remote machine</i> , nome della macchina remota cui inviare le stampe, da specificare come stringa contenente l'hostname o l'indirizzo IP (in notazione dotted decimal).
rp	<i>remote printer</i> , nome della coda di stampa su una macchina remota, da specificare come stringa.
sd	<i>spool directory</i> , nome della directory dove vengono mantenuti i file da stampare presenti nella coda.
sh	<i>suppress header</i> , sopprime la stampa delle intestazioni, è un valore logico.

Tabella 3.9: I principali campi di `/etc/printcap`.

Nel caso di LPRng il demone **lpd** viene controllato anche da altri due file di configurazione, usualmente mantenuti, insieme a tutti gli altri file del sistema, in `/etc/lprng`. Il primo file, **lpd.conf**, contiene la configurazione generale del demone, ma dato che tutti i valori sono ad un default corretto di norma questo non deve essere modificato.

Si può usare questo file per specificare valori di default validi in generale per i parametri di `/etc/printcap`, ma la sintassi prevede che il file contenga l'assegnazione di un parametro per ogni riga (al solito righe vuote e tutto quello che inizia per **#** viene ignorato) anziché separati da “:”. I parametri hanno gli stessi nomi usati in `/etc/printcap`, e l'assegnazione, oltre alle modalità previste per quest'ultimo, prevede anche quella di utilizzare il carattere **@** per identificare un valore logico.

Il file permette inoltre di specificare una serie di configurazioni generali del demone usando delle opportune direttive che non corrispondono a nessun parametro di **printcap**, ad esempio con la direttiva **lockfile** si specifica la locazione del file di lock utilizzato per indicare la presenza di un demone che gira nel sistema, con **printcap_path** si indica la posizione di default del file **printcap**, mentre con la direttiva **include** si possono includere altri file. Al solito si rimanda alla pagina di manuale, accessibile con **man lpd.conf**, per i dettagli e l'elenco completo delle direttive.

Il secondo file di configurazione di **lpd** è **lpd.perms** che viene utilizzato per il controllo degli accessi. Questa è la più grossa differenza con il sistema di stampa di BSD classico, questo infatti utilizza semplicemente il file `/etc/hosts.lpd` (alternativamente viene usato, se presente, anche

`host.equiv`) per indicare le macchine da cui è consentito collegarsi per stampare da remoto, mentre con `LPRng` nessuno di questi file viene utilizzato.

Al solito righe vuote e tutto quello che segue un “#” viene ignorato, se si deve usare quest’ultimo carattere in una configurazione lo si deve pertanto proteggere con una barra inversa “\”. Il file contiene una serie di direttive tutte nella forma:

```
ACCEPT [[not] chiave = valore[,valore]* ]*
REJECT [[not] chiave = valore[,valore]* ]*
DEFAULT ACCEPT
DEFAULT REJECT
```

dove con le parentesi quadre seguite dall’asterisco si intende che una combinazione può essere ripetuta più volte (per specificare ulteriori condizioni).

Le direttive `REJECT` e `ACCEPT` specificano, come il nome rende evidente, se accettare o meno un certo accesso sulla base della condizione espressa di seguito; il controllo dei permessi viene fatto usando un insieme di chiavi il cui valore viene confrontato con quanto specificato nel file, l’uso della direttiva `DEFAULT` serve a stabilire cosa succede quando non sono state specificate condizioni.

Valore	Significato
P	capacità di inviare alla stampante il contenuto della coda.
R	richiesta di <code>lpr</code> da un host remoto.
C	richiesta di controllo remoto tramite <code>lpc</code> .
M	richiesta di rimozione di una stampa tramite <code>lprm</code> .
Q	richiesta di elencazione delle stampe tramite <code>lpq</code> .
S	richiesta dello stato dello spool di stampa con <code>lpc</code> .
U	operazione amministrativa di un utente.
X	connessione remota.

Tabella 3.10: Valori del campo `SERVICE` in `lpd.perms`.

La chiave più importante è `SERVICE`, che permette di classificare il *tipo* di richiesta di accesso, quando non la si specifica si intende applicare la condizione seguente a qualunque tipo di accesso; questa può prendere uno o più dei valori identificata da una lettera, i cui valori possibili, e relativo significato sono riportati in tab. 3.10.

Valore	Significato
REMOTEHOST	lista degli IP da cui accettare connessioni remote (indicato alternativamente come <code>REMOTEIP</code>).
REMOTEPORT	porta da cui viene effettuata la connessione (indicata alternativamente come <code>PORT</code>).
REMOTEUSER	utente da cui proviene la richiesta di accesso.
SERVER	vera quando la richiesta è locale.
LPC	per le richieste di controllo specifica quali dei comandi di <code>lpc</code> possono essere eseguiti
PRINTER	nome della stampante (o meglio della coda) cui viene richiesto l’accesso.
USER	utente cui appartiene la stampa richiesta (definita per le operazioni su stampe esistenti).
HOST	macchina da cui è stato inserito la stampa richiesta (definita per le operazioni su stampe esistenti).
SAMEUSER	la richiesta proviene dallo stesso utente cui appartiene la stampa (definita per le operazioni su stampe esistenti).
SAMEHOST	la richiesta proviene dalla stessa macchina da cui è stato inserito la stampa (definita per le operazioni su stampe esistenti).
FORWARD	equivalente a <code>NOT SAMEHOST</code>

Tabella 3.11: Principali chiavi per le restrizioni di accesso in `lpd.perms`.

Tutte le chiavi partono con un valore nullo, ma ad ogni richiesta di accesso alcune di esse (a seconda del tipo di accesso) vengono inizializzate con i valori relativi ai parametri della richiesta, che poi saranno confrontati con le condizioni indicate all'interno di `lpd.perms`. In tab. 3.11 si sono riportate le principali chiavi ed il relativo significato, per un elenco dettagliato si può fare riferimento alla pagina di manuale accessibile con `man lpd.perms`.

Il file esprime sempre un elenco di condizioni, che vengono controllate sequenzialmente, quando una di queste viene verificata gli ulteriori controlli vengono evitati. Un esempio di `lpd.perms`, estratto dalla versione installata su una Debian Sid, è il seguente:

```
REJECT NOT SERVER
ACCEPT SERVICE=C SERVER REMOTEUSER=root
ACCEPT SERVICE=C LPC=lpd,status,printcap
REJECT SERVICE=C
ACCEPT SERVICE=M SAMEHOST SAMEUSER
ACCEPT SERVICE=M SERVER REMOTEUSER=root
REJECT SERVICE=M
REJECT SERVICE=R FORWARD
DEFAULT ACCEPT
```

In questo caso la configurazione si apre vietando ogni accesso che non sia locale. Dopo di che viene consentito, con le regole per il servizio “C”, solo a root di controllare il demone con `lpc` in maniera completa, mentre per chiunque altro viene consentito (con la direttiva `LPC`) di utilizzare solo i comandi in essa elencati. Per quanto riguarda invece la capacità di cancellare un file (il servizio “M”) lo si permette solo a richieste provenienti dallo stesso utente e dalla stessa macchina, o in maniera generica all'amministratore. Si elimina poi possibilità di connettersi da qualunque altra macchina che non sia quella locale.

3.5.4 Il *Common Unix Printing System*

Un approccio completamente nuovo e diverso rispetto a quello di LPD è quello realizzato da CUPS (acronimo di *Common Unix Printing System*), che realizza tutta l'interfaccia al sistema di stampa attraverso un apposito servizio (in genere posto sulla porta 631) con cui si comunica via rete usando il protocollo HTTP. Rispetto ad un server HTTP però un server CUPS prevede una comunicazione con caratteristiche specifiche che sono state standardizzate nell'RFC 3239 in cui è definito il protocollo IPP, acronimo di *Internet Printing Protocol*.

L'uso di un protocollo di rete che è comunque basato su HTTP ha il grande vantaggio che con CUPS buona parte dell'amministrazione del sistema di stampa, come la creazione il controllo e la rimozione di stampe e code di stampa o la configurazione delle stampanti, può essere eseguita direttamente tramite un browser qualunque, che basterà puntare sulla porta 631 della macchina da amministrare, e questa resta la modalità principale, anche se è possibile usare degli opportuni comandi di shell per eseguire gli stessi compiti.

Il servizio è fornito da un demone, `cupsd`, che è controllato da un file di configurazione, `cupsd.conf`, che di norma viene installato in `/etc/cups`. Questo è il file configura il servizio nel suo insieme, come dicevamo infatti la gestione di stampanti, code e stampe avviene attraverso l'interfaccia che il servizio mette a disposizione. Il formato del file è simile a quello del file di configurazione di Apache;³⁸ la maggior parte sono una serie di direttive elementari nella forma:

NomeDirettiva valore

dove il nome della direttiva è composto da normali caratteri, e il valore viene specificato di seguito separato da uno spazio; al solito righe vuote o inizianti per un “#” vengono ignorate.

³⁸il server web più diffuso, che tratteremo in sez. ??.

Con la direttiva **Port** si può specificare la porta su cui il servizio si pone in ascolto; una delle caratteristiche più interessanti di CUPS però è la sua capacità di riconoscere automaticamente le stampanti presenti su una rete (il cosiddetto *browsing*). Questa funzionalità è attivata di default (ma può essere disattivata impostando ad **off** la direttiva **Browsing**) e può essere controllata con la direttiva **BrowseAddress** per indicare l'indirizzo di broadcast cui inviare le richieste di riconoscimento. Normalmente tutte le richieste vengono accettate, ma con le direttive **BrowseAllow** e **BrowseDeny** si possono indicare quali sono le macchine (o le reti) le cui richieste vengono accettate o rifiutate.

Direttiva	Significato
AccessLog	il file su cui vengono registrati gli accessi; richiede come valore il nome del file o la parola chiave syslog che invia le informazioni all'omonimo servizio.
BrowseAddress	indirizzo di broadcast cui inviare le richieste di riconoscimento.
BrowseAllow	indirizzo o rete da cui accettare le richieste di riconoscimento.
BrowseDeny	indirizzo o rete da cui non accettare le richieste di riconoscimento.
Browsing	abilita o disabilita il riconoscimento automatico delle stampanti presenti (prende i valori on o off).
DataDir	la directory dove sono mantenuti i dati di CUPS.
DefaultCharset	il set di caratteri di default.
DocumentRoot	la directory radice per i documenti HTTP (come la documentazione) che vengono messi a disposizione attraverso l'interfaccia.
ErrorLog	il file su cui vengono registrati gli errori; richiede come valore il nome del file o la parola chiave syslog che invia le informazioni all'omonimo servizio.
Group	gruppo per conto del quale viene eseguito cupsd (di default lpadmin).
Include	permette di includere un altro file nella configurazione.
LogLevel	livello dei messaggi da parte del demone.
PageLog	il file su cui vengono registrati i dati sulle pagine stampate; richiede come valore il nome del file o la parola chiave syslog che invia le informazioni all'omonimo servizio.
Port	specifica la porta su cui si pone in ascolto il demone cupsd (il default è 631).
Printcap	specifica su quale file riprodurre un file printcap che faccia da sostituto per /etc/printcap per i programmi che necessitano di quest'ultimo per riconoscere le code di stampa.
ServerRoot	specifica la directory radice per il demone cupsd , di default è /etc/cups .
TempDir	la directory in cui inserire i file temporanei, in sostanza lo <i>spool</i> di stampa (di default è /var/spool/cups/tmp).
User	utente per conto del quale viene eseguito cupsd (di default lp).
MaxCopies	numero massimo di copie che un utente può richiedere, il default è 0 e significa nessun limite.
MaxJobs	numero massimo di stampe in coda, il default è 0 e significa nessun limite.
MaxJobsPerUser	numero massimo di stampe per utente, il default è 0 e significa nessun limite.

Tabella 3.12: Principali direttive per il file **/etc/cups/cupsd.conf**.

In tab. 3.12 sono riportate le più importanti direttive elementari di **/etc/cups/cupsd.conf**, per l'elenco completo con una descrizione sommaria si può fare riferimento alla pagina di manuale; inoltre in genere le distribuzioni installano una versione predefinita di questo file in cui tutte le opzioni (anche quelle non attivate esplicitamente) sono abbondantemente commentate. Per una documentazione completa invece si può fare riferimento ai manuali on-line direttamente accessibili attraverso l'interfaccia del sistema (un'altro dei vantaggi di avere un protocollo basato su HTTP) o reperibili su <http://www.cups.org/>.

Oltre alle direttive appena illustrate, che usano la sintassi elementare in cui si esprime la corrispondenza di un valore ad una parola chiave, **cupsd.conf** prevede la presenza di direttive con una sintassi più complessa; in particolare queste sono quelle usate effettuare il controllo degli accessi all'interfaccia di rete fornita da **cupsd** con cui è possibile gestire il controllo del sistema di stampa attraverso il protocollo IPP. Queste direttive sono in genere specificate in coda al file, ed hanno una forma diversa, del tipo:

<Location /percorso>

```
DirettivaAccesso1 valore1
DirettivaAccesso2 valore2
...
</Location>
```

Una direttiva **Location** viene utilizzata per identificare quella che si chiama una *collocazione*; l'accesso alle varie funzionalità del sistema di stampa via IPP viene infatti gerarchizzato in una struttura ad albero analoga a quella dei file, ed espressa appunto con dei “percorsi” a partire da una radice “/” avendo accesso alla quale si diventa in grado di compiere qualunque operazione.

Percorso	Descrizione
/	Accesso a tutte le operazioni.
/admin	Accesso operazioni amministrative (creazione, cancellazione, impostazione e avvio delle stampanti e delle code).
/classes	Accesso alle operazioni sulle classi.
/classes/name	Accesso alle operazioni sulla classe name .
/jobs	Accesso alle operazioni sulle stampe.
/printers	Accesso alle operazioni sulle stampanti.
/printers/name	Accesso alle operazioni sulla stampante name .

Tabella 3.13: I percorsi per l'accesso alla gestione delle operazioni definibili all'interno di una *collocazione*.

Specificando un *percorso* a partire da detta radice si possono definire le proprietà di accesso per insiemi sempre più ristretti di operazioni. I principali *percorsi* utilizzabili in una *collocazione* sono illustrati in tab. 3.13; per l'elenco completo si può fare riferimento alla documentazione in linea di CUPS.

Una volta che con la scelta di un percorso nella direttiva **Location** si è definito a quale classe di operazioni si vuole che siano applicate le regole di accesso, queste ultime devono essere specificate all'interno della *collocazione* stessa; questo viene fatto attraverso una serie di ulteriori direttive, specifiche per l'indicazione di regole di accesso, che hanno una sintassi identica alle direttive elementari viste in precedenza, ma che sono utilizzabili solo all'interno di una *collocazione*.

Direttiva	Significato
Order	Definisce la modalità con cui vengono gestite le autorizzazioni con le direttive Deny e Allow , prende come valori: Allow,Deny in cui si accettano tutte le richieste eccetto quelle negate da un direttiva Deny e Deny,Allow in cui si accettano solo le richieste consentite con una direttiva Allow .
AuthClass	definisce il livello di autenticazione richiesto, prende come valori: Anonymous che disabilita l'autenticazione, User che richiede utente e password, System che inoltre richiede che l'utente appartenga al gruppo sys (o a quello specificato con la direttiva SystemGroup), Group che richiede che l'utente appartenga al gruppo definito dalla direttiva AuthGroupName .
AuthType	definisce le modalità di autenticazione, prende come valori: None che non effettua nessuna autenticazione, Basic che effettua l'autenticazione usando e credenziali standard della macchina, Digest e BasicDigest che utilizzano le credenziali specifiche memorizzate in <code>/etc/cups/passwd.md5</code> .
Allow	consente l'accesso, prende come valore un numero IP di una singola macchina o di una rete (in notazione CIDR, vedi sez. 7.2.2) o un nome a dominio, consentendo notazioni del tipo <code>*.dominio.it</code> o <code>.dominio.it</code> per indicare tutti le macchine in un dominio, o la parola chiave All .
Deny	nega l'accesso, usa gli stessi valori appena descritti per Allow .
Encryption	abilita la cifratura della connessione, prende come valori: Never , IfRequested , Required il cui significato è autoesplicativo.

Tabella 3.14: Le direttive di accesso usate in una *collocazione*.

Una *collocazione* prevede sempre la presenza di una direttiva **Order** (posta per chiarezza come prima direttiva al suo interno), che definisce la politica di accesso: se l'accesso è consentito

per default si utilizza il valore **Allow,Deny**, mentre se è vietato si utilizza il valore **Deny,Allow**. Alla direttiva **Order** devono poi seguire successive direttive **Allow** o **Deny** per indicare le macchine da cui vietare o consentire gli accessi, e direttive come **AuthType** e **AuthClass** per indicare le eventuali modalità di autenticazione. I valori utilizzabili per queste direttive sono riportati in tab. 3.14, in cui sono illustrate le principali direttive di accesso, per una descrizione più dettagliata di tutte le direttive e della relativa sintassi si può fare nuovamente riferimento alla documentazione di CUPS.

Un esempio del contenuto di `/etc/cups/cupsd.conf`, estratto (togliendo i commenti) dalla versione installata con il pacchetto `cupsys` su una Debian Sarge, è il seguente:

```
DefaultCharset notused
LogLevel info
Printcap /var/run/cups/printcap
Port 631
<Location />
    Order Deny,Allow
    Deny From All
    Allow From 127.0.0.1
</Location>
<Location /admin>
    AuthType Basic
    AuthClass System
    Order Deny,Allow
    Deny From All
    Allow From 127.0.0.1
</Location>
```

In questo esempio gran parte delle opzioni sono lasciate al valore di default, e ci si limita in sostanza a consentire l'accesso all'interfaccia via rete solo in locale, richiedendo per l'amministrazione un'autenticazione basata sul corrispondente utente sulla macchina su cui si sta operando.

Oltre alla configurazione lato server è possibile specificare una configurazione generale sul lato client (per tutti i programmi cioè che si appoggiano a CUPS per la stampa), che viene mantenuta a livello di sistema nel file `/etc/cups/client.conf`, e può essere personalizzata dal singolo utente in un file `.cupsrsrc` nella propria home directory.

In genere le direttive principali che si utilizzano in detti file sono **ServerName** che prende come valore il nome della macchina cui rivolgersi come server di stampa e **Encryption** che prende gli stessi valori della omonima vista in tab. 3.14 per il server, che indica la modalità con cui eseguire una eventuale cifratura della connessione.

Come accennato anche se la gran parte della gestione e della configurazione del sistema di stampa viene gestita attraverso l'interfaccia web fornita da CUPS è comunque possibile utilizzare anche una serie di comandi di shell; il primo di questi è `lpadmin`, che è quello che si può utilizzare per definire le proprietà generali delle stampanti presenti nel sistema. Il concetto di base è quello di definire delle *stampanti logiche*,³⁹ cui fare sempre riferimento con l'opzione `-p` seguita dal nome. In pratica tutte le invocazioni di `lpadmin` richiedono l'uso di questa opzione, tranne il caso in cui si usi al suo posto `-x` che richiede la cancellazione di una stampante logica.

Per creare una stampante logica oltre ad identificarla con `-p` occorrerà anche definire con quale dispositivo vi si accede, cosa che si fa attraverso l'opzione `-v`; questa prende come argomento quello che la documentazione chiama un *device-URI*, cioè una stringa nella forma di un indirizzo simile a quello di un sito web, che indica con quale modalità è possibile raggiungere la stampante stessa (se sulla porta parallela, seriale, USB, o via rete secondo vari protocolli di comunicazione), un breve elenco di possibili valori è riportato in tab. 3.15.

³⁹le si chiamano così in quanto si possono associare più *stampanti logiche* ad una stessa stampante fisica, una stampante logica ha una rozza equivalenza alla coda di stampa vista per LPD.

Nome	Descrizione
<code>ipp://remote.printer.ip</code>	stampante di rete raggiungibile con protocollo IPP.
<code>smb://remote.printer.ip</code>	stampante di rete windows raggiungibile tramite Samba.
<code>lpd://remote.printer.ip</code>	stampante di rete raggiungibile con protocollo LPD.
<code>parallel:/dev/lp0</code>	stampante locale sulla porta parallela.
<code>usb:/dev/lp0</code>	stampante locale su porta USB.
<code>serial:/dev/lp0</code>	stampante locale su porta seriale.

Tabella 3.15: Le diverse possibili specificazioni per gli indirizzi del dispositivo associato ad una stampante (*device-URI*).

Oltre a definire come si può parlare con la stampante è necessario anche specificare quale “linguaggio” usare, questo si fa con l’opzione `-P` che associa alla stampante logica il relativo file PPD (*PostScript Printer Description*) che ne descrive le capacità; questo è un po’ l’equivalente del filtro di stampa, in quanto serve a definire quali sono le modalità con cui alla fine i file vengono stampati.

Una delle caratteristiche più interessanti di CUPS è quella di permettere di definire delle *classi* di stampanti, in sostanza una classe definisce un gruppo di stampante che può essere trattato come un insieme, così che indicando di stampare su quella classe la stampa viene effettuata su una delle stampanti che ne fanno parte. La definizione delle classi viene fatta tramite le opzioni `-c` e `-r` di `lpadmin` che permettono rispettivamente di inserire o eliminare una stampante dalla classe passata come argomento; se una classe non esiste viene creata all’inserimento della prima stampante, se si elimina l’ultima stampante la classe viene cancellata.

Opzione	Descrizione
<code>-p stampante</code>	indica la stampante logica su cui si sta operando.
<code>-x stampante</code>	rimuove una stampante logica.
<code>-d stampante</code>	imposta la stampante come stampante di default.
<code>-c classe</code>	inserisce in una classe una stampante logica.
<code>-r classe</code>	rimuove una stampante logica da una classe.
<code>-v URI</code>	indica il dispositivo da usare per parlare con la stampante (secondo il formato illustrato in tab. 3.15).
<code>-P file.ppd</code>	definisce il file di descrizione della stampante da utilizzare.
<code>-E</code>	abilita una stampante logica rendendola disponibile per la stampa.
<code>-h host</code>	Indica la macchina remota su far operare il comando.

Tabella 3.16: Principali opzioni del comando `lpadmin`.

Le principali opzioni del comando sono riportate in tab. 3.16; al solito si può fare riferimento alla pagina di manuale o alla documentazione di CUPS per un elenco completo e per la descrizione dettagliata.

Una volta definita una stampante logica questa può essere abilitata o disabilitata alla stampa rispettivamente con i comandi `enable` e `disable` che prendono come argomento il nome della stessa; il comando `reject` inoltre supporta l’opzione `-c` che permette di rimuovere tutte le stampe pendenti, e `-r` che prende come argomento una stringa in cui le motivazioni per cui la stampante è stata disabilitata (che sarà riportata dai comandi diagnostici).

Si tenga presente comunque che abilitare o disabilitare una stampante logica significa sostanzialmente abilitare o disabilitare l’esecuzione delle stampe su di essa, e che questa è una operazione diversa rispetto a quella di accettare o rifiutare l’immissione di nuove stampe sulla relativa coda; questa seconda operazione viene fatta invece con i comandi `accept` e `reject`, che di nuovo richiedono come argomento il nome della stampante; come per `disable` anche `reject` supporta l’opzione `-r` con lo stesso significato.

Il comando per richiedere una stampa è invece `lp` (anche se il sistema supporta l’uso di `lpr`

con la stessa sintassi vista in sez. 3.5.2), che prende come argomento il file da stampare, e lo invia sulla stampante di default riportando a video un identificatore associato alla stampa in maniera analoga a `lpr`.

In questo caso per indicare la stampante da usare si usa l'opzione `-d` che prende come parametro il nome della stampante o della classe, mentre se si vuole stampare su una macchina remota specifica occorre specificarla come parametro per l'opzione `-h`. Altre due opzioni utili sono `-n` che permette di specificare il numero di copie da stampare e `-q` che permette di impostare una priorità (un numero da 1 a 100, con valore di default pari a 50). Le opzioni principali sono riportate in tab. 3.17, per l'elenco completo ed i dettagli si può fare riferimento alla pagina di manuale.

Opzione	Descrizione
<code>-E</code>	forza una comunicazione cifrata col server.
<code>-d</code>	specifica la stampante o la classe da usare.
<code>-h</code>	specifica la macchina remota cui inviare la stampa.
<code>-i</code>	specifica la stampa su cui operare.
<code>-n</code>	specifica il numero di copie da stampare.
<code>-q</code>	specifica una priorità.
<code>-H</code>	specifica quando eseguire una stampa: hold la blocca indefinitamente, un tempo in formato HH:MM la blocca ad allora, resume la avvia immediatamente e restart riesegue una stampa completata (gli ultimi due valori richiedono l'uso di <code>-i</code> per identificare la stampa cui si fa riferimento).
<code>-P</code>	specifica quali pagine stampare (passati come lista separate da virgole o intervalli separati da "-").

Tabella 3.17: Principali opzioni del comando `lp`.

Per cancellare una stampa si può invece usare il comando `cancel`, che prende come argomento l'identificatore della stampa, o se specificato, con l'opzione `-a` una stampante o una classe, nel qual caso saranno cancellate tutte le stampe presenti su di essa. Il comando supporta anche l'opzione `-h` con lo stesso significato di `lp`; al solito per i dettagli si faccia riferimento alla pagina di manuale.

Infine per ottenere informazioni riguardo il sistema di stampa ci sono due comandi, il primo è `lpinfo` che permette di vedere i dispositivi fisici ed i supporti per le stampanti disponibili all'interno di CUPS; il comando richiede una delle opzioni `-v` o `-m` per stampare rispettivamente la lista dei dispositivi utilizzabili e quella delle stampanti supportate. Ad esempio se si vuole ottenere una lista dei dispositivi visibili da riutilizzare con l'opzione `-v` di `lpadmin` si può eseguire il comando:

```
monk:~# lpinfo -v
network http
network ipp
network lpd
direct parallel:/dev/lp0
direct usb:/dev/usb/lp0
...
direct usb:/dev/usb/lp15
network smb
```

Il secondo comando è `lpstat` che riporta le informazioni relative alle singole stampanti, classi e stampe attive. Il comando usa le opzioni `-E` e `-h` con lo stesso significato visto in tab. 3.17. L'opzione `-c` richiede la stampa delle informazioni relative ad una classe (o a tutte se non si specifica quale classe), l'opzione `-a` riporta lo stato di una coda (e delle eventuali stampe in attesa), mentre l'opzione `-p` riporta lo stato di una stampante, entrambe richiedono il nome di

una stampante logica da interrogare, e riportano lo stato di tutte quelle definite se questa non viene specificata. Ulteriori dettagli sul comando e l'elenco completo delle opzioni si trovano al solito sulla pagina di manuale

Capitolo 4

Amministrazione ordinaria del sistema

4.1 Archiviazione e backup

Una delle attività fondamentali della amministrazione di un sistema è quella dei backup. In questa sezione prenderemo in esame i programmi per la gestione degli archivi di dati, per la gestione dei backup e la duplicazione dei dati dei dispositivi. Tratteremo le modalità con cui si possono eseguire i backup sia di file e directory che di interi filesystem.

4.1.1 Criteri generali per il backup

Quella dei backup è probabilmente la principale attività di ogni sistemista. Nel mondo dell'informatica i dati sono il principale prodotto del lavoro svolto, e la perdita degli stessi significa spesso la perdita di grandi quantità di ore di lavoro. Per questo motivo occorre prendere le opportune misure per assicurarsi contro questa evenienza, ed il backup è la principale di queste misure.

Ci possono essere varie ragioni per cui si possono perdere dei dati, in generale si tende a classificarle in quattro categorie diverse: disastri naturali, attività umane, errori del software e guasti dell'hardware; ciascuna di queste comporta rischi specifici ed opportune contromisure.

Oggi l'hardware tende ad essere sufficientemente affidabile, ciò non di meno i guasti continuano ad esserci, e la sfortuna è in agguato per farli avvenire nel peggior momento possibile. In genere in questo caso i dispositivi più critici sono gli hard disk, che per quanto ben costruiti hanno una loro fragilità meccanica e sono comunque dispositivi basati sulla lettura di campi magnetici di bassissima intensità, per cui il rischio di malfunzionamento, per quanto ridotto, non è trascurabile. Nel caso dei dischi una possibile assicurazione contro il malfunzionamento è quello dell'uso del RAID (vedi sez. 6.1), ma questo copre soltanto un caso particolare di guasti dell'hardware.

Al contrario dell'hardware l'affidabilità del software non sembra subire particolari miglioramenti, ed un programma solido, affidabile e sicuro è sempre un caso piuttosto raro, ed inoltre l'affidabilità di un singolo programma è relativamente inutile se un altro va a sovrascrivere per errore i suoi dati. In questo la struttura di un sistema Unix, con la separazione dei privilegi, e l'uso di utenti distinti può essere di aiuto, ma di nuovo non è la soluzione al problema di eventuali perdite di dati.

L'affidabilità delle persone è sempre una variabile difficile da quantificare, sia in termini delle competenze che riducano la eventualità di errori nelle operazioni (che però possono sempre accadere anche al più esperto degli amministratori) sia nell'affidabilità personale diretta in caso

di eventuali azioni dolose che possono causare, direttamente o indirettamente, la distruzione di dati o il danneggiamento dei relativi supporti.

Per quanto riguarda i disastri naturali, ed altre eventualità remote completamente distruttive di ogni tipo, occorre sempre valutare quanto la distruzione completa dei propri dati all'interno di un evento catastrofico possa incidere in termini di perdite complessive e determinare se vale la pena di prevedere delle contromisure specifiche per tali evenienze (che spesso comportano un discreto aggravio di spese sia in termini di attrezzature che di gestione).

Come accennato, mantenere dei backup è la strategia più semplice ed efficace per proteggersi dalle molteplici evenienze che possono causare la perdita di dati; avere a disposizione copie multiple dei propri dati rende meno critica la perdita di una di esse, il costo della perdita si riduce semplicemente a quello del ripristino degli stessi da backup, e non della ricreazione da zero degli stessi.

Perché questa strategia sia efficace comunque occorre che i backup siano effettuati in maniera appropriata; come ogni altra cosa del mondo reale infatti essi possono fallire e al solito tenderanno a farlo nel peggior modo e momento possibili, per cui potreste trovarvi senza disco e scoprire che un nastro è illeggibile, o che vi siete dimenticati di inserire nel backup alcune informazioni essenziali, o che l'unica unità a nastro in grado di rileggere i vostri backup si è rotta pure lei.

Per questo motivo nell'effettuare dei backup il primo punto è quello di mettere a punto una opportuna strategia. Ci sono infatti parecchie variabili da tenere sotto controllo, come la quantità di dati che si vogliono mettere sotto controllo, la frequenza dei backup, il periodo che si intende coprire con il backup, il mezzo utilizzato, le modalità di gestione degli stessi.

In genere un primo passo è quello di definire cosa si intende inserire nel backup: ad esempio non è quasi mai strettamente indispensabile effettuare un backup dei programmi di sistema, se non nel caso in cui una reinstallazione da zero non comporti un carico di lavoro eccessivo, mentre può essere importante salvare delle configurazioni ottenute come risultato finale di parecchie ore di lavoro. Sono senz'altro da salvare tutti i dati di sistema, gli archivi della posta e di un eventuale database e del web, e quant'altro non è replicabile in maniera automatica, insieme ai dati personali dei singoli utenti. Questo ad esempio vuol dire, nell'ambito di un sistema GNU/Linux, che si devono salvare i contenuti delle directory `/etc`, `/var` e `/home`, mentre non è necessario salvare quelli di `/usr`.

Una volta determinata la dimensione totale dei dati che si vogliono archiviare, occorre stabilire anche il periodo di tempo che si intende coprire con i backup, e la frequenza con cui si effettuano questi ultimi. Se infatti si avesse un sistema di backup contenente solo le informazioni necessarie a ripristinare la situazione ad un certo istante, non si potrebbero recuperare eventuali dati cancellati in precedenza. Nella pratica invece il caso più comune di recupero di dati da un backup è proprio quello di dati cancellati inavvertitamente, che, se assenti alla data dell'ultimo backup disponibile, sarebbero irrimediabilmente perduti.

Questo comporta che deve essere determinata la frequenza con cui si vogliono effettuare i backup, in modo da poter essere in grado di ricostruire la situazione dei propri dati ad istanti diversi nel passato. La frequenza dei backup dipende ovviamente anche da quella con cui variano i dati: in genere le configurazioni di sistema o un indirizzario variano senz'altro molto meno della posta elettronica o dei dati di accesso a servizi web. Un altro fattore che incide sulla frequenza è il tempo che occorre a creare il backup, dato che se questo dovesse eccedere il periodo fra due backup successivi si avrebbe una situazione in cui un backup non si completa prima dell'inizio del successivo, e non avrebbe senso usare una frequenza maggiore.

Infine la scelta del periodo da coprire con i backup dipende dall'uso degli stessi, ad esempio se servono solo a mantenere una copia di riserva che permetta il recupero di eventuali cancellazioni accidentali passate inosservate, o se si vuole mantenere una storia completa per ragioni di archiviazione. Ovviamente più lungo sarà il periodo, più risorse dovranno essere utilizzate nella manutenzione dei backup.

Per limitare l'occupazione di risorse ed il tempo di esecuzione dei backup viene in aiuto la possibilità di effettuare i cosiddetti *backup incrementali*, dei backup cioè in cui si archiviano solo i file che sono stati modificati rispetto ad un backup precedente. In questo modo è possibile ricostruire la situazione ad un dato istante senza dover registrare tutte le volte l'insieme completo dei dati. Ai vantaggi in risparmio di tempo di esecuzione e di utilizzo delle risorse corrispondono però degli svantaggi: anzitutto si complica la procedura di recupero (che necessiterà di un numero di passaggi crescente con il numero di backup incrementali coinvolti), ma soprattutto il fallimento di un backup incrementale renderà inutilizzabili tutti quelli successivi basati su di esso, per questo qualunque strategia di backup prevede comunque l'effettuazione periodica di *backup completi*.

Un'altra decisione cruciale nella strategia di backup è la scelta del supporto da utilizzare per gli stessi. Le variabili in gioco sono il costo, l'affidabilità, la velocità e l'utilizzabilità. Il costo incide direttamente sulla quantità di dati che si possono salvare, per cui è opportuno avere un mezzo poco costoso. L'affidabilità invece è fondamentale, dato che un backup inaffidabile è assolutamente inutile, ma l'affidabilità dipende anche dal tipo di problema che ci si trova ad affrontare, in genere i dischi sono molto affidabili, ma non sono adatti come mezzi di backup in quanto stanno nello stesso computer dei dati originali che può andare distrutto, danneggiato o manomesso.¹ La velocità è senz'altro utile, ma diventa cruciale solo qualora un mezzo troppo lento rendesse impossibile effettuare i backup con la frequenza necessaria. L'utilizzabilità viene spesso data per scontata, ma occorre invece riflettere su quanto possa essere pericoloso avere uno splendido sistema di backup perfettamente funzionante, che funziona però solo con una vecchia scheda hardware fuori produzione.

Tenendo conto di tutte queste caratteristiche alla fine i supporti più utilizzati per i backup sono usualmente i nastri. I CD-R infatti, benché funzionali e poco costosi, hanno dei grossi problemi di stabilità nel lungo periodo, i floppy hanno capacità ridotta e scarsa affidabilità, mentre i vari tipi di dischi rimovibili presentano in genere un costo troppo elevato con capacità insufficienti. In genere inoltre si utilizzano unità a nastro SCSI, dato che questa interfaccia ha dimostrato di mantenersi utilizzabile nel tempo, anche in presenza di nuove tecnologia per le schede.

Oltre alla scelta dei supporti una strategia di backup deve comunque anche prevedere la definizione delle modalità di conservazione degli stessi. Ad esempio può non essere troppo saggio tenere i propri supporti in un armadio nel corridoio, sotto una vecchia tubatura del riscaldamento. Inoltre qualunque sia la modalità di stoccaggio utilizzata occorre anche prevedere verifiche periodiche del funzionamento dei backup (effettuando dei ripristini di controllo), dato che accorgersi che qualcosa è andato storto nel momento del bisogno non è molto simpatico.²

A titolo di esempio una strategia comunemente utilizzata è quella di effettuare backup incrementali a frequenza giornaliera, intervallati da backup completi a cadenza settimanale o mensile, archiviando i backup mensili per un periodo di tempo più o meno lungo a secondo delle esigenze "storiche" che si possono avere.

4.1.2 Il comando tar

Il comando più usato per la gestione di archivi di file è **tar**, chiamato così dall'inglese **Tape ARchive** dato che il comando è nato apposta per archiviare i file su nastro magnetico. Al di là del suo uso per la creazione di archivi, **tar** è anche il programma più diffuso per la distribuzione

¹ovviamente a questo fanno eccezione i dischi estraibili, ma questo comporta altri problemi, fra cui quello del costo che, dovendosi utilizzare dischi estraibili a caldo, è solitamente molto alto.

²è rimasto famoso l'esempio di un sistemista coscenziioso che aveva accuratamente pianificato tutta la sua strategia, prevedendo anche lo stoccaggio dei nastri in un locale sicuro in una locazione remota, che si accorse la prima volta che dovette recuperare dei dati che i suoi nastri erano stati rovinati dal campo magnetico generato dai fili del riscaldamento del sedile della macchina su cui appoggiava i nastri quando li portava al sito di stoccaggio.

di pacchetti di software dai sorgenti (come vedremo in sez. 4.2.1). La caratteristica del comando è che è in grado di salvare (e ripristinare) il contenuto e la struttura di una certa directory e di tutto il suo contenuto, mantenendo invariate anche le caratteristiche specifiche dei file (come permessi, proprietari, tempi, ecc.).

Il comando `tar` supporta una grande varietà di opzioni ed argomenti, e la versione GNU è stata dotata di una serie di ulteriori estensioni. In generale però si possono distinguere le opzioni del comando in due classi principali: quelle che indicano delle *operazioni* e quelle che specificano dei *parametri*. In ogni invocazione si può specificare non più di una operazione, mentre si può avere un numero qualunque di parametri.

Il comando ha tre operazioni principali, `-c` per creare un archivio, `-t` per verificarne il contenuto, e `-x` per estrarlo, ma le operazioni sono in totale otto, e sono illustrate in tab. 4.1 (sia in forma breve che estesa). Si è usata la sintassi delle opzioni in forma GNU, ma `tar` è uno dei comandi storici di Unix, e ne esistono molte versioni. Per questo anche la versione GNU (la sola che tratteremo) supporta varie sintassi, ed in particolare si può sempre fare a meno di utilizzare il `-` per specificare una opzione in forma breve.

Opzione		Significato
<code>-c</code>	<code>--create</code>	crea un nuovo archivio.
<code>-x</code>	<code>--extract</code>	estrae i file da un archivio.
<code>-t</code>	<code>--list</code>	elenca i file contenuti in un archivio.
<code>-A</code>	<code>--concatenate</code>	aggiunge un archivio in coda ad un altro.
<code>-d</code>	<code>--diff</code>	confronta i membri di un archivio con la controparte sul filesystem e riporta ogni differenza in dimensioni, permessi, proprietari e tempi.
<code>-u</code>	<code>--update</code>	aggiunge file in coda all'archivio, ma solo se sono più recenti della versione già presente nell'archivio o non sono presenti in esso.
<code>-r</code>	<code>--append</code>	aggiunge file in coda all'archivio.
	<code>--delete</code>	cancella i file da un archivio (questo non può funzionare sui nastri).
	<code>--compare</code>	identica a <code>--diff</code> .
	<code>--get</code>	identica a <code>--extract</code> .

Tabella 4.1: Operazioni del comando `tar`.

Oltre alle operazioni `tar` supporta una enorme quantità di opzioni che modificano il comportamento del programma. Di queste ne esistono due di uso comune; la prima è `-v` (o `--verbose`), che aumenta la *prolissità* del comando, facendogli stampare a video durante l'esecuzione molte più informazioni (ad esempio la lista dei file inseriti o estratti dall'archivio). È in genere possibile aumentare ulteriormente la quantità di informazioni stampate inserendo la stessa opzione una seconda volta.

La seconda è `-f` (`--file`) che serve a specificare il nome del file da usare come archivio da cui leggere o su cui scrivere; in generale, a meno di non specificare il file con `-f` il comando usa come default `/dev/st0` o il primo dispositivo che riconosce come una unità a nastri, per evitare confusione è pertanto opportuno specificare sempre esplicitamente il nome dell'archivio con questa opzione.

Come accennato per creare un archivio il comando necessita dell'opzione `--create` o `-c`; in questo caso dovrà anche essere specificato con `-f` il nome del file su cui saranno archiviati i file, di norma si utilizza anche l'estensione `.tar` per identificare questi file. Date queste due opzioni il comando prende come argomenti una lista dei singoli file da archiviare, se però fra questi si inserisce una directory il comando archiverà anche il contenuto di quest'ultima, e di tutte le directory sottostanti. Ad esempio si possono archiviare alcuni file con:

```
piccardi@anarres:~/Truelite/documentazione/corso$ tar -cvf dispense.tar *.tex
Struttura.tex
advadmin.tex
config.tex
corso.tex
fdl.tex
netadmin.tex
netbase.tex
netinter.tex
ordadmin.tex
ringraziamenti.tex
shell.tex
struttura.tex
```

e si noti come si sia usata anche l'opzione `-v` per stampare la lista dei file archiviati; allo stesso modo si potrà archiviare tutta una directory con:

```
piccardi@anarres:~/Truelite/documentazione$ tar -cvf dispense.tar corso
corso/
corso/CVS/
corso/CVS/Root
corso/CVS/Repository
corso/CVS/Entries
corso/images/
corso/images/CVS/
corso/images/CVS/Root
corso/images/CVS/Repository
corso/images/CVS/Entries
corso/images/dir_links.eps
corso/images/dir_links.dia
...
corso/ordadmin.tex.~1.5.~
corso/netbase.tex
corso/netinter.tex
corso/dispense.tar
corso/#ordadmin.tex#
```

e si noti come in questo caso vengano archiviate ricorsivamente anche tutte le directory sottostanti ed il relativo contenuto.

Si tenga presente che quando si crea un archivio con `-c` il file di destinazione viene sovrascritto, se si vuole invece aggiungere dei file ad un archivio preesistente occorre usare l'opzione `-r`. Inoltre, a meno di non specificare un pathname assoluto, il file viene creato nella directory corrente, questo significa anche che se quest'ultima è inclusa nell'archivio `tar` si troverà ad affrontare il problema di come archiviare l'archivio stesso; il comando è sufficientemente scaltro da accorgersi di questa situazione ed evitare di inserire nell'archivio che sta creando il file dell'archivio stesso:

```
piccardi@anarres:~/Truelite/documentazione/corso$ tar -cf dispense.tar .
tar: ./dispense.tar: file is the archive; not dumped
```

Una volta creato un archivio se ne potrà verificare il contenuto con `-t` (o `--list`); di nuovo si dovrà utilizzare `-f` per indicare il file contenente l'archivio che in questo caso sarà letto. Se non si passano altri argomenti il risultato sarà la stampa a video della lista di tutti i file contenuti nell'archivio, anche senza la presenza dell'opzione `-v`, specificando la quale si otterrà invece una lista in formato esteso, analogo a quello di `ls -l`. Si può verificare la presenza di un singolo file o di una lista passando come argomento il nome (o i nomi), si tenga presente però che in

questo caso deve essere specificato il nome completo del file così come è stato archiviato (quello stampato dall'opzione `-v` in fase di creazione), il confronto infatti viene eseguito solo nei termini della corrispondenza fra l'argomento ed il nome così questo viene visto nell'archiviazione.

Per estrarre i dati inseriti in un archivio si deve utilizzare l'opzione `--extract` o `-x`, di nuovo è necessario specificare l'archivio da cui si estraggono i file con `-f`. Come per la creazione l'uso dell'opzione `-v` permette di ottenere la lista dei file che vengono estratti dall'archivio. L'estrazione avviene sempre nella directory corrente, e qualora si siano archiviate intere directory ne viene ricreata l'intera gerarchia. Se non si specifica nessun argomento il comando estrae il contenuto completo dell'archivio, ma è possibile estrarre un singolo file fornendone il nome (o il pathname se questo è collocato in una sottodirectory), che si può ottenere dalla lista fornita dall'opzione `-t`.

Si tenga presente infine che nell'estrazione il comando ricrea i file con tutte le caratteristiche che essi avevano nel sistema su cui sono stati creati; questo significa anche che vengono mantenuti i permessi e utente e gruppo proprietario del file, utente e gruppo che nel sistema corrente possono non esistere o essere del tutto diversi.³

Di questo può essere necessario dover tenere conto quando si estraggono degli archivi creati da altri (come quelli dei pacchetti sorgenti di cui parleremo in sez. 4.2). Inoltre dato che alcuni filesystem (come il `vfat` di Windows) non supportano la presenza di permessi, utenti e gruppi, il programma può lamentarsi dell'impossibilità di ricreare questi attributi dei file; un altro errore che viene segnalato, in caso di impostazione sbagliata dell'ora di sistema, è quando i file che vengono creati hanno dei tempi situati nel futuro.

4.1.3 Il comando `cpio`

Un comando alternativo per la gestione dell'archiviazione di file e directory è `cpio`, che permette di estrarre ed inserire file in un archivio, per il quale è anche in grado di usare lo stesso formato usato da `tar`. A differenza di `tar` però prende sempre come argomento di ingresso una lista di file.

Il comando `cpio` ha tre modalità operative, che devono essere sempre indicate ad ogni invocazione del programma. In modalità *copy-out* il comando copia i file dentro l'archivio; per attivare questa modalità deve essere specificata l'opzione `-o` o `--create`. Il programma legge una lista di file dallo standard input e scrive l'archivio risultante sullo standard output. Pertanto una invocazione tipica è qualcosa del tipo:

```
cpio < lista_file > archivio
```

è uso comune generare la lista direttamente con l'uso di `find`.

In modalità *copy-in* il comando estrae i file contenuti in un archivio o ne elenca il contenuto; per attivare questa modalità deve essere specificata l'opzione `-i` o `--extract`. L'archivio viene letto dallo standard input, se non si specifica nessuna opzione tutti i file vengono estratti, altrimenti tutti gli argomenti che non sono opzioni vengono considerati come *pattern* secondo la sintassi⁴ del *filename globbing* della shell (vedi sez. 2.1.4) e vengono estratti solo i file che corrispondono. Pertanto una invocazione tipica in questa modalità è del tipo:

```
cpio < archivio
```

³utente e gruppo vengono mantenuti soltanto se l'archivio viene estratto dall'amministratore, gli utenti normali infatti non possono creare file con proprietario diverso da se stessi e con gruppo diverso da quelli a cui appartengono.

⁴con due eccezioni, vengono considerati validi come caratteri sui quali verificare una corrispondenza anche un "." all'inizio del file e il carattere "/", così da considerare nella corrispondenza anche i file nascosti e intere sezioni di pathname.

In modalità *pass-through* il comando copia dei file da una directory ad un'altra, combinando in sostanza le due modalità *copy-in* e *copy-out* in un unico passo senza creare un archivio; per attivare questa modalità deve essere specificata l'opzione **-p** o **--pass-through**. Il comando legge la lista dei file da copiare dallo standard input e li copia nella directory che deve essere specificata come argomento non opzionale. Pertanto una invocazione tipica in questa modalità è del tipo:

```
cpio < lista directory
```

Il comando, a parte quelle usate per specificarne la modalità di operazione, supporta molte altre opzioni. In particolare con **-v** si richiede una maggiore prolissità nelle operazioni, con la stampa dei nomi dei file che vengono processati; con **-t** si richiede di stampare la lista del contenuto di un archivio, con **-A** di aggiungere i file ad un archivio esistente (invece di sovrascriverlo).

Opzione	Significato
-0	prende in ingresso una lista di stringhe terminate dal carattere NUL (generabili con find) per permettere l'archiviazione di file il cui nome contiene un carattere di a capo.
-a	ripristina il tempo di ultimo accesso sui file appena letti perché non risultino tali.
-d	ricrea le directory in cui si trovano i file (altrimenti questi verrebbero creati nella directory corrente).
-f	copia solo i file che non corrispondono al pattern specificato.
-F	permette di specificare un nome di file da usare come archivio al posto dello standard input o dello standard output.
-m	preserva i precedenti tempi di ultima modifica quando ricrea i file in fase di estrazione.
-t	stampa la lista dei file contenuti nell'archivio.
-u	sovrascrive eventuali file già esistenti con quelli dell'archivio senza chiedere conferma.
-v	stampa informazioni aggiuntive sui file che vengono processati dal comando (in combinazione con -t viene usato un formato analogo a quello di ls -l).

Tabella 4.2: Principali opzioni del comando **cpio**.

L'elenco delle principali opzioni è riportato in tab. 4.2, in cui si sono tralasciate le tre opzioni principali per selezionare le modalità di operazione; al solito l'elenco completo delle opzioni è disponibile nella pagina di manuale accessibile con **man cpio**.

4.1.4 I comandi **dump** e **restore**

I comandi **tar** e **cpio** appena illustrati, vengono usati per archiviare i file presenti all'interno delle directory specificate, ma esistono comandi, come **dump** e **restore** che permettono di effettuare l'archiviazione a livello di un intero filesystem.

Il limite di un programma come **dump** è che questo funziona solo per un filesystem che ne supporti le operazioni salvando le informazioni necessarie. Questo nel caso di Linux è vero solo per il filesystem *ext2* e la sua estensione *journalled*, *ext3*. Inoltre per poter utilizzare questa funzionalità occorre anche montare opportunamente il filesystem, e come accennato in sez. 1.2.4, per questo il file **/etc/fstab** prevede un campo apposito, il quinto, che deve essere inizializzato opportunamente ad un valore non nullo.

Una volta abilitato il supporto per il *dump* del filesystem tramite **fstab** effettuando operazioni di backup dello stesso tramite **dump** saranno salvate le informazioni necessarie a ricostruire

quali file sono stati cambiati da un backup all'altro, così da permettere dei backup incrementali. Il funzionamento del programma prevede la possibilità di 10 diversi livelli di *dump*, numerati da 0 a 9, da specificare con le opzioni omonime (cioè -0, -1, ... -9), e se non specificato viene usato il livello 9.

Il livello 0 indica un backup completo, in cui si sono salvati tutti i file, e stabilisce un punto di partenza per tutti gli altri livelli. Un livello superiore richiede il salvataggio di tutti i file che sono cambiati dall'ultimo backup effettuato con un livello inferiore. Questo significa che un backup di livello 9 salverà sempre solo i file modificati dopo l'ultimo backup precedente.

Quindi per fare un esempio,⁵ se si esegue un backup con livello 2 seguito da uno di livello 4 si avranno in quest'ultimo solo i file cambiati dal momento in cui si è eseguito il precedente backup di livello due; se poi si eseguisse un dump di livello 3 sarebbero salvati di nuovo tutti i file modificati dopo il backup di livello 2 (e il livello 4 sarebbe ignorato), se al posto del dump di livello 3 se ne fosse fatto uno di livello 5 invece si sarebbero salvati solo i file modificati dopo il backup di livello 4.

Questo permette di poter eseguire i backup incrementali in maniera efficiente per ridurre al minimo sia il tempo di recupero che il numero di supporti utilizzati. Una possibile strategia di backup è quella riportata nella pagina di manuale, in cui si fanno backup incrementali giornalieri, usando un insieme di supporti da ruotare settimanalmente, usando un algoritmo di tipo *torre di Hanoi* modificato.

A parte l'opzione che ne specifica il livello il comando richiede la presenza dell'opzione -f seguita da un parametro che indica il file su cui effettuare il backup (in genere si usano unità a nastri, e cioè /dev/st0, ma si può specificare un file ordinario o usare - per indicare lo standard output), e prende come argomento il *mount point* del filesystem di cui si vuole eseguire il backup.⁶ Oltre quelle citate **dump** prende numerose altre opzioni, molte delle quali sono relative alla gestione dei nastri su cui usualmente vengono salvati i backup. Si sono riportate le principali in tab. 4.3, per l'elenco completo si può al solito fare riferimento alla pagina di manuale.

Opzione	Significato
-D	permette di cambiare il file su cui viene mantenuta l'informazione relativa ai precedenti backup (completi o incrementali) che di solito è /var/lib/dumpdates.
-e	esclude dal backup una lista di <i>inode</i> (vedi sez. 1.2.1) passati per numero.
-E	esclude una lista di <i>inode</i> elencati nel file specificato come parametro.
-f	scrive il backup sul file passato come parametro, si possono specificare più file separandoli con virgole, e saranno considerati come volumi successivi.
-L	usa la stringa passata come parametro come etichetta del backup.
-M	abilita il salvataggio multivolume (in cui si usa -f per specificare più di un volume di backup).
-u	aggiorna il file /var/lib/dumpdates che contiene la tabella in cui sono scritte i filesystem per i quali e le date in cui sono stati effettuati i backup, e a quale livello.
-v	aumenta la <i>prolissità</i> del comando facendogli stampare a video più informazioni riguardo le operazioni correnti.

Tabella 4.3: Principali opzioni del comando **dump**.

Il comando che permette di recuperare i dati archiviati con **dump** è **restore**, che esegue

⁵si suppone che prima si sia partiti con un dump di livello 0, un passaggio iniziale infatti è sempre necessario.

⁶in realtà si può anche specificare una directory contenente i file che si vogliono archiviare e non un *mount point*, in questo caso però sono consentiti solo backup completi di livello 0 e tutto il contenuto della directory deve risiedere nello stesso filesystem.

esattamente il compito inverso, e come `dump` usa l'opzione `-f` per indicare il file (ordinario o di dispositivo) da cui recuperare i dati da ripristinare. Il comando inoltre prevede la necessaria presenza di una serie di opzioni che ne indicano la modalità di operazione. Le principali sono `-C` che effettua un confronto fra i file passati come argomento e quelli sul backup, `-i` che porta in modalità interattiva (dove possono essere dati una serie di comandi, descritti nella pagina di manuale), `-r` che permette di ricostruire un intero filesystem da zero, `-t` che verifica se i file passati come argomento sono nel backup o stampa a video il contenuto integrale dello stesso se invocato senza argomenti, e `-x` che effettua il ripristino dal contenuto dell'archivio: di una directory, se questa viene passata come argomento, o di tutto l'archivio se non vengono passati argomenti.

Opzione	Significato
<code>-f</code>	indica il file in cui è stato memorizzato il backup.
<code>-C</code>	esegue un confronto fra i file presenti nel filesystem e quelli nel backup.
<code>-i</code>	attiva la modalità interattiva.
<code>-r</code>	ricostruisce un intero filesystem da zero.
<code>-t</code>	presenta un elenco di file nel backup.
<code>-x</code>	estrae una directory dal backup (o tutto il contenuto).
<code>-h</code>	estrae il contenuto di una directory, e non di quelle in essa contenute.
<code>-M</code>	abilita il ripristino da un archivio multivolume.
<code>-v</code>	stampa una maggiore quantità di informazione durante le operazioni.

Tabella 4.4: Principali opzioni del comando `restore`.

Come `dump` anche `restore` prevede numerose opzioni, parecchie delle quali servono a controllare proprietà relative all'uso dei nastri. Oltre a quelle brevemente spiegate in precedenza si sono riportate in tab. 4.4 le più rilevanti; al solito tutti i dettagli e l'elenco completo delle opzioni (e dei comandi disponibili in modalità interattiva) sono riportati nella pagina di manuale.

4.2 La gestione dei pacchetti software

Affronteremo in questa sezione le varie modalità in cui un amministratore può installare e rimuovere software dal sistema, in particolare esaminando le funzionalità di gestione automatizzata dei pacchetti che permettono di tenere traccia di tutto quello che si è installato nel sistema.

4.2.1 L'installazione diretta

Uno dei grandi vantaggi del software libero è che avendo a disposizione i sorgenti dei programmi che si usano è sempre possibile effettuare una *installazione diretta* dei pacchetti software che ci servono a partire da questi ultimi.

In genere il software viene distribuito in forma sorgente in degli archivi compressi chiamati genericamente “*tarball*” in quanto creati con il programma `tar`, trattato in sez. 4.1.2. In genere gli archivi vengono pure compressi, per cui di norma li si trovano distribuiti in file che usano l'estensione `.tar.gz`⁷ dato che il programma di compressione più usato è `gzip`; una alternativa che si inizia a diffondere per gli archivi più grandi è quella dell'uso di `bzip2` (che comprime molto di più, anche se è più lento), nel qual caso l'estensione usata è `.tar.bz2`.

Per installare un pacchetto dai sorgenti la prima cosa da fare è scaricarsi l'archivio degli stessi dal sito di sviluppo: è consigliato usare, se ci sono, eventuali *mirror*, dato che questi di

⁷talvolta abbreviata in `tgz`, per l'uso di sistemi obsoleti che hanno problemi coi nomi di file che hanno troppi caratteri “.” al loro interno.

norma sono meno *occupati* del sito originale, e probabilmente anche più “vicini” a voi e quindi con una maggiore velocità di accesso. Una volta scaricato il pacchetto se ne può verificare il contenuto o scompattarlo usando il comando `tar` usando la sintassi trattata in sez. 4.1.2. Pertanto se il nostro pacchetto software è `pacchetto.tar.gz` lo si potrà decomprimere con `tar -xvzf pacchetto.tar.gz`; in genere questo crea nella directory corrente una sottodirectory (quella che conteneva i file sulla macchina in cui è stato creato l’archivio) con il contenuto dell’archivio, che di solito ha lo stesso nome del pacchetto.

A questo punto per l’installazione occorre eseguire le relative operazioni. Queste possono essere molto diverse da pacchetto a pacchetto; in genere chiunque distribuisce software fornisce anche le relative informazioni per l’installazione che si trovano insieme ai sorgenti del pacchetto nei file `README` o `INSTALL`. Se il pacchetto usa gli *autotool* GNU⁸ o segue la procedura di installazione standard tutto quello che c’è da fare è di entrare nella directory dove si è scompattato il contenuto della *tarball* ed eseguire la sequenza di comandi:

```
./configure
make
make install
```

Il primo comando (`./configure`) esegue uno script di shell che verifica che nel sistema sia presente tutto quello che serve per compilare il pacchetto. Questo è il punto più critico della procedura, in quanto se manca qualcosa lo script si interromperà dicendo che non esiste il supporto per quella funzionalità o mancano i file di dichiarazione relativi ad una certa libreria che serve al programma (o la libreria stessa).

Uno dei problemi più comuni è che, pur avendo installato una libreria, non si sono installati i file di dichiarazione che sono usati dal compilatore per poter accedere, quando compila il pacchetto, alle funzioni della stessa. In genere infatti, quando si installano pacchetti binari già compilati, questi non sono necessari, e pertanto nella installazione standard di una distribuzione normalmente vengono tralasciati. Di norma questi sono disponibili nella vostra distribuzione come pacchetti con lo stesso nome di quello contenente le librerie corrispondenti, estesi con un `-dev` per indicare che sono necessari per lo sviluppo.⁹ In questo caso non che c’è altro da fare che installare anche questi pacchetti con il meccanismo di gestione fornito dalla vostra distribuzione, secondo le modalità che vedremo più avanti.

Una volta che `./configure` ha completato con successo le sue operazioni potremo eseguire il secondo comando nella sequenza, `make`. Questo è un programma per la *costruzione* di altri programmi, che usa meccanismo molto sofisticato che permette di creare i file nella giusta sequenza. Il funzionamento di `make` va al di là di quanto sia possibile affrontare qui, basti dire quello che si otterrà dal comando è una lunga serie di linee di uscita da parte del compilatore, fino a quando tutte le operazioni saranno completate ed i binari del pacchetto saranno stati creati. Se qualcosa va storto in questa procedura avete due strade, scrivere un *bug report* a chi ha creato il pacchetto (evitando troppi accidenti e fornendogli le righe in cui si è verificato l’errore) o provare a correggere l’errore voi stessi (ma dovete essere pratici di programmazione, nel caso probabilmente non starete a leggere questo manuale).

A questo punto con `make install` si dice allo stesso programma di eseguire le istruzioni di installazione. Si tenga presente che la procedura standard prevede di installare i pacchetti compilati dai sorgenti sotto `/usr/local/`, per cui per poterli utilizzare occorre avere le directory di questa gerarchia secondaria nel `PATH` della shell (vedi sez. 2.1.4) e si deve essere in grado di usare le eventuali librerie installate dal pacchetto (secondo quanto illustrato in sez. 3.1.2).

⁸un insieme di programmi che consente di usare delle procedure automatizzate per l’installazione controllando che sulla macchina sia presente tutto quello che serve per creare il pacchetto.

⁹servono infatti non solo per compilare un programma per l’installazione, ma anche qualora si volessero sviluppare programmi che utilizzano le corrispondenti librerie.

In questo modo è possibile installare pacchetti generici, il problema di tutto ciò, oltre al tempo perso a compilare i programmi (che per pacchetti piccoli è forse trascurabile, ma per pacchetti importanti come il server X può essere, a seconda della potenza della macchina, dell'ordine delle ore o dei giorni), è che dovete ricordarvi di cosa avete installato, dove e quando, e che se installate una nuova versione dovete verificare che la sovrapposizione non generi problemi.

Inoltre se installare è facile, è più problematico disinstallare. Alcuni pacchetti (una minoranza purtroppo) forniscono uno speciale *bersaglio*¹⁰ per **make** che consente la disinstallazione con **make unistall**. Ma se questo non c'è occorre tracciarsi a mano i file che sono stati installati e cancellarli. Per questo, come vedremo nelle sezioni seguenti, gran parte delle distribuzioni utilizzano un sistema di gestione dei pacchetti che permette di tenere traccia di cosa si installa, di dove sono messi i file, per poi permettere una cancellazione pulita del pacchetto quando lo si vuole disinstallare.

4.2.2 La gestione dei pacchetti con rpm

Uno dei primi sistemi evoluti per la gestione dell'installazione e rimozione di pacchetti software è RPM, realizzato da RedHat per la sua distribuzione.¹¹ Data la sua efficacia esso è stato adottato da molte altre distribuzioni, divenendo uno dei sistemi di gestione dei pacchetti più diffusi.

Con RPM si intende sia il programma **rpm**, che permette di eseguire l'installazione e la rimozione dei pacchetti, che il formato¹² dei file con cui sono distribuiti i pacchetti, normalmente caratterizzato dalla estensione **.rpm**. In realtà il comando **rpm** fa molto di più che installare e disinstallare: mantiene la lista dei pacchetti installati, e di dove sono stati installati i singoli file che ne fanno parte; così può accorgersi se un pacchetto va in conflitto con altri cercando di installare gli stessi file. Inoltre è in grado di eseguire degli script in fase di installazione e disinstallazione che possono effettuare delle procedure di configurazione automatica del pacchetto stesso.

Il comando è complesso e prende molteplici opzioni, come molteplici sono le funzionalità che fornisce. Le due opzioni più semplici sono **-i** che installa un pacchetto e **-e** che lo rimuove. Nel primo caso deve essere specificato il file **.rpm** che contiene il pacchetto, nel secondo caso il nome del pacchetto stesso. Se il pacchetto è già installato si può usare **-U** che esegue l'*upgrade*.

Inoltre con l'opzione **-q** è possibile eseguire una serie di interrogazioni sul database dei pacchetti installati; se non si specifica altro l'opzione prende un nome di pacchetto di cui verifica l'esistenza, mentre con **rpm -qa** si ottengono tutti i pacchetti installati nel sistema. Se invece si usa **-qf**, seguito dal nome del pacchetto, si stampano i file in essi contenuti. Si può anche leggere il contenuto di un pacchetto non installato con l'opzione **-qpf** specificando il file dello stesso. Il comando prende molte altre opzioni, le principali delle quali sono riportate in tab. 4.5. La descrizione dettagliata del comando con l'elenco completo delle opzioni sono al solito disponibili nella relativa pagina di manuale.

Una delle funzionalità più importanti di un gestore di pacchetti è quella di essere in grado gestire le cosiddette *dipendenze* di un pacchetto da un altro, deve cioè essere in grado di accorgersi se per installare un certo pacchetto è necessaria la presenza di un altro (ad esempio perché si possa usare un programma che usa l'interfaccia grafica dovrà prima essere installata quest'ultima).

¹⁰si chiamano così, dall'inglese *target*, gli argomenti che si passano a **make** e che normalmente indicano quali delle varie sezioni di comandi di costruzione e installazione devono essere usati; non specificare nulla usa il primo dei *target* disponibili, che esegue di norma il compito principale, di solito ne esistono anche degli altri, uno comune ad esempio è **clean** che serve a cancellare i risultati della compilazione per riportare la directory del pacchetto nelle condizioni iniziali precedenti alla costruzione dello stesso.

¹¹la sigla originariamente stava per *RedHat Package Manager*, essendo utilizzato da altre distribuzioni oggi viene chiamato *RPM Package Manager*.

¹²in sostanza si tratta di archivi **cpio**.

Opzione	Significato
-i	Installa il pacchetto.
-r	Rimuove il pacchetto.
-l	Interroga il database per la presenza di un pacchetto.
-L	Stampa i file contenuti in un pacchetto.
-S	Ricerca i pacchetti che contengono un file.
-s	Stampa lo <i>stato</i> di un pacchetto.
-p	Stampa informazioni su un pacchetto installato.
-I	Stampa informazioni su un <i>.deb</i> .
-c	Stampa il contenuto di un <i>.deb</i> .

Tabella 4.6: Principali opzioni del comando `dpkg`.

così possibile richiedere l'installazione automatica non solo di un singolo pacchetto, ma anche di tutti quelli da cui questo dipende. Inoltre in genere i pacchetti vengono distribuiti direttamente via rete, con una modalità che permette il download automatizzato degli stessi.

Il programma di gestione principale per i pacchetti non è allora `dpkg`, ma `apt-get`, che serve appunto da front-end per tutto il sistema dell'*Advanced Package Tool*. In una distribuzione Debian tutto quello che si deve fare è mantenere una lista degli appropriati *repository* dei pacchetti nel file `/etc/apt/sources.list`. Un *repository* non è altro che una directory (locale o remota) che contiene i vari pacchetti e le relative informazioni organizzati in maniera opportuna. Basterà poter accedere al repository (cosa che può essere fatta in una molteplicità di modi, i principali dei quali sono via HTTP o via FTP), per ottenere automaticamente tutto quello che serve.

Una volta impostati i vari repository da cui si vogliono recuperare i pacchetti basterà eseguire il comando `apt-get update` per scaricare la lista aggiornata dei pacchetti. A questo punto sarà possibile installare un pacchetto con il comando `apt-get install nome`, il programma si incaricherà di effettuare automaticamente il download dello stesso, e di quelli necessari per soddisfare eventuali dipendenze, e di installare il tutto.

Inoltre in Debian è stato pure creato un sistema generico per la auto-configurazione automatica dei pacchetti (chiamato *debconf*) che permette, una volta scaricati i pacchetti, di richiedere all'utente tutte le informazioni necessarie per la configurazione degli stessi. Così oltre alla installazione viene anche eseguita la configurazione di base con la creazione dei relativi file, che molto spesso è tutto quello che c'è da fare.

Qualora si voglia rimuovere un pacchetto il comando è `apt-get remove nome`, ed in questo caso, se altri pacchetti dipendono da quello che si vuole rimuovere, il comando chiede conferma della volontà di rimuovere anche loro, ed in caso di conferma procede alla rimozione completa. Una delle caratteristiche di Debian è che la rimozione di un pacchetto non comporta normalmente la rimozione dei relativi file di configurazione, che potrebbero tornare utili in una successiva reinstallazione. Se si vuole rimuovere definitivamente anche questi occorre usare l'opzione `-purge`.

Infine il sistema consente una estrema facilità di aggiornamento del sistema, basta infatti usare il comando `apt-get upgrade` dopo aver usato `apt-get update` per ottenere l'installazione automatica delle eventuali nuove versioni presenti sul repository di tutti i pacchetti che sono installati nel sistema. Il comando però non rimuove mai un pacchetto già presente dal sistema, anche quando questo può essere stato sostituito da un altro. Per risolvere questo tipo di situazione (che si incontra di solito quando si passa da una versione di Debian ad un'altra) si può usare il comando `apt-get dist-upgrade` che esegue una risoluzione intelligente dei conflitti ed è in grado di effettuare l'upgrade di pacchetti importanti a scapito di quelli secondari, che possono essere disinstallati.

Le opzioni principali di `apt-get` sono riportate in tab. 4.7, l'elenco completo, insieme alla descrizione dettagliata di tutte le caratteristiche del comando, è al solito disponibile nella relativa pagina di manuale.

Opzione	Significato
<code>install</code>	Installa un pacchetto.
<code>remove</code>	Rimuove un pacchetto.
<code>clean</code>	Cancella l'archivio dei pacchetti scaricati.
<code>update</code>	Scarica la lista aggiornata dei pacchetti.
<code>upgrade</code>	Esegue l'upgrade dei pacchetti aggiornati.
<code>dist-upgrade</code>	Esegue l'upgrade della distribuzione.
<code>autoclean</code>	Cancella dall'archivio i pacchetti con vecchie versioni.

Tabella 4.7: Principali opzioni del comando `apt-get`.

4.3 La gestione di utenti e gruppi

Tratteremo in questa sezione la gestione degli utenti e dei gruppi presenti nel sistema: vedremo i comandi utilizzati per crearli, eliminarli, e modificarne le proprietà ed esamineremo quali sono le modalità con cui vengono mantenute all'interno del sistema le informazioni ad essi relative.

4.3.1 Una visione generale

Dato che GNU/Linux è un sistema multiutente abbiamo finora dato per scontato la presenza di utenti che potessero utilizzarlo. In realtà la questione non è affatto così immediata. Il kernel infatti supporta la presenza di utenti e gruppi associando a questi degli identificatori numerici che poi vengono usati nella gestione dei processi e dei file, ma come abbiamo abbondantemente ripetuto in sez. 1.1 tutta la gestione del sistema, compresa quella che permette il riconoscimento e l'accesso agli utenti, viene realizzata da appositi programmi.

Questo vale in particolare anche, come accennato in sez. 1.1.2, per quella procedura di collegamento al sistema che porta ad avere dei processi che vengono eseguiti dal kernel per conto di un certo utente. Perché questa sia effettuata però occorrono le *opportune informazioni* che permettano ai vari programmi che gestiscono la procedura (come `login`) di sapere quali utenti ci sono, come riconoscerli e quali risorse assegnargli.

Questo ci porta ad una delle caratteristiche fondamentali di un qualunque sistema multiutente: il concetto di *account*. Come avviene per una banca, per poter utilizzare il sistema si deve avere un “conto” presso di lui, che ci conceda l'accesso e l'uso delle risorse. Ovviamente perché questo accada, non solo occorrerà, come accennavamo in sez. 1.4.1, che un utente si identifichi appropriatamente, ma dovranno anche essere identificate le risorse che gli si mettono a disposizione.

Una delle parti più critiche dell'amministrazione del sistema è allora quella che permette di creare e mantenere le informazioni relative agli *account* degli utenti presenti, che dovranno contenere sia le informazioni necessarie all'identificazione degli stessi che quelle relative alle risorse messe loro a disposizione.

Tradizionalmente in un sistema unix-like l'autenticazione degli utenti viene fatta utilizzando un *username*, che è il nome che il sistema associa all'utente, ed una *password* segreta la cui conoscenza serve a dimostrare l'autenticità della propria identità. Come accennato in sez. 1.4.1 il sistema prevede anche la presenza di gruppi di utenti, che a loro volta vengono identificati¹⁴ da altri nomi (i *groupname*). Inoltre, come visto in sez. 1.2.2, ogni utente ha a disposizione una home directory per i propri file, e quando si collega al sistema gli viene messa a disposizione una shell (la *shell* di *login*, cui abbiamo accennato a pag. 72). Tutte queste informazioni sono quelle che vengono mantenute ed utilizzate dai programmi per la gestione di utenti e gruppi.

Infine si tenga presente che utenti e gruppi vengono utilizzati dal sistema non solo per gestire gli *account* delle persone fisiche che utilizzano il sistema, ma anche per associare una *identità* ad

¹⁴si ricordi che il kernel conosce solo degli identificativi numerici, i nomi degli utenti e dei gruppi sono informazioni disponibili solo in user-space, proprio per la presenza di un sistema di gestione degli stessi.

alcuni servizi: ad esempio molti programmi server vengono eseguiti per conto di un utente a loro riservato; questo permette di mantenere separati i loro file rispetto a quelli di altri servizi e di non dare loro privilegi amministrativi,¹⁵ aumentando la sicurezza del sistema.

Si ricordi infatti che dal punto di vista del kernel esistono solo gli identificatori numerici di utenti e gruppi illustrati in sez. 1.4.1; che uno di questi sia associato ad un utente fisico, o semplicemente utilizzato da un programma di servizio per il kernel è del tutto indifferente,¹⁶ e sta ai programmi di gestione degli utenti stabilire una politica che allochi opportunamente gli identificatori.

Nelle prime versioni di Unix tutte le informazioni relative agli utenti ed ai gruppi presenti nel sistema erano memorizzate su due file, `/etc/passwd` e `/etc/group`, che tratteremo meglio in sez. 4.3.3. Dato che questi, come tutti i file di configurazione del sistema, sono file di testo, in teoria non ci sarebbe nessuna necessità di programmi specifici per la loro gestione, dato che possono essere modificati a mano con un qualunque editor.¹⁷

Nei sistemi moderni però il meccanismo di gestione di utenti e gruppi è stato completamente modularizzato attraverso sia l'uso del *Name Service Switch*, visto in sez. 3.1.3, che di PAM (*Pluggable Authentication Method*), su cui torneremo in sez. 4.3.4. In questo modo è possibile mantenere le informazioni ed effettuare i relativi controlli usando i supporti più disparati (server NIS, vari database, server LDAP, ecc.), e quindi in generale non è più possibile andare ad effettuare le modifiche a mano con un editor.

L'uso di un supporto modulare però fa sì che in molti casi si possano utilizzare buona parte degli stessi comandi che in origine operavano solo sui file di testo in maniera trasparente rispetto al meccanismo con cui sono effettivamente gestite le informazioni.

4.3.2 I comandi per la gestione di utenti e gruppi

Il primo comando di gestione che prendiamo in esame è `useradd`, che permette di aggiungere un nuovo utente al sistema (cioè di creare un *account*). Il comando prende come argomento il nuovo username. Si tenga presente che di default il comando si limita a creare il nuovo utente, ma non imposta la password (che resta disabilitata), non crea la home directory e non imposta una shell di login.

È possibile comunque impostare ciascuna di queste proprietà (e molte altre) attraverso le opportune opzioni: ad esempio `-p` permette di specificare la password dell'utente,¹⁸ `-s` la shell, `-g` il gruppo iniziale, `-G` eventuali altri gruppi di appartenenza, `-m` richiede la creazione della home directory, con tanto di creazione di tutti i file contenuti nella directory `/etc/skel` (vedi sez. 3.2.2).

Infine con `-u` si può impostare un valore specifico per lo user ID, altrimenti il comando assegnerà all'utente un valore predefinito corrispondente al primo numero maggiore di 999¹⁹ e più grande di tutti gli altri valori utilizzati per gli altri utenti.

Le principali opzioni sono riportate in tab. 4.8, l'elenco completo che comprende anche quelle più sofisticate, legate all'uso delle *shadow password* che permettono di impostare alcune proprietà delle password (come durata, lunghezza minima, ecc.) insieme a tutti i dettagli

¹⁵nella maggior parte dei casi questi, quando servono, sono garantiti solo all'avvio del servizio, ed eliminati non appena non sono più necessari.

¹⁶l'unica differenza che il kernel riconosce fra i vari utenti è quella fra l'amministratore e gli altri.

¹⁷cosa che in certi casi è comunque utile saper fare, ad esempio per togliere una password di amministratore dal sistema contenuto in un disco montato a mano usando un sistema di recupero, poiché in quel caso i comandi andrebbero ad operare sulla configurazione del sistema di recupero, e non di quello che si vuole riparare.

¹⁸che deve essere specificata in forma cifrata, per cui di norma non si usa mai questa opzione, ma si provvede ad eseguire il comando `passwd` in un secondo tempo.

¹⁹questa è la scelta di debian, altre distribuzioni si possono fermare a valori inferiori (RedHat ad esempio si fermava a 499); i valori fra 0 e 999 sono usati normalmente per gli utenti corrispondenti ad alcuni servizi di sistema.

Opzione	Significato
-b	imposta la home directory.
-d	imposta la home directory dell'utente.
-u	specifica un valore numerico per l'user ID.
-p	imposta la password.
-s	imposta la shell di default.
-m	copia il contenuto di <code>/etc/skel</code> nella home.
-g	imposta il gruppo di iniziale.
-G	imposta eventuali gruppi aggiuntivi.
-o	permette di specificare un user ID già esistente.

Tabella 4.8: Principali opzioni del comando `useradd`.

sul funzionamento del comando sono disponibili nella pagina di manuale accessibile con `man useradd`.

Oltre che a creare un nuovo utente il comando può essere anche usato per modificare le proprietà di un utente già esistente, nel qual caso deve essere invocato con l'opzione `-D`. Per questo però è disponibile anche il comando `usermod`, che è del tutto analogo all'uso di `useradd` con l'opzione `-D`, e prende le opzioni elencate in tab. 4.8 per `useradd`, ma prima di operare si assicura che l'utente che si va a modificare non sia collegato alla macchina. Inoltre `usermod` supporta le opzioni `-L` e `-U` usate rispettivamente per bloccare e sbloccare l'accesso di un utente inserendo un carattere `!` nel campo che contiene la sua password criptata (su questo torneremo in sez. 4.3.3).

Analogo ad `useradd` è `groupadd` che permette di creare un nuovo gruppo. In questo caso le uniche opzioni sono `-g` che permette di specificare un group ID specifico (il valore di default è impostato con gli stessi criteri visti per l'user ID) e `-o` che unito al precedente permette di specificare un group ID già in uso.

L'omologo di `usermod` per i gruppi è invece il comando `groupmod` che permette di modificare un gruppo, in particolare il comando permette di cambiare il group ID usando `-g`, supportando sempre l'opzione `-o` per poter specificare un gruppo già assegnato, e di cambiare il nome del gruppo con l'opzione `-n`.

I comandi `userdel` e `groupdel` permettono invece di cancellare rispettivamente un utente e un gruppo, da specificare come argomento. Nel caso di `groupdel` il comando non ha nessuna opzione specifica, mentre `userdel` si limita a cancellare l'utente, se si vuole anche rimuoverne la home directory²⁰ si dovrà usare l'opzione `-r` che è l'unica opzione supportata dal comando.

Per ciascuno di questi comandi per la creazione e rimozione di utenti e gruppi Debian mette a disposizione una interfaccia più amichevole attraverso gli analoghi comandi `adduser` ed `addgroup` e `deluser` ed `delgroup`. I primi permettono una impostazione automatica di tutti gli attributi di un nuovo utente, secondo quanto specificato nel file di configurazione `/etc/adduser.conf`, richiedendo quando serve le informazioni necessarie come la password ed il nome reale. I secondi eseguono la rimozione dell'utente e del gruppo²¹ seguendo lo schema specificato nel file di configurazione `/etc/deluser.conf`. Le informazioni complete sono disponibili al solito nelle relative pagine di manuale.

Oltre ai precedenti comandi generali, esistono una serie di comandi diretti che permettono di modificare i singoli attributi, il più comune è `passwd` che permette di cambiare la password. L'utilizzo più comune è quello di invocare il comando per cambiare la propria password, nel qual caso non è necessario nessun argomento; il comando chiederà la password corrente e poi la nuova password per due volte (la seconda per conferma, onde evitare errori di battitura). Se si specifica un argomento questo indica l'utente di cui si vuole cambiare la password, ma si

²⁰si tenga presente che questo non assicura la cancellazione di tutti i file di proprietà dell'utente che potrebbero essere in altre directory diverse dalla sua home.

²¹gestendo anche, rispetto ai corrispondenti programmi base, l'eliminazione dei file non contenuti nella home.

tenga presente che solo l'amministratore può cambiare la password di un altro utente, gli utenti normali possono cambiare solo la propria e solo dopo essersi autenticati con la vecchia.

Oltre alla semplice operazione di cambiamento della password il comando supporta molte altre funzionalità di gestione delle stesse, in particolare per la gestione delle caratteristiche delle *shadow password* (su cui torneremo in sez. 4.3.3), dette opzioni con il relativo significato sono riportate in tab. 4.9, e possono essere utilizzate solo dall'amministratore.

Opzione	Significato
-x	imposta in numero massimo di giorni per cui la password rimane valida, passati i quali l'utente sarà forzato a cambiarla.
-n	imposta il numero minimo di giorni dopo il quale una password può essere cambiata, l'utente non potrà modificarla prima che siano passati.
-w	imposta il numero di giorno per i quali l'utente viene avvertito prima della scadenza della password.
-i	imposta il numero massimo di giorni per cui viene accettato il login dopo che la password è scaduta, passato i quali l'account sarà disabilitato.
-e	fa scadere immediatamente una password, forzando così l'utente a cambiarla al login successivo.

Tabella 4.9: Opzioni del comando `passwd` per la gestione delle informazioni relative alle *shadow password*.

Oltre alle funzioni di gestione delle *shadow password* il comando permette anche di bloccare e sbloccare l'uso di un account rispettivamente con le opzioni `-l` e `-u`, inoltre l'opzione `-d` consente di cancellare la password lasciandola vuota, in tal caso però chiunque può entrare nel sistema conoscendo l'username.

Infine, benché sia una funzionalità poco nota, il comando permette anche, usando l'opzione `-g`, di cambiare (o mettere, dato che di norma non viene impostata) la password per un gruppo. Se si specifica anche `-r` la password presente sul gruppo viene rimossa. Solo l'amministratore o l'amministratore del gruppo possono eseguire questo comando.

Quest'ultimo è uno degli utenti del gruppo cui è stato dato il compito di amministrare lo stesso, e oltre a poterne cambiare la password è in grado di aggiungere altri utenti al gruppo usando il comando `gpasswd`. Questo, oltre alla capacità di cambiare la password di un gruppo, permette all'amministratore di aggiungere utenti a un gruppo con l'opzione `-M`, o definire degli utenti amministratori del gruppo con l'opzione `-A`. Questi ultimi potranno con le opzioni `-a` e `-d` aggiungere o rimuovere altri utenti dal gruppo e disabilitare l'accesso al gruppo da parte di esterni con `-R`.

Inserire una password su un gruppo significa consentire ad altri utenti che non sono nel gruppo di farne parte attraverso l'uso del comando `newgrp`. Questo comando permette infatti di cambiare il proprio group ID assumendo quello di un gruppo, ma solo quando è stata impostata una password per il gruppo, che deve essere fornita (altrimenti l'operazione non è consentita). Se invece si fa già parte del gruppo non è necessaria nessuna password.

Oltre a `newgrp` si può cambiare gruppo anche con il comando `sg` così come il comando `su` permette di assumere l'identità di un altro utente. Entrambi i comandi vogliono come argomento il nome del (gruppo o dell'utente) del quale si vogliono avere i diritti e richiedono la relativa password. Se usati con l'opzione `-c` si può specificare un comando da eseguire con i diritti del gruppo o dell'utente richiesto. Al solito la pagina di manuale riporta la documentazione completa.

Gli altri comandi per la gestione delle proprietà degli utenti sono `chsh`, che permette ad un utente di cambiare la sua shell di login di (ma solo fra quelle elencate in `/etc/shells`, vedi sez. 3.2.2) e `chfn` che permette di cambiare le informazioni mantenute nel campo chiamato *Gecos*

(vedi sez. 4.3.3), in cui si scrivono il nome reale e altri dati relativi all'utente. Al solito si faccia riferimento alle relative pagine di manuale per la descrizione completa.

4.3.3 Il database di utenti e gruppi

Come accennato in sez. 4.3.2 nelle prime versioni di Unix tutte le informazioni relative ad utenti e gruppi venivano tenute in due soli file. Benché questo schema si sia evoluto con l'introduzione prima delle *shadow password* e poi di PAM e del *Name Service Switch*, a tutt'oggi la modalità più comune per mantenere il database degli utenti su un sistema GNU/Linux resta quella di scrivere le relative informazioni su alcuni file di testo.

Il primo file su cui sono mantenute queste informazioni è `/etc/passwd` chiamato così perché è al suo interno che nelle prime versioni di Unix venivano memorizzate le password. Il file deve essere leggibile da tutti perché oltre alle password memorizza anche la corrispondenza fra l'username ed il relativo identificativo numerico, è da qui che normalmente i programmi come `ls` o `ps` ottengono l'username che corrisponde ad un certo user ID. Nel file sono mantenute poi anche altre informazioni come la shell di login, la home directory, ecc.

Il fatto che il file sia leggibile da tutti può far sorgere il dubbio di come si possa avere una cosa del genere per un file in cui sono memorizzate delle password. In realtà queste non sono memorizzate in chiaro (il testo della password non viene mai scritto da nessuna parte nel sistema), quello che viene memorizzato è solo il risultato della cifratura della password, quello che si chiama un *hash* crittografico.

Tutte le volte che si richiede una autenticazione quello che il sistema fa è semplicemente ricalcolare questo valore per la password che gli si fornisce e verificare che coincida con quello memorizzato; dato che solo se si è fornita la password originaria si potrà riottenere lo stesso risultato, in caso di coincidenza si è ottenuta l'autenticazione. Siccome poi dal punto di vista matematico è praticamente impossibile (ogni metodo è equivalente a quello provare tutte le possibilità) risalire dal valore cifrato alla password originale, il rendere leggibile quest'ultimo non costituiva un problema.

Benché per la gestione normale degli utenti ci siano gli opportuni programmi di shell (che abbiamo visto in sez. 4.3.2) abbiamo accennato come in casi di emergenza può essere utile modificare a mano il file; per questo occorre sapere come vengono mantenute le informazioni. 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 (o *username*).
- password cifrata (opzionale).
- identificatore numerico dell'utente (user ID).
- identificatore numerico del gruppo di default (group ID).
- nome e cognome dell'utente ed eventuali altri campi di commento separati da virgole; questo campo è detto anche *Gecos*.
- home directory dell'utente (pathname assoluto).
- shell di login.

ed un esempio di questo file può essere il seguente:

```
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
```

i dettagli del formato del file possono essere trovati nella relativa pagina di manuale accessibile con `man 5 passwd`.²²

La presenza di una `x` nel secondo campo del nostro esempio indica che sono attive le *shadow password*, su cui torneremo fra poco. 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 modificare 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.

Analogo a `/etc/passwd` per mantenere l'elenco dei gruppi c'è il file `/etc/group` che contiene le informazioni ad essi relative. In questo caso i campi sono soltanto quattro (sempre separati da `“:”`) ed indicano rispettivamente:

- il nome del gruppo (o *groupname*).
- la password del gruppo (si ricordi quanto visto in sez. 4.3.2).
- il valore numerico del gruppo (group ID).
- la lista degli username degli utenti appartenenti al gruppo, separati da virgole.

un esempio di questo file è il seguente:

```
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:
tty:x:5:piccardi,admin
...
piccardi:x:1000:
...
```

anche le informazioni mantenute da questo file sono usate da tutti i programmi che devono tradurre l'identificatore numerico del gruppo in un nome e pertanto deve essere mantenuto aperto in lettura.

Benché l'algoritmo crittografico con cui si calcolano le password sia piuttosto robusto,²³ mantenere leggibili le password, anche se cifrate, espone comunque ad un attacco a forza bruta (cioè alla possibilità che qualcuno tenti di provarle tutte). Se quando è stato creato Unix questa eventualità, dati i computer dell'epoca, era impraticabile, con la potenza dei computer di oggi lo è molto meno. Inoltre anche se oggi è diventato possibile usare altri algoritmi di crittografia che rendono più difficile un compito del genere, c'è sempre da fare i conti con la pigrizia degli utenti che tendono ad usare password come *pippo* o *ciccio* e simili.

Per cui anche se spesso gli attacchi a forza bruta non sono praticabili, è comunque piuttosto semplice (e ci sono un sacco di programmi molto efficienti nel farlo) utilizzare quello che si chiama un attacco a dizionario, in cui invece di tutte le combinazioni si provano solo quelle relative ad un dizionario di possibili password. E non giovano neanche trucchetti come quello di scrivere le parole alla rovescia, invertire delle lettere o mettere un `“3”` al posto di una `“e”` o un `“1”` al posto di una `“i”`. Tutti trucchetti ampiamente noti che qualunque programma decente di password cracking applicherà alle parole del suo dizionario.

²²si ricordi quanto detto in sez. 2.3.1 riguardo le sezioni delle pagine di manuale.

²³per i curiosi l'algoritmo originale si chiama DES, oggi è poco usato per la sua debolezza, il suo problema infatti è che prevede delle chiavi di dimensione ridotta ed ha dei limiti sul numero di caratteri delle password (vengono usati solo i primi 8 caratteri) per cui in genere si usano algoritmi alternativi come MD5 che prende password di dimensioni qualsiasi.

Per questo motivo non è comunque molto sicuro lasciare leggibili a tutti le password anche nella forma cifrata, dato che potrebbero essere soggette ad una analisi di questo tipo. Per questo motivo alcuni anni fa, nella reimplementazione dei meccanismi di autenticazione, venne introdotto quello che è stato chiamato il sistema delle *shadow password* che oltre a consentire di spostare le password in un file a parte ha pure aggiunto una serie di funzionalità di sicurezza ulteriori.

Se nel sistema sono state abilitate le *shadow password* (in genere lo si fa in fase di installazione, anche se ormai questa non è neanche più una opzione e vengono usate di default) nel secondo campo di `/etc/passwd` viene posta una “x” e le password cifrate vengono spostate in `/etc/shadow`.

In questo caso, dato che le altre informazioni necessarie ai programmi restano disponibili in `/etc/passwd` si può proteggere questo file in lettura così che solo l'amministratore possa accedervi. Oltre alle password sono memorizzate in `/etc/shadow` una serie di ulteriori informazioni che permettono un controllo molto più dettagliato su di esse, come date di scadenza, date in cui sono state cambiate ecc. Il formato del file è analogo a quelli già visti, ci sono 9 campi separati con dei “:”, il cui rispettivo contenuto è:

- nome di login (o *username*).
- password cifrata.
- giorno in cui è stata cambiata password l'ultima volta (espresso in numero di giorni dal 1/1/1970).
- numero di giorni che devono passare dall'ultimo cambiamento prima che la password possa essere cambiata nuovamente.
- numero di giorni dall'ultimo cambiamento dopo i quali la password scade e deve essere necessariamente cambiata.
- numero dei giorni precedenti quello di scadenza della password in cui gli utenti vengono avvisati.
- numero dei giorni successivi a quello di scadenza della password dopo i quali l'utente viene disabilitato.
- giorno in cui l'utente è stato disabilitato (espresso in numero di giorni dal 1/1/1970).
- campo riservato.

ed un esempio di questo file è:

```
root:n34M1zgKs8uTM:12290:0:99999:7:::
daemon*:11189:0:99999:7:::
bin*:11189:0:99999:7:::
sys*:11189:0:99999:7:::
sync*:11189:0:99999:7:::
games*:11189:0:99999:7:::
...
piccardi:$1$KSRp21Z3$s9/C2ms0Ke9UTaPpQ98cv1:11189:0:99999:7:::
...
```

al solito i dettagli sul significato dei campi si trovano nella pagina di manuale, accessibile con `man shadow`.

Si noti come per alcuni utenti la password sia sostituita dal carattere * (talvolta viene usato anche !), questo è un modo, dato che detti caratteri non corrispondono ad un valore possibile per una password cifrata, di impedire che il corrispondente utente possa eseguire un login, e lo si usa normalmente quando si vuole disabilitare l'accesso ad un utente o per gli utenti dei servizi di sistema che non necessitano di eseguire un login.

L'utilità del sistema delle *shadow password*, è che questo consente anche di impostare delle politiche di gestione delle stesse. Ad esempio si può impostare un tempo di scadenza forzando

gli utenti a non utilizzare sempre la stessa password, cosa che a lungo andare fa aumentare le possibilità che essa venga scoperta.

Infine dato che anche i gruppi hanno le loro password, anche queste sono state spostate in un altro file, `/etc/gshadow`. Il formato è sempre lo stesso, i campi in questo caso sono 4, ed indicano rispettivamente:

- nome del gruppo
- password cifrata
- amministratore del gruppo
- utenti appartenenti al gruppo

ed un esempio di questo file è il seguente:

```
root:::
daemon:::
bin:::
sys:::
adm:::
tty:::piccardi,admin
...
piccardi:x::
...
```

Tutte le distribuzioni recenti installano di default le *shadow password*, è comunque possibile effettuare una conversione automatica dal sistema tradizionale usando il comando `pwconv`. Questo prima rimuove da `/etc/shadow` tutte le voci non presenti in `/etc/passwd`, poi riporta le password presenti in esso all'interno di `/etc/shadow`, ed infine le sovrascrive con una `x`.

Inoltre il programma usa i valori di `PASS_MIN_DAYS`, `PASS_MAX_DAYS` e `PASS_WARN_AGE` presi da `login.defs` (vedi sez. 3.1.4) per impostare i rispettivi campi di `/etc/shadow`. Si tenga presente che `login.defs` non viene usato se si utilizza PAM e le relative informazioni sono prese direttamente dalla configurazione di quest'ultimo.

Analogo a `pwconv`, ma operante sui gruppi (e relativi file `/etc/group` e `/etc/gshadow`) è `grpconv`; essendo le password dei gruppi poco utilizzate anche questo comando non è molto usato. È possibile inoltre effettuare anche la conversione inversa, tornando dalle *shadow password* al sistema tradizionale con i comandi `pwunconv` e `grpunconv`, in questo caso le password verranno reinserite in `/etc/passwd` ed `/etc/shadow`, ma le informazioni aggiuntive (come i dati sulle scadenze delle password) verranno persi.

Si ricordi che, come accennato all'inizio, l'uso dei file appena descritti, pur restando ancora oggi quello più diffuso, è solo uno dei metodi per mantenere le informazioni riguardo gli *account* degli utenti. Per questo motivo esaminare il contenuto di questi file non è detto sia sufficiente ad identificare tutti gli utenti di un sistema, in quanto le informazioni potrebbero essere mantenute anche altrove.

Per questo motivo si può utilizzare un apposito comando, `getent`, che, quando le informazioni sono distribuite su più supporti, consente di interrogare il sistema dal *Name Service Switch* per richiedere una lista completa. Il comando richiede come argomento una delle classi di informazioni gestite dal *Name Service Switch* fra quelle di tab. 3.1, e stampa a video il risultato.

In questo modo si otterrà la stampa delle informazioni degli utenti nello stesso formato in cui sono mantenute nel corrispondente file tradizionale; ad esempio si avrà:

```
piccardi@monk:~/truedoc/corso$ getent passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
...
piccardi:x:1002:1002:Simone Piccardi,,,:/home/piccardi:/bin/bash
admin:x:1003:1003:Utente Amministrativo,,,:/home/admin:/bin/bash
```

4.3.4 Il *Pluggable Authentication Method*

La necessità di una maggiore flessibilità della gestione degli utenti non riguarda soltanto la possibilità di mantenere su diversi supporti i dati loro associati, ma anche, e soprattutto, quella di permettere la gestione di diverse modalità per gestire l'autenticazione. A questo provvede il sistema chiamato *Pluggable Authentication Method*, in breve PAM, introdotto inizialmente su Solaris e poi portato anche su GNU/Linux.

L'idea è quella di realizzare un meccanismo che ricopra per i programmi di autenticazione il ruolo che ha il *Name Service Switch* per quelli che richiedono una corrispondenza fra identificatori numerici e simbolici. Dato che in questo caso la richiesta è più sofisticata il sistema sarà più complesso, ma in sostanza si tratta sempre delle definizioni di una interfaccia di programmazione generica per cui tutti i programmi che necessitano di compiere una qualche forma di autenticazione (come `login` o `passwd`) lo fanno attraverso una libreria che fa da intermediario rispetto al meccanismo con cui poi le operazioni vengono effettivamente svolte, così che questo possa essere, in maniera totalmente trasparente ai programmi scelti, sostituito con un altro.

ipologia	Significato
account	fornisce il servizio di verifica della presenza di un <i>account</i> e della relativa validità (le password sono scadute, si può accedere al servizio, ecc.).
auth	fornisce il servizio di verifica dell'identità di un utente, in genere sulla base di un meccanismo di sfida/risposta (come dare la password), ma può essere esteso in maniera generica (permettendo l'uso anche di dispositivi hardware).
password	fornisce il servizio di aggiornamento dei meccanismi di autenticazione (come cambiare la password), ed è strettamente collegato ad auth .
session	fornisce i servizi di preparazione per le risorse che devono essere messe a disposizione dell'utente (ad esempio il montaggio della home directory) all'inizio di una sessione di lavoro e quelli relativi alla liberazione delle stesse alla fine della sessione.

Tabella 4.10: Le quattro tipologie di servizi forniti da PAM.

Nel gestire l'insieme dei metodi per l'autenticazione e l'accesso al sistema PAM suddivide il lavoro in quattro tipologie di servizio indipendenti fra loro, illustrate in tab. 4.10, dove si sono riportate anche le quattro parole chiave usate nella configurazioni per identificarle.

La flessibilità di PAM è dovuta al fatto che il sistema è implementato come una libreria dinamica che permette di caricare una serie di *moduli* (usualmente mantenuti in `/lib/security`) che realizzano diverse funzionalità in una o più delle classi di tab. 4.10. Una delle caratteristiche più interessanti di PAM è questi moduli possono essere *impilati* in modo da combinare le funzionalità di ciascuno nella fornitura di un determinato servizio. Inoltre il sistema permette di utilizzare impostazioni diverse per le differenti applicazioni che lo usano.

La configurazione di PAM può essere effettuata in due modi; quello più comune è l'uso di una serie di file (uno per ciascuna applicazione che usa il sistema) posti nella directory `/etc/pam.d`. In questo modo infatti è molto più semplice per le singole applicazioni mantenere un proprio file di configurazione.

Ogni applicazione che necessita dell'uso del sistema sarà controllata da un apposito file in `/etc/pam.d` il cui nome corrisponde a quello dell'applicazione stessa (si avranno cioè file come `login`, `su`, `passwd`, ecc.). Ciascun file contiene una regola per riga, usata per invocare un determinato modulo; si può proseguire una regola su più righe terminandole con il carattere "\", ed al solito righe vuote ed inizianti per "#" vengono ignorate. Si possono utilizzare più moduli invocandoli con altrettante regole nella sequenza in cui queste sono scritte all'interno del file.

Una regola è composta da almeno tre campi separati da spazi (o tabulatori), all'ultimo dei quali, che indica il modulo da utilizzare, possono seguire altri campi che specificano degli argomenti; la forma generica è pertanto:

```
type control module-path [arguments ...]
```

Il primo campo, **type**, indica (con le parole chiave di tab. 4.10), la classe di servizi a cui la regola fa riferimento. Il secondo campo, **control**, indica la modalità di comportamento che le applicazioni devono assumere in caso di successo o fallimento del compito assegnato al modulo, questi prevedono una sintassi semplificata, in cui si utilizza direttamente uno dei valori riportati tab. 4.11, o la sintassi completa illustrata nella pagina di manuale di PAM, accessibile con **man 7 pam**.

ipologia	Significato
requisite	il fallimento del modulo invocato comporta l'immediata terminazione della procedura di valutazione, gli altri eventuali moduli non verranno invocati.
required	il fallimento del modulo comporta il fallimento della procedura di autenticazione, ma gli altri moduli specificati nelle regole seguenti vengono comunque invocati.
sufficient	il successo del modulo comporta il successo dell'operazione, fintanto che nessun precedente modulo required è fallito.
optional	il successo o il fallimento del modulo non ha nessuna influenza sul risultato finale a meno che questo non sia l'unico modulo utilizzato.

Tabella 4.11: I principali valori per il campo di controllo dei risultati usati nei file di configurazione di PAM.

Il terzo campo indica il file (in genere una libreria condivisa, quindi un **.so**) con il codice del modulo, normalmente espresso con un pathname relativo alla directory **/lib/security/**;²⁴ al nome del modulo può seguire una lista (separata da spazi) degli argomenti da passare a quest'ultimo; in genere questi sono specifici del modulo, ma esistono alcuni argomenti speciali messi a disposizione dal sistema stesso, che si sono riportati in tab..

ipologia	Significato
use_first_pass	il modulo non deve richiedere una password all'utente ma usare quella già fornita in una precedente invocazione di un altro modulo; se questa non è valida l'utente non saranno fatte ulteriori richieste (l'argomento è valido solo per i servizi auth e passwd).
try_first_pass	il modulo deve usare la password già fornita in una precedente invocazione di un altro modulo, ed eseguire direttamente una richiesta qualora questa non sia valida (l'argomento è valido solo per i servizi auth e passwd).
expose_account	segnala al modulo che può essere più permissivo nel rivelare informazioni riguardo l'utente (in genere questo viene evitato per non fornire appigli ad un attaccante).
use_authtok	forza il modulo a non richiedere una nuova password (quando si esegue un cambiamento della stessa) ma ad usare quella proveniente da un modulo precedente (l'argomento è valido solo per il servizio passwd).

Tabella 4.12: Alcuni argomenti generici che possono essere passati ai moduli di PAM.

Una forma alternativa di configurazione è quella dell'uso del file **/etc/pam.conf** in cui si inseriscono tutte le regole di gestione che starebbero dentro i file di **/etc/pam.d.**²⁵ Il file prevede

²⁴si può comunque usare un file qualunque, specificando un pathname assoluto.

²⁵se quest'ultima è presente il file viene ignorato.

una linea per ciascuna regola ed al solito si ignorano le righe vuote e quelle inizianti per `#`; a differenza delle precedenti queste sono composta da cinque campi separati da spazi, nella forma:

```
service type control module-path [arguments ...]
```

dove **service** indica il nome dell'applicazione cui si applica la regola (quello che con `/etc/pam.d` è il nome del file).

Capitolo 5

Amministrazione straordinaria del sistema

5.1 La gestione di kernel e moduli

Tratteremo in questa sezione la gestione del kernel, in tutti i suoi aspetti: dalla scelta delle diverse versioni, al tipo di kernel da utilizzare, la sua ricompilazione, l'installazione, la gestione dei moduli, l'utilizzo delle *patch* e tutto quanto attiene la manutenzione dello stesso.

5.1.1 Le versioni del kernel

Uno dei primi problemi che ci si trova ad affrontare nella gestione del kernel è quello della scelta di versione quale usare. Nella maggior parte dei casi il kernel viene installato dalla propria distribuzione durante l'installazione, e molti, non avendo necessità specifiche (ad esempio la mancanza di supporto per un qualche dispositivo) evitano di installarne un altro.

Le esigenze che portano all'installazione di un nuovo kernel sono in genere due, la prima è ottimizzare il kernel per renderlo più adatto alla propria configurazione hardware; molti kernel di installazione infatti sono compilati con un supporto generico (per tipo di processore o per il chipset della piastra madre) per poter essere impiegati su qualunque PC; pertanto può essere utile ricompilarli per eliminare il supporto di funzionalità superflue non disponibili e attivare quello per la versione specifica del proprio hardware.

In questo caso si hanno due scelte, si può ricompilare il kernel della propria distribuzione (in genere tutte forniscono i relativi sorgenti), od utilizzare un kernel *ufficiale*.¹ In genere infatti le varie distribuzioni installano una propria versione del kernel, modificata applicando vari *patch*² che si ritiene migliorino le prestazioni o la stabilità ed aggiungono funzionalità reputate rilevanti, ma non ancora incluse nel kernel ufficiale.

Qualora si scelga il kernel della propria distribuzione c'è solo da procurarsi i relativi sorgenti, i file di configurazione e provvedere alla ricompilazione secondo le istruzioni di sez. 5.1.3. Se invece si vuole installare un kernel ufficiale (ad esempio per avere le funzionalità aggiunte nello sviluppo effettuato nel frattempo) occorre scegliere una versione adeguata.

La scelta della versione di kernel da utilizzare è in linea generale abbastanza semplice, occorre prendere l'ultima versione stabile. Per stabilire di quale versione si tratta basta andare sul sito ufficiale del kernel. Conviene comunque dare alcune spiegazioni sul significato dei numeri di versione del kernel: essi sono espressi sempre da tre numeri separati da punti.

¹si chiama così il kernel pubblicato su <http://www.kernel.org>, curato dal *maintainer* ufficiale (lo stesso Linus o chi lui ha delegato al compito).

²si chiamano così le modifiche, in forma di file prodotti dal programma *diff*, da applicare ai sorgenti tramite il comando omonimo, per ottenere una nuova versione degli stessi.

Il primo numero esprime la *major version*, un numero di versione che cambia solo in caso di fondamentali modifiche strutturali dell'infrastruttura, cosa avvenuta finora una sola volta (nel passaggio dei formati dei binari dall'*a.out* all'*ELF*). Al momento la *major version* è la 2 e non sembrano esserci all'orizzonte modifiche tali da giustificare una 3.

Il secondo numero esprime il cosiddetto *patchlevel*, ma indica più propriamente una serie di sviluppo, che è quella che invece cambia periodicamente. La convenzione scelta dagli sviluppatori è che un numero pari indica una versione *stabile*, mentre un numero dispari indica la versione *sperimentale*, di sviluppo, in cui vengono introdotte tutte le nuove funzionalità e le modifiche infrastrutturali che porteranno alla successiva versione stabile. Per esempio i kernel della serie 2.4.x indicano i kernel stabili, usati per le macchine in produzione, lo sviluppo dei quali³ è volto alla eliminazione dei bug e alla stabilizzazione del sistema, mentre i kernel della serie 2.5.x indicavano i kernel sperimentali, nati a partire da un precedente kernel stabile, nei quali sono state introdotte le nuove funzionalità, riscritte parti che non si consideravano soddisfacenti, ecc.

L'ultimo numero di versione è infine il numero progressivo che identifica i kernel all'interno di una serie, e che viene aggiornato periodicamente con il relativo sviluppo, nella direzione della stabilizzazione per le serie pari, nella direzione delle nuove funzionalità ed infrastrutture per quelle di sviluppo. Si tenga presente che ogni versione stabile ha in genere un suo *maintainer* (quello delle versioni di sviluppo finora è sempre stato Linus), che ne cura lo sviluppo ed il rilascio delle nuove versioni.

Al momento della scrittura di queste dispense (gennaio 2004) l'ultimo kernel "stabile" è il 2.6.1, mentre non esistono ancora kernel in versione instabile: siamo cioè in quel periodo che segue il rilascio di una nuova serie stabile in cui non si è ancora dato vita ad una nuova versione di sviluppo.⁴ Questo ci dice che in realtà il nuovo kernel stabile non è lo poi così tanto, dato che la nuova serie è appena nata; in genere ci vuole sempre un po' di tempo perché le nuove versioni stabili maturino e possano sostituire completamente le versioni precedenti. Per questo al momento è senz'altro più opportuno utilizzare l'ultima versione stabile precedente, cioè il kernel 2.4.24.

Questo ci dice che anche se genericamente valida, l'indicazione di utilizzare l'ultimo kernel della serie stabile, va presa comunque con prudenza. Possono esistere anche delle buone ragioni (macchine con software vecchio che non gira sulle nuove versioni e che non si può aggiornare) motivi di spazio (i nuovi kernel tendono a consumare più risorse e a non supportare più piattaforme hardware particolarmente *'data'*) che spingono a mantenere l'utilizzo di vecchie serie, come la 2.0.x e la 2.2.x, che sono a tutt'ora sviluppate, sia pure solo a livello di correzione dei pochi errori restanti.

5.1.2 Sorgenti e *patch*

Una volta scelta la versione del kernel da utilizzare, il passo successivo è quello di scaricare i sorgenti e ricompilarli. Come accennato il sito per la distribuzione delle versioni ufficiali è <http://www.kernel.org>, che in genere ha molto carico, per cui si consiglia l'uso di uno dei vari mirror italiani disponibili, la cui lista è segnalata sulla stessa pagina.

In genere i sorgenti vengono distribuiti nella directory `/pub/linux` (vi si accede sia in FTP

³accade spesso che gli sviluppatori si lascino comunque prendere la mano e introducano comunque nuove funzionalità, o eseguano *backporting* di codice dalla versione di sviluppo, si può dire comunque che in generale in una versione *stabile* viene curata molto la stabilità del sistema e la correzione degli errori rispetto all'inserimento di nuove funzionalità, che avviene solo quando esse sono state abbondantemente verificate.

⁴dopo alcuni mesi dal rilascio della nuova serie 2.6.x, (nel novembre 2004, alla serie 2.6.9), pare essere cambiata la modalità di sviluppo del kernel, che non prevede per il momento la creazione di nuove serie instabili, ma il passaggio da una versione stabile alla successiva, integrando anche grossi cambiamenti del kernel, e l'eventuale introduzione di un quarto livello di numerazione per tenere conto di eventuali *stabilizzazioni* di queste release successive.

che in HTTP) e sono disponibili in tre forme,⁵ le prime due sono degli archivi completi in formato `tar` compressi o con `bzip2` o con `gzip`, il cui nome sarà qualcosa del tipo `linux-2.4.24.tar.bz2` o `linux-2.4.24.tar.gz` (il primo è più compresso e si scarica più velocemente, ma sono sempre una ventina di megabyte abbondanti) oppure attraverso dei *patch* che permettono di passare da una versione precedente alla successiva, in modo che sia possibile evitare di riscaricare da capo l'archivio completo tutte le volte. Così ad esempio una volta che si abbiano i sorgenti del kernel 2.4.23 si potrà passare al 2.4.24 scaricando soltanto il file `patch-2.4.24.gz` (di norma viene compresso anche questo). Così diventa possibile aggiornare alla versione successiva senza dover effettuare dei download di enormi dimensioni.

Una volta scaricati gli archivi si dovranno scompattare questi ultimi che creeranno una directory `linux-2.4.24` nella directory corrente. A seconda dei casi il comando da usare è `tar -xvjf linux-2.4.24.tar.bz2` o `tar -xvzf linux-2.4.24.tar.gz`. In genere si tende a mettere detti sorgenti in `/usr/src` ma nella procedura di compilazione ed installazione niente obbliga a questa scelta, anzi, dato che non è necessario usare `root` per la compilazione, l'uso della propria home directory potrebbe anche essere una scelta migliore.

Un discorso diverso va fatto qualora si vogliano utilizzare i *patch*. Questo tra l'altro vale sia per il passaggio da una versione di kernel all'altra, che per l'applicazione di *patch* relativi all'installazione di funzionalità aggiuntive che possono interessare, ma che non sono ancora incluse nei sorgenti del kernel.

Per questo occorre capire cos'è un *patch*: questo è definito sulla base della differenza fra due file (in genere dei sorgenti, ma la cosa vale per qualunque file di testo), così come prodotta dal comando `diff`, che permette di indicare quali righe sono cambiate dall'uno all'altro e salvare il tutto su un file. Così si può passare da una versione di un programma alla successiva trasmettendo solo le differenze nel relativo sorgente. Il comando `diff` può inoltre essere eseguito ricorsivamente su due intere directory, registrando le differenze sia per quanto riguarda i vari file che esse contengono, che per l'aggiunta o la rimozione di alcuni di essi. Si può poi salvare il tutto su unico file, che verrà a costituire per l'appunto il *patch*; un esempio è il seguente:

```
--- linux-2.4.25/arch/i386/defconfig      2004-02-18 05:36:30.000000000 -0800
+++ linux-2.4.26/arch/i386/defconfig      2004-04-14 06:05:25.000000000 -0700
@@ -2,7 +2,6 @@
 # Automatically generated make config: don't edit
 #
 CONFIG_X86=y
-CONFIG_ISA=y
 # CONFIG_SBUS is not set
 CONFIG_UID16=y

@@ -31,12 +30,14 @@
 # CONFIG_MPENTIUM4 is not set
 # CONFIG_MK6 is not set
 # CONFIG_MK7 is not set
+# CONFIG_MK8 is not set
 # CONFIG_MELAN is not set
 # CONFIG_MCRUSOE is not set
 # CONFIG_MWINCHIP6 is not set
 # CONFIG_MWINCHIP2 is not set
 # CONFIG_MWINCHIP3D is not set
 # CONFIG_MCYRIXIII is not set
+# CONFIG_MVIAC3_2 is not set
 CONFIG_X86_WP_WORKS_OK=y
 CONFIG_X86_INVLPG=y
 CONFIG_X86_CMPXCHG=y
...
```

⁵in realtà esiste anche una forma di distribuzione tramite il protocollo `rsync`, che permette di ridurre la quantità di dati da scaricare.

la prima riga qui indica il file originale, mentre la seconda la nuova versione, si noti che si tratta di un pathname relativo, che ha come origine la directory in cui si trovano i due diversi alberi quando è stato eseguito il `diff`. Le righe che iniziano per `@@` indicano a quale riga nei due file fanno riferimento i dati riportati di seguito, nel caso 7 righe del primo file a partire dalla 12, che diventano 6 del secondo a partire sempre dalla stessa riga. Le differenze sono mostrate apponendo un `+` alle righe aggiunte nel secondo file ed un `-` a quelle tolte.

Con il comando `patch` si può invece compiere l'operazione inversa, e cioè applicare ad un certo file le differenze ottenute con il metodo precedente, in modo da convertirlo nella nuova versione. La cosa può essere effettuata anche ricorsivamente, su un intero albero di file e directory. Così se chi dispone dei nuovi sorgenti del kernel 2.4.24 mantiene anche quelli della versione 2.4.23, potrà generare un *patch* delle differenze, che applicato a questi ultimi li trasformerà in quelli del 2.4.24. L'utilità dei *patch* è che in questo modo anche chi cura la manutenzione di ulteriori funzionalità non presenti nei kernel ufficiali potrà limitarsi a distribuire il *patch* che contiene le relative aggiunte e modifiche, così che un utente possa, se lo desidera, applicarle senza dover scaricare tutto un nuovo albero dei sorgenti.

Il meccanismo è del tutto generale, ed inoltre il comando `patch` è sufficientemente intelligente da essere in grado di applicare anche più *patch* distinti in successione, fintanto che questi non andranno ad operare esattamente sulle stesse righe degli stessi file eseguendo modifiche incompatibili fra di loro. Così diventa possibile inserire nel kernel anche più funzionalità aggiuntive, fintanto che queste non interferiscono fra loro, o saltare da una versione di kernel ad un'altra che non sia la successiva applicando in successione più *patch*.

Come accennato il comando per applicare un *patch* è appunto `patch`. Nel caso più semplice in cui si deve operare su un singolo file la sintassi è immediata e si può eseguire il comando con un qualcosa del tipo:

```
patch original patch.diff
```

nel qual caso, a meno che non si sia specificata l'opzione `-b` (o `--backup`) per richiedere un backup, la nuova versione prenderà il posto dell'originale.

Quando però si ha a che fare con un *patch* che coinvolge più file (come quelli che si applicano ad un albero di sorgenti) i nomi dei file cui esso va applicato è riportato nel file stesso, e non devono quindi essere specificati; inoltre in questo caso il comando leggerà il contenuto del *patch* dallo standard input, per cui occorrerà usare una redirectione.

In questo caso per capire il funzionamento del comando occorre rifarsi all'esempio di *patch* mostrato in precedenza, il comando ricerca (a partire dalla directory corrente) il file che considera la vecchia versione e cerca di applicarvi le differenze. Il problema che molto spesso ci si trova di fronte è che si ha a disposizione solo la versione di partenza e non quella di arrivo, ad esempio si sono scompattati i sorgenti nella directory `linux-2.4.23` ma non si ha la directory `linux-2.4.24`. Per questo motivo di norma bisogna dire al comando, usando l'opzione `-p`, da quale livello di directory nell'albero dei sorgenti si vuole partire per applicare il *patch*. Il livello 0 usa semplicemente quanto specificato nel *patch* stesso, ma nel caso appena illustrato questo non funzionerebbe, in quanto non si sarebbe in grado di trovare il file di destinazione, se però ci si ponesse direttamente dentro la directory `linux-2.4.23` cancellando il primo livello di directory tutti i pathname relativi sarebbero risolti; pertanto di norma per applicare un *patch* sui sorgenti del kernel quello che si fa è:

```
cd /usr/src/linux
patch -p1 < /path/to/patch/patch.diff
```

Si tenga presente che se non si specifica un livello, il default di `patch` è di utilizzare solo il nome del file, ignorando le directory presenti nel pathname relativo, per cui in genere l'applicazione fallirà. Se il comando non riesce ad applicare un *patch* (ad esempio perché se ne è già applicato

uno incompatibile, o si è sbagliato file) genererà dei automaticamente dei file terminanti in `.rej` che contengono le modifiche che è stato impossibile effettuare. Inoltre `patch` è in grado di rilevare il caso in cui si prodotto il `patch` invertendo le versioni, nel qual caso avvisa richiedendo il permesso di applicare il `patch` alla rovescia; questo può essere richiesto esplicitamente con l'opzione `-R` (o `--reverse`). Si tenga presente però che se si tenta di applicare lo stesso `patch` una seconda volta si avrà proprio questo comportamento, ma proseguire nell'applicazione non sarebbe corretto, per questo esiste l'opzione `-N` (o `--forward`) che indica di ignorare i `patch` che sembrano invertiti o già applicati.

Il comando `patch` prende molte altre opzioni ed è in grado di utilizzare vari formati per i `patch` ed anche di interagire direttamente con vari programmi per il controllo di versione per identificare quali sono i file su cui operare. Per tutti i dettagli sul funzionamento del comando e sul significato delle opzioni si può al solito fare riferimento alla pagina di manuale, accessibile con `man patch`.

5.1.3 La ricompilazione del kernel

Una volta che si sono scompattati i sorgenti ed applicati gli eventuali `patch` ritenuti opportuni si può passare alla compilazione del kernel. Questa, come per la maggior parte dei pacchetti che si installano dai sorgenti, viene eseguita tramite il comando `make`, ma nel caso non viene utilizzata la procedura illustrata in sez. 4.2.1, in quanto nel caso del kernel non esiste uno script di configurazione, ma tutto viene gestito attraverso una procedura di costruzione dedicata, creata dagli stessi sviluppatori. Pertanto tutto la procedura è controllata dal `Makefile` principale presente nella base della directory dei sorgenti, e le varie operazioni sono compiute invocando gli opportuni `target`⁶ del comando `make`.

Una delle caratteristiche peculiari di Linux (torneremo sull'argomento in dettaglio anche in sez. 5.1.4) è quella di essere *modulare*. A differenza cioè degli altri sistemi unix-like in cui il kernel è un unico programma *monolitico*, caricato in memoria all'avvio del sistema ed in cui devono essere inserite tutte le funzionalità che si vogliono usare, Linux può partire con un kernel contenente le sole funzionalità di base e poi caricare da disco in maniera dinamica delle ulteriori sezioni di codice, dette moduli, che aggiungono le funzionalità ulteriori o il supporto per l'uso di certi dispositivi, solo quando servono.

Questa è una delle caratteristiche più rilevanti di Linux, che gli permette una flessibilità di utilizzo che gli altri kernel non hanno. È possibile infatti modularizzare lo sviluppo del kernel separandone le funzionalità, evitare di mantenere permanentemente in memoria parti di codice che sono utilizzate solo per limitati periodi di tempo (ad esempio il codice per accedere a CDROM o floppy occupa inutilmente memoria se non li si stanno utilizzando), indicare opzioni specifiche per la gestione di un dispositivo in fase di caricamento, o modificarle senza bisogno di un riavvio (basta rimuovere il modulo e ricaricarlo con le nuove opzioni).

L'uso dei moduli ha pertanto una grande rilevanza e deve essere pianificato accuratamente in fase di compilazione e configurazione. Un primo aspetto dell'uso dei moduli è che quando si usano diverse versioni del kernel devono essere usate anche diverse versioni dei moduli. Ciò comporta che ogni kernel deve avere la sua versione dei moduli, che sono identificati, come quest'ultimo, per la relativa versione, quella che viene mostrata dal comando `uname -r`.⁷ Sorge allora un problema quando si vogliono ottenere due (o più) kernel diversi a partire dagli stessi sorgenti, dato che in questo caso la versione sarà la stessa.

Per risolvere questo problema è allora possibile definire una versione “personalizzata”. La versione del kernel è indicata dai sorgenti, ed è codificata nelle prime righe del `Makefile` principale, che per i kernel ufficiali sono nella forma:

⁶si ricordi quanto accennato in sez. 4.2.1 relativamente al funzionamento di questo comando.

⁷in realtà come vedremo in sez. 5.1.4, i moduli sono identificati soprattutto per la directory in cui sono mantenuti, che è `/lib/modules/‘uname -r’`.

```

VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 27
EXTRAVERSION =

KERNELRELEASE=$(VERSION) . $(PATCHLEVEL) . $(SUBLEVEL) $(EXTRAVERSION)

```

e come si vede è qui che viene definita la variabile `KERNELRELEASE` che poi sarà usata in tutto il resto della procedura. Si noti allora la presenza, appositamente predisposta, della variabile `EXTRAVERSION`, che serve appunto a specificare un ulteriore identificativo di versione, che permetta di tenere separati kernel diversi (ad esempio per le opzioni di compilazione che si sono scelte) ottenuti a partire dagli stessi sorgenti. Nelle release ufficiali `EXTRAVERSION` resta sempre non definita,⁸ è cura di chi esegue una ricompilazione definirla adeguatamente, evitando di usare caratteri speciali, come lo spazio, che possono essere interpretati negli script (il caso più comune è aggiungere un “-qualcosa”).

In genere questa è l'unica modifica⁹ che può essere necessario fare a mano (anche se il `kernel-package` di Debian fornisce il comando `make-kpkg` che è in grado di farla automaticamente), tutte le altre configurazioni sono gestite in maniera indipendente, attraverso la modalità che vedremo più avanti. Le uniche altre eventuali (anche se poco probabili) modifiche che si possono voler fare al `Makefile` riguardano la variabile `CROSS_COMPILE`, che può essere usata per compilare kernel per una architettura diversa dalla propria (ad esempio un kernel per PowerPC su una macchina Intel), e le opzioni per le ottimizzazioni del `gcc` (che è meglio lasciar stare al valore di default, che è sicuro, a meno di non sapere esattamente quello che si sta facendo, o essere in vena di sperimentazione).

Un secondo aspetto dell'uso dei moduli che occorre tener presente è che per poterli utilizzare occorre anzitutto poter caricare in memoria il loro codice; il che significa che si deve essere in grado di leggere i relativi file oggetto,¹⁰ dato che questi non sono altro che codice, come quello dei programmi ordinari, anche se un po' particolare.¹¹ Questo comporta allora che le funzionalità del kernel necessarie ad accedere al supporto su cui si trovano i moduli non possono essere ottenute con l'uso di questi ultimi (e dovranno essere inserite all'interno del kernel in maniera *monolitica*).

Per capire quali sono queste funzionalità occorre ricordare quali sono i due compiti di base eseguiti dal kernel all'avvio: montare la directory radice ed eseguire `init`. Per il primo compito occorre il supporto per accedere al dispositivo su cui si trova la radice e quello per il relativo filesystem, per il secondo il supporto per l'uso del formato binario di esecuzione dei programmi. Quanto necessario a svolgere questi due compiti, anche se modularizzabile, dovrà comunque essere inserito permanentemente nel kernel, pena il fallimento dell'avvio con un *kernel panic*.¹²

A parte le eventuali modifiche del `Makefile` per modificare la `EXTRAVERSION`, il primo passo per la compilazione del kernel è quello della configurazione, in cui si scelgono quali funzionalità

⁸questo non vale per le release intermedie e per i sorgenti mantenuti indipendentemente da altri sviluppatori, che usano appunto questa variabile per marcare le loro versioni.

⁹in realtà il nel kernel 2.6.x è prevista una opzione per impostarla direttamente anche nell'interno del programma di configurazione.

¹⁰un file oggetto, in genere identificato dall'estensione `.o`, è un file che contiene il codice compilato di una o più funzioni, in cui però gli indirizzi non sono stati assegnati, in questo modo, attraverso un procedimento successivo detto *collegamento* (o meglio *linking*) si possono unire insieme più funzioni per dar luogo a quello che poi andrà a costituire un eseguibile; questo è anche quello che si fa quando si produce l'immagine del kernel, e l'uso dei moduli consente di ripetere il procedimento sul codice del kernel che sta girando.

¹¹ed infatti a partire dalla serie 2.6.x li si è distinti dai normali file *oggetto* usando l'estensione `.ko` al posto di `.o` (ma si ricordi che l'estensione in Unix è solo una convenzione, nel caso conta solo il contenuto).

¹²si chiama così un crash fatale del kernel; questo può avvenire solo per errori fatali durante l'esecuzione dello stesso (in caso di bug particolarmente gravi, molto rari, o di problemi hardware, assai meno rari), o in fase di boot quando mancano le componenti essenziali per l'avvio del sistema.

attivare e quali no, quali mettere direttamente dentro il kernel, e quali utilizzare come moduli. Una volta che si sia specificato quanto voluto, il kernel verrà costruito di conseguenza.

Tutto questo viene fatto, dal punto di vista della compilazione e della costruzione del kernel, tramite il contenuto del file `.config`, sempre nella directory base dei sorgenti, dove sono memorizzate tutte le opzioni di configurazione. Un estratto del contenuto del file è il seguente:

```
#
# Automatically generated make config: don't edit
#
# CONFIG_UID16 is not set
# CONFIG_RWSEM_GENERIC_SPINLOCK is not set
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_HAVE_DEC_LOCK=y
```

Le opzioni di configurazione sono tutte dichiarate come variabili nella forma `CONFIG_XXX`, quelle attivate non sono commentate ed assegnate al relativo valore, alcune indicano dei valori generici (come il tipo di processore o la codifica NLS¹³ usata di default) ma la maggior parte possono avere come valori possibili solo “y” che ne indica l’inclusione nel kernel (o la semplice attivazione), o, qualora l’opzione faccia riferimento ad una funzionalità che può essere modularizzata, “m”.

Come scritto nell’estratto illustrato in precedenza normalmente `.config` non deve essere scritto a mano, ma opportunamente generato, dato che, a meno di non sapere esattamente quello che si sta facendo, si rischia di attivare opzioni incompatibili fra di loro o inconsistenti. Per questo la configurazione viene eseguita invocando `make` con uno dei *target* di configurazione.

Il primo *target* è `config`, questo avvia uno script di shell che effettua la configurazione chiedendo di immettere sul terminale uno per uno i valori da assegnare varie opzioni che si vogliono attivare. Ovviamente eviterà di eseguire ulteriori domande qualora non si attivi una che le prevede opzione, ma il procedimento è comunque molto scomodo in quanto non esiste un meccanismo per correggere una impostazione una volta che si sia fatto un errore, per cui occorre ricominciare da capo. Pertanto è oggi praticamente in disuso, a parte per una sua versione modificata, invocabile con il *target* `oldconfig` che si limita a rileggere e riprocessare il file di configurazione precedente ricavando da questo, invece che dalle nostre risposte sul terminale, le risposte alle domande. Questo può risultare utile qualora si siano effettuate modifiche a mano del file `.config` e si voglia essere sicuri di ottenere un file di configurazione coerente.

Gli altri due *target* sono `menuconfig` e `xconfig` che attivano invece due interfacce utente, testuale la prima e grafica la seconda, con finestre e menù che permettono di selezionare interattivamente le varie opzioni ed effettuare le relative scelte in maniera casuale, senza serializzare le domande. Le due interfacce, a parte l’apparenza, sono sostanzialmente equivalenti, per cui tratteremo solo la prima. A partire dal kernel 2.6.x le interfacce grafiche sono diventate 2, la prima, sempre accessibile con `make xconfig` è basata sulle librerie QT, la seconda, accessibile con `make gconfig`, è basata sulle librerie GTK.

Eseguendo `make menuconfig` nella directory dei sorgenti del kernel si otterrà la pagina di avvio del programma di configurazione, mostrata in fig. 5.1. In genere il programma viene compilato la prima volta che si esegue il relativo bersaglio, questo talvolta fallisce in quanto per la compilazione necessitano le librerie *ncurses* su cui è basata l’interfaccia a finestre. Di norma queste vengono installate, ma non altrettanto avviene per i file di dichiarazione necessari alla compilazione, nel qual caso andrà installato il relativo pacchetto¹⁴ (e quelli delle *glibc*, qualora anch’essi fossero assenti).

La finestra di avvio riporta nella prima riga in alto la versione del kernel, se si è modificata la `EXTRAVERSION` questa dovrà comparire. Subito sotto c’è il titolo della sezione in cui ci si trova

¹³il *Native Language Support*, indica la codifica dei vari codici ASCII per le stringhe, come `iso8859-1`.

¹⁴le librerie *ncurses* sono presenti in tutte le distribuzioni, per cui è sempre il caso di usare i relativi pacchetti, per Debian sono `ncurses` e `libncurses-dev`.

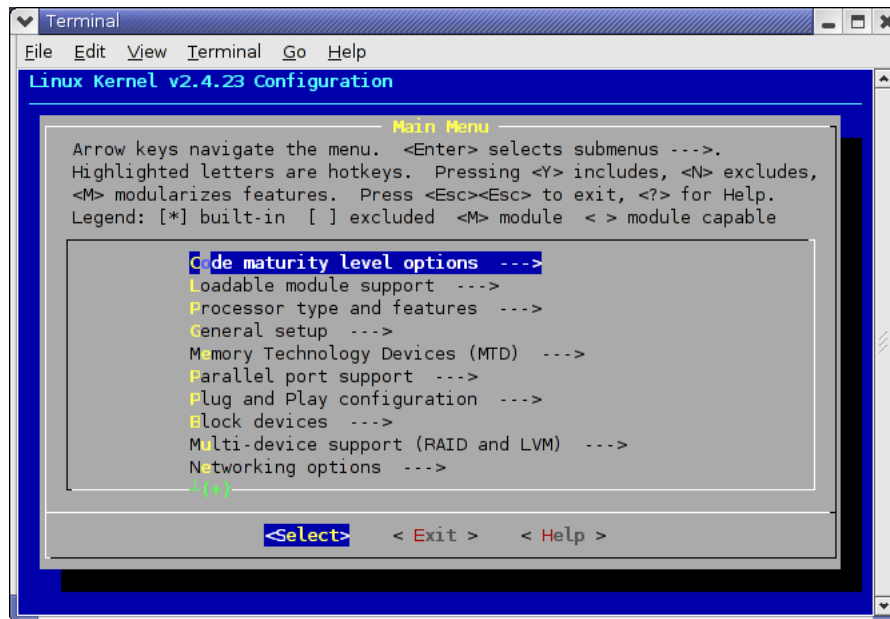


Figura 5.1: Schermata di avvio della configurazione del kernel con `make menuconfig`.

(nel caso è il menù principale), seguito da un breve riassunto dei principali comandi disponibili. Le frecce verticali permettono di spostarsi nella finestra centrale che contiene le varie sezioni in cui sono state suddivise le opzioni di configurazione. Nella parte bassa ci sono le tre opzioni principali che permettono di selezionare una opzione, uscire dalla finestra corrente e ottenere una finestra di aiuto (contestuale all'opzione selezionata), che possono essere cambiate con le frecce orizzontali.

Dal menù principale è possibile selezionare una sezione premendo invio (a meno di non aver cambiato l'opzione di selezione), e questo ci porterà nella finestra di configurazione delle relative opzioni, un esempio della quale è mostrato in fig. 5.2. I valori delle opzioni sono riportati all'inizio di ogni riga, quelli indicati fra parentesi tonde sono per le opzioni che richiedono un valore generico, che può essere selezionato da un menù a tendina o inserito da una finestra di immissione che si attivano quando l'opzione viene selezionata.

I valori fra parentesi quadre indicano le opzioni per le quali è possibile solo scegliere fra l'attivazione o meno e la scelta può essere fatta premendo il tasto "y" per attivare e "n" per disattivare, premendo la barra si può ciclare fra le due opzioni. Un asterisco indica che la funzionalità è attivata, uno spazio vuoto che non è attivata. Si tenga presente che se si tratta del supporto per funzionalità specifiche del kernel questo implica che il relativo codice sarà incluso monoliticamente, molte di queste opzioni però servono anche per attivare ulteriori configurazioni o specificare caratteristiche di un'altra opzione (che può anche essere modulare), nel qual caso le successive descrizioni appariranno indentate.

I valori fra parentesi angolari indicano invece le opzioni relative a funzionalità che possono essere anche modularizzate; rispetto alle precedenti possono presentare anche il valore "M" (attivabile direttamente premendo "m" o ciclando fra i valori possibili con la barra) che indica che si è optato per la creazione del relativo modulo, se invece si ha un "*" la funzionalità sarà inserita direttamente nel kernel.

Infine per alcune sezioni sono presenti delle ulteriori sottosezioni (sono indicate dalla assenza del valore delle opzioni e dal fatto che terminano con una freccia, come nel menù principale). Una volta che si sono effettuate le proprie scelte selezionando la voce di uscita si può tornare al menù precedente, se si è già nel menù principale invece si uscirà effettivamente dalla configurazione, e comparirà una finestra che richiede se si vuole che le nuove configurazioni siano salvate o

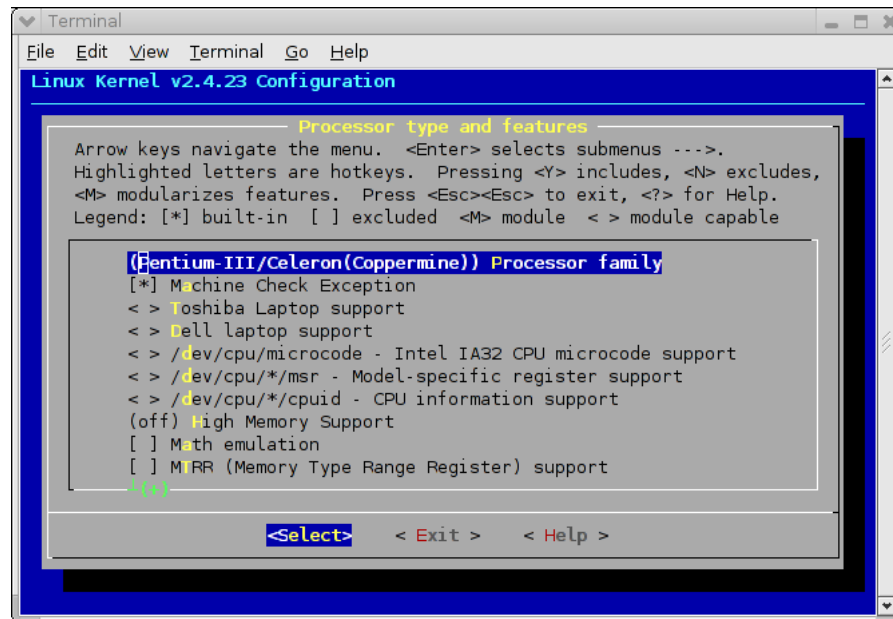


Figura 5.2: Schermata di configurazione del kernel con `make menuconfig`.

scartate, sovrascrivendo un nuovo `.config`.

Due opzioni specifiche per il menù principale sono poi quelle disponibili separatamente in fondo allo stesso, che permettono di salvare i valori della configurazione su, o caricarli da, un file specificabile dalla solita riga di immissione dei dati. Questo può comunque essere ottenuto con delle semplici copie del file `.config`.

Data la quantità (alcune centinaia) delle opzioni disponibili non è possibile commentarle tutte, pertanto ci limiteremo ad una descrizione sommaria del contenuto delle varie sezioni del menù principale, quelle disponibili con i kernel della serie 2.4.x¹⁵ sono le seguenti:

Code maturity level options

Questa sezione contiene una sola opzione che se attivata permette di vedere tutte le opzioni che sono classificate come sperimentali; in realtà è sempre il caso di attivarla in quanto buona parte delle opzioni sperimentali sono ampiamente utilizzate e perfettamente funzionanti; le opzioni “pericolose” vengono ampiamente segnalate nelle relative descrizioni.

Loadable module support

In questa sezione si attivano le opzioni per abilitare la gestione dei moduli ed il relativo supporto nel kernel. Le opzioni sono tre e in generale si possono attivare tutte, si può disattivare la seconda, che introduce un controllo di versione per i moduli in modo che non vengano caricati per errore moduli compilati per un'altra versione del kernel;¹⁶ questo può comportare problemi qualora si vogliano utilizzare moduli compilati a parte o distribuiti in forma binaria.

Processor type and features

Questa sezione contiene le opzioni relative alla scelta del tipo di CPU presente, con le opzioni per il supporto di vari insiemi di istruzioni estese (**MTRR (Memory Type Range Register) support**). Di particolare importanza è poi l'opzione per il **Symmetric multi-processing** che consente l'uso di macchine multiprocessore. Sempre

¹⁵alcune di queste sono presenti solo nelle versioni più recenti.

¹⁶il meccanismo funziona aggiungendo una *checksum* a tutti i nomi dei simboli del kernel, cosicché le relative funzioni possono essere chiamate solo all'interno dello stesso kernel.

qui vanno attivate le opzioni per l'uso di grandi quantità di memoria (**High Memory Support**) quando si ha più di 1Gb.

General setup

Questa è la sezione dove si attiva il supporto per le funzionalità principali del kernel, in particolare per i vari tipi di bus, per la rete, e per alcuni servizi interni, il formato degli eseguibili. In genere deve essere sempre abilitato il supporto per il bus PCI e per la rete (**Networking support** e **PCI support**). Il supporto per il formato ELF (**Kernel support for ELF binaries**) deve essere sempre incluso nel kernel (è il formato standard degli eseguibili, senza il quale non è possibile lanciare nessun programma). Altre due opzioni essenziali (necessarie al funzionamento moltissimi programmi) sono **System V IPC** e **Sysctl support**.

Memory Technology Devices

Questa sezione contiene le opzioni relative ai supporti di memoria opzionali come flash, memorie a stato solido ecc. utilizzate prevalentemente nei sistemi embedded.

Parallel port support

Questa sezione contiene le opzioni relative al supporto per la porta parallela, necessarie per poter utilizzare i dispositivi (ad esempio una stampante) ad essa collegati. Può essere completamente modulare.

Plug and Play configuration

Questa sezione contiene le opzioni per abilitare il supporto all'uso del *Plug and Play* per le schede che lo supportano. Può essere completamente modulare.

Block devices

Questa sezione contiene le opzioni per abilitare il supporto di una serie di dispositivi a blocchi (i floppy, vari *disk-array* e RAID hardware, i RAM disk e il loopback). A meno di non avere la radice su uno di questi dispositivi il supporto può essere modulare. Sono in genere da attivare **Normal floppy disk support**, **RAM disk support** e **Loopback device support**.

Multi-device support

Questa sezione contiene le opzioni per abilitare il supporto del RAID software (vedi sez.) e del Logical Volume Manager (vedi sez. 6.2). A meno di non avere la radice su uno di questi dispositivi il supporto può essere modulare.

Networking options

Questa sezione contiene le opzioni relative alla rete, in particolare il supporto per i vari protocolli di rete e tipi di socket e le funzionalità relative al filtraggio dei pacchetti (**Network packet filtering**) ed al routing avanzato. Sono da attivare **TCP/IP networking**, **Packet socket** e **Unix domain sockets**, per gli ultimi due è possibile farlo anche in maniera modulare. Se si usa **dhclient** è altresì necessario il supporto per il **Socket Filtering**.

Telephony Support

Questa sezione contiene le opzioni relative al supporto di schede telefoniche dedicate, che consentono di telefonare direttamente dal computer, ed usare questo come ponte fra linea telefonica normale e VoIP.

ATA/IDE/MFM/RLL support

Questa sezione contiene le opzioni per il supporto del bus IDE, di tutti i relativi dispositivi (dischi, CDROM, ecc.) e dei vari chipset. Se come nella maggior parte dei casi si hanno dischi IDE occorre abilitare ed inserire nel kernel **ATA/IDE/MFM/RLL**

support e nella sottosezione **IDE, ATA and ATAPI Block devices** le due opzioni **Enhanced IDE/MFM/RLL disk/cdrom/tape/floppy support** e **Include IDE/ATA-2 DISK support**, oltre al supporto per il proprio chipset, tutto il resto può essere modulare. È buona norma lasciare attivi **Generic PCI IDE Chipset Support** e **Generic PCI bus-master DMA support** che permettono l'avvio con un supporto generico. Inoltre è utile attivare l'opzione **Use PCI DMA by default when available** altrimenti l'I/O su disco risulterebbe estremamente rallentato.

SCSI support

Questa sezione contiene le opzioni per il supporto dei dispositivi SCSI (dischi, CD, nastri), del relativo protocollo, e dei vari controller. L'opzione **SCSI support** deve essere abilitata anche se non si hanno dispositivi SCSI in quanto il protocollo viene usato da altri sistemi, come i programmi per la masterizzazione (che usano l'emulazione IDE-SCSI) e le chiavi di memoria USB (che usano il protocollo SCSI per vedere la memoria come un disco). Se non si ha la radice su un disco SCSI tutto tranne **SCSI support**, **SCSI disk support** ed il supporto per il proprio controller può essere modulare. È presente una sottosezione per selezionare il supporto per i vari tipi di controller.

Fusion MPT device support

Questa sezione contiene le opzioni di configurazione per una scheda *LSI Logic Fusion*.

IEEE 1394 (FireWire) support

Questa sezione contiene le opzioni per il supporto delle interfacce e dei protocolli per il bus *Firewire*.

I2O device support

Questa sezione contiene le opzioni per il supporto del bus di comunicazione I2O usato per la comunicazione a basso livello fra i vari dispositivi presenti sulla scheda madre (ad esempio i sensori di temperatura).

Network device support

Questa sezione contiene le opzioni di configurazione per le varie schede di rete utilizzabili con Linux (sia ethernet che di altro tipo), più il supporto per alcuni protocolli di comunicazione di basso livello (PPP, SLIP, ecc.) e dispositivi virtuali. Il supporto può anche essere modulare, a meno di non avere la radice su un filesystem di rete.

Amateur Radio support

Questa sezione contiene le opzioni per la configurazione del protocollo AX.25, detto anche *Packet radio*, usato dai radioamatori per la trasmissione dati via radio.

IrDA (infrared) support

Questa sezione contiene le opzioni per il supporto dei dispositivi di comunicazione ad infrarossi (le porte IrDA).

ISDN subsystem

Questa sezione contiene le opzioni per il supporto dei dispositivi ISDN e dei relativi protocolli.

Old CD-ROM drivers

Questa sezione contiene le opzioni per il supporto dei vecchi CDROM pilotati direttamente dalla schede audio. Ampiamente in disuso.

Input core support

Questa sezione contiene le opzioni per il supporto per mouse, tastiere, joystick ed altri dispositivi di interazione (detti *Human Interface Device* su USB).

Character devices

Questa sezione contiene le opzioni di configurazione per una serie di dispositivi a caratteri. Sono essenziali le opzioni per il supporto dei terminali (**Virtual terminal** e **Support for console on virtual terminal**) che servono all'avvio per la console di sistema. Per poter utilizzare connessioni da remoto (ad esempio con **ssh**) è poi necessario il supporto per gli pseudo-terminali (**Unix98 PTY support**). É sempre in questa sezione che si abilita il supporto per le porte seriali **Standard/generic serial support** (e se si vuole la console sulla seriale anche **Support for console on serial port**). Qui può essere abilitato il supporto per la stampante su parallela (**Parallel printer support**), per l'uso del bus AGP (**/dev/agpgart (AGP Support)**) e per il supporto delle accelerazioni grafiche (**Direct Rendering Manager**) attraverso la interfaccia DRI di XFree86. Può essere inoltre utile abilitare il supporto per l'orologio in tempo reale (**Enhanced Real Time Clock Support**).

Multimedia devices

Questa sezione contiene le opzioni per il supporto di schede TV e schede radio.

File systems

Questa sezione contiene le opzioni per il supporto di un gran numero di diversi filesystem. Si deve essere sicuri di inserire nel kernel il supporto per il filesystem della radice (/), qualunque esso sia (i più usati sono ext2, ext3 e reiserfs). Qui si può anche abilitare il supporto per il filesystem dei CDROM (**ISO 9660 CDROM file system support**), e per i vari filesystem di Windows. É sempre opportuno abilitare il supporto per il filesystem **/proc** (che è usato da moltissimi programmi) e per gli pseudo-terminali (**/dev/pts**). Una sottosezione a parte è dedicata ai filesystem di rete (NFS, SMB ed altri), dove può essere configurato il relativo supporto.

Console drivers

Questa sezione contiene le opzioni per il supporto di tutta una varietà di diversi dispositivi a caratteri. Di norma basta configurare solo l'opzione **VGA text console**, a meno di non avere necessità del framebuffer (come avviene per le macchine che non usano la VGA, come gli Apple).

Sound Questa sezione contiene le opzioni relative al supporto delle schede sonore per Linux.

USB support

Questa sezione contiene le opzioni per il supporto dei vari chipset del bus USB e dei vari dispositivi che si possono inserire su di esso.

Bluetooth support

Questa sezione contiene le opzioni per il supporto dei protocolli e dei dispositivi *Bluetooth*.

Kernel hacking

Questa sezione contiene le opzioni per la configurazione del supporto di alcune funzionalità utilizzate principalmente dagli sviluppatori per il debug del kernel. Può essere utile abilitare l'opzione **Magic SysRq key** che permette l'uso di particolari combinazioni di tasti (a partire appunto da *SysRq*) per tentare un recupero in estremo dei dati in caso di crash del kernel.

Cryptographic options

Questa sezione contiene le opzioni per il supporto di vari algoritmi crittografici all'interno del kernel. In genere viene utilizzato per supportare filesystem cifrati e IPSEC.

Library routines

Questa sezione contiene le opzioni per la configurazioni di alcune librerie usate dal kernel.

Una volta completata la configurazione, qualunque sia il metodo con cui la si è effettuata, viene salvato il nuovo `.config`.

A questo punto la procedura cambia a seconda che si stia utilizzando un kernel dalla serie 2.4.x o precedenti o un 2.6. Infatti con la serie 2.6.x anche la procedura di ricompilazione è stata profondamente modificata (e migliorata), pertanto quanto segue si applica soltanto ai kernel precedenti la serie 2.6.x.

Se è la prima volta che si compila il kernel, il primo passo è creare le dipendenze con il comando `make dep`. Il comando esegue due compiti, il primo è creare le dipendenze¹⁷ per la compilazione, il secondo, se si è abilitato il controllo della versione dei moduli, è calcolare le informazioni per il versionamento nei simboli.¹⁸ Pertanto quando se non si è abilitato il versionamento è necessario eseguire questo comando soltanto la prima volta che si effettua una compilazione, altrimenti deve essere eseguito ogni volta che si cambia la configurazione, in quanto l'informazione sulla versione dei simboli dipende da questa.

Il passo successivo è normalmente quello di compilare il kernel, sui normali PC questo si fa con il comando `make bzImage`, che crea l'omonima immagine compressa del kernel nella directory `arch/i386/boot/`; su altre architetture si usa in genere `make vmlinux` che crea una immagine non compressa nel file omonimo nella directory corrente. Un altro bersaglio possibile è `make zImage`, che crea una immagine compressa, valido anche in per altre architetture.

Questo era il bersaglio originale per la creazione delle immagini del kernel, e può essere ancora usato fintanto che il kernel è di dimensione inferiore a 512kb. Se la dimensione è superiore occorre invece usare `make bzImage`, non tanto, come qualcuno ancora ritiene, perché così l'immagine viene compressa di più,¹⁹ quanto perché nel primo caso il kernel viene caricato nella cosiddetta *low memory* (cioè sotto i primi 640kiB) e viene utilizzato un meccanismo d'avvio diverso, mentre nel secondo caso viene caricato sopra 1MiB.

Al giorno d'oggi l'unica ragione per usare `zImage` è quella della compatibilità con alcune vecchie versioni di LILO ed alcuni vecchi BIOS che non supportano la procedura di avvio di `bzImage`, che non risente del limite di 512k nella dimensione, ed è anche più veloce. Questo ovviamente vale solo per l'architettura PC, se si usano altre architetture ci possono essere altri bersagli o può non essere necessario l'uso di un kernel compresso (è il caso dell'architettura PPC dei Mac).

Una volta compilata l'immagine del kernel il passo successivo è, se li si sono abilitati, passare alla compilazione dei moduli. Questo viene fatto con il comando `make modules`.

La compilazione del kernel (e dei moduli) è in genere un processo piuttosto lungo e che utilizza pesantemente le risorse (memoria e CPU) della macchina. Pertanto viene spesso anche usato come test di efficienza.²⁰ Il procedimento può essere velocizzato usando l'opzione `-j` del comando `make` che consente di *parallelizzare* la compilazione. Questa è una funzionalità del tutto generale di `make`²¹ che consente di specificare come parametro dell'opzione un numero di

¹⁷cioè determinare quali file di dichiarazione (i `.h`) sono necessari per produrre i relativi file binari (i `.o`) contenenti il codice di kernel e moduli.

¹⁸si chiamano così i nomi delle funzioni che vengono dichiarate all'interno di un modulo, ma possono essere usati da altri; per far questo si dice che il simbolo deve essere *esportato*, il versionamento funziona aggiungendo al nome di ciascun simbolo un hash unico che impedisce di chiamare da un kernel diverso le suddette funzioni.

¹⁹la compressione è identica nei due casi e viene sempre effettuata con `gzip`, nonostante il nome `bzImage` possa trarre in inganno, `bzip2` non viene mai usato.

²⁰ad esempio può facilmente causare il surriscaldamento della CPU, per cui viene utilizzato spesso per verificare se un *overclocking* è andato a buon fine.

²¹in realtà la funzionalità è della versione GNU di `make`, non è detto la si ritrovi su altre versioni.

Target	Significato
<code>config</code>	interfaccia di configurazione a linea di comando.
<code>oldconfig</code>	interfaccia di configurazione a linea di comando, che riutilizza i valori precedentemente immessi.
<code>menuconfig</code>	interfaccia di configurazione a grafica testuale, basata sulle librerie <code>ncurses</code> .
<code>xconfig</code>	interfaccia di configurazione grafica.
<code>dep</code>	crea le dipendenze per la compilazione e le informazioni per il versionamento dei moduli.
<code>depend</code>	identico a <code>dep</code> .
<code>zImage</code>	crea una immagine compressa del kernel (valida su tutte le architetture).
<code>bzImage</code>	crea una immagine compressa del kernel (su architettura PC) con una diversa procedura di avvio.
<code>vmlinux</code>	crea una immagine non compressa del kernel.
<code>modules</code>	compila i moduli.
<code>modules_install</code>	installa i moduli nella relativa directory.
<code>clean</code>	cancella tutti i file oggetto (i <code>.o</code>) presenti prodotti da una precedente compilazione.
<code>mrproper</code>	oltre a quanto esegue <code>clean</code> , cancella anche le informazioni sulle dipendenze.
<code>distclean</code>	oltre a quanto esegue <code>mrproper</code> e cancella ulteriori file prodotti cercando di riportare l'albero dei sorgenti identico allo stato immediatamente dopo la scompattazione.

Tabella 5.1: Principali *target* del comando `make` per la compilazione del kernel fino alla versione 2.4.x.

processi da eseguire in parallelo,²² ciascuno dei quali compilerà parti indipendenti,²³ così da avere in generale sempre qualche processo in compilazione anche quando gli altri sono bloccati sull'I/O.

Una volta eseguita la compilazione i passi successivi riguardano l'installazione; il primo passo è installare i moduli, questo viene fatto usando un ulteriore *target*, con il comando `make modules_install`. I moduli vengono sempre installati sotto `/lib/modules`, in una directory diversa per ciascuna versione del kernel, con lo stesso nome della versione del kernel (così come definita nel `Makefile`); così nel caso si installino i moduli del kernel 2.4.24 i moduli e tutti i relativi file saranno installati in `/lib/modules/2.4.24/`.

I passi successivi sono l'installazione della nuova immagine del kernel e di file relativi. Per questo vengono anche forniti alcuni *target* per `make`, ma in genere è preferibile eseguire l'operazione manualmente, anche perché in questo caso le operazioni sono indipendenti dal *bootloader* che si intende usare.

Come spiegato in sez. 1.2.3 i file necessari all'avvio del sistema sono mantenuti in `/boot`, pertanto è qui che deve essere copiata l'immagine del kernel, che nella maggior parte dei casi è `arch/i386.boot/bzImage`; la convezione è chiamare il file con il nome `vmlinuz-versione`, dove la versione è quella impostata con il `Makefile`. In genere è utile anche copiare il file `System.map`, questo contiene le informazioni che permettono di identificare per nome (e non tramite l'indirizzo in memoria) le varie routine del kernel, la cosiddetta *mappa dei simboli*.²⁴

Il contenuto di questo file è utilizzato dal processo interno al kernel che invia al servizio del *syslog* gli eventuali errori riscontrati nell'esecuzione del kernel; questi contengono degli indirizzi binari, ed il file viene usato per poter ottenere i nomi delle funzioni coinvolte invece dei loro

²²la cosa è particolarmente efficiente su macchine multiprocessore.

²³questo è possibile sfruttando appunto le informazioni sulle dipendenze usate da `make` per affidare a processi diversi la compilazioni di sorgenti indipendenti.

²⁴è in questo file cioè che viene mantenuto l'elenco completo dei nomi delle funzioni *esportate* (cioè rese visibili anche alle altre funzioni, così che queste possano chiamarle) all'interno del kernel, sia quelle presenti nell'immagine di avvio che quelle presenti nei moduli.

indirizzi.²⁵ La sua assenza perciò non comporta problemi di funzionamento del sistema, ma solo una maggiore difficoltà per chi dovrà andare ad analizzare gli errori.

Lo stesso file è utilizzato anche in sede di installazione dei moduli quando vengono calcolate le dipendenze di un modulo da un altro, ed il comando `depmod` (che vedremo a breve) darà degli errori in caso di sua mancanza o di non corrispondenza con il kernel attivo. Come per l'immagine del kernel questo viene di norma copiato su `/boot` appendendo un “-” e la versione; questo fa sì che ogni kernel sia in grado di trovare ed utilizzare la sua mappa.

Infine è buona norma, tutte le volte che si installa un nuovo kernel, salvare anche le opzioni di configurazione con cui lo si è prodotto, questo significa copiare anche il file `.config`, di norma questo si fa (ad esempio è la scelta di Debian) copiandolo sempre sotto `/boot`, con il nome `config-versione`.

Ricapitolando, l'insieme dei vari passi per ottenere un nuovo kernel ed installare tutti i file relativi, è il seguente:

```
make config
make dep
make -j 3 bzImage
make -j 3 modules
make modules_install
cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.24-my
cp System.map /boot/System.map-2.4.24-my
cp .config /boot/config-2.4.24-my
```

dopo di che occorrerà configurare il proprio *bootloader* (vedi sez. 5.3) per l'uso del nuovo kernel.

5.1.4 La gestione dei moduli

Come accennato nella sezione precedente una delle caratteristiche più significative del kernel Linux è la modularità, che permette, tutte le volte che si richiede una funzionalità mancante, di tentare, prima di restituire un errore, il caricamento del modulo che la provvede. Come abbiamo visto questo comporta la configurazione del relativo supporto nel kernel e la compilazione come moduli delle varie funzionalità che si vogliono utilizzare in questo modo; inoltre occorre anche l'installazione di una serie di programmi in user space che permettono di gestire questa funzionalità: il pacchetto `modutils`.

Nelle vecchie versioni del kernel la gestione dei moduli era fatta attraverso un apposito demone, `kernelld`, che riceveva le richieste dal kernel ed eseguiva il caricamento dei moduli. A partire dalla serie 2.4.x il meccanismo è stato realizzato con un apposito sottosistema del kernel, detto *kmod*.

Il meccanismo si basa su una apposita funzione interna al kernel²⁶ che prende come parametro una stringa indicante il modulo che si vuole caricare (ma con questa si possono anche richiedere, come vedremo fra poco, funzionalità generiche) e crea un processo temporaneo interno al kernel che consente di invocare un apposito programma in user space il quale si incarica di tutte le operazioni necessarie al caricamento del modulo.²⁷ Questo è di norma `modprobe` (che esamineremo in dettaglio a breve) ma si può specificare un qualunque altro programma sia attraverso l'interfaccia del `sysctl` che scrivendolo direttamente in `/proc/sys/kernel/modprobe`.

²⁵la funzionalità di per sé non è essenziale, ma senza questa informazione diventa molto complesso per chi sviluppa il kernel capire dove si è verificato l'errore, ed anche più difficile per voi chiedere aiuto o cercare informazioni relativamente all'errore stesso.

²⁶la funzione è `request_module`, inizialmente oltre a questa `kmod` era un processo interno al kernel che girava in permanenza, poi però l'interfaccia è stata semplificata in modo da usare solo questa funzione e creare il processo su richiesta.

²⁷data la complessità delle operazioni non è possibile eseguire un compito del genere in kernel space, mentre usando un programma in user space si possono avere a disposizione tutte le funzionalità del sistema.

Il programma fondamentale per l'uso dei moduli è **insmod**, che si incarica di caricare un modulo all'interno del kernel, leggendolo dal disco, effettuando la risoluzione dei simboli, e *collegandolo* al codice del kernel. Il comando prende come parametro il **nome** del modulo, e per trovare il file il comando cerca il corrispondente file oggetto (cioè **nome.o**) sotto la directory **/lib/modules/`uname -r`**,²⁸ a meno che non si sia usata la variabile di ambiente **MODPATH** o una diversa opzione nel file di configurazione **/etc/modules.conf** per indicare una directory diversa.

Se il modulo li prevede possono essere ulteriormente specificati dei parametri nella forma **parametro=valore** dove il **parametro** dipende dal modulo (la lista dei parametri disponibili per ciascun modulo si può ottenere tramite il comando **modinfo**), ed il **valore** può essere una stringa o un numero intero, quest'ultimo specificabile sia in forma decimale (17), che ottale (021), che esadecimale (0x11).

Si tenga presente che **insmod** consente di inserire nel kernel solo un modulo alla volta, e per farlo ha bisogno di risolvere tutti i simboli necessari al modulo stesso, se alcuni di questi non sono presenti nel kernel, ma in altri moduli, il comando fallirà con un errore di “**unresolved symbol**”.

Come norma di sicurezza il comando non carica i moduli se i relativi file non appartengono all'amministratore, onde evitare che il contenuto di un modulo possa essere sovrascritto in caso di compromissione dell'utente cui appartiene, con la conseguente possibilità di far eseguire direttamente al kernel il codice che si vuole; questo comportamento può essere disabilitato con l'opzione **-r** (ad uso principalmente degli sviluppatori).

Inoltre **insmod** di norma controlla che la versione del kernel corrente e quella del modulo combacino, in questo modo si evita di caricare moduli che appartengano a kernel diversi; anche questo comportamento può essere disabilitato con l'opzione **-f**. Si tenga presente però che in questo caso viene evitato soltanto il controllo sulla versione del kernel, il controllo sull'uso dei nomi dei simboli non può essere evitato, questo significa che se si sono compilati i moduli con il supporto per il versionamento (che crea dei nomi di simboli contenenti una checksum) non sarà comunque possibile utilizzarli.

Per l'elenco completo di tutte le opzioni (alcune sono comunque obsolete, facendo riferimento al vecchio **kernelld**) con le relative spiegazioni dettagliate si può consultare al solito la pagina di manuale, accessibile con **man insmod**; le principali opzioni si sono comunque riportate in tab. 5.2 con una breve spiegazione.

Opzione	Significato
-f	Evita il controllo della corrispondenza fra versione del kernel e versione del modulo.
-L	Usa il file locking per prevenire tentativi simultanei di caricare lo stesso modulo.
-h	Stampa un sommario del comando e relative opzioni.
-n	Esegue tutta la procedura eccettuato il caricamento finale del modulo.
-r	Disabilita la condizione che il modulo da caricare sia di proprietà di root.
-v	Abilita la stampa di un maggior numero di informazioni.
-k	imposta il flag di <i>auto-clean</i> per il modulo, che viene controllato da kernelld per rimuovere i moduli non più in uso.

Tabella 5.2: Principali opzioni del comando **insmod**.

Come accennato **insmod** consente di inserire un modulo solo quando tutti i simboli di cui questo ha bisogno possono essere referenziati; questo comporta che se alcuni di questi sono

²⁸si è indicata la directory con questa notazione in quanto **uname -r** restituisce appunto la stringa con il nome della versione del kernel, ed è proprio con tale nome che vengono cercati i moduli.

definiti da un altro modulo, si avrà un problema di dipendenze. Per ovviare a questo problema c'è un secondo comando, **modprobe**, che permette di risolvere anche tutte le dipendenze, trovare quali sono gli altri moduli che servono per poterne utilizzare uno, e caricare preventivamente anche questi.

Il meccanismo con cui **modprobe** è in grado di risolvere le dipendenze si basa sul contenuto del file **modules.dep** che si trova nella directory in cui sono installati i moduli. Questo viene di norma prodotto in fase di installazione degli stessi (tramite il comando **depmod** su cui torneremo più avanti) ed ha un formato del tipo:

```
/lib/modules/2.4.23/kernel/fs/vfat/vfat.o: /lib/modules/2.4.23/kernel/fs/fat/fat.o
```

che assomiglia a quello di un **Makefile**, dove per ciascun modulo viene indicato la lista degli altri da cui dipende.

Come **insmod** anche **modprobe** effettua la ricerca dei moduli da caricare fra quelli compilati per il kernel corrente, nella directory **/lib/modules/`uname -r`**, dove questi vengono installati con **make modules_install**. In genere i moduli vengono poi suddivisi in ulteriori sottodirectory; questa suddivisione cambia a seconda della versione del kernel. Ad esempio a partire dal kernel 2.4 i moduli sono installati sotto la directory **kernel**, e all'interno di questa suddivisi per categorie: nel caso avremo **fs** per il supporto dei filesystem, **driver** per il supporto delle periferiche, **net** per il supporto dei protocolli di rete, **crypto** per gli algoritmi di crittografia. A loro volta i moduli installati sotto **drivers** sono suddivisi per tipologia di hardware.

La potenza di **modprobe** è che il comando, oltre alla risoluzione automatica delle dipendenze, è in grado anche di caricare più moduli in contemporanea e, sfruttando la suddivisione delle sottodirectory appena illustrata, anche uno fra tutti quelli che forniscono una certa funzionalità.

Di norma infatti **modprobe** prevede come argomento il nome (o i nomi) dei moduli da caricare, (da indicare senza l'estensione **.o** finale), se invece si specifica l'opzione **-t** si indica di trattare il parametro successivo come un pattern di ricerca all'interno della directory dei moduli, in questo caso il comando tenterà di caricare in sequenza tutti i moduli il cui pathname corrisponde al pattern, fermandosi al primo che viene caricato con successo. Questo consente ad esempio di chiedere il caricamento del driver di una scheda di rete (senza dover specificare quale) con un comando del tipo:

```
modprobe -t drivers/net \*
```

dato che in questo caso verranno provati tutti i moduli presenti in quella directory.

Specificando anche l'opzione **-a** la stessa operazione verrà eseguita per tutti i moduli della lista senza fermarsi al primo che è stato caricato successo. Con l'opzione **-l** invece si avrà la lista dei moduli che corrispondono. Infine con l'opzione **-r** si può richiedere la rimozione dell'intera pila di moduli caricati in dipendenza dal modulo specificato (sempre che nel frattempo non siano stati utilizzati).

Come nel caso di **insmod** anche con **modprobe** si può specificare un parametro da passare al modulo che viene caricato, il vantaggio di **modprobe** è che attraverso l'uso del suo file di configurazione si possono passare dei valori di default senza doverli scrivere esplicitamente. Le altre opzioni del comando sono riportate in tab. 5.3, l'elenco completo ed una descrizione dettagliata delle stesse è come sempre disponibile nella pagina di manuale, accessibile con **man modprobe**.

Un comando essenziale per il funzionamento di **modprobe** è **depmod** che crea il file **modules.dep** che identifica le dipendenze fra i vari moduli passati come argomenti sulla riga di comando. È grazie a questo file che è possibile determinare quali sono i moduli che contengono i simboli necessari per poter poi caricare un altro modulo, così da poter effettuare il caricamento di tutti i moduli nella giusta sequenza. In genere il comando viene sempre invocato senza argomenti e con l'opzione **-a**,²⁹ dato che in tal caso esegue il calcolo delle dipendenze con i moduli presenti

²⁹ nelle ultime versioni questa è opzionale.

Opzione	Significato
-t	Usa una lista di moduli da caricare che corrispondono ad un pattern.
-a	Carica tutti i moduli della lista specificata con -t invece di fermarsi al primo.
-l	Stampa la lista dei moduli che corrispondono ad un certo pattern specificato con -t.
-n	Esegue tutta la procedura eccettuato il caricamento finale del modulo.
-r	Rimuove il modulo specificato e l'insieme di moduli da cui esso dipende, o esegue l' <i>autoclean</i> .
-v	Abilita la stampa di un maggior numero di informazioni.
-C	Permette di usare un differente file di configurazione (da passare come parametro per l'opzione).
-c	mostra i valori della configurazione corrente.

Tabella 5.3: Principali opzioni del comando `modprobe`.

in tutte le directory specificate in `modules.conf`. Con l'opzione `-A` il calcolo viene effettuato controllando preventivamente i tempi dei file, aggiornando `modules.dep` solo se qualcosa è cambiato.

Una volta che i moduli non sono più utilizzati possono essere rimossi con il comando `rmmod`, che prende come parametro il nome di un modulo. Ovviamente perché il comando abbia successo il modulo in questione non deve essere in uso, né contenere simboli usati da un altro modulo (cioè non devono esserci altri moduli che *dipendano* da esso).

Se però si usa l'opzione `-r` il comando esegue una rimozione ricorsiva, cercando di rimuovere anche i moduli che dipendono dal modulo indicato (diventa così possibile effettuare l'operazione inversa di `modprobe`). L'uso dell'opzione `-a` attiva invece l'*autoclean*, marca cioè i moduli inutilizzati come "*ripulire*" e rimuove i moduli che erano già stati marcati come tali. In questo modo si può compiere l'operazione in due passi diminuendo la probabilità di rimuovere moduli temporaneamente inutilizzati. Al solito l'elenco completo delle opzioni con le relative descrizioni è disponibile nella pagina di manuale accessibile con `man rmmod`.

Il comportamento del comando `modprobe`, e con esso dell'intero meccanismo di caricamento automatico dei moduli, che viene realizzato attraverso questo programma, è determinato dal file di configurazione `/etc/modules.conf`.³⁰ Qui si possono specificare una serie di direttive che permettono di controllare sia le modalità con cui vengono caricati i moduli, che le directory dove effettuare le ricerche. Il formato del file prevede anche la presenza di direttive condizionali e l'uso di variabili, con sintassi analoga a quella della shell, ma queste funzionalità non sono molto usate.

La direttiva principale che si trova nel file è `alias`, che permette di associare un modulo ad una certa funzionalità. In realtà la direttiva consente semplicemente di associare un nome (un *alias* appunto) ad un modulo (indicato al solito con il nome del relativo file oggetto, ma senza estensione). In questo modo si può usare il nome dell'*alias* al posto di quello del modulo nella invocazione di `modprobe`. La potenza reale della direttiva sta nel fatto che il kernel, quando necessita dell'uso di una certa funzionalità, utilizza `kmod` per invocare `modprobe`, passandogli come parametro un opportuno identificativo, e si capisce subito allora che basta usare detto identificativo come *alias* per un certo modulo per ottenere l'associazione di quest'ultimo alla relativa funzionalità.

Il problema è allora di sapere quali sono gli identificativi utilizzati dal kernel; un certo numero di essi sono predefiniti,³¹ ed già associati all'unico modulo che può essere utilizzato. Esistono

³⁰in alcuni sistemi più vecchi può essere usato invece il file `/etc/conf.modules`, che è deprecato e non deve essere più utilizzato.

³¹invocando `modprobe -c` specificando un file di configurazione vuoto (con `-C`) si può stampare la configurazione

però tutta una serie di funzionalità che non sono necessariamente associate ad un unico modulo: il caso classico è quello del controller SCSI (vedi sez. 5.4.2, identificato come `scsi-hostadapter`, che deve essere fatto corrispondere al modulo specifico della scheda SCSI di cui si dispone, ad esempio se si ha una Adaptec si potrà usare una riga del tipo:

```
alias scsi-hostadapter aic7xxx
```

Il problema è che non esiste una lista di riferimento che indichi i nomi delle varie funzionalità, per cui si possono dare solo indicazioni generali. Allora per quanto riguarda ethernet si possono associare le singole interfacce ad un certo modulo (relativo ad una certa scheda) usando il nome dell'interfaccia stessa. Per le schede sonore invece si può usare la `sound-slot`, per il controller del bus USB (vedi sez. 5.4.4) `usb-interface`, ecc. Inoltre per le periferiche associate ad un file di dispositivo si può usare la notazione generica `char-major-NN`, o `block-major-NN-MM` o direttamente il file di dispositivo stesso; così un estratto del file potrebbe essere:

```
...
alias sound-slot-0 dmasound_pmac
alias char-major-14-3 dmasound_pmac
alias /dev/dsp dmasound_pmac
alias sound-service-0-0 i2c-keywest
alias char-major-14-0 i2c-keywest
alias /dev/mixer i2c-keywest
...
alias eth0 sungem
alias eth1 airport
...
```

Si tenga presente infine che si possono tranquillamente creare `alias` facendo riferimento ad altri `alias`. Inoltre ci sono due parole chiave che si possono specificare al posto del modulo, con `off` si indica a `modprobe` di ignorare le richieste di caricare quel modulo, con `null` invece si fa sì che la richiesta abbia comunque successo, anche se non viene caricato niente.

Una seconda direttiva è `option`, che prende come primo argomento il nome di un modulo (o di un `alias`) seguito dai parametri da passare al suddetto modulo quando viene caricato. Alla direttiva si può apporre un `add` che fa sì che i parametri specificati vengano aggiunti ad altri eventualmente già presenti.

Infine la direttiva `path` permette di specificare le directory in cui eseguire la ricerca dei moduli, questa ha due diverse forme possibili:

```
path=/lib/modules/2.0.*/
path[net]=/lib/modules/`uname -r`/net
```

nella prima si indica semplicemente una directory, mentre nella seconda si specifica anche, fra parentesi quadre, una etichetta che identifica una classe di moduli (se non specificata si assume il valore `misc`). Se la direttiva non viene utilizzata vengono usate le directory predefinite che sono le seguenti:

```
path[boot]=/lib/modules/boot
path[toplevel]=/lib/modules/`uname -r`
path[toplevel]=/lib/modules/`kernelversion`
path[toplevel]=/lib/modules/default
path[toplevel]=/lib/modules
```

ma anche una singola occorrenza della direttiva `path` sovrascrive questi valori con quanto indicato; se si vuole che queste vengano mantenute si utilizzare la direttiva `keep` prima di specificare qualunque direttiva `path`.

di default.

Un elenco delle altre principali direttive, con relativa descrizione, è riportato in tab. 5.4, per l'elenco completo e delle spiegazioni più dettagliate si può al solito fare riferimento alla pagina di manuale, accessibile con `man modules.conf`.

Opzione	Significato
<code>path</code>	definisce una directory dove cercare i moduli.
<code>keep</code>	mantiene le directory predefinite per la ricerca dei moduli.
<code>option</code>	definisce un parametro opzionale da passare al modulo quando viene caricato.
<code>alias</code>	definisce un nome da associare a un modulo.
<code>pre-install</code>	definisce un comando da eseguire prima di caricare il modulo specificato.
<code>install</code>	definisce un comando da usare al posto di <code>insmod</code> per caricare il modulo specificato.
<code>post-install</code>	definisce un comando da eseguire dopo aver caricato il modulo specificato.
<code>pre-remove</code>	definisce un comando da eseguire prima di rimuovere il modulo specificato.
<code>remove</code>	definisce un comando da usare al posto di <code>rmmmod</code> per caricare il modulo specificato.
<code>post-remove</code>	definisce un comando da eseguire dopo aver rimosso il modulo specificato.

Tabella 5.4: Principali direttive del file di configurazione `modules.conf`.

Si tenga comunque presente che con il kernel 2.6 il meccanismo di caricamento dei moduli è stato completamente riscritto, e che quanto illustrato finora fa riferimento solo alle versioni precedenti (per essere precisi fino al kernel 2.5.48, quando è stato introdotto il nuovo sistema).

Un altro file utilizzato da Debian per la gestione dei moduli è `/etc/modules`, che contiene la lista dei moduli che si vuole siano 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
```

Oltre ai comandi per il caricamento dei moduli, il pacchetto `modutils` contiene altri comandi di gestione. Abbiamo visto che molti moduli possono prendere dei parametri che consentono di specificarne il comportamento. Per sapere quali sono si può usare il comando `modinfo` che consente di esaminare il file oggetto del modulo ed estrarne una serie di informazioni, la principale delle quali è la lista dei parametri supportati dal modulo. Un esempio del comando è il seguente:

```

anarres:/home/piccardi# modinfo radeon
filename:      /lib/modules/2.4.23-ben1/kernel/drivers/char/drm/radeon.o
description:   "ATI Radeon"
author:        "Gareth Hughes, Keith Whitwell, others."
license:       "GPL and additional rights"
parm:          drm_opts string

```

e come si vede questo modulo ha un parametro `drm_opts` il cui valore è una stringa. Il comando supporta una serie di opzioni che gli permettono di stampare solo alcune delle informazioni disponibili, per la descrizione completa si può fare riferimento alla relativa pagina di manuale.

Infine il comando `lsmod` permette di ottenere la lista dei moduli caricati in memoria e tutte le informazioni ad essi relative; il comando non prende opzioni (a parte le classiche `-h` e `-V`) né argomenti; un esempio di questo comando è:

```

anarres:/home/piccardi# lsmod
Module                Size  Used by    Not tainted
hid                   23156    0 (unused)
radeon                111132    1
agpgart               18012    3
ds                    8284    1
yenta_socket         11936    1
dmasound_pmac         65408    1 (autoclean)
dmasound_core        12832    1 (autoclean) [dmasound_pmac]
soundcore             4184    3 (autoclean) [dmasound_core]
pcmcia_core           44328    0 [ds yenta_socket]
airport               3348    1 (autoclean)
orinoco               38616    0 (autoclean) [airport]
hermes                9088    0 (autoclean) [airport orinoco]
usb-ohci              22848    0 (unused)
usbcore               72628    1 [hid usb-ohci]

```

La prima colonna indica il nome del modulo, mentre la seconda le sue dimensioni. La terza colonna indica quante volte è usato il modulo, mentre l'ultima colonna indica se il modulo è inutilizzato o marcato per la rimozione (fra parentesi tonde) e quali altri moduli dipendono da lui (fra parentesi quadre).

Un modulo il cui conteggio di utilizzo non sia nullo non può essere rimosso, lo stesso vale se il modulo ha altri moduli che dipendono da lui, a meno di non usare l'opzione `-r` di `rmmod`.

5.2 La gestione dei dischi e dei filesystem

In questa sezione prenderemo in esame la gestione dei dischi a partire da alcune nozioni relative all'hardware, per passare al partizionamento ed ai relativi programmi. Tratteremo inoltre la gestione dei filesystem per quanto riguarda quelle tutte le operazioni (come creazione, riparazione e modifica dei parametri interni) che non sono direttamente connesse all'uso ordinario (montaggio e smontaggio), che è già stato trattato in sez. 1.2.4.

5.2.1 Alcune nozioni generali

Prima di affrontare l'uso dei programmi per la gestione di dischi e filesystem occorre introdurre qualche concetto relativo al funzionamento fisico dei dischi rigidi. In genere, e per questo sono dispositivi a blocchi, lo spazio disponibile sui piatti magnetici di un disco viene suddiviso in blocchi elementari di dati, delle dimensioni di 512 byte, chiamati *settori*. Dal disco si può sempre leggere un settore alla volta (e non un singolo byte), questo viene fatto direttamente dal kernel

(o dal BIOS) che dice all'interfaccia hardware di leggere un determinato settore fornendogli l'indirizzo dello stesso.

Le prime interfacce IDE separavano fisicamente le linee di indirizzamento che consentivano richiedere la lettura di un particolare settore all'interno del disco in tre gruppi. Un primo gruppo serviva per indirizzare le testine dei diversi piatti, da cui il nome *head* e la sigla *H*; in sostanza vedendo il disco come un cilindro questo parametro indicava la coordinata in altezza lungo l'asse dello stesso, facendo riferimento ad un determinato piatto nel disco.

Il secondo gruppo serviva per muoversi lungo la coordinata angolare all'interno di un piatto, da cui il nome *sector* e la sigla *S*; infine l'ultimo gruppo indicava come muoversi lungo la coordinata radiale, da cui il nome *cylinder* e la sigla *C*. Per questo ancora oggi si parla di *geometria* di un disco, e quando se ne vuole identificare le dimensioni si indica i rispettivi valori massimi di questi tre parametri (la terna chiamata CHS), moltiplicando i quali si ottiene il numero totale dei settori, e quindi la capacità del disco.

Il problema è che nelle prime interfacce IDE c'era una separazione fisica delle linee che indicavano questi indirizzi, 10 linee servivano per indicare il cilindro, 4 per le testine, e 6 per i settori, per un totale di 1024x16x63 settori, pari a 504Mib. Anche il BIOS usava questa suddivisione nella routine di accesso al disco (la INT13), che richiedeva, per accedere ad un certo settore, una terna di valori indicanti appunto il numero di cilindro, testina e settore. Questi venivano passati tramite il contenuto di alcuni registri del processore,³² ed in particolare 10 bit erano utilizzati per indicare il cilindro, 6 bit per indicare il settore ed 8 bit per la testina.

La corrispondenza diretta fra valori di *head*, *sector* e *cylinder* e coordinate fisiche del settore indirizzato è andata persa quasi subito, non appena i dischi han cominciato a superare le dimensioni massime previste dall'interfaccia IDE originale. Ma per mantenere la compatibilità con i sistemi operativi che usavano il BIOS, l'interfaccia di accesso di quest'ultimo è rimasta invariata, l'accesso ad un settore veniva eseguito direttamente dall'hardware in maniera trasparente, indipendentemente dalla geometria reale del disco. Dato che il parametro indicante la testina è di 8 bit, la dimensione massima ottenibile con questa interfaccia è di un totale di 1024x256x64 settori, pari a circa 8.4GiB, a lungo considerato un limite irraggiungibile.

Il limite irraggiungibile è stato però raggiunto piuttosto presto, ma a questo punto ci si è trovati di fronte al limite non più superabile delle restrizioni sui valori massimi di cilindro, testina e settore, da cui deriva il famoso problema di non poter accedere a dischi oltre il 1024-simo cilindro che affligge i BIOS più vecchi, ed i sistemi operativi e le vecchie versioni di LILO che sono in grado di usare soltanto l'interfaccia originaria.

Tutto questo è stato superato con l'introduzione del *Linear Block Addressing*, in cui anche per i dischi IDE l'accesso è eseguito, come per i dischi SCSI,³³ specificando semplicemente un numero di settore che cresce linearmente da 0 al valore massimo. Questo però comporta che la vecchia interfaccia non è più utilizzabile; per questo nei BIOS più recenti al posto della vecchia INT13 può essere usata anche una "INT13 estesa" che utilizza direttamente il numero del settore in forma lineare. Resta il problema che alcuni sistemi operativi e vari programmi non sono in grado di utilizzare questa nuova interfaccia, e per loro deve essere usata la vecchia interfaccia.

In genere il problema della geometria non si pone assolutamente per Linux, dato che il kernel è in grado di accedere nativamente all'interfaccia IDE ed indirizzare direttamente l'accesso ai singoli settori, per cui non risente affatto delle limitazioni del BIOS, il problema invece si pone per un *bootloader* come LILO che dovendo stare nei pochi byte a disposizione nel *Master Boot Record* (vedi sez. 5.2.2) non può implementare un suo accesso indipendente. Il che significa che se si mette il kernel in una zona del disco oltre il 1024-simo cilindro con dei BIOS che

³²per i dettagli sui registri e su tutte le varie limitazioni storicamente susseguites si può leggere il *Large Disk HOWTO*.

³³il protocollo SCSI non ha mai avuto problemi di geometria, dato che ha sempre previsto l'uso di un valore lineare per indicare il settore, anche se poi il BIOS si doveva inventare una geometria per poter usare le sue routine di accesso.

non supportano LBA, LILO non sarà in grado di leggerlo. Questo è un problema che in tutti i computer moderni è abbondantemente superato, ma che si può presentare ancora quando si mettono dischi nuovi su macchine molto vecchie.

In tal caso si deve avere l'accortezza di mettere il kernel in una sezione di disco che sia entro il 1024-simo cilindro (cioè nella parte iniziale del disco). Questo può voler dire la necessità di creare una partizione iniziale (di solito la si monta sotto `/boot`) su cui si mettono le immagini del kernel; ma può essere un problema quando nella prima partizione si è installato Windows e questa è troppo grossa.

5.2.2 Il partizionamento

Uno dei compiti di preparazione che occorre eseguire tutte le volte che si installa un nuovo disco è quello di suddividerne lo spazio in opportune *partizioni*. In genere questo viene fatto all'installazione del sistema attraverso il programma di installazione, che in tutte le distribuzioni più recenti è in grado di eseguire il compito in maniera semi-automatica. Quando si installa un nuovo disco, o se si vuole effettuare l'operazione manualmente per avere un maggior controllo, occorre utilizzare un apposito programma, `fdisk`.

Prima di entrare nel dettaglio del funzionamento di `fdisk` e derivati è comunque opportuno dare alcuni cenni sui criteri base del partizionamento. Anzitutto occorre ricordare che nell'architettura del PC esiste (a causa delle limitazioni storiche) un limite massimo al numero di partizioni fisiche effettive (dette anche *partizioni primarie*), queste vengono memorizzate nel primo settore di ciascun disco, quello che viene chiamato, dato che è sempre lì che viene installato il *bootloader* per l'avvio del sistema, il *Master Boot Record*. Una parte di questo settore costituisce la cosiddetta *tabella delle partizioni*, e dato lo spazio limitato questa non può contenerne più di quattro. Qui vengono memorizzate settore di inizio e di fine delle partizioni.

Architetture meno obsolete (come SPARC, PowerPC Apple, o Alpha) non hanno questo limite, ed il numero di partizioni è sostanzialmente arbitrario. Per superare questa limitazione dell'architettura PC sono state introdotte le cosiddette *partizioni logiche* o *estese*. In tal caso quello che si fa è di indicare nella tabella delle partizioni sull'MBR l'utilizzo di una *partizione estesa* e poi utilizzare il settore di inizio di quest'ultima per installarvi una ulteriore tabella delle partizioni, in modo da poterne utilizzare di altre, che vengono chiamate *logiche* o *secondarie*, in numero più elevato.

Come accennato in sez. 1.2.4 le partizioni vengono indicate nel file di dispositivo che si riferisce ad un disco con il relativo numero. Le partizioni primarie sono sempre numerate da 1 a 4, mentre quelle logiche iniziano dal 5, così ad esempio `/dev/hda2` è la seconda partizione primaria del primo disco del primo canale IDE, mentre `/dev/hdb5` è la prima partizione logica del secondo disco del primo canale IDE.

Partizioni primarie e secondarie possono coesistere, nei termini in cui si usa una partizione primaria per indicare la tabella delle partizioni secondarie. In genere questo lo si fa creando tre partizioni primarie ed usando la quarta per le partizioni secondarie, ma si può usare una partizione primaria qualunque, (ad esempio la seconda) tenendo presente però che si è utilizzata una partizione primaria per indicare le partizioni estese, le partizioni primarie successive (nel caso la terza e la quarta) non saranno più utilizzabili.

Il comando che tradizionalmente viene utilizzato per partizionare un disco³⁴ è `fdisk`, sulla sua base sono stati creati anche altri programmi,³⁵ ma noi faremo riferimento solo a questo ed alla variante `cfdisk`. Entrambi operano esclusivamente sulla tabella delle partizioni, ed in

³⁴su un PC, non tratteremo qui le altre architetture, che hanno programmi diversi.

³⁵comprese alcune versioni dotate di interfaccia grafica, di uso più intuitivo, come `qtparted` o `gparted`, che in realtà si appoggiano più a `parted` (vedi sez. 6.2.5) che a `fdisk`.

genere se si effettuano modifiche il contenuto delle stesse³⁶ viene perduto. Per questo motivo il loro utilizzo è in genere limitato al momento dell'installazione del sistema o di uno nuovo disco.

Il comando prende come argomento il file di dispositivo che indica un disco (ad esempio `/dev/hdc` per un disco IDE o `/dev/sda` per un disco SCSI), l'unica opzione che ha senso usare è `-l` che si limita a stampare la tabella delle partizioni presente. Le altre opzioni servono per specificare le caratteristiche del disco e non servono più in quanto i valori sono determinati automaticamente (e per quanto riguarda numero di cilindri, testine e settori possono essere modificati anche all'interno del comando). Quando il comando viene lanciato stampa un messaggio di benvenuto, e avvisa di possibili problemi (vedi sez. 5.3.2) quando il numero dei cilindri è maggiore di 1024. Un esempio del risultato del comando è il seguente:

```
monk:/root# fdisk /dev/sda

The number of cylinders for this disk is set to 2213.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help):
```

dove dopo la stampa iniziale si pone in attesa dei comandi che vengono specificati con le relative lettere. Così se vogliamo una lista dei comandi possibili possiamo premere `m`, ottenendo:

```
Command (m for help): m
Command action
  a   toggle a bootable flag
  b   edit bsd disklabel
  c   toggle the dos compatibility flag
  d   delete a partition
  l   list known partition types
  m   print this menu
  n   add a new partition
  o   create a new empty DOS partition table
  p   print the partition table
  q   quit without saving changes
  s   create a new empty Sun disklabel
  t   change a partition's system id
  u   change display/entry units
  v   verify the partition table
  w   write table to disk and exit
  x   extra functionality (experts only)

Command (m for help):
```

ed allora con `p` potremo stampare la tabella delle partizioni corrente:

```
Command (m for help): p

Disk /dev/sda: 255 heads, 63 sectors, 2213 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1            1         2176    17478688+   83   Linux
/dev/sda2          2177         2213     297202+   82   Linux swap
```

³⁶a meno di non aver fatto un semplice ridimensionamento e trattato opportunamente il filesystem, vedi sez. 6.2.5.

che mostra due partizioni, di cui una normale, una per la *swap*. Si noti anche come viene riportata una geometria per il disco; questa è quella che è vista dal kernel, ed in genere coincide con quanto visto dal BIOS, ma è possibile cambiarla con uno dei comandi delle funzionalità avanzate.

A questo punto si può usare **d** per cancellare una partizione (ne chiederà il numero), ed **n** per crearne una nuova, nel qual caso prima chiederà se primaria o secondaria e poi il relativo numero. Una volta scelta la partizione il programma chiederà il settore di partenza (proponendo come default il primo che risulta libero) e la dimensione. Quest'ultima può essere specificata in varie unità di misura o direttamente con il settore finale (il comando propone come default l'ultimo settore del disco).

Il comando **t** permette di impostare il valore numerico che identifica il *tipo* di partizione; questo viene in genere utilizzato per indicare che tipo di filesystem sarà contenuti nella stessa. Per i filesystem di Linux il valore è **83** (in esadecimale) e vale per tutti i filesystem. I filesystem Windows invece usano vari codici diversi a seconda del tipo di filesystem. Il valore **82** è invece usato per indicare una partizione destinata all'uso come *swap* (vedi sez. 5.2.5). Altri valori usati da Linux sono **fd** per indicare le partizioni usate per il RAID software (vedi sez. 6.1) e **8e** per quelle usate per il *Logical Volume Manager* (vedi sez. 6.2).

Qualunque modifica si effettui sulla tabella delle partizioni non sarà mai registrata effettivamente su disco fintanto che non si dà il comando **w**, il quale comunque chiederà conferma avvertendo che si possono perdere tutti i dati presenti sul disco. Si esce da **fdisk** con il comando **q**, che però non salva le modifiche effettuate.

Un comando alternativo a **fdisk** è **cdisk**, una versione più *amichevole* che permette l'uso una interfaccia testuale semigrafica ed interattiva in cui si possono eseguire le varie operazioni spostandosi su comandi e partizioni con l'uso delle frecce. In fig. 5.3 è mostrata la schermata di **cdisk**, dalla quale è poi possibile dare tutti i comandi.

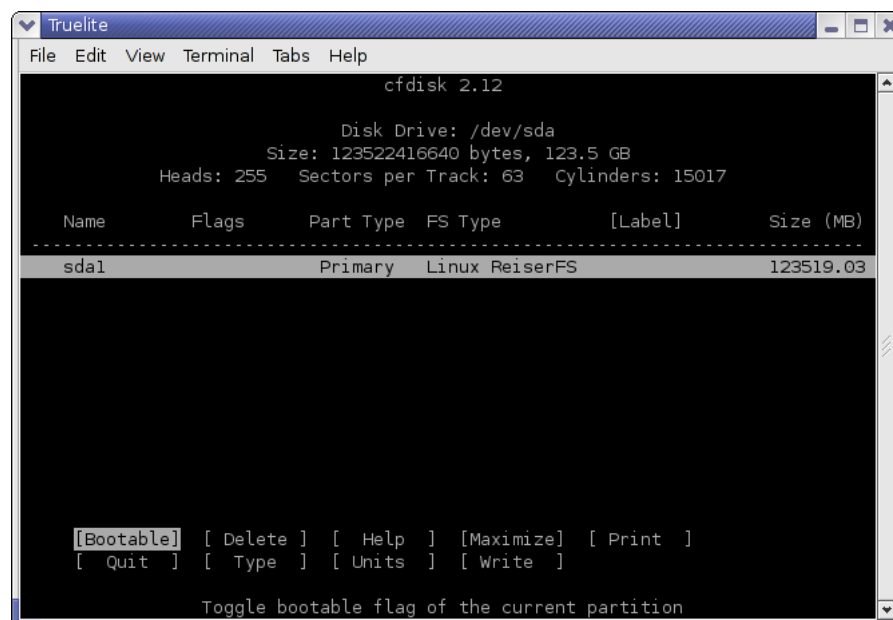


Figura 5.3: Schermata del comando **cdisk**.

Ovviamente il partizionamento di un disco è una operazione privilegiata, che può compiere solo l'amministratore. È anche un compito delicato, in quanto se si cambia la tabella delle partizioni cancellando o modificando una di quelle presenti il relativo contenuto andrà perso.³⁷

³⁷al riavvio successivo, in quanto il kernel legge la tabella delle partizioni all'avvio e la mantiene in memoria; per questo se si ripartizione un disco in uso occorre riavviare il sistema.

Occorre quindi stare molto attenti ed evitare di salvare le modifiche fintanto che non si è assolutamente sicuri di quanto si è fatto; infatti ogni filesystem viene creato per stare esattamente nelle dimensioni di una partizione, e se se ne modificano le dimensioni o la posizione le informazioni del filesystem non saranno più utilizzabili.

Pertanto è sempre opportuno pianificare bene le dimensioni delle partizioni fin dall'installazione del sistema, perché modificarle in un secondo tempo comporta spesso la necessità di dover fare un backup dei contenuti. Esiste comunque il comando `parted` (che tratteremo in sez. 6.2.5), che permette di ridimensionare un filesystem e restringere le relative partizioni, anche se l'operazione resta complessa e delicata.

Tutto questo significa che nell'installazione occorre decidere una *strategia di partizionamento*, cioè decidere quante partizioni fare e come assegnarle; per farlo occorre tenere ben presenti le indicazioni sul contenuto delle varie directory di sistema illustrate in sez. 1.2.3. Una delle strategie più semplici è quella di creare una partizione unica e mettere tutto quanto su di essa. Questo schema ha il vantaggio di non doversi preoccupare del dimensionamento delle altre partizioni né del rischio di esaurire lo spazio su una mentre le altre sono vuote. Ha però lo svantaggio che se si deve reinstallare il sistema i dati degli utenti e quelli di sistema (come pagine web, posta archiviata e tutto quanto in genere si tiene sotto `/var`) verrà cancellato. Inoltre se si hanno più dischi non è ovviamente possibile mettere tutto su una sola partizione.

Per questo di solito si preferisce creare almeno una partizione su cui montare `/home` per i dati degli utenti, ed un'altra per `/var`. Se poi si vuole separare anche quanto strettamente necessario all'avvio dal resto del sistema si può mettere anche `/usr` su una partizione separata. Infine in certi casi (per il solito problema del 1024 cilindro visto in sez. 5.3.2) può essere necessario creare una partizione separata per `/boot`.

Infine se si opera spostando tutto il possibile su partizioni separate si avrà una riduzione del contenuto della radice che consente di attuare una strategia di ridondanza in cui si tengono due copie della stessa su due partizioni diverse. In questo modo se per qualche motivo si danneggiano file essenziali per il sistema, si avrà sempre a disposizione l'altra partizione con un sistema in grado di partire.

Usare varie partizioni (e più dischi) ha comunque una serie di controindicazioni; anzitutto si può sprecare inutilmente spazio disco quando una partizione si riempie mentre le altre sono vuote, e a questo punto si ha solo l'alternativa di ripartizionare o di spostare pezzi da una partizione all'altra e farvi riferimento con dei link simbolici, che non è il massimo dell'eleganza, e può comportare una serie di problemi per i programmi di backup che di norma vengono usati imponendo di archiviare solo i file presenti sul filesystem corrente, per evitare di inserire nel backup cose che non c'entrano nulla.

Inoltre una volta che si sono spostate `/home`, `/var` e `/usr` (ed eventualmente `/opt`) non è che resti molto altro, e per quanto grandi si possano fare le partizioni, non si risolverà mai il problema di esaurire lo spazio in quelle con la maggior parte dei contenuti, se l'occupazione cresce oltre la capacità del più grande dei dischi che si hanno. Per questo spesso ad esempio si è costretti ad usare più dischi per le home directory degli utenti, che non possono essere più tenute sotto una directory unica, ma andranno suddivise su pathname diversi, complicandone la gestione.

Come si vede l'uso delle partizioni comporta una serie di potenziali problemi di gestione dovuti alla necessità di pianificare adeguatamente in fase di installazione la gestione dello spazio disco; per risolverli alla radice è disponibile, a partire dai kernel della serie 2.4.x il sistema del *Logical Volume Manager* (che tratteremo in sez. 6.2) che permette di unire partizioni³⁸ e dischi fisici diversi in dei *volumi logici* all'interno dei quali creare i relativi filesystem.

³⁸si noti come in fig. 5.3 compaia appunto una partizione di tipo **Linux LVM**, che è quella da usare per i volumi fisici che verranno usati con LVM come componenti di un volume logico.

5.2.3 La creazione di un filesystem

Abbiamo visto in sez. 1.2.4 come si può rendere disponibile il contenuto di un filesystem nell'albero delle directory; quando si installa il sistema però (o quando si aggiunge un disco) i filesystem devono essere creati, in modo da strutturare lo spazio disponibile (che sia una partizione o un dispositivo fisico o virtuale) per l'uso, operazione comunemente detta *formattazione*.

Filesystem	Opzione	Descrizione
reiserfs	mkreiserfs	
	-b N	Specifica la dimensione dei blocchi.
	-s N	Specifica la dimensione del giornale.
	-j file	Specifica un file di dispositivo per il giornale.
	-h hash	Specifica un algoritmo di <i>hashing</i> per la ricerca veloce dei file all'interno delle directory.
msdos	mkfs.msdos	
	-F N	Specifica la dimensione della FAT (<i>File Allocation Table</i>).
	-n name	Specifica un nome per il volume (11 caratteri).
	-c	Esegue il controllo del dispositivo per eventuali settori difettosi.
vfat	mkfs.vfat	
		Identiche a mkfs.msdos .
ext2	mkfs.ext2	
	-b N	Specifica la dimensione dei blocchi.
	-F	Forza successivo controllo del filesystem.
	-i N	Specifica ogni quanti byte di spazio disco deve essere creato un inode, non può essere inferiore alle dimensioni di un blocco.
	-j	Crea il giornale per il filesystem (equivalente ad invocare il comando come mkfs.ext3).
	-J	Permette di specificare i parametri per la gestione del file di giornale.
	-m N	Specifica la percentuale di spazio disco riservata per l'amministratore che gli utenti normali non possono usare.
	-c	Esegue il controllo del dispositivo per eventuali settori difettosi.
	-L	Associa una etichetta al filesystem (che può essere utilizzata per farvi riferimento al posto del file di dispositivo) .
ext3	mkfs.ext3	
		Identiche a mkfs.ext2 .

Tabella 5.5: Le principali opzioni per i comandi di creazione dei filesystem.

Si tenga presente che questa è una operazione diversa dalla formattazione *fisica* del dispositivo che divide lo spazio sul disco in tracce e settori, come quella che di solito si fa sui dischetti. In genere per gli hard disk questa operazione non è più necessaria dato che viene eseguita direttamente dal fabbricante una volta per tutte. Si deve invece eseguirla per i floppy, con il comando **fdformat**, ma questo non comporta che poi si possa utilizzarlo, occorrerà anche creare il filesystem: in Linux infatti la formattazione fisica del dispositivo e la creazione del filesystem, sono delle operazioni distinte, che vengono sempre tenute separate, nonostante alcuni sistemi operativi le eseguano insieme.

La creazione di un filesystem su un dispositivo si esegue con il comando **mkfs**, questo prende come parametro il file di dispositivo su cui si vuole creare il filesystem, mentre il tipo di filesystem che si vuole creare si specifica con l'opzione **-t**. In realtà il comando è solo un front-end, che chiama, per ciascun tipo di filesystem, un comando specifico. Così ad esempio se si vuole formattare un filesystem **msdos** con **mkfs -t msdos** verrà invocato **mkfs.msdos**. In generale i programmi di formattazione sono forniti insieme agli altri programmi di supporto per il filesystem (come quelli per il controllo di consistenza) e possono avere un nome specifico (ad esempio per Reiser FS il programma si chiama **mkreiserfs**) che può essere invocato direttamente, ma perché la forma generica **mkfs -t type** funzioni deve essere presente il corrispondente **mkfs.type**.

Le altre opzioni dipendono dal filesystem scelto, e possono essere trovate nella relativa pagina

di manuale, accessibile al solito con `man mkfs.type`. Un elenco delle principali opzioni disponibili con i principali filesystem è riportato in tab. 5.5.

Benché oggi esistano molteplici alternative (Reiser, JFS, XFS), il filesystem più usato con Linux è *ext2*, o il suo fratello maggiore *ext3*, che mantiene una identica struttura dei dati su disco, e viene gestito semplicemente in maniera diversa a livello del kernel. Per questo motivo esamineremo, anche nelle sezioni seguenti, i vari comandi specifici della gestione di questo filesystem (che valgono anche per *ext3*), a partire da quello usato per la creazione, `mke2fs`, un cui esempio è:

```

anarres:/home/piccardi# mke2fs /dev/hda3
mke2fs 1.35-WIP (07-Dec-2003)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
25064 inodes, 100000 blocks
5000 blocks (5.00%) reserved for the super user
First data block=1
13 block groups
8192 blocks per group, 8192 fragments per group
1928 inodes per group
Superblock backups stored on blocks:
    8193, 24577, 40961, 57345, 73729

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

Se usato senza specificare altro che il dispositivo³⁹ da formattare il comando si limita a creare il filesystem, stampando tutta una serie di informazioni ad esso relative. È particolarmente significativa quella relativa a dove sono stati posti i vari backup del *superblock*⁴⁰ questo infatti è un blocco speciale che contiene tutte le informazioni relative alla configurazione del filesystem, ed è essenziale per poterlo montare. La perdita del *superblock* causerebbe la perdita completa del filesystem per cui ne vengono mantenute varie copie in modo che in caso di distruzione o corruzione si possa usare una delle copie. Pertanto è bene annotarsi la posizione delle varie copie, anche se è possibile recuperarla in un secondo momento con `dumpe2fs`.

Al di là della invocazione diretta appena mostrata il comando prevede numerose opzioni che permettono di impostare tutta una serie di caratteristiche del filesystem che si va a creare. Una delle principali è la dimensione dei *blocchi* in cui lo spazio disco viene suddiviso, da specificare tramite l'opzione `-b`. Con *blocco* si indica in genere l'unità minima di spazio disco che viene allocata per inserirvi i dati di un file; in genere⁴¹ ogni file consuma almeno un blocco, e le dimensioni dei file sono dei multipli interi di questo valore. Valori tipici sono 1024, 2048 o 4096. Qualora si ometta questo parametro il suo valore viene stabilito con una valutazione euristica

³⁹anche se volendo si può anche usare un file normale che potrà poi essere montato in loopback.

⁴⁰la spiegazione esatta del significato dei vari parametri come quelli relativi a *group blocks* e *fragments* va al di là di quanto sia possibile fare qui, in breve però si può dire che un filesystem *ext2* divide lo spazio disco in blocchi che a loro volta vengono raggruppati nei *group blocks*, che contengono al loro interno le tabelle degli inode e lo spazio per i blocchi di dati e le informazioni per evitare la frammentazione dello spazio disco, per una spiegazione più completa di può leggere il file `filesystem/ext2.txt` distribuito nella documentazione allegata ai sorgenti del kernel.

⁴¹per alcuni filesystem più evoluti, come Reiser, esistono dei meccanismi automatici che permettono di mettere il contenuto di diversi file di piccole dimensioni in un solo blocco.

sulla base della dimensione del disco e del tipo di uso che si vuole fare del filesystem.⁴² Qualora si usi un valore negativo questo viene preso come limite inferiore per la dimensione di un blocco.

Un altro parametro importante è quello del numero di inode da creare (si ricordi che, come detto in sez. 1.2.2, ogni file è sempre associato ad un inode). Questi di norma vengono mantenuti in una sezione apposita del filesystem, le cui dimensioni vengono determinate in sede di creazione del filesystem.⁴³ Il comando permette, con l'uso dell'opzione `-i` di impostare ogni quanti byte di spazio disco deve essere creato un inode. Si deve dare cioè la valutazione della dimensione media di un file sul disco, specificare un valore troppo alto creerà pochi inode, con il rischio di esaurirli, un valore troppo basso ne creerà troppi, con conseguente spreco di risorse.

Si tenga presente che per la gestione dei file in un filesystem blocchi e inode sono due risorse indipendenti, si possono cioè esaurire sia i blocchi liberi che gli inode disponibili (anche se il primo caso è di gran lunga quello più comune), e trovarsi così nella condizione di non poter più creare nuovi file. Per questo motivo per una gestione ottimale del disco di solito occorre una scelta opportuna sia delle dimensioni dei blocchi che del rapporto fra blocchi disponibili e inode. Tenere le dimensioni dei blocchi basse riduce lo spreco di spazio per quelli occupati parzialmente, ma implica un maggiore lavoro per la gestione (e maggiore spazio per le informazioni relative). Usare pochi inode permette di risparmiare spazio ed essere più veloci nella ricerca, ma si corre il rischio di finirli prima di esaurire lo spazio disco.

Altre caratteristiche aggiuntive di un filesystem *ext2* possono essere impostate attraverso l'opzione `-O`, che prende una lista di parametri, separata da virgole se sono più di uno, che specificano la caratteristiche da abilitare. I valori possibili per i parametri di `-O` sono illustrati in tab. 5.6. Di default vengono attivate le due caratteristiche `sparse_super` e `filetype`, che però non sono compatibili con le implementazioni di *ext2* antecedenti i kernel della serie 2.2, per cui se si deve utilizzare il filesystem con un kernel 2.0 è necessario specificare come parametro il valore `none` che disabilita tutte le caratteristiche aggiuntive.

Parametro	Significato
<code>dir_index</code>	usa una struttura ad alberi ed hash per la gestione del contenuto delle directory.
<code>filetype</code>	memorizza il tipo di file nelle voci delle directory.
<code>has_journal</code>	crea anche il file per il giornale (equivalente all'uso di <code>-j</code>).
<code>journal_dev</code>	richiede che sul dispositivo indicato sia creato un giornale invece di un filesystem.
<code>sparse_super</code>	crea un filesystem con meno copie del <i>superblock</i> per non sprecare spazio sui filesystem molto grandi.

Tabella 5.6: Parametri per l'attivazione delle estensioni dei filesystem *ext2* attivabili mediante l'opzione `-O` di `mke2fs`.

Nel caso si voglia creare un filesystem di tipo *ext3*, si deve utilizzare `-j` che permette di creare un file apposito per il giornale (affronteremo il tema dei filesystem *journalled* in sez. 5.2.4), a questa si abbina di solito `-J` che permette di specificarne delle ulteriori caratteristiche. Qualora non venga utilizzata sono utilizzati i valori di default, altrimenti questi possono essere specificati nella forma `parametro=valore`; un elenco dei parametri specificabili con `-J` è illustrato in tab. 5.7, con relativa spiegazione.

Un problema che si può avere nella gestione dei dischi è quello dei settori difettosi. In genere l'avvenire di errori di questo indica che il disco si sta degradando, ed è bene provvedere ad un backup ed alla sua sostituzione. È comunque possibile, con *ext2* ed *ext3*, creare il filesystem in modo da non utilizzare i blocchi contenenti settori difettosi attraverso l'opzione `-c` che esegue un controllo per verificarne la presenza ed escluderne dall'uso qualora rilevati. Qualora l'opzione

⁴² questo può essere indicato tramite l'opzione `-T`, che prende i tre valori `news`, `largefile` e `largefile4` e assegna un diverso rapporto fra numero di inode e spazio disco disponibile.

⁴³ alcuni filesystem più evoluti permettono di cambiare questo parametro anche in un secondo tempo.

Parametro	Significato
size	Specifica la dimensione del giornale, da specificare in MiB. Deve essere un minimo di 1024 blocchi, e non può superare 102400 blocchi.
device	permette di impostare un dispositivo esterno per mantenere il giornale. Al posto di un file di dispositivo si può specificare una l'etichetta associata al filesystem (con l'opzione -L). Il dispositivo che fa da giornale deve essere inizializzato con il <code>mke2fs -O journal_dev</code> .

Tabella 5.7: Parametri per la gestione del giornale di un filesystem *ext3* specificabili tramite l'opzione -J di `mke2fs`.

venga specificata due volte il controllo viene eseguito in maniera distruttiva (ed estremamente lenta), scrivendo e rileggendo tutto il disco. Specificando l'opzione -l invece di eseguire il controllo direttamente su una lista di questi blocchi letta da un file; di norma questo può essere prodotto indipendentemente dal programma `badblocks` che è quello che viene eseguito anche quando di usa -c per effettuare il controllo. Le altre opzioni principali di `mke2fs` sono riportate nella lista di tab. 5.5, per un elenco si faccia al solito riferimento alla pagina di manuale.

Se lo si usa indipendentemente da `mke2fs` anche `badblocks` vuole come argomento il file di dispositivo da esaminare, ed opzionalmente il blocco di partenza e quello finale del controllo. Si deve aver cura di specificare con l'opzione -b la dimensione dei blocchi, che dovrà ovviamente coincidere con quella usata con `mke2fs`, per cui in genere è sempre meglio non usare direttamente il programma, ma invocarlo tramite `mke2fs` o `e2fsck`.

L'opzione -c permette di selezionare quanti blocchi vengono controllati alla volta (il default è 16), aumentandolo si accresce la velocità di esecuzione del comando, ma anche il suo consumo di memoria. Con l'opzione -i si indica un file contenente una lista nota di blocchi danneggiati (quella presente nel filesystem è ottenibile con `dumpe2fs`), mentre con -o si indica un file su scrivere i risultati.

Infine ci sono una serie di opzioni per indicare il tipo di test da eseguire, normalmente viene eseguita una lettura dei blocchi per verificare errori, se non ce ne sono si prosegue. Con -p che può specificare quante volte ripetere la scansione di un blocco prima di decidere che non ci sono errori. Con -w si richiede di eseguire il controllo scrivendo dei dati e poi rileggendoli per verificarne l'integrità; il metodo è molto più preciso, ma anche molto più lento, inoltre in questo modo il controllo è distruttivo in quanto sovrascrive i dati, pertanto non può essere usato su un filesystem montato. Si può eseguire questo stesso tipo di test in modalità non distruttiva con l'opzione -n, nel qual caso però il controllo diventa *estremamente* lento. Per le rimanenti opzioni del comando si può al solito fare riferimento alla pagina di manuale.

Una volta creato il filesystem, come anche mostrato nell'output di `mke2fs`, si possono modificare alcune caratteristiche con l'uso del comando `tune2fs`. L'uso più comune di `tune2fs` è probabilmente quello per trasformare un filesystem *ext2* in un filesystem *ext3* creando il giornale; questo è ottenibile immediatamente con l'opzione -j, e si può usare -J con le stesse opzioni di tab. 5.7 per indicare le caratteristiche del giornale.

Un'altra opzione di uso comune è -c che permette di impostare il numero di volte che il filesystem può essere montato senza che venga forzato un controllo di consistenza con `fsck`, analoga a questa è -i che specifica l'intervallo massimo, in giorni, mesi o settimane, fra due controlli. Con -C si può anche modificare a mano il contatore del numero di volte che un filesystem è stato montato, in modo da poter forzare il controllo ad un successivo riavvio.

Con l'opzione -O si possono modificare in un secondo tempo le caratteristiche avanzate specificando come parametro una lista dei valori già illustrati in tab. 5.6, in questo caso per disabilitare una funzionalità si può apporre un carattere "~" al corrispondente parametro. Si tenga presente che se si modificano i parametri `sparse_super` o `filetype` sarà poi necessario

eseguire una riparazione del filesystem con **e2fsck**.

Opzione	Significato
-c	Imposta il numero di volte che un filesystem può essere rimontato prima di subire un controllo; prende come parametro un numero.
-C	Modifica il contatore del numero di volte che un filesystem è stato montato; prende come parametro un numero.
-e	Modifica il comportamento del kernel quando viene rilevato un errore sul filesystem, prende come parametro uno dei valori: continue (continua ignorando l'errore), remount-ro (rimonta il filesystem in sola lettura), panic (si ferma causando un <i>kernel panic</i>).
-i	Imposta l'intervallo di tempo fra due controlli successivi; prende come parametro un numero di giorni, di settimane se si pospone il carattere "w", di mesi se si pospone il carattere "m".
-j	aggiunge un giornale per il filesystem.
-J	sovrascrive i parametri del giornale, prende come parametro uno dei valori di tab. 5.7.
-l	stampa i contenuti del <i>superblock</i> (come dumpe2fs).
-L	Imposta il nome del volume, per un massimo di 16 caratteri. È questo nome (detto anche <i>label</i>) che può essere usato da vari comandi per indicare il filesystem al posto del dispositivo.
-m	Imposta la percentuale di blocchi riservati (messi a disposizione dell'utente specificato con -u, di norma l'amministratore).
-O	Imposta le caratteristiche avanzate del filesystem, prende come parametri uno dei valori di tab. 5.6.
-r	Imposta il numero di blocchi riservati.
-u	Imposta l'utente che può usare i blocchi riservati.
-g	Imposta il gruppo che può usare i blocchi riservati.
-U	Imposta l' <i>universally unique identifier</i> (UUID) del filesystem, un numero (espresso in cifre esadecimali) che identifica il filesystem (in maniera analoga alla <i>label</i>).

Tabella 5.8: Principali opzioni per il comando **tune2fs**.

Infine si può usare l'opzione **-L** per impostare una etichetta sul filesystem (il nome del volume), con essa diventa possibile fare riferimento in **/etc/fstab** a questo nome invece che al file di dispositivo per indicare il filesystem. Le altre opzioni principali sono illustrate in tab. 5.8, al solito se ne può ottenere l'elenco completo ed una descrizione dettagliata nella pagina di manuale del comando.

5.2.4 Controllo e riparazione di un filesystem

Benché il sistema sia molto stabile e problemi di questo tipo siano piuttosto rari, vengono forniti anche una serie di comandi diagnostici per il controllo e la riparazione di un filesystem che possa essersi danneggiato. In questa sezione tratteremo in dettaglio quelli disponibili per *ext2* (ed *ext3*), anche se, in particolare per la riparazione, esistono programmi analoghi per qualunque filesystem.

Come già accennato in sez. 5.2.3 il comando **dumpe2fs** può essere usato per recuperare una serie di informazioni riguardo ad un filesystem *ext2*. Se invocato senza opzioni il comando stampa una lunga lista di informazioni ricavate dal contenuto del *superblock* e dei *group block*, come esempio riportiamo solo la prima parte dell'output del comando, quella relativa alle sole informazioni del *superblock*, ottenibili specificando l'opzione **-h**:

```

anarres:~# dumpe2fs -h /dev/hda4
dumpe2fs 1.35-WIP (07-Dec-2003)
Filesystem volume name:   <none>
Last mounted on:         <not available>
Filesystem UUID:         d0cb4773-dbf8-4898-94e8-bb2acc41df0d
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal filetype needs_recovery sparse_super
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              4169760
Block count:              8324194
Reserved block count:     416209
Free blocks:              5365536
Free inodes:              3935324
First block:              0
Block size:               4096
Fragment size:            4096
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         16352
Inode blocks per group:   511
Last mount time:          Fri Jan 30 20:59:45 2004
Last write time:          Fri Jan 30 20:59:45 2004
Mount count:              16
Maximum mount count:      22
Last checked:             Sun Dec 21 14:45:22 2003
Check interval:           15552000 (6 months)
Next check after:         Fri Jun 18 15:45:22 2004
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               128
Journal inode:            13
First orphan inode:       2421098
Journal backup:           inode blocks

```

Le altre opzioni principali del comando sono **-b**, che restituisce l'elenco dei blocchi marchiati come danneggiati in un filesystem, **-ob** che permette di specificare il blocco (indicato per numero) da usare come *superblock* al posto di quello standard (in caso di corruzione di quest'ultimo), **-oB** per indicare la dimensione dei blocchi (anche questa è una informazione necessaria solo in caso di corruzione del filesystem). Infine **-i** permette di leggere le informazioni invece che dal filesystem da una immagine create con il comando **e2image**. Per un elenco completo e relative spiegazioni si faccia al solito riferimento alla pagina di manuale.

Una delle misure di precauzione che si possono prendere per tentare un recupero in caso di corruzione di un filesystem *ext2* è quella di crearne una immagine con il comando **e2image**. Questo comando permette di salvare su un file i dati critici del filesystem in modo da poterli riutilizzare con programmi di riparazione come **e2fsck** o **debugfs**. Il comando prende come argomenti il dispositivo su cui si trova il filesystem ed il nome del file su cui salvare l'immagine. L'unica opzione è **-r** che crea una immagine binaria che può essere usata dai vari programmi di controllo come se fosse l'intero filesystem.⁴⁴

In generale il funzionamento dei filesystem in Linux è estremamente stabile, ed una volta

⁴⁴ per far questo viene creato uno *sparse file* delle stesse dimensioni del filesystem; uno *sparse file* è un file in cui non sono state scritte le parti vuote, pertanto anche se la sua dimensione può essere enorme, pari appunto ad un intero filesystem, in realtà viene occupato su disco solo lo spazio relativo alle parti non vuote.

creati è praticamente impossibile⁴⁵ danneggiarli nel corso delle normali operazioni. Se però si ha un black-out improvviso, o qualcuno inciampa nel cavo di alimentazione del server, è normale che il filesystem, dal momento in cui l'aggiornamento dei dati su disco è stato interrotto brutalmente, si possa trovare in uno stato incoerente.

In questo caso si può avere un danneggiamento della struttura del filesystem, che deve essere riparato. In genere ogni filesystem prevede l'esistenza di un flag su disco, attivato quando viene montato, che indica che è in uso, e che viene azzerato solo quando il filesystem viene smontato (nel qual caso si è certi che tutte le operazioni sospese sono state completate e lo stato è coerente). Se c'è stata un'interruzione della corrente questo flag resterà attivo ed il sistema potrà rilevare, al successivo tentativo di montaggio, che qualcosa è andato storto.

Quello che può succedere in questi casi dipende dal filesystem. Coi filesystem tradizionali `mount` rileva l'errore e non monta il filesystem o lo monta in sola lettura (a seconda delle opzioni scelte). A questo punto occorre usare l'opportuno programma di controllo per verificare lo stato del filesystem ed eventualmente riparare gli errori. Di norma, in caso di rilevamento di un errore durante la procedura di avvio del sistema, questo viene lanciato automaticamente.

La procedura di controllo e riparazione può essere molto lunga e complessa, specie per filesystem di grandi dimensioni, in quanto prevede una serie di verifiche dettagliate per identificare informazioni incoerenti e parziali, che comportano varie scansioni del contenuto del filesystem stesso. Questo può significare dei tempi che possono diventare ore o addirittura giorni per i dischi più grandi. Per ovviare a questo inconveniente alcuni filesystem più avanzati supportano il cosiddetto *journalling*, un sistema che permette di riportare il filesystem in uno stato coerente con grande velocità.

Il concetto fondamentale del *journalling* è che le operazioni sul filesystem vengono prima registrate su un *giornale*, (un file apposito a questo dedicato) e poi riportate sul filesystem. Così se si ha una interruzione improvvisa si hanno due casi: se l'interruzione è avvenuta durante l'aggiornamento del filesystem, il giornale sarà intatto e lo si potrà utilizzare per completare l'aggiornamento interrotto. Se invece l'interruzione è avvenuta durante la scrittura nel giornale sarà questo ad essere scartato; si perderanno così le ultime modifiche, ma il filesystem resterà in uno stato coerente.

In questo modo quando si ha un *filesystem journalled* si può evitare il lungo procedimento di riparazione. In realtà in questo non è che il procedimento non avvenga, solo che grazie alla presenza del giornale questo viene eseguito con grande rapidità, non dovendo effettuare il controllo completo del filesystem. In generale però, specie se si usa *ext3*, è opportuno mantenere un controllo periodico sul filesystem, in quanto errori sul disco, cavi difettosi o bug del kernel potrebbero comunque aver corrotto le informazioni.

Il programma generico per il controllo di un filesystem è `fsck` (da *File System ChecK*) che, come `mkfs`, non è altro che un front-end per i singoli programmi specifici di ciascun tipo di filesystem. Questi vengono attivati attraverso l'opzione `-t` seguita dal nome del filesystem. Il programma prende come argomenti un elenco di filesystem da controllare specificati per dispositivo (o *mount point*, se sono citati in `/etc/fstab`). Se si specifica più di un filesystem la lista dei relativi tipi, separata da virgole, deve essere specificata come parametro per `-t`.

Quando viene invocato con l'opzione `-A` (di solito questo viene fatto nella procedura di avvio) il comando esegue una scansione di `/etc/fstab` e cerca di controllare ogni filesystem ivi elencato, a meno che il sesto campo del file non sia impostato a zero (si ricordi quanto detto in sez. 1.2.4). Il comando prima proverà ad eseguire il controllo per il filesystem radice e passerà poi a tutti gli altri in ordine di valore crescente del suddetto campo.

Come per `mkfs` le opzioni disponibili dipendono dallo specifico filesystem, di solito sono definite due opzioni generiche, `-a` che cerca di eseguire le riparazioni in modo automatico, senza chiedere l'intervento dell'amministratore, e `-r` che invece esegue le riparazioni in modalità inte-

⁴⁵ a meno di non usare kernel sperimentali; in questo caso si stanno cercando rogne, ed è anche possibile trovarle.

rattiva. Per le altre opzioni si può fare riferimento alle pagine di manuale dei vari programmi dedicati di ciascun filesystem.

Nel caso di Linux tutti i filesystem più recenti (ReiserFS, JFS e XFS) supportano nativamente il *journaling*; ed anche il tradizionale *ext2* ha ottenuto questa funzionalità con la nuova versione *ext3*. L'uso di un giornale ha però un impatto sulle prestazioni del filesystem, e non è detto che sia sempre utilizzato. Questo comporta che la diffusione di *ext2* è ancora molto ampia, e continua ad avere senso per i filesystem che possono essere utilizzati in sola lettura (come la radice e */usr*).

Inoltre partendo da un filesystem *ext2* la conversione ad *ext3* è molto semplice, basta aggiungere il giornale al filesystem con `tune2fs -j` e rimontarlo come *ext3*. Per questo motivo si possono sostanzialmente considerare *ext2* ed *ext3* (che nella gestione dei dati è identico) come il filesystem più diffuso. Per questo motivo ci concentreremo sulla versione dei programmi di riparazione e controllo specifici di questo filesystem.

Il programma di controllo e riparazione del filesystem *ext2* è **e2fsck**, che prende come argomento il dispositivo da controllare. Il comando supporta le opzioni generiche **-a** e **-r** illustrate in precedenza, che sono state mantenute solo per compatibilità, in realtà **-a** è deprecata in favore della più recente **-p** mentre **-r** non fa niente dato che il comando viene sempre eseguito in modalità interattiva. È comunque possibile usare l'opzione **-y** per ottenere per far rispondere “yes” automaticamente a tutte le domande in modo da poter eseguire il comando in modalità non interattiva (ad esempio da uno script), mentre con **-n** si fa la stessa cosa aprendo il filesystem in sola lettura,⁴⁶ e dando una risposta di “no” a tutte le domande.

In caso di filesystem pesantemente corrotto si possono poi specificare il *superblock* da utilizzare con **-b** e la dimensione di un blocco con **-B**. Se si è usato un giornale posto su un altro dispositivo questo deve essere specificato con l'opzione **-j**. Quando si è usata l'opzione **-b** e il filesystem non è stato aperto in sola lettura **e2fsck** si cura anche di ripristinare anche il superblock al completamento del controllo.

Usando l'opzione **-c** si può richiedere anche la scansione per il rilevamento di settori difettosi, e specificandola due volte viene usato il metodo non distruttivo di scansione con scrittura e riletture. Con l'opzione **-l** si può specificare un file con una lista da aggiungere a quella dei blocchi difettosi, mentre con **-L** il file indica la nuova lista di blocchi difettosi.

Le altre opzioni principali del comando sono riportate in tab. 5.9, per un elenco completo e la relativa documentazione si può al solito fare riferimento alla pagina di manuale del comando.

Opzione	Significato
-b	Specifica un superblocco alternativo.
-B	Specifica le dimensioni di un blocco.
-c	Esegue il controllo del dispositivo per eventuali settori difettosi.
-f	forza il controllo del filesystem.
-j	Specifica il dispositivo di un giornale esterno.
-l	Specifica il file con una lista di blocchi difettosi da aggiungere a quelli già presenti.
-L	Specifica il file con la nuova lista dei blocchi difettosi (sovrascrivendo quella presente).
-n	Esegue il controllo rispondendo automaticamente <i>no</i> a tutte le domande.
-t	Stampa delle statistiche di esecuzione.
-y	Esegue il controllo rispondendo automaticamente <i>yes</i> a tutte le domande.

Tabella 5.9: Principali opzioni per il comando **e2fsck**.

In genere non c'è necessità di eseguire direttamente **e2fsck**, in quanto di norma questo viene eseguito dagli script di avvio. Nel caso lo si voglia eseguire comunque, oltre a specificare l'opzione **-f** occorre assicurarsi che il filesystem relativo non sia montato, o sia montato in sola lettura. Di norma l'esecuzione automatica (quella ottenuta con **-a** o **-p**) prevede che il filesystem sia

⁴⁶eccetto il caso in cui si siano specificate le opzioni **-c**, **-l** o **-L** nel qual caso il filesystem viene aperto in scrittura per aggiornare la lista dei settori difettosi.

riparato senza necessità di intervento da parte dell'utente. Ci sono casi comunque in cui questo non è possibile, nel qual caso il programma si ferma.

Quando questo avviene durante la procedura di avvio di norma il sistema viene mandato in *Single User Mode* (si veda sez. 5.3.4) e viene richiesto di rieseguire il programma manualmente. In tal caso di norma il filesystem non è montato, a meno che il filesystem danneggiato non sia la radice, nel qual caso esso deve essere comunque montato,⁴⁷ ma in sola lettura.

Qualora anche la riparazione con **e2fsck** eseguito manualmente fallisca ci si trova di fronte ad un filesystem pesantemente danneggiato. In questo caso l'ultima risorsa è quella di utilizzare **debugfs** per provare ad eseguire manualmente la riparazione. Il comando permette di aprire anche un filesystem danneggiato, e di eseguire su di esso delle operazioni. Non è detto che si riesca a riparare completamente il filesystem⁴⁸, ma si possono tentare delle operazioni di ripulitura che potrebbero portare lo stesso in uno stato riparabile da **e2fsck**.

Il comando prende come argomento il dispositivo contenente il filesystem da esaminare; in caso di filesystem pesantemente danneggiato l'opzione **-b** permette di specificare una dimensione dei blocchi, l'opzione **-s** permette di specificare la *superblock*, ed infine l'opzione **-i** permette di specificare una immagine creata con **e2image**.

Infine si tenga presente che per Linux non esistono programmi di uso comune per la *deframmentazione* del disco.⁴⁹ Questa infatti è un problema solo per quei filesystem la cui architettura è talmente inadeguata da renderlo tale. In generale un qualunque filesystem unix-like è in grado di gestire la allocazione dello spazio disco in maniera da evitare il sorgere del problema fin dall'inizio.

5.2.5 La gestione della *swap*

Come ultimo argomento relativo alla gestione dei dischi affronteremo quello della gestione dello spazio dedicato alla *swap*. Questa costituisce un complemento alla gestione della memoria virtuale, il meccanismo con cui il kernel controlla l'allocazione della memoria fisica ai vari processi.

Come accennato in sez. 1.1 e poi approfondito in sez. 1.3.1 il kernel gestisce, con l'aiuto della MMU (*Memory Management Unit*) del processore una rimappatura dello spazio degli indirizzi dei processi sulla memoria fisica effettivamente disponibile. In questo modo ciascun processo può usare un suo spazio di indirizzi virtuale (usando un qualunque valore fa quelli possibili), e tramite la MMU accederà ad un indirizzo reale della memoria effettivamente disponibile.

Una delle caratteristiche del meccanismo è che in questo modo ciascun processo mantiene un suo spazio di indirizzi separato; se un processo usa un indirizzo che non corrisponde a nessuna pagina di memoria si avrà quello che si chiama un *segmentation fault* ed il sistema si accorgerà immediatamente dell'errore inviando al processo un segnale di **SIGSEGV**. Ma la potenza del meccanismo della memoria virtuale consiste nel fatto che esso consente di utilizzare anche dello spazio disco come supporto aggiuntivo rispetto alla memoria fisica, quando l'uso di questa diventa eccessivo.

È questo lo scopo della cosiddetta *area di swap*, una sezione di disco (in genere una partizione, ma si può usare anche un file) su cui il kernel può salvare le pagine di memoria meno utilizzate di un processo, in modo che altri possano utilizzare la memoria fisica che così viene liberata. Chiaramente quando si andrà a rieseguire il processo le cui pagine di memoria sono state salvate

⁴⁷chiaramente se il filesystem è danneggiato così gravemente da non poter neanche essere montato si avrà un *kernel panic*.

⁴⁸per questo occorre una conoscenza dettagliata della struttura di un filesystem *ext2*, in effetti il programma, come dice il nome, non è uno strumento per il controllo quanto per il debugging, ad uso dei cosiddetti *filesystem guru*.

⁴⁹in effetti ci sono alcuni programmi per eseguire questa operazione, ma normalmente non vengono usati, proprio perché assai poco utili.

sull'area di swap e questo cercherà di accedere a quelle, si avrà quello che si chiama un *page fault*⁵⁰ a questo punto il sistema della memoria virtuale provvederà a recuperare la pagina dall'area di swap e a rimetterla in memoria, così che il processo possa proseguire la sua esecuzione come se nulla fosse.⁵¹

Perché il kernel possa usare un'area di swap questa deve essere opportunamente inizializzata, come se fosse un filesystem;⁵² a questo provvede il comando **mkswap** che prende come argomento il file da usare come area di swap. In genere si usa il comando specificando come argomento un file di dispositivo indicante una partizione appositamente riservata (vedi sez. 5.2.2), in questo caso infatti l'accesso al disco è diretto (è quindi più veloce) rispetto al caso, comunque possibile, in cui si usa un file normale.⁵³

Dato che le prestazioni del sistema della memoria virtuale dipendono direttamente, in caso di utilizzo dell'area di swap, dai tempi di accesso a quest'ultima e dalla velocità di trasferimento dei dati, è in genere una pessima idea usare un disco vecchio e lento per questo scopo. È per questo motivo inoltre che a volte si suggerisce di usare per l'area di swap l'ultima partizione disponibile, considerata come quella che contiene i settori più esterni del disco, dove la velocità di trasferimento è maggiore.

Il comando prevede (per compatibilità con le vecchie versioni) un secondo argomento che specifica la dimensione dell'area di swap; se non lo si specifica viene automaticamente utilizzato tutto lo spazio disponibile. Come per **mkfs** l'opzione **-c** richiede un controllo della partizione per la presenza di settori difettosi. L'opzione **-p** permette di specificare come parametro la dimensione delle pagine di memoria (in genere 4096 byte, ma dipende dall'architettura hardware), l'elenco delle opzioni è riportato in tab. 5.10.

Opzione	Significato
-c	esegue un controllo per la presenza di settori difettosi.
-f	forza l'esecuzione del comando anche se si sono dati argomenti sbagliati (come un dimensione dell'area maggiore di quella della partizione).
-p	specifica, con il valore passato come parametro, la dimensione delle pagine di memoria.
-v	specifica la versione della formattazione dell'area di swap, con -v0 usa il vecchio stile (deprecato), con il -v1 usa il nuovo.

Tabella 5.10: Opzioni per il comando **mkswap**.

Si tenga presente che per i kernel precedenti il 2.4.10 erano possibili fino ad un massimo di 8 aree di swap, a partire da esso sono state portate a 32; un elenco di quelle attive è visibile in **/proc/swaps**. La dimensione massima dell'area di swap dipende sia dalla versione utilizzata che dall'architettura hardware. Con la vecchia versione il massimo dipendeva solo dalla dimensione delle pagine (ed era di 128MiB con le pagine di 4096 byte della architettura standard dei PC) con la nuova versione dipende solo dall'architettura ed è di 2GiB per la maggior parte di esse.

Una volta creata un'area di swap questa non verrà utilizzata dal kernel fintanto che non la si attiva con il comando **swapon**; questo prende come argomento il file da usare come area di swap, che verrà immediatamente attivata. In generale però il comando non viene mai invocato direttamente, ma chiamato con l'opzione **-a** negli script di avvio, ed in questo caso attiverà tutti i dispositivi che sono marcati come **swap** all'interno di **/etc/fstab**. Se invece si vuole avere

⁵⁰quello che succede è che quando la MMU si accorge che la pagina richiesta non è mappata su della memoria fisica invia un segnale al processore, così che questo possa eseguire la opportuna sezione del kernel, quest'ultimo si occuperà o di recuperare la pagina dalla *swap* (se l'indirizzo era giusto) o di inviare un segnale di **SIGSEGV** (se l'accesso era sbagliato).

⁵¹in realtà ovviamente tutto questo richiede tempi che sono ordini di grandezza superiori rispetto all'accesso diretto alla RAM, per cui il nulla è solo teorico, in pratica il programma sarà molto più lento.

⁵²dato che l'area deve essere solo utilizzata per copiarci pagine in questo caso non sono affatto necessarie tutte le infrastrutture mostrate in sez. 1.2.2.

⁵³per il quale si dovrebbe prima passare attraverso il relativo filesystem.

una lista dei dispositivi attualmente utilizzati si potrà usare l'opzione `-s`, ottenendo qualcosa del tipo:

```
monk:/home/piccardi/truedoc/corso# swapon -s
Filename                                Type              Size    Used    Priority
/dev/hda2                              partition         498004  1936    -1
```

Un'altra opzione importante è `-p`, che permette di impostare una priorità (un parametro di valore fra 0 e 32767) per l'uso della partizione. Quando si ha una sola area questo valore non è significativo in quanto il kernel si limiterà ad usare la prima sezione libera che trova, se le aree sono più di una l'algoritmo di utilizzo prevede che il kernel usi a turno quelle della stessa priorità (realizzando così una forma di RAID-1, vedi sez. 6.1.1) e passi ad una di priorità inferiore solo quando quelle di priorità superiore sono piene.

Se non si specifica nulla le priorità vengono assegnate automaticamente andando a diminuire nell'ordine di attivazione (così che la prima area attivata è quella a priorità maggiore), usando `-p` si può forzare la stessa priorità ottenendo un uso più efficace e maggiore velocità. Si può ottenere questo risultato anche per le aree di swap attivate con `swapon -a` specificando in `/etc/fstab` nel relativo campo l'opzione `pri=N` dove `N` è lo stesso valore che si userebbe con `-p`.

Infine si può disattivare un'area di swap usando il comando `swapoff`, anche in questo caso occorre fornire come argomento il file di dispositivo indicante l'area da disabilitare; il comando consente anche l'uso dell'opzione `-a` nel qual caso non sarà necessario passare argomenti e saranno disattivate tutte le aree di swap presenti, così come indicate in `/proc/swaps` e `/etc/fstab`.

5.3 La gestione dell'avvio del sistema

In questa sezione prenderemo in esame la procedura di avvio del sistema, dettagliando i vari passaggi che portano dall'accensione della macchina al pieno funzionamento del sistema, e come si possa intervenire (trattando i vari programmi coinvolti ed i relativi meccanismi di configurazione) a ciascuno stadio di questa procedura.

5.3.1 L'avvio del kernel

Una volta accesa la macchina è il BIOS che si incarica di lanciare il sistema operativo. Le modalità possono dipendere da tipo e versione di BIOS, ed in genere le versioni più moderne permettono di avviare il sistema da una varietà di supporti (ivi compreso l'avvio via rete). In genere comunque tutti i BIOS supportano l'avvio da floppy, disco e CDROM (anche se alcuni dei più vecchi non supportano il CDROM).

Per quanto riguarda l'avvio da CDROM e floppy non esiste nessuna configurazione specifica per Linux, occorre semplicemente impostare opportunamente il BIOS perché vengano usati tali dispositivi. In genere questo si fa tramite l'apposito menù di configurazione che permette di selezionare la sequenza in cui vengono controllati i vari dispositivi per l'avvio. Il primo su cui si trovano i dati opportuni verrà utilizzato.

Il caso più comune è quello in cui l'avvio viene fatto dal disco rigido, ed è su questo che ci soffermeremo con maggior dettaglio nei paragrafi successivi. La procedura prevede che il BIOS legga il primo settore del disco⁵⁴ scelto per l'avvio⁵⁵ che viene detto *Master Boot Record*, o MBR. Questo contiene un programma apposito, chiamato *bootloader*, che si incarica di effettuare il lancio del sistema operativo.

⁵⁴si sottintende disco IDE, se si vuole usare un disco SCSI occorre comunque impostare il BIOS per l'avvio su SCSI, ed avere un controller SCSI che supporta il boot.

⁵⁵per molti BIOS questo può essere solo il primo disco IDE, anche se BIOS più recenti permettono di usare anche altri dischi.

In generale il *bootloader* è un programma elementare il cui unico compito è quello di trovare sul disco il file che contiene il sistema operativo, caricarlo in memoria ed eseguirlo, passandogli eventuali parametri di avvio. Una delle caratteristiche di Linux infatti è che, essendo in fin dei conti anche questo un programma, può prendere degli argomenti, solo che in questo caso non esiste una riga di comando da cui darglieli, se non quella che può mettere a disposizione il *bootloader*⁵⁶ prima di eseguirlo.

Normalmente non c'è necessità di passare argomenti specifici al kernel, in generale questi servono a passare valori ad alcuni sottosistemi⁵⁷ o ad impostare delle funzionalità usate all'avvio; essi sono sempre nella forma **chiave=valore**, dove **valore** può essere un valore numerico (espresso in notazione decimale, ottale o esadecimale) o una stringa, o una lista di stringe separate da virgole. Un elenco dei principali argomenti è riportato in tab. 5.11, un elenco dettagliato può essere trovato nel file **kernel-parameters.txt** della documentazione allegata ai sorgenti del kernel⁵⁸ o nel *boot-prompt-HOWTO*.

Argomento	Significato
vga=mode	dove mode è un valore intero che stabilisce la <i>modalità video</i> in cui utilizzare la console, o il valore ask per fare eseguire una scansione.
mem=size	imposta la dimensione della memoria da utilizzare, veniva utilizzato per sopprassedere il malfunzionamento del sistema di autorilevamento del kernel.
initrd=file	specifica un <i>RAM disk</i> iniziale (che verrà usato come radice provvisoria), è compito del <i>bootloader</i> interpretare il significato di file .
root=dev	specifica (in genere col nome del file di dispositivo) il disco (o la partizione) da montare come radice.
init=command	specifica il programma da lanciare come primo processo del sistema (di default è /sbin/init).
console=dev	specifica un file di dispositivo da usare come console per i messaggi di errore.

Tabella 5.11: Principali argomenti per la linea di comando del kernel.

Una delle caratteristiche dei *bootloader* (tratteremo più avanti i due più comuni) è che essi, oltre che nel *Master Boot Record*, possono essere installati anche nel settore iniziale di ciascuna partizione. Questo permette allora di concatenare più *bootloader* (in particolare questa è la tecnica usata normalmente per lanciare il *bootloader* di Windows) in modo che il primo lanci il successivo, e così via.

Infine si tenga conto che tutto questo si applica solo ai normali PC. Linux però gira su moltissime architetture hardware diverse, come Alpha, Sparc, HP-UX, PowerPC, ecc. In tal caso la procedura di avvio è spesso gestita direttamente dall'equivalente del BIOS per quell'architettura, come SILO per la Sparc, MILO per le Alpha e OpenFirmware per i Mac, ognuno dei quali ha un suo diverso meccanismo di funzionamento; non ci occuperemo di questi casi.

Una volta avviato il kernel effettuerà una serie di operazioni preliminari, come le inizializzazioni delle infrastrutture generiche, e dei dispositivi il cui supporto è stato compilato direttamente nel kernel. Una volta completata l'inizializzazione dell'hardware tutto quello che resta da fare è semplicemente montare il dispositivo su cui si trova la directory radice e lanciare il programma di avvio del sistema, che di norma è **/sbin/init**. Le operazioni di avvio, per quanto riguardo il kernel, si concludono con il lancio di **init**, tutto il resto verrà eseguito da quest'ultimo in user space.

⁵⁶che in questo caso svolge il ruolo della shell, e deve anche essere in grado, per alcuni argomenti, di fornire una opportuna interpretazione.

⁵⁷ad esempio i moduli che supportano l'uso di parametri in fase di caricamento (vedi sez. 5.1.4), qualora compilati all'interno del kernel possono ricevere in questo modo i valori dei loro parametri.

⁵⁸nella directory *Documentation*.

La procedura con cui il BIOS carica in memoria il sistema dipende ovviamente dal supporto da cui lo prende, e questo deve ovviamente contenere i dati in un formato adeguato. Il caso del floppy è analogo a quello di un disco, il BIOS legge il primo settore del floppy ed esegue il programma che c'è contenuto.

Il vantaggio di questo metodo è che se si crea una immagine compressa del kernel (con la procedura vista in sez. 5.1.3) viene automaticamente inserito nei primi 512 byte della stessa un programma di avvio che si limita a caricare il resto del contenuto del floppy in memoria, scompattarlo ed eseguirlo. Pertanto basta l'uso di `dd` per copiare direttamente l'immagine del kernel su un dischetto con un comando del tipo:

```
dd if=bzImage of=/dev/fd0
```

per ottenere un floppy di avvio.

Il problema con un disco d'avvio fatto in questo modo è che non si possono fornire argomenti all'avvio. In particolare questo significa che il sistema userà come radice quella in uso quando viene compilata l'immagine. In realtà alcuni argomenti di avvio in questo caso possono essere modificati con il comando `rdev`.

Se usato senza argomenti il comando si limita a stampare una riga in riporta quale è la partizione impostata come directory radice per il sistema corrente. Se si specifica un solo argomento questo deve essere un file contenente l'immagine di un kernel e allora stamperà la radice predisposta per quel kernel; se si specifica un secondo argomento questo dovrà essere il dispositivo contenente la nuova directory radice che andrà a sovrascrivere la precedente.

Il comando permette anche di modificare anche altri due parametri che sono mantenuti nell'immagine del kernel, la *modalità video* usando l'opzione `-v`, o invocandolo come `vidmode`, e la dimensione del *RAM disk* iniziale, usando l'opzione `-r` o invocandolo come `ramsize`. Per i dettagli si faccia riferimento alla pagina di manuale.

Nel caso di un CDROM il meccanismo di avvio è sostanzialmente lo stesso dei floppy, posto che il CDROM sia stato masterizzato con le opportune estensioni. In sostanza viene inserita nel CD l'immagine di un floppy e l'avvio viene eseguito alla stessa maniera. In questo caso però di solito non viene usata una immagine di un floppy contenente soltanto il kernel, ma viene utilizzato il programma `syslinux` per creare una immagine di avvio contenuta in un filesystem FAT.

In tal caso basterà copiare i file necessari (come l'immagine del kernel o del *RAM disk* iniziale) sul floppy⁵⁹ di destinazione e poi eseguire il programma, che prende come argomento il file del dispositivo da inizializzare (in genere `/dev/fd0`). Il comando sovrascriverà opportunamente il settore iniziale del disco e ci copierà un file `LDLINUX.SYS` che contiene il programma di avvio. Questo mette a disposizione una riga di comando analoga a quella di LILO (vedi sez.5.3.2), ed è controllato dal file di configurazione `SYSLINUX.CFG` che basta inserire nella radice del floppy. Per i dettagli si può fare riferimento alla pagina di manuale di `syslinux` al solito accessibile con `man syslinux`.

5.3.2 L'uso di *LILO*

Il primo *bootloader* creato per Linux è stato LILO, da *Linux LOader*, ed a lungo è stato anche l'unico presente. È un bootloader con una architettura elementare, che si affida al BIOS per la lettura del disco, e pertanto risente di tutti i limiti che questo può avere. Uno di questi problemi è che, a causa delle restrizioni ereditate dai primi PC illustrate in sez. 5.2.1, alcuni vecchi BIOS non sono capaci di leggere i dischi oltre il 1024-simo cilindro.

In genere LILO viene installato nell'MBR alla fine della procedura di installazione di una distribuzione. In generale è possibile, come accennato in precedenza, installarlo all'interno di

⁵⁹o su qualunque altro dispositivo si stia utilizzando; il vantaggio di utilizzare un filesystem consente, una volta che lo si è montato, di accedere direttamente ai contenuti.

una partizione, se si dispone di un altro *bootloader* in grado di eseguirlo. Una volta eseguito (sia direttamente dal BIOS, che da un altro *bootloader*) LILO si incarica di trovare l'immagine del kernel sul disco, (in sostanza in fase di installazione viene memorizzata la posizione su disco cui essa si trova) caricarla in memoria, passare tutti gli eventuali parametri di avvio specificati in fase di configurazione, e poi eseguirla.⁶⁰ Da questo punto in poi il controllo passa al kernel, che si incarica di tutto il resto.

La configurazione di LILO è gestita tramite il file `/etc/lilo.conf`, questo contiene sia le opzioni passate al kernel in fase di avvio, che le direttive che permettono di controllare direttamente il comportamento del *bootloader*. Si tenga presente però che, al contrario di quanto avviene in genere per altri file di configurazione, non basta modificarlo perché i cambiamenti diventino effettivi al riavvio successivo. Si deve invece far girare il programma `lilo` che reinstalla il *bootloader* con le nuove opzioni.

In genere si ha a che fare con questo file tutte le volte che si vuole usare un nuovo kernel. Infatti per poter lanciare un kernel occorre specificarne la posizione su disco al *bootloader* in modo che questo possa caricarlo. La cosa va fatta anche se si è semplicemente sovrascritto un kernel precedente con un altro (e pertanto non si è neanche dovuto modificare `/etc/lilo.conf`); si ricordi infatti che LILO (il *bootloader*, quello che sta nell'MBR) conosce solo la posizione fisica del kernel nel disco, non quella logica nel filesystem; pertanto se si sovrascrive un file non è affatto detto che la posizione fisica del nuovo file sia la stessa (anzi di norma non lo è affatto) per cui al successivo riavvio senz'altro LILO non potrebbe non essere più in grado di trovare il kernel, con la conseguente impossibilità di lanciare il sistema.

Il formato di `/etc/lilo.conf` è simile a quello degli altri file di configurazione, le linee vuote o che iniziano per `#` vengono ignorate. Le altre linee indicano una opzione o una direttiva. Un esempio di questo file è il seguente:

```
# 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
# Installs the specified file as the new boot sector
#
install=/boot/boot.b
# Specifies the location of the map file
#
map=/boot/map
# Specifies the number of deciseconds (0.1 seconds) LILO should
# wait before booting the first image.
#
delay=30
# message=/boot/bootmess.txt
#       prompt
#       single-key
#       delay=100
#       timeout=30
# Kernel command line options that apply to all installed images go
# here. See: The 'boot-prompt-HOWTO' and 'kernel-parameters.txt' in
# the Linux kernel 'Documentation' directory.
```

⁶⁰qualora lo si sia richiesto, si può caricare in memoria anche l'immagine di un ram-disk, che serve come filesystem iniziale per il kernel.

```

#
# append=""
# Boot up Linux by default.
#
default=linux
image=/boot/vmlinuz-2.2.21
    label=linux
    read-only
    optional
image=/boot/vmlinuz-2.2.20
    label=linuxold
    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 direttive sono divise in tre classi principali: le direttive che specificano opzioni globali, che si applicano al comportamento generale del *bootloader*, le direttive che specificano opzioni relative alle singole immagini dei sistemi che si vogliono lanciare, e le direttive che permettono di passare delle opzioni al kernel (Linux).

La maggior parte delle direttive ricade nella prima classe; di queste forse la più importante è *boot*, che specifica il dispositivo su cui installare il *bootloader*, nel caso è l'MBR del primo disco IDE, indicato tramite il suo file di dispositivo come */dev/hda*. Si può specificare allo stesso modo una partizione, ad esempio se si volesse installare LILO sulla seconda partizione del primo disco si sarebbe potuto usare */dev/hda2*.

La direttiva *map* specifica il file (di norma *boot.map*) che contiene la mappa della posizione su disco delle varie immagini del kernel; questa viene definita automaticamente alla installazione del pacchetto e ricostruita tutte le volte che si lancia il comando *lilo*. La direttiva *install* invece specifica qual'è il file che contiene il programma del *bootloader*, ne possono esistere varie versioni, che presentano una interfaccia di avvio semplificata a menù o semplicemente testuale. In genere tutti questi i file referenziati da queste direttive vengono tenuti, come illustrato in sez. 1.2.3, nella directory */boot*.

Le altre direttive globali più usate sono quelle che controllano le modalità di avvio. Se si specifica *prompt* il *bootloader* si ferma all'avvio presentando una riga di comando da cui l'utente può immettere dei comandi (si tratta in genere di scrivere il nome di uno dei kernel predefiniti, seguito dalle eventuali opzioni che gli si vogliono passare). Specificando un numero con *timeout* si introduce un tempo massimo (in decimi di secondo) dopo il quale, pur avendo specificato *prompt*, si procederà automaticamente all'avvio. Se non si specifica *prompt* invece l'avvio verrà effettuato automaticamente, senza possibilità di intervento dell'utente,⁶¹ dopo il numero di decimi di secondo specificato con *delay*.

Direttive come *lba32* e *linear* servono ad indicare a LILO quale metodo utilizzare per caricare il kernel. Per trovare l'immagine del kernel sul disco infatti LILO utilizza la posizione dello stesso specificata dalla *boot map* che contiene la lista degli indirizzi sul disco da cui leggere una immagine del kernel. Di default questi indirizzi sono indicati nella notazione classica, specificando cilindro, testina e settore, ed il caricamento del kernel viene utilizzando l'interfaccia classica illustrata in sez. 5.2.1, questo comporta che si avranno problemi tutte le volte che si installa un kernel al di là del limite del 1024-simo cilindro. Con i BIOS (e le versioni di LILO)

⁶¹in realtà è comunque possibile ottenere la riga di comando se ti tiene premuto il tasto di shift.

più recenti, si deve sempre specificare la direttiva **lba32**, in tal caso gli indirizzi nella *boot map* saranno tenuti in forma lineare, e verrà usata la nuova interfaccia di accesso della *INT13 estesa*, e ci si può dimenticare di tutti i problemi relativi alla geometria dei dischi.

Qualora il BIOS non supporti queste estensioni, la direttiva **linear** mantiene sempre gli indirizzi nella *boot map* in forma lineare, ma per l'accesso al disco esegue una conversione al vecchio formato, usando la *INT13* originale. Questo può essere utile in quanto non è detto che Linux ed il BIOS concordino sempre sulla geometria del disco; se questo accade e non si è usato **linear** all'avvio LILO userà dei valori per cilindro, testina e settore relativi ad una geometria diversa a quella vista dal BIOS, con relativo fallimento. Usando **linear** invece l'idea della geometria che ha il kernel viene diventa irrilevante, e sarà LILO ad usare all'avvio la conversione dell'indirizzo lineare usando la geometria che gli fornisce il BIOS. L'uso di questa direttiva comunque ha senso solo per i vecchi BIOS che non supportano LBA.

Con la direttiva **default** si può scegliere quale, fra le varie immagini del kernel installate o gli altri sistemi operativi presenti su altre partizioni, viene selezionato come default per l'avvio (e lanciato quando scade il tempo specificato da **timeout**). Per farlo basta riferimento all'etichetta che si è associata con la direttiva **label** a ciascuna immagine o altro sistema presente.

Un'altra direttiva fondamentale è **root** che definisce il dispositivo da cui montare la directory radice, nel nostro esempio `/dev/hdb5`.⁶² Di solito la si specifica all'inizio del file per utilizzarla come valore di default per tutti i kernel, ma può essere anche specificata per ciascuna immagine.

La direttiva che permette di identificare un kernel da lanciare è **image**, che prende come parametro il file che contiene l'immagine del kernel. Si ricordi che benché qui essa sia specificata tramite il pathname del file che la contiene, LILO vi accederà direttamente usando posizione di quest'ultima nel disco. Questo significa che se si copia su una di queste immagini un altro file (ad esempio per un aggiornamento), essendo questo creato altrove (quando si sovrascrive con **cp** prima viene creata la copia, poi cancellato il file preesistente e assegnato il suo nome al nuovo), fintanto che non si ricrea la *boot map* e la si reinstalla, LILO continuerà ad accedere alla posizione del vecchio file. Questo significherà nel migliore dei casi, se si è fortunati e nessuno ha sovrascritto i dati, che si utilizzerà ancora la vecchia immagine, mentre in caso di sovrascrittura l'avvio fallirà (in maniera più o meno spettacolare a seconda dei casi).

Per ciascuna direttiva **image** vanno poi fornite una serie di ulteriori direttive specifiche da applicare all'avvio di quella immagine. Una sempre necessaria è **label** che definisce l'etichetta che identifica l'immagine, da usare nella fase di avvio e per la direttiva **default**.

La direttiva **initrd** specifica il file da utilizzare come ram-disk per l'avvio, la direttiva **append** permette di specificare dei parametri di avvio al kernel, analoghi a quelle che si possono scrivere al prompt, la direttiva **read-only** fa montare la radice in sola lettura (in genere è cura degli script di avvio di rimontarla anche in scrittura).

Nel caso si voglia lanciare un altro sistema operativo (cosa che di norma si fa tramite un altro bootloader) si deve usare la direttiva **other** specificando la partizione su cui si trova quest'ultimo. Anche questa direttiva deve essere seguita da una **label**. L'elenco completo si trova al solito nella pagina di manuale, accessibile con `man lilo.conf`.

Una direttiva utile per la sicurezza è **password** che permette di proteggere con una password la procedura di avvio. Dato che la password è scritta in chiaro nel file di configurazione, qualora si usi questa funzione si abbia anche la cura di proteggerne l'accesso in lettura (il comando **lilo** in ogni caso stampa un avviso se questa direttiva viene utilizzata con un file accessibile in lettura).

L'uso di questa direttiva permette di controllare le modalità con cui si fanno partire le varie immagini (o i sistemi operativi alternativi), attraverso ulteriori direttive da indicare nelle relative sezioni del file di configurazione. Specificando **bypass** non viene applicata nessuna restrizione; specificando **restricted** si impedisce all'utente di passare dei parametri al kernel sul prompt, a

⁶²si noti che non è assolutamente detto che questa debba stare sullo stesso disco da cui si lancia il kernel.

meno di non fornire la password; specificando **mandatory** è necessario fornire la password anche per avviare la relativa immagine.

Una volta che si sono definite le impostazioni in `/etc/lilo.conf`, si può attivarle con il comando `lilo`. Questo si limita a leggere il suddetto file e a reinstallare il *bootloader* con i nuovi valori. Il comando prende varie opzioni che permettono di soprassedere i valori specificati da `lilo.conf`, al solito si può fare riferimento alla pagina di manuale, accessibile con `man lilo`, per l'elenco completo.

Vale la pena però di ricordarne due, la prima è `-r` che permette di specificare una directory nella quale eseguire un **chroot** prima di eseguire il comando. Essa è molto utile quando si avvia il sistema da un disco di recupero (ad esempio se si è fatto casino con LILO ed il sistema non parte più) perché permette di correggere i valori nel `lilo.conf` del disco, e ripristinare il sistema reinstallando il bootloader come se lo si fosse fatto direttamente dal disco originale; ovviamente in tal caso occorre montare il disco da qualche parte nel sistema usato come recupero e poi usare `lilo -r` sulla directory su cui lo si è montato.

La seconda opzione è `-R` che invece specifica una riga di comando da passare a LILO al successivo riavvio, come se la si scrivesse direttamente dal prompt. L'utilità dell'opzione è che questo avviene una *sola* volta, per cui solo il riavvio successivo userà le nuove opzioni; così se qualcosa non va, ed il riavvio fallisce, basta far riavviare la macchina per riavere i valori precedenti, che si presume siano funzionanti.

5.3.3 L'uso di GRUB

Il secondo *bootloader* disponibile per Linux è GRUB (da *Grand Unified Bootloader*); GRUB è nato come *bootloader* per il sistema HURD⁶³ ma è in grado di avviare qualunque altro tipo di sistema (compreso Linux e BSD).

La caratteristica principale rispetto a LILO è che GRUB è formato da diverse parti, dette *stadi*, ciascuno dei quali, come gli omonimi dei razzi, viene usato per lanciare il successivo. In sostanza poi gli stadi sono due, il primo è un analogo di LILO e viene installato sul bootloader; il suo unico compito è quello di lanciare lo stadio successivo, che poi si incaricherà di effettuare tutte le operazioni successive.

Il grande vantaggio di GRUB è che siccome il primo stadio deve solo occuparsi di trovare il secondo ed è quest'ultimo che fa tutto il lavoro, tutte le restrizioni che si applicano ad un programma che deve stare nell'MBR scompaiono. Per questo GRUB non solo non soffre del problema del 1024-simo cilindro, ma è in grado di leggere nativamente i principali filesystem, cosicché non solo si può fare riferimento alle immagini del kernel attraverso un pathname, ma si ha anche a disposizione una micro-shell che consente di navigare attraverso un filesystem ed esplorarne il contenuto. Questo vuol dire che basta cambiare il file di configurazione di GRUB (o sovrascrivere una immagine del kernel) perché la nuova versione sia usata al successivo riavvio, e non è necessario utilizzare un comando apposta come per LILO (evitando i guai che nascono tutte le volte che ci si dimentica di farlo).

Un altro grande vantaggio è che GRUB può ricevere i comandi avvio non solo dalla console, ma anche via seriale e via rete, dato che non ha bisogno del BIOS per gestire l'I/O, il che lo rende molto più flessibile. Inoltre non è limitato a lanciare il sistema dal primo canale IDE, ma, essendo di nuovo indipendente dal BIOS, può usare uno qualunque dei dischi presenti.

Tutti i file di GRUB sono mantenuti in `/boot/grub`, in questa directory si trovano i file `stage1` e `stage2` che contengono i due stadi standard usati nell'avvio, e una serie di `stage1_5` che contengono gli stadi intermedi che GRUB utilizza per accedere ai contenuti di vari filesystem (sono supportati tutti i principali filesystem di Linux). Il file `device.map` contiene invece la lista

⁶³HURD è un sistema libero sperimentale realizzato dalla FSF, basato su una architettura microkernel, con funzionalità molto avanzate e completamente modulare; il suo sviluppo però è ancora piuttosto limitato.

dei dispositivi riconosciuti da GRUB in fase di installazione, e come questi vengono mappati nella notazione interna di GRUB; un esempio di questo file è il seguente:

```
piccardi@monk:/boot/grub$ cat device.map
(fd0)    /dev/fd0
(hd0)    /dev/hda
```

che ci mostra come su questa macchina sia presente un floppy ed un hard disk.

GRUB identifica i dispositivi con un nome fra parentesi tonde, il floppy viene sempre identificato con `fd0` (nel caso ci siano due floppy ci sarebbe anche `fd1`) mentre i dischi vengono identificati, in ordine di rilevazione, con `hd0`, `hd1`, ecc. Si tenga presente che GRUB non distingue i nomi per i dischi SCSI, quindi anche questi verrebbero identificati con la sigla `hdN` (con `N` dipendente dall'ordine di rilevamento).

Il file `device.map` viene generato con il comando di installazione di GRUB, `grub-install`: questo prende come parametro il dispositivo su cui si vuole installare GRUB. Il comando copia anche nella directory `/boot` tutti i file di GRUB, si può dirgli di usare una radice diversa con l'opzione `--root-directory=DIR`. Il comando esegue anche una scansione dei dispositivi e crea `device.map`.

In realtà GRUB non ha un vero e proprio file di configurazione, infatti se avviato normalmente mette a disposizione una shell limitata da cui eseguire i vari comandi interni. Questa è disponibile anche da Linux, se lo si lancia con il comando `grub`, la differenza con LILO è che essa è disponibile anche quando viene lanciato all'avvio. I comandi di GRUB sono molteplici ed oltre a quelli usati per impostare l'avvio dei vari kernel, ce ne sono altri con capacità di ricerca dei file, di visualizzazione e confronto del loro contenuto, e la shell di GRUB offre funzionalità avanzate come l'auto-completamento di comandi e nomi, ed un *help on line* con il comando `help`. È però possibile inserire una lista di questi comandi nel file `menu.lst` (sempre sotto `/boot/grub`) e questi verranno eseguiti automaticamente all'avvio, per cui questo diventa una sorta di file di configurazione.

Il vantaggio di usare `menu.lst` è che questo permette di creare automaticamente un menù semigrafico (in formato testo) dal quale scegliere quale kernel (o altro sistema) avviare. Un contenuto tipico di questo file è:

```
## default num
# Set the default entry to the entry number NUM. Numbering starts from 0, and
# the entry number 0 is the default if the command is not used.
#
# You can specify 'saved' instead of a number. In this case, the default entry
# is the entry saved with the command 'savedefault'.
default      0

## timeout sec
# Set a timeout, in SEC seconds, before automatically booting the default entry
# (normally the first entry defined).
timeout      5

# Pretty colours
color cyan/blue white/blue

## password ['--md5'] passwd
# If used in the first section of a menu file, disable all interactive editing
# control (menu entry editor and command-line) and entries protected by the
# command 'lock'
# e.g. password topsecret
#      password --md5 $1$gLhU0/$aW78kHK1QfV3P2b2znUoe/
# password topsecret
```

```

title          Debian GNU/Linux, kernel 2.4.21
root           (hd0,0)
kernel        /boot/vmlinuz-2.4.21 root=/dev/hda1 ro
savedefault

title          Debian GNU/Linux, kernel 2.4.21 (recovery mode)
root           (hd0,0)
kernel        /boot/vmlinuz-2.4.21 root=/dev/hda1 ro single
savedefault

```

e su Debian si può anche usare il comando `update-grub`, che effettua una ricerca dei kernel presenti in `/boot/` e crea automaticamente una lista con le relative voci in `menu.lst`.

I vari kernel sono introdotti da una direttiva `title`, che specifica una stringa che comparirà nel menù iniziale, ad essa si associa la direttiva `root`, che specifica su quale partizione cercare il file. Questa viene indicata con la notazione di GRUB, in cui si indica fra parentesi tonda il dispositivo, seguito da una virgola e dal numero progressivo della partizione a partire da zero. Nel caso precedente allora si dice di cercare il kernel nella prima partizione del primo disco, in sostanza `/dev/hda1`.

L'immagine da caricare si specifica con la direttiva `kernel`, seguita dal pathname del file. Si tenga presente che questo è relativo alla partizione stessa (non esiste in concetto di radice in GRUB), per cui se ad esempio si è posto `/boot` su un'altra partizione non è più necessario specificarla nel nome del file. Al nome del file si devono poi far seguire le opzioni da passare al kernel all'avvio, fra cui occorre comunque specificare il dispositivo da montare come radice (nell'esempio con `root=/dev/hda1`).

La direttiva `default` permette di stabilire quale, nella lista di immagini dichiarate, deve essere lanciata se l'utente non interviene, dopo un tempo di attesa specificato con `timeout`. L'elenco completo delle funzionalità di GRUB è disponibile nel *GRUB-HOWTO*.

5.3.4 Il sistema di inizializzazione alla SysV

Come accennato in sez. 5.3.1 una volta lanciato il kernel si cura solo dell'inizializzazione dell'hardware, di montare la directory radice e di lanciare il programma di avvio del sistema, che per tradizione è `/sbin/init`.⁶⁴ Questo è uno dei motivi per cui questo programma (come tutti quelli essenziali all'avvio, deve stare nella radice, in `/sbin`.

Due degli errori più comuni, che comportano l'impossibilità di proseguire nella procedura di avvio, sono proprio quelli relativi all'impossibilità di montare la radice (ad esempio perché ci si è dimenticati di inserire nel kernel il supporto per accedere al dispositivo su cui essa si trova o quello per il suo filesystem) o all'impossibilità di lanciare `init` (ad esempio perché si è danneggiato il programma, o qualche libreria, o si è indicato come radice una partizione sbagliata).

Una volta lanciato `init` il kernel non esegue più direttamente nessun altro compito, tutto il resto viene effettuato attraverso gli opportuni programmi invocati da `init`. È a questo punto che emerge una delle principali differenze fra i vari sistemi che si ispirano a Unix. Essa origina dalla divisione che ci fu negli anni '70, fra la versione sviluppata alla AT/T, che poi diventerà SysV, e quella sviluppata a Berkley, che darà origine alla famiglia dei BSD. Una delle differenze principali fra i due sistemi è quello del procedimento di avvio.

La differenza non è tanto nel meccanismo di funzionamento del sistema, dato si che tratta sempre di lanciare gli opportuni programmi, quando nella modalità in cui questi vengono avviati. Nei sistemi derivati da BSD questo viene fatto attraverso l'esecuzione di alcuni script. Attivare o meno un servizio dipende dall'inserimento o meno (al posto "giusto") delle opportune righe di

⁶⁴in realtà passando l'opzione `init=...` si può lanciare un programma qualunque indicandone il pathname; usare `init=/bin/sh` è uno dei metodi più comuni per reimpostare una password di root dimenticata.

avvio all'interno di essi. Nel caso di Linux poche distribuzioni (Slackware è la più nota), hanno adottato questa modalità, la gran parte han preferito, per la sua maggiore flessibilità, lo stile di avvio di SysV.

I sistemi derivati da SysV usano un sistema più complesso, ma che offre maggiore funzionalità e soprattutto è più “*modulare*”. La gran parte delle distribuzioni di GNU/Linux⁶⁵ ha adottato questo sistema.

L'avvio “*alla SysV*” avviene sulla base dei cosiddetti *runlevel*, una sorta di *modalità operative* del sistema in cui vengono lanciati un particolare insieme di programmi, che garantiscono la presenza di un certo gruppo di servizi. È possibile selezionare sia quali programmi (ed in che ordine) lanciare in ciascun *runlevel*, sia quale *runlevel* utilizzare. Inoltre è possibile passare da un *runlevel* all'altro. È compito di *init* portare il sistema in un certo *runlevel*, secondo quanto specificato nel suo file di configurazione, */etc/inittab*.

I *runlevel* validi sono 8, quelli effettivamente utilizzati sono numerati da 0 a 6. A questi si aggiunge il *runlevel* S, che riveste un ruolo speciale e serve ad indicare quali di servizi che devono essere attivati comunque. La standardizzazione di LSB prevede che tre *runlevel* siano riservati; il *runlevel* 0 viene usato per fermare il sistema, mentre il *runlevel* 6 per riavviarlo. Infine il *runlevel* 1 serve per andare nel cosiddetto *single user mode*, la modalità di recupero in cui può entrare nel sistema solo l'amministratore e solo dalla console, con tutti i servizi disattivati e la directory radice montata in sola lettura.

Per gli altri *runlevel* le caratteristiche possono variare da distribuzione a distribuzione (con o senza servizi di rete, avvio terminante con il login direttamente da X, etc.). Per Debian da 2 a 5 sono tutti equivalenti, RedHat usa il 2 per l'avvio in console senza rete, 3 per l'avvio in console con la rete e 5 per l'avvio in modalità grafica.

Tutto il procedimento di avvio eseguito viene controllato dal file di configurazione di *init* che è */etc/inittab*. Il formato di questo file è molto semplice, come al solito le linee vuote o che iniziano per “#” vengono ignorate, le altre prevedono quattro campi separati dal carattere “:”, nella forma:

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

dove il campo *id* deve essere una sequenza unica di 1-4 caratteri che identifica la linea (ed in certi casi delle azioni speciali), il campo *runlevel* la lista dei *runlevel* (espressa coi numeri di cui sopra) cui si applica l'azione specificata dalla parola chiave del campo *action* mentre il campo *process* indica il programma che deve essere lanciato (e relative opzioni ed argomenti).

I principali valori utilizzabili per il campo *action* sono riportati in tab. 5.12, i dettagli e l'elenco completo si trovano al solito nella pagina di manuale, accessibile con *man inittab*. Un esempio di questo file 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.
10:0:wait:/etc/init.d/rc 0
11:1:wait:/etc/init.d/rc 1
12:2:wait:/etc/init.d/rc 2
13:3:wait:/etc/init.d/rc 3
14:4:wait:/etc/init.d/rc 4
15:5:wait:/etc/init.d/rc 5
16:6:wait:/etc/init.d/rc 6
# What to do when CTRL-ALT-DEL is pressed.
```

⁶⁵GNU/Linux è stato sviluppato da zero, per cui non deriva da nessuna delle due famiglie di Unix, per questo in genere si è cercato di prendere il meglio da entrambe.

Argomento	Significato
respawn	il processo viene eseguito all'ingresso nel <i>runlevel</i> e viene riavviato tutte le volte che termina.
wait	il processo viene eseguito una volta all'ingresso nel <i>runlevel</i> e si aspetta la sua terminazione prima di proseguire.
once	il processo viene eseguito una volta all'ingresso nel <i>runlevel</i> senza aspettare la sua terminazione prima di proseguire.
boot	il processo viene eseguito una sola volta all'avvio del sistema; il valore del <i>runlevel</i> viene ignorato.
initdefault	indica quale <i>runlevel</i> deve essere utilizzato all'avvio, il campo indicante il programma da eseguire viene ignorato.
ctrlaltdel	esegue il processo specificato quando init riceve un segnale di SIGINT o se si utilizza la combinazione di tasti ctrl-alt-del , viene usato in genere per invocare shutdown .
powerwait	eseguito quando viene notificata ad init la caduta della linea elettrica da parte del programma di gestione del gruppo di continuità.
powerokwait	eseguito quando viene notificato ad init il ritorno della corrente sulla linea elettrica.
powerfailnow	eseguito quando viene comunicato ad init da parte del programma di gestione del gruppo di continuità che le batterie si stanno esaurendo.

Tabella 5.12: Principali valori delle azioni indicabili nell'omonimo campo di `/etc/inittab`.

```

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

```

Il file viene letto da **init** e le azioni specificate vengono eseguite nella procedura di avvio (e ad ogni cambio di *runlevel* forzato con il comando **telinit**). Se il numero del *runlevel* scelto corrisponde con almeno uno di quelli indicati nel secondo campo viene eseguito il comando indicato nella ultima colonna secondo la modalità specificata dal terzo campo. Specificando come azione **wait** si richiede che il programma sia eseguito una sola volta, attendendo che esso sia concluso prima di passare all'azione successiva; nell'esempio è questo il caso con gli script `/etc/init.d/rc` (che, come vedremo fra poco, sono quelli usati per avviare e fermare i servizi). Se invece non è necessario attendere la conclusione si può usare **once**.

Specificando **respawn** si chiede che il comando sia messo in esecuzione immediatamente, senza attendere la sua conclusione per proseguire coi successivi, si richiede inoltre che esso sia rilanciato automaticamente ogni volta che se ne termina l'esecuzione. Questo è quello che nell'esempio viene fatto con **getty** per avere i terminali di login attivi sulle varie console: ogni volta che ci si collega al sistema **getty** eseguirà **login** per l'autenticazione e questo eseguirà

la shell,⁶⁶ all'uscita dalla shell, **init**, accorgendosi della terminazione del processo, rilancerà di nuovo **getty**.

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** impostano proprio il runlevel di default. Inoltre il campo **id** può dover essere soggetto a restrizioni come nel caso delle righe di **getty** che richiedono il numero della console.

In genere non c'è molto da fare con questo file, l'unica cosa che 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.

Come si può notare dall'esempio precedente, attraverso **inittab** si possono impostare solo alcune azioni elementari, tutto il procedimento di avvio di SysV viene effettuato, come accennato, grazie all'uso dello script **/etc/init.d/rc**. Come si può notare questo viene invocato, per ciascun *runlevel*, con un argomento pari al numero dello stesso.

Il meccanismo di avvio di SysV si basa infatti sulla presenza, per ciascun programma o servizio che si vuole attivare, di uno specifico script di avvio, la cui locazione, secondo il FHS, è in **/etc/init.d**. Questi script prendono sempre come parametri una serie di comandi: **start**, che avvia il servizio, **stop** che lo ferma, **restart** che lo ferma e lo riavvia, **reload** che fa rileggere la configurazione.

Se si va ad analizzare il contenuto di **/etc/init.d/rc** ci si accorgerà che questo verifica la presenza di una directory **rcN.d**,⁶⁷ dove N corrisponde al numero del *runlevel* passato come argomento e legge la lista dei file ivi presenti. Se vediamo il contenuto di queste directory vedremo che esse contengono tutte una serie di link simbolici agli script di **/etc/init.d**, con lo stesso nome ma preceduto da una sigla composta da una S od una K seguite da un numero di due cifre.

Dopo aver letto la lista dei file presenti **/etc/init.d/rc** si limita ad eseguire tutti questi script in ordine alfabetico, passando l'argomento **stop** a quelli che iniziano per K (che sta per *kill*) ed l'argomento **start** a quelli che iniziano per S. In questo modo i servizi indicati dai rispettivi script vengono fermati o avviati, e nell'ordine stabilito dalle due cifre usate nella sigla.

In questo modo è possibile, per ogni servizio, avere una procedura d'avvio personalizzata, da mettere in **/etc/init.d**, ed avviarlo o fermarlo a piacere nei vari *runlevel* con un semplice link simbolico. Si può anche, grazie alle due cifre, decidere anche in quale punto della sequenza di avvio lanciare un certo servizio (ad esempio un server di rete dovrà essere lanciato sempre dopo che questa è stata attivata).

Si noti come la presenza dei servizi presenti in un determinato *runlevel* possa dipendere da come si è arrivati ad esso; nel passaggio da un runlevel ad un altro infatti i servizi avviati dal primo vengono fermati dal secondo solo in presenza di uno corrispondente script iniziante per K, altrimenti resteranno attivi, anche se non sono previsti fra quelli che verrebbero avviati se si andasse direttamente in quel *runlevel*.

Come accennato attivare o disattivare un servizio su un certo *runlevel* è semplicemente questione di creare un link simbolico con l'opportuna sigla iniziale, sono comunque disponibili

⁶⁶Si ricordi che mettere in esecuzione un nuovo programma non comporta la creazione di un nuovo processo, nella catena di esecuzioni successive il processo resterà sempre lo stesso.

⁶⁷direttamente sotto **/etc**, come richiesto dalla LSB, ma alcune vecchie distribuzioni ls mantengono ancora sotto **/etc/init.d**.

innumerevoli programmi in grado di automatizzare il compito, come `update-rc.d` per Debian o `chkconfig` per RedHat, per l'uso i quali rimandiamo alle rispettive pagine di manuale.

5.4 La gestione di interfacce e periferiche

Tratteremo in questa sezione i programmi e le funzionalità disponibili per la configurazione di una serie di interfacce hardware di cui i moderni computer sono dotati (escludendo le interfacce di rete che saranno trattate a parte in sez. 7.3) per le quali, al di là dell'accesso ai dispositivi finali secondo l'interfaccia classica per cui in un sistema unix-like *tutto è un file*, sono necessari ulteriori modalità di accesso e controllo per le funzionalità che non sono comprese in questa astrazione.

5.4.1 Gestione delle interfacce di espansione

Una delle caratteristiche essenziali dell'architettura hardware dei computer moderni è quella di disporre di opportune interfacce di espansione in cui inserire delle schede dedicate che permettano l'uso di nuove periferiche; si ha così la possibilità di ampliare le capacità di un computer utilizzando degli appositi dispositivi in grado effettuare i più svariati compiti.

Una interfaccia di espansione in sostanza non fornisce direttamente delle funzionalità finali, che sono specifiche del singolo dispositivo che si pone su di essa, quanto una interfaccia hardware comune che consente una comunicazione fra la CPU e la memoria e tutti i dispositivi che sono posti su detta espansione. In questo modo il sistema operativo può inviare comandi e scambiare dati con le singole periferiche, in modo da poterne utilizzare le capacità specifiche.

In Linux il supporto per queste interfacce è ovviamente fornito direttamente dal kernel, che predispone tutta l'infrastruttura software con cui è possibile leggere i dati relativi a dette interfacce, e poi dialogare con i singoli dispositivi su di esse presenti. Questi ultimi, seguendo il criterio fondamentale dell'architettura di un sistema unix-like per cui *tutto è un file*, potranno poi essere acceduti attraverso l'interfaccia generica illustrata in sez. 1.2.1.

Le due interfacce di espansione tradizionalmente più utilizzate nel mondo dei computer basati sulla architettura classica dei cosiddetti PC (in sostanza la piattaforma basata su Intel o compatibili) sono la ormai superata ISA (*Industry Standard Architecture*) e la più recente ed ampiamente utilizzata PCI (*Peripheral Component Interconnect*). Entrambe queste interfacce definiscono uno standard sia hardware (consistente in piedinatura, dimensioni, specifiche dei segnali elettrici ecc.) che software (i comandi che la CPU deve dare per comunicare con l'interfaccia ed i dispositivi presenti sulla stessa).

La struttura di base di queste interfacce è quella di definire uno speciale canale di comunicazione (il cosiddetto *bus*) sul quale far passare sia i dati che i comandi che vanno dal *sistema centrale* (CPU e memoria) ai dispositivi inseriti nell'interfaccia stessa (le *periferiche* appunto), e viceversa. Le interfacce forniscono inoltre un opportuno meccanismo che permetta di comunicare in maniera indipendente (identificando opportunamente i flussi di dati sul *bus*) con ciascuna delle periferiche presenti sull'interfaccia.

Una descrizione dettagliata del funzionamento di queste interfacce va ben oltre lo scopo di queste dispense, la loro gestione è infatti completamente realizzata all'interno del kernel attraverso il codice relativo al loro supporto. Quello che interessa dal punto di vista dell'amministrazione di sistema è capire quali sono le risorse utilizzate e come allocarle e le funzionalità messe a disposizione del kernel che consentono di accedere alle informazioni relative a dette interfacce ed ai dispositivi su di esse presenti a scopo di configurazione o di controllo.

Le risorse fondamentali usate da una interfaccia sono sostanzialmente due gli *interrupt* e i *canali DMA*. Un *interrupt* è un un meccanismo con cui un dispositivo hardware può inviare un

segnale al processore⁶⁸ (usando quella che si chiama una *linea di interrupt*) così che questo possa *interrompere* le operazioni e rispondere all'esigenza di attenzione così manifestata dal dispositivo. In genere un processore ha un numero limitato di linee di interrupt (originariamente nei PC erano 8, oggi sono state portate a 15), in altre architetture sono 32 o 64.

La seconda risorsa è quella dei *canali DMA*, questi sono un meccanismo hardware che consente dei trasferimenti diretti di dati da una periferica alla memoria senza che questi debbano essere letti direttamente con l'intervento della CPU, che può essere utilizzata in altre operazioni. Con l'uso dei canali DMA il sistema si limita a richiedere solo un intervento iniziale della CPU per indicare al dispositivo in quale zona di memoria inviare i dati, che saranno poi trasferiti in maniera asincrona. In genere l'uso di un canale DMA si abbina sempre a quello di un interrupt che serve a segnalare la conclusione del trasferimento, così che la CPU sistema possa utilizzare i dati disponibili in memoria.

Uno dei problemi relativi alla gestione delle interfacce di espansione è allora proprio quello della allocazione degli interrupt e dei canali DMA ai dispositivi esistenti,⁶⁹ che di norma può essere eseguita a livello di BIOS. In particolare alcuni interrupt sono assegnati staticamente a periferiche presenti sui PC da prima che fosse possibile una allocazione dinamica, come l'interfaccia IDE per i dischi, le seriali e la parallela; gli altri poi possono essere lasciati liberi per l'uso da parte delle interfacce di espansione o di altre interfacce interne come USB (che vedremo in sez. 5.4.4).

I problemi che possono sorgere sono allora quelli dell'allocazione di queste risorse; il kernel permette di esaminare lo stato corrente di queste allocazioni attraverso i due file `/proc/interrupts` per gli interrupt e `/proc/dma` per i canali DMA; un esempio potrebbe essere il seguente:

```
# cat /proc/interrupts
          CPU0
 0:       584706      XT-PIC  timer
 1:       15435      XT-PIC  keyboard
 2:           0      XT-PIC  cascade
 8:          4      XT-PIC  rtc
 9:          2      XT-PIC  usb-uhci, usb-uhci, btaudio, btv
10:     933383      XT-PIC  EMU10K1, eth0
11:       2170      XT-PIC  ide2, ide3, aic7xxx
12:     319229      XT-PIC  PS/2 Mouse
14:         56      XT-PIC  ide0
15:     162630      XT-PIC  ide1
NMI:         0
LOC:     584659
ERR:       848
MIS:         0
```

che mostra l'allocazione degli interrupt, il cui numero progressivo è riportato in prima colonna, rispettivamente ai dispositivi ad essi associati (riportati nell'ultima); nella seconda colonna sono riportati il numero di interrupt registrati al momento ed un sommario di questa statistica è riportato nelle ultime quattro righe. Analogamente abbiamo:

```
# cat /proc/dma
4: cascade
```

che mostra l'allocazione dei canali DMA.

⁶⁸in senso fisico, viene alzato un livello su uno dei piedini del processore a questo dedicato.

⁶⁹in realtà questo è un problema che si ha quasi esclusivamente sui PC intel-compatibili, che hanno pochi interrupt e canali DMA e tutta una serie di limitazioni ereditate dall'architettura originaria che su altre piattaforme non esistono.

Una terza risorsa è quella delle *porte di I/O*, una modalità di comunicazione diretta fra la CPU e le stesse effettuata attraverso l'accesso ad alcune locazioni di memoria riservate (chiamate appunto in questo modo) leggendo e scrivendo dalle quali si andava a leggere e scrivere direttamente sui dispositivi. Anche in questo caso alcune di queste porte sono allocate staticamente a periferiche standard come le seriali, mentre altre devono essere allocate opportunamente quando si inserisce il relativo dispositivo su una interfaccia di espansione; la situazione corrente dell'allocazione è riportata dal kernel nel file `/proc/ioports`, un cui esempio è:

```
# cat /proc/ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
02e8-02ef : serial(set)
02f8-02ff : serial(set)
0376-0376 : ide1
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(set)
0cf8-0cff : PCI conf1
5c20-5c3f : ALi Corporation M7101 PMU
...
```

L'interfaccia ISA nasce come estensione dei bus dei vecchi PC, che consentivano l'utilizzo di schede di espansione, originariamente prevedeva un bus a 8 bit, quasi subito portato a 16; usata principalmente per modem interni e schede sonore, al giorno d'oggi è sostanzialmente in disuso, e si trova soltanto sulle macchine più vecchie.

Le prime versioni dell'interfaccia prevedevano la presenza di opportuni interruttori sulle schede stesse (i cosiddetti *jumper*) che permettevano di selezionare in maniera manuale gli interrupt, i canali DMA o le porte di I/O da utilizzare. In questo caso era cura del sistemista allocare queste risorse in maniera compatibile fra le varie schede (con le opportune impostazioni sui *jumper*); in particolare gli interrupt che a differenza del più recente bus PCI non possono essere condivisi fra schede diverse.

Per evitare questo problema e facilitare l'uso delle schede di espansione da parte degli utenti meno esperti venne creato uno standard per l'autoconfigurazione delle schede chiamato *Plug'n Play*,⁷⁰ in cui le precedenti risorse potevano essere allocate dinamicamente dal sistema operativo o dal BIOS ed impostate sulle schede. Il supporto per questa funzionalità prevede un meccanismo di configurazione detto *isolation*,⁷¹ in cui ad ogni scheda venga assegnato un *Card Serial Number* (in breve CSN) che la identifica e ne vengono rilevate le caratteristiche, dopo di che alle varie schede vengono assegnate le risorse da utilizzare in modo che non ci sia conflitto.

In questo modo un driver potrà utilizzare la scheda conoscendo i parametri che indicano quale risorsa utilizzare; nel caso di Linux buona parte dei moduli delle schede ISA utilizzano i parametri `irq`, `ioport` e `dma` che permettono di utilizzare la scheda sapendo quali sono le risorse ad essa allocate. Il problema è che se un modulo viene caricato prima che il PnP abbia eseguito l'allocazione la scheda non sarà utilizzabile, inoltre se il meccanismo fallisce ci si può trovare con

⁷⁰quasi immediatamente ribattezzato in *Plug'n Pray* visto che spesso il meccanismo non funzionava e le risorse venivano allocate in maniera non compatibile con altre schede, con conseguente impossibilità di usare le espansioni.

⁷¹una descrizione più dettagliata può essere trovata nel Plug-and-Play HOWTO.

delle schede presenti ma non utilizzabili, se poi si cerca di riallocare le risorse di una scheda in uso gli effetti possono essere anche peggiori.⁷²

Le modalità con cui le schede vengono configurate sono sostanzialmente due: alcuni BIOS sono in grado di eseguire da soli il procedimento di allocazione delle risorse, e presentare il risultato finale al sistema operativo;⁷³ altri non sono in grado di farlo⁷⁴ ed il procedimento dovrà allora essere effettuato direttamente dal sistema operativo prima di poter utilizzare i driver.⁷⁵

In genere⁷⁶ quando il BIOS esegue la configurazione delle schede PnP questa viene salvata nella memoria non volatile (la cosiddetta ESCD, *Extended System Configuration Data*, dove vengono mantenute tutte le configurazioni del BIOS) in modo da poter essere riutilizzata al riavvio successivo; quando viene inserita una nuova scheda *Plug'n Play* questa verrà riconosciuta e configurata e la configurazione sarà aggiunta nella ESCD. Sebbene in teoria questo permetta di evitare la riconfigurazione tutte le volte che si riavvia la macchina, con Linux c'è il problema che quando la configurazione viene eseguita in un secondo tempo i nuovi valori non vengono salvati nella ESCD.

Per tutta questa serie di motivi il meccanismo del *Plug'n Play* finisce con il complicare notevolmente le cose, dato che molte delle funzionalità vengono a dipendere dalla versione di BIOS disponibile e da come questo e le relative schede supportano lo standard. Per questo motivo con Linux è in genere preferibile usare le capacità di configurazione fornite direttamente dal sistema.

Prima dei kernel della serie 2.4.x, l'unico modo di eseguire la configurazione era grazie al programma `isapnp`, a partire da essi il supporto per la configurazione è stato introdotto nel kernel grazie al modulo `isa-pnp` che viene chiamato dai vari driver dei dispositivi per eseguire l'allocazione delle risorse. Il programma viene comunque ancora utilizzato (e lo descriveremo a breve) anche se non è più strettamente necessario, almeno fin quando i driver ed il supporto sulle schede funzionano correttamente.

Il comando prende come unico argomento il nome di un file di configurazione (che in genere è `/etc/isapnp.conf`), ed esegue le impostazioni secondo le direttive in esso contenute. Le uniche opzioni sono `--help` e `--version` il cui significato è evidente; a partire dalla versione 1.18 si può usare anche `-` come nome di file, per indicare una lettura dallo standard input.

Eseguendo il comando si esegue la configurazione delle schede e l'assegnazione delle risorse secondo quanto specificato nella configurazione. In genere dovrebbe essere eseguito ad ogni riavvio, in quanto con Linux non c'è modo di salvare le impostazioni precedenti. Se non si fa così si corre il rischio, usando Windows (95 o 98) sulla stessa macchina che questo configuri le schede in maniera differente, così che al successivo riavvio con Linux queste diventino inutilizzabili.

Il formato di `isapnp.conf` è piuttosto complesso, e non staremo qui a descriverlo (gli interessati possono fare riferimento alla pagina di manuale, accessibile con `man isapnp.conf`) dato che in genere questo file viene prodotto automaticamente grazie al comando `pnpdump`. Questo comando permette infatti di ricavare i dati delle schede presenti sulla macchina eseguendo una scansione del bus ISA alla ricerca di schede che supportano il *Plug'n Play* e leggendo da ciascuna di esse l'elenco delle risorse necessarie (il PnP prevede che esse siano memorizzate all'interno della scheda).

Il problema è che in certi casi le informazioni riportate non sono accurate; questo perché fintanto che le schede non erano necessarie all'avvio alcuni produttori sono stati piuttosto pigri

⁷²con buona probabilità il blocco completo del sistema.

⁷³questo nel caso di Linux significa solo che si dovranno individuare quali sono le risorse assegnate, passando gli opportuni valori ai moduli che le utilizzano.

⁷⁴o si può dire al BIOS di non farlo specificando che si ha un sistema operativo *Pnp enabled* che si occuperà del compito nella apposita sezione di configurazione.

⁷⁵questo, nel caso di Linux, significa che la configurazione dovrà essere effettuata prima di caricare i moduli relativi a dette schede.

⁷⁶alcuni BIOS meno evoluti non supportano questa funzionalità.

nella specificazione delle risorse necessarie all'interno della scheda, contando sull'uso del driver (ovviamente fornito solo per DOS/Windows) per eseguire le impostazioni. Questo significa che in certi casi le informazioni ricavate da `pnpdump` non sono corrette, e potrebbe essere necessario controllare quali sono le risorse utilizzate sotto Windows per ricavare dati esatti.

Il comando comunque esegue la scansione effettuando vari tentativi di comunicazione con le eventuali schede, interrogando in fila tutte le porte di I/O possibili (nell'intervallo fra `0x203` e `0x3ff`, riservato alle schede ISA), per identificare quali di queste corrispondono ad un dispositivo presente sul BUS, cui richiedere le informazioni. Si tenga presente però che in certi casi si possono avere conflitti (l'interrogazione cioè può interferire con altre schede non PnP o già configurate) con risultati che vanno dalla successiva inutilizzabilità delle suddette schede al blocco completo del sistema. Pertanto è sempre opportuno eseguire il comando in *single user mode* (vedi sez. 5.3.4).

Se invocato senza argomenti il comando stampa sullo standard output il risultato della scansione, direttamente nel formato utilizzato per la configurazione del programma `isapnp`; di default però tutte le istruzioni sono commentate, pertanto è comune salvare il file e poi editarlo per togliere i commenti. Se però lo si invoca con l'opzione `-c` il comando stesso cerca di determinare le impostazioni più sicure e fornisce una versione pronta (cioè senza necessità di togliere i commenti) del file stesso. Si tenga presente però che se l'allocazione risulta impossibile il comando può bloccarsi indefinitamente nel tentativo di trovare una soluzione.

Se il comando viene invocato con un solo argomento questo viene interpretato come il valore minimo della porta di I/O da cui iniziare la scansione sul bus ISA, si può così limitare l'intervallo evitando di interrogare schede già configurate o non PnP. L'argomento può essere specificato come valore esadecimale (se inizia con le cifre `0x`), ottale (se inizia per `0`) o decimale.

Se si usano due argomenti in questo caso il primo dei due assume il significato di numero di schede già identificate e configurate dal BIOS, nel qual caso il programma non esegue la procedura di ricerca, e inizia la scansione per nuove schede a partire dall'indirizzo passato come secondo argomento. Le opzioni principali del comando sono riportate in tab. 5.13. Al solito per l'elenco completo ed i dettagli si faccia riferimento alla pagina di manuale.

Opzione	Significato
<code>-c</code>	tenta di determinare delle impostazioni sicure per i dispositivi, e produce un output direttamente utilizzabile senza necessità di rimuovere i commenti.
<code>-d</code>	scarica i valori dei registri interni per ogni scheda e li stampa sullo standard output, in questo modo è possibile vedere le eventuali impostazioni fatte dal BIOS o mantenute di default dalle schede stesse.
<code>-h</code>	stampa un messaggio di aiuto.
<code>-i</code>	ignora gli errori di <i>checksum</i> nel riconoscimento degli indirizzi delle porte di I/O; in alcuni casi questo non è dovuto a conflitti e porta a classificare come non valide tutte le porte, ignorandolo si potranno trovare lo stesso le schede presenti.
<code>-v</code>	stampa la versione
<code>-o</code>	scrive il risultato sul file passato come parametro invece che sullo standard output.

Tabella 5.13: Principali opzioni per il comando `pnpdump`.

Lo standard PCI *Peripheral Connect Interface* nasce per fornire un bus di espansione generico che fosse adatto all'evoluzione dei computer, ed in grado di supportare velocità di trasferimento dei dati molto superiori al precedente ISA. Il bus nasce a 32 bit e prevede una frequenza per le operazioni di 33.3MHz, per una banda passante teorica di circa 1Gbit/s; dello standard sono state proposte varie estensioni (fra cui il bus AGP delle schede video che è sostanzialmente un PCI con frequenze operative più alte).

A differenza del bus ISA nel caso di PCI la allocazione delle risorse è dinamica; in genere gli

interrupt vengono associati sulla base dello slot su cui è inserita la scheda di espansione, inoltre il bus supporta la condivisione degli interrupt. Un'altra caratteristica specifica del bus PCI è che ogni scheda riporta al suo interno (nel firmware) una serie di informazioni fra cui da una coppia di numeri che specificano sia il produttore che il tipo della scheda, che possono essere confrontati con un database delle schede prodotte (usualmente mantenuto nel file `/usr/share/misc/pci.ids`).

In un bus PCI ciascuna periferica viene identificata univocamente grazie ad un indirizzo che è composto da tre numeri: il *numero di bus*, il *numero di dispositivo* ed il *numero di funzione*. Il *numero di bus* identifica su quale bus si trova la scheda, infatti lo standard supporta la possibilità di avere più bus (fisicamente distinti) sulla stessa macchina, fino ad un massimo di 256. Una caratteristica comune del PCI è che i bus possono essere collegati in fra di loro attraverso dei dispositivi appositi detti *PCI bridge*, per cui con una scheda apposita si può controllare un altro bus PCI contenente altre schede. Per ciascun bus si possono poi inserire fino a 32 schede diverse (identificate per *numero di dispositivo*) che possono supportare fino a 8 diverse periferiche cadauna (in caso di schede *multifunzione*); il tutto compone un numero a 16 bit che viene a costituire l'*indirizzo* hardware della periferica sul bus PCI.

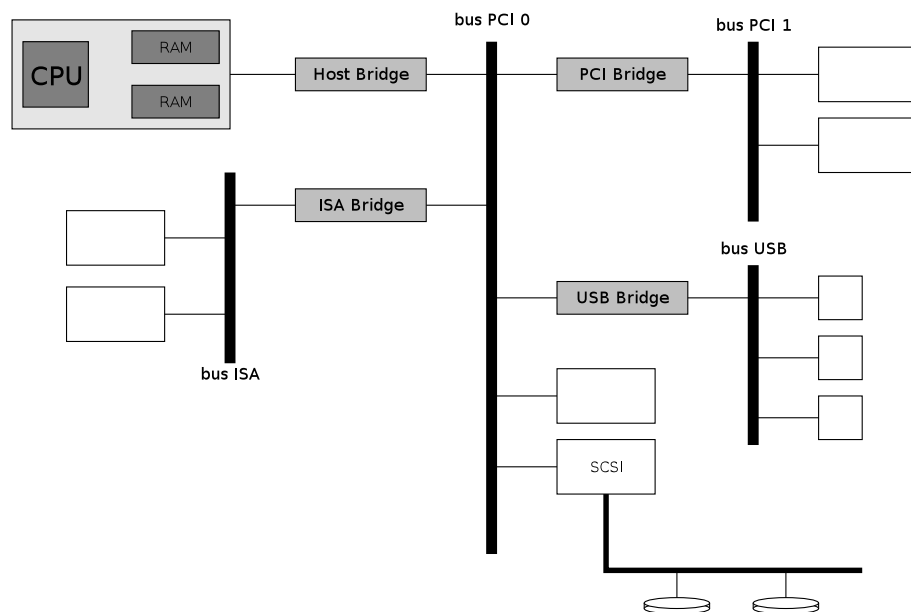


Figura 5.4: Schema della disposizione del bus PCI e delle varie interfacce di espansione.

I computer più recenti in genere hanno sempre almeno due bus (uno dei quali viene usato principalmente per l'AGP), la CPU è collegata attraverso il cosiddetto *host bridge* sul bus principale che è sempre il bus 0, ulteriori bus sono agganciati su questo con un *PCI bridge* e su di essi possono essere agganciati ulteriori bus in modo da formare un albero. Uno schema della disposizione dei bus più comune nelle architetture PC è in fig. 5.4.

In generale il kernel riporta le informazioni relative a tutte le periferiche disponibili sul bus PCI all'interno del filesystem *proc* nella directory `/proc/bus/pci`, che contiene una directory (chiamata con il numero corrispondente per ciascun bus presente nel sistema, più il file `devices` in cui sono riportate le informazioni relative a tutte le schede presenti (ed agli eventuali moduli cui sono associate). Questo però vale per le versioni di kernel più recenti, nelle versioni più vecchie esisteva solo il file `/proc/pci`, attualmente deprecato, benché ancora presente per compatibilità, che contiene una lista descrittiva delle varie schede presenti.

La lista dei dispositivi presenti sul bus può essere ottenuta anche con il comando `lspci` (nei nuovi kernel questo è il metodo canonico, e sarà l'unico supportato in futuro). Il comando non

prende argomenti e stampa sullo standard output la lista delle schede rilevate sul bus PCI, con qualcosa del tipo:

```
[root@gont corso]# lspci
00:00.0 Host bridge: VIA Technologies, Inc. VT8363/8365 [KT133/KM133] (rev 03)
00:01.0 PCI bridge: VIA Technologies, Inc. VT8363/8365 [KT133/KM133 AGP]
00:07.0 ISA bridge: VIA Technologies, Inc. VT82C686 [Apollo Super South] (rev 4
0)
00:07.1 IDE interface: VIA Technologies, Inc. VT82C586/B/686A/B PIPC Bus Master
IDE (rev 06)
00:07.2 USB Controller: VIA Technologies, Inc. USB (rev 16)
00:07.3 USB Controller: VIA Technologies, Inc. USB (rev 16)
00:07.4 Host bridge: VIA Technologies, Inc. VT82C686 [Apollo Super ACPI] (rev 4
0)
00:09.0 SCSI storage controller: Adaptec AHA-2940U/UW/D / AIC-7881U (rev 01)
00:0f.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8139/8139C/813
9C+ (rev 10)
01:00.0 VGA compatible controller: nVidia Corporation NV11 [GeForce2 MX/MX 400]
(rev a1)
```

Il comando riporta nella prima colonna l'indirizzo di ciascun dispositivo ordinato per numero di bus, numero di dispositivo e numero di funzione seguito da una descrizione sommaria della periferica relativa; usando l'opzione `-v` si può avere una descrizione più dettagliata comprensiva delle risorse utilizzate dalla periferica stessa, con qualcosa del tipo di:

```
0000:00:12.0 Ethernet controller: VIA Technologies, Inc. VT6102 [Rhine-II] (rev 74)
Subsystem: Micro-Star International Co., Ltd.: Unknown device 7120
Flags: bus master, medium devsel, latency 32, IRQ 23
I/O ports at dc00 [size=256]
Memory at dffffe00 (32-bit, non-prefetchable) [size=256]
Capabilities: <available only to root>
```

e ripetendo una seconda volta l'opzione si possono avere ancora più dettagli.

Con l'opzione `-t` si può invece avere una stampa della struttura ad albero del bus, mentre con `-s` si può selezionare quali dispositivi guardare, passando un parametro che esprime il relativo indirizzo nella forma `bus:slot.func` (il formato della prima colonna del precedente esempio) dove ciascun numero identificativo può essere sostituito dal carattere jolly `"*"`. Le altre opzioni principali sono riportate in tab. 5.14, per un elenco completo si faccia al solito riferimento alla pagina di manuale del comando.

Opzione	Significato
<code>-v</code>	aumenta le informazioni stampate (può essere ripetuto due volte).
<code>-n</code>	riporta il valore numerico degli identificatori delle schede invece di usare la descrizione testuale riportata nel database.
<code>-b</code>	riporta la lista degli interrupt per come li vede il bus (con APIC nel kernel vengono rimappati).
<code>-t</code>	mostra una schematizzazione ad albero della disposizione dei dispositivi.
<code>-s</code>	mostra le informazioni relative ad uno o dispositivi che corrispondono ad un certo indirizzo sul bus espresso un parametro nella forma <code>bus:slot.func</code> .
<code>-d</code>	mostra le informazioni relative ai dispositivi di un singolo produttore (usando gli identificativi della scheda sulla base del valore del parametro <code>vendorID:deviceID</code> dove entrambi gli identificativi sono espressi come numeri esadecimali).
<code>-i</code>	usa il file passato come parametro come database degli identificativi delle schede PCI.

Tabella 5.14: Principali opzioni per il comando `lspci`.

Un secondo comando utilizzabile per operare sul bus PCI è **setpci**, che può essere utilizzato per interrogare e configurare i singoli dispositivi presenti sul bus. Il comando necessita sempre di una opzione di selezione per indicare su quale dispositivo operare, questa può essere sia **-s** per usare l'indirizzo che **-d**) per usare l'identificativo della scheda; entrambe usano la stessa sintassi già vista in tab. 5.14 per **lspci**.

Il comando prende come argomento il nome del registro⁷⁷ su cui si vuole operare. Indicando solo il nome di un registro ne sarà stampato il valore corrente, indicando una assegnazione nella forma **registro=valore** ne sarà invece cambiato il contenuto (si tenga conto che il valore deve essere espresso in esadecimale). Al nome del registro si possono aggiungere i suffissi **.B**, **.W** e **.L** per indicare che si vuole eseguire l'operazione su un registro di dimensione pari ad un byte, una parola (16 bit) o una parola lunga (32 bit). Al posto del nome può anche essere usato un valore esadecimale che ne indica la posizione nello spazio dei registri.

Un elenco sommario dei principali registri e del relativo significato è riportato in tab. 5.15, un elenco completo di tutti i nomi definiti è riportato nella pagina di manuale di **setpci** (i nomi sono riportati in maiuscolo, ma possono essere specificati anche in minuscolo), per il relativo significato si può fare riferimento alla dichiarazione delle costanti omonime in **/usr/include/linux/pci.h** o alle specifiche dello standard PCI.

Registro	Significato
device_id	identificativo del dispositivo.
vendor_id	identificativo del produttore del dispositivo.
latency_timer	imposta un temporizzatore scaduto il quale il dispositivo rilascia l'uso del bus (così si permette in uso più corretto del bus da parte di altri dispositivi presenti).
min_gnt	valore (in sola lettura) del tempo minimo per il quale deve essere garantito l'uso del bus al dispositivo (in unità di quarti di microsecondo).
max_lat	valore (in sola lettura) che specifica quanto spesso il dispositivo necessita di accedere al bus (in unità di quarti di microsecondo).

Tabella 5.15: Costanti identificative di alcuni registri PCI usate dal comando **setpci**.

5.4.2 Gestione delle interfacce SCSI

L'interfaccia SCSI (*Small Computer System Interface*) potrebbe essere inserita fra le interfacce di espansione generiche trattate in sez. 5.4.2 in quanto anche essa definisce una struttura a *bus* su cui vengono innestati dispositivi multipli. La trattiamo a parte in quanto in realtà questa non è necessariamente una interfaccia collegata ad un solo PC⁷⁸ e di norma viene realizzata tramite l'uso di apposite schede (i *controller SCSI*) che si inseriscono in una delle interfacce precedentemente citate (ormai esclusivamente PCI, anche se alcuni vecchi controller usavano l'interfaccia ISA) e vengono utilizzati tramite esse.

Una seconda differenza con le interfacce generiche di sez. 5.4.1 è che nonostante sia possibile inserire dispositivi diversi su un bus SCSI, l'interfaccia è utilizzata quasi esclusivamente per l'accesso a periferiche di stoccaggio di dati (principalmente dischi e nastri, ma anche CD e masterizzatori) e non all'uso di periferiche generiche,⁷⁹ ed il protocollo stesso della trasmissione dei dati sul bus è dedicato a questo tipo di lavoro. Infine il bus non è di solito cablato in soluzione

⁷⁷lo standard PCI prevede che tutte le schede debbano avere una serie di registri di configurazione, che contengono informazioni o controllano vari aspetti del loro funzionamento, come il tempo massimo che un dispositivo può tenere il bus.

⁷⁸è infatti possibile, e viene fatto normalmente con sistemi di dischi condivisi, collegare più PC alla stessa interfaccia SCSI.

⁷⁹unica eccezione ancora in uso è quella degli primi scanner che, non essendo all'epoca disponibile una interfaccia più semplice con sufficienti capacità, venivano pilotati da una interfaccia SCSI.

unica su una piastra madre, ma può essere realizzato anche con un insieme di connettori che collegano fra loro le varie periferiche.

Uno dei problemi maggiori con le interfacce SCSI è che dalla prima definizione dello standard (SCSI-1, del 1986) si sono susseguite molte modifiche (lo SCSI-2, cui segue lo SCSI-3 che però non è mai stato rilasciato come tale ed è stato suddiviso in parti distinte dell'interfaccia) ma anche all'interno degli stessi standard le modalità di realizzazione delle cablature, dei connettori e dei segnali sono varie il che ha portato ad una discreta confusione.

Lo standard originario (SCSI-1) prevedeva un bus basato su un connettore a 50 pin, di cui 8 (più uno di parità) erano riservati per la trasmissione dei dati, la frequenza di trasmissione era di 5MHz, con una corrispondente velocità massima di trasferimento di 5MiB/s, erano poi previste 4 linee per gli indirizzi, consentendo fino ad un massimo di 8 periferiche sul bus.

Una prima modifica, chiamata *Fast SCSI* venne fatta aumentando la frequenza del bus a 10MHz per raddoppiare la velocità di trasferimento. Altre modifiche vennero fatte riguardo la modalità di gestione dei segnali sui cavi ed il tipo di cablatura, questo portò allo sviluppo di un secondo standard (lo SCSI-2). Lo standard prevedeva una lunga serie di estensioni, molte delle quali relative alla gestione dei segnali sui cavi ed alla composizione della cablatura stessa (terminazione attiva, segnali differenziali, nuovi connettori), oltre ad un aumento dei comandi disponibili e all'introduzione del sistema del *command queing* che permetteva ad un singolo dispositivo di accettare più comandi, in modo da poterne ottimizzare l'ordine di esecuzione.

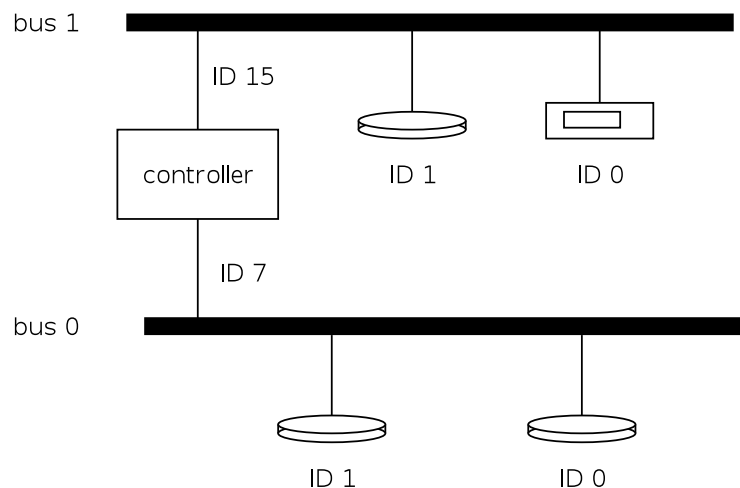


Figura 5.5: Schema della struttura delle interfacce SCSI.

Una delle modifiche principali (detta *Wide SCSI*), che richiede un connettore diverso a 68 pin, prevedeva la possibilità di usare un bus per il trasferimento dati a 16 bit (detto *wide*, in contrapposizione con il precedente, detto *narrow*) raddoppiando anche il numero di periferiche inseribili nel bus. In questo modo si otteneva a parità di frequenza (con il cosiddetto *Fast Wide SCSI*) un raddoppio della velocità di trasferimento 20MiB/s. Passi successivi sono state l'introduzione di bus con frequenze sempre maggiori consentendo velocità di trasferimento sempre maggiori: *Ultra SCSI* a 40MiB/s, *Ultra2* a 80MiB/s, *Ultra160* a 160MiB/s e ultimamente pure *Ultra 320* a 320MiB/s.

Lo schema classico di una interfaccia SCSI è riportato in fig. 5.5, il bus prevede l'esistenza di almeno una *unità di controllo*, la scheda, detta *controller SCSI*, che in genere si mette sul bus PCI del computer, e a cui si collegano le altre periferiche utilizzando gli appositi connettori. Un singolo controller può gestire anche più bus, nel qual caso sarà in grado di alloggiare più connettori. Tutte le periferiche sul bus, compreso l'unità di controllo, vengono identificate attraverso un indirizzo di identificazione, detto SCSI ID.

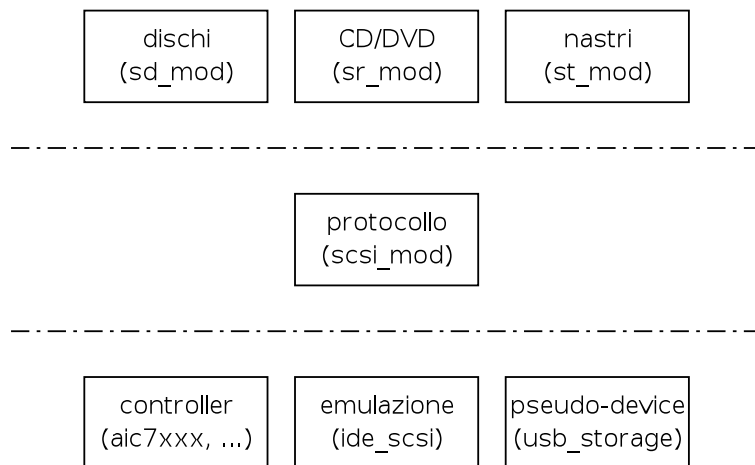


Figura 5.6: Strutturazione del supporto alle interfacce SCSI nel kernel.

Nel caso di Linux il supporto SCSI è presente fin dalle prime versioni del kernel, ma dato che lo standard prevede vari tipi di periferiche a partire dal kernel 2.4 il sistema è stato organizzato su tre livelli, secondo lo schema illustrato in fig. 5.6. Al livello più alto, utilizzati direttamente dai programmi quando accedono ai relativi file di dispositivo, stanno i driver per le periferiche che si trovano sul bus, come CD, nastri o dischi, ciascuno dei quali richiede una modalità d'accesso distinta; a questi si aggiunge il supporto generico che viene usato come interfaccia per inviare direttamente comandi ad una periferica (che viene usato per pilotare dispositivi particolari come gli scanner o i masterizzatori).

Il livello intermedio costituisce il collante fra i driver dei dispositivi finali ed i driver che invece si occupano di gestire i controller per l'accesso al bus, che saranno diversi a seconda della scheda SCSI utilizzata. Oltre a quest'ultimi però si situano a questo livello anche tutti i moduli che consentono di utilizzare il protocollo SCSI per controllare dispositivi posti su altre interfacce; ad esempio il protocollo viene utilizzato sia per accedere ai dischi sul bus USB (che siano memorie a stato solido o veri e propri hard disk), che per utilizzare i masterizzatori IDE attraverso un meccanismo di emulazione; in tal caso i comandi del protocollo SCSI, invece di diventare segnali elettrici su connettore attaccato ad un controller, saranno inviati su un bus virtuale fornito dal driver per il supporto di dette funzionalità.

Le periferiche accedute tramite interfaccia SCSI sono identificate attraverso quattro numeri: il primo è l'*host adapter number* che identifica, come il nome stesso indica, quale è l'interfaccia (in genere la scheda col controller, ma può essere anche l'interfaccia di un bus virtuale) con cui si accede ai dispositivi; il numero viene assegnato dal kernel all'avvio a seconda dell'ordine in cui rileva le schede presenti o di quando viene abilitato il supporto per bus virtuali che si utilizzeranno. L'*host adapter number* è un numero progressivo che parte da zero.

Il secondo numero è il cosiddetto *channel number*, che identifica ciascun bus (detto anche, in altra nomenclatura, *canale SCSI*) associato ad un *host adapter*; alcuni controller infatti possono gestire più di un bus, e questi saranno numerati progressivamente a partire da zero, nell'ordine in cui l'adattatore stesso li presenta (che dipenderà ovviamente da come è fatto quest'ultimo). Anche questo è un numero progressivo che parte da zero.

All'interno di ciascun *canale* si avrà poi l'identificativo del singolo dispositivo posto su di esso (lo SCSI ID di cui abbiamo già parlato). In genere sui dispositivi questo viene stabilito dall'utente con degli appositi interruttori; di norma sono presenti dei jumper che permettono di impostare le linee corrispondenti, anche se in alcuni confezionamenti sono disponibili dei selettori. Per i controller invece l'identificativo può essere modificato dal BIOS di configurazione, ma è in genere preimpostato al valore più alto (in genere 7) che è quello che ha maggiore priorità.

Infine alcuni dispositivi più sofisticati (come le unità a nastro dotate di libreria, o i juke-box di CDROM) possono avere al loro interno diverse funzionalità che vengono allora indirizzate da un quarto numero detto *Logical Unit Name*, o LUN. In questo caso la periferica da usare (ad esempio l'unità a nastro o il meccanismo di controllo della libreria) viene identificata completamente usando anche il LUN; per la maggioranza dei dispositivi, che non hanno più periferiche a bordo, questo è nullo.

Pertanto una modalità di indicare le periferiche SCSI è quella di fornire una quadrupletta di numeri che indicano interfaccia, canale, ID e LUN, del tipo di:

```
<scsi(_adapter_number), channel, id, lun>
```

e quando esse vengono riconosciute dal kernel in fase di avvio si avrà in messaggio del tipo:

```
scsi0 : SCSI emulation for USB Mass Storage devices
  Vendor: IC35L120  Model: AVV207-0          Rev:  0 0
  Type:   Direct-Access                      ANSI SCSI revision: 02
USB Mass Storage device found at 2
...
SCSI device sda: 241254720 512-byte hdwr sectors (123522 MB)
sda: assuming drive cache: write through
sda: sda1
Attached scsi disk sda at scsi0, channel 0, id 0, lun 0
```

e si noti come in questo caso sia stato rilevato un hard-disk su USB, in cui l'interfaccia di emulazione viene vista come `scsi0`, ed dispositivo viene riconosciuto (non essendo qui disponibile un meccanismo di assegnazione degli ID ogni dispositivo verrà associato ad una diversa interfaccia) con valori del canale, ID e LUN nulli.

Il kernel mantiene le informazione relative alle interfacce SCSI disponibili nella directory `/proc/scsi`, ed in particolare nel file `scsi`; un esempio del contenuto del file, quando si ha una interfaccia cui sono agganciati più dispositivi è il seguente:

```
root@ellington:~# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: MATSHITA Model: CD-R   CW-7502   Rev: 4.10
  Type:   CD-ROM                ANSI SCSI revision: 02
Host: scsi0 Channel: 00 Id: 01 Lun: 00
  Vendor: PIONEER  Model: CD-ROM DR-U16S   Rev: 1.01
  Type:   CD-ROM                ANSI SCSI revision: 02
Host: scsi0 Channel: 00 Id: 02 Lun: 00
  Vendor: SEAGATE  Model: ST118202LW      Rev: 0004
  Type:   Direct-Access          ANSI SCSI revision: 02
```

ed in questo caso si noti come ci siano tre diversi ID per tre diverse periferiche.

Come già mostrato nel precedente esempio sul contenuto dei messaggi di avvio ogni volta che una periferica viene rilevata su una interfaccia SCSI gli viene assegnato un file di dispositivo, quest'ultimo dipende dal tipo di dispositivo, nel caso di dischi essi sono tutti nella forma `/dev/sdX` dove `X` è una lettera progressiva a partire da `a`; le eventuali partizioni saranno accedute per numeri crescenti a partire da 1, esattamente come per i dischi IDE.

Qualora invece si abbia a che fare con dei CDROM si avranno dispositivi diversi, in particolare nel caso del secondo esempio si avrà (andando a controllare i messaggi di avvio) un risultato del tipo:

```
Attached scsi CD-ROM sr0 at scsi0, channel 0, id 0, lun 0
Attached scsi CD-ROM sr1 at scsi0, channel 0, id 1, lun 0
```

i CD infatti vengono acceduti con i dispositivi `/dev/srN` dove `N` è un numero progressivo a partire da 0; un nome equivalente, anch'esso utilizzato, è `/dev/scdN`. Gli altri dispositivi utilizzati sono `/dev/sgN` per l'utilizzo del driver generico (quello con cui si pilotano ad esempio masterizzatori e scanner) e `/dev/stN` per i nastri (con l'alternativa di `/dev/nstN`, in cui il nastro non viene riavvolto quando arriva alla fine).

Una caratteristica speciale del file `/proc/scsi/scsi` è che questo può essere acceduto anche in scrittura, si possono così rimuovere e aggiungere dispositivi, funzionalità che può essere utile qualora si disponga di periferiche *hot-swap* che possono essere estratte a caldo dal bus⁸⁰; questo può essere fatto con comandi come:

```
echo "scsi remove-single-device H C I L" > /proc/scsi/scsi
echo "scsi add-single-device H C I L" > /proc/scsi/scsi
```

dove `H`, `C`, `I` ed `L` sono la solita quadrupletta di numeri che identifica il dispositivo. Ovviamente un dispositivo potrà essere rimosso solo se non è al momento utilizzato.

Le altre informazioni presenti nella directory `/proc/scsi` sono una directory per ciascuna interfaccia utilizzata, con il nome del relativo modulo di kernel, contenente un file con nome corrispondente al numero di ciascun *host adapter number* presente con quella interfaccia, nella forma:

```
/proc/scsi/<driver_name>/<scsi_adapter_number>
```

così nelle situazioni dei due esempi illustrati potremo trovare la directory `aic7xxx` in caso e la directory `usb-storage` nell'altro, e in quest'ultimo caso, quando utilizziamo oltre al disco fisso anche una penna USB, otterremo al suo interno due file, corrispondenti alle due istanze della stessa interfaccia viste come *host adapter* diversi.⁸¹

Rispetto agli equivalenti dispositivi IDE in genere dischi e CDROM SCSI hanno prestazioni superiori per la capacità del bus di gestire l'invio di comandi multipli, vista però la maggiore complessità del protocollo sono anche in genere più costosi (per questo CD e masterizzatori sono ormai praticamente inesistenti, e rimangono sostanzialmente solo dischi e nastri per l'uso professionale).

Un secondo aspetto da tenere presente nel caso di dispositivi SCSI è che in genere il cablaggio degli stessi è più delicato, alcuni bus infatti devono essere opportunamente *terminati* per consentire una corretta trasmissione dei segnali, inoltre la presenza di diversi standard sullo stesso bus può portare a degrado di prestazioni. Infine occorre tenere sotto controllo l'allocazione degli ID, che è di norma a carico degli utenti, la presenza di due dispositivi con lo stesso ID infatti li rende entrambi inutilizzabili.

Un programma storicamente utilizzato per ricavare le informazioni relative ai dispositivi SCSI è `scsi_info`, (che in Debian fa parte del pacchetto `pcmcia-cs`); questo permette di ricavare dal dispositivo le informazioni ad esso sottostanti come:

```
monk:/home/piccardi/truedoc/corso# scsi_info /dev/sda
SCSI_ID="0,0,0"
HOST="0"
MODEL="IC35L120 AVV207-0"
FW_REV=" 0 0"
```

Gran parte dei programmi di gestione dell'interfaccia SCSI sono però distribuiti con il pacchetto `scsitools`; il principale è `scsiinfo` che permette di interrogare un dispositivo (passato come argomento) per ricavarne tutta una serie di proprietà (per l'elenco completo e le relative

⁸⁰questo è possibile solo se il bus e le periferiche sono cablati con dei connettori appositi (ad 80 pin) che supportino questa funzionalità.

⁸¹si avrebbe lo stesso risultato nel caso si installassero più schede con lo stesso controller.

spiegazioni si consulti la pagina di manuale), è invece da segnalare l'uso dell'opzione `-l`, da fare senza specificare un dispositivo, che riporta quelli presenti.

Infine sempre con il pacchetto `scsitools` viene anche fornito lo script `rescan-scsi-bus.sh` che esegue una scansione del bus abilitando nuovi dispositivi eventualmente aggiunti dopo il boot, utilizzando i comandi da inviare sul file `/proc/scsi/scsi` illustrati in precedenza.

5.4.3 Gestione delle interfacce seriali

Le interfacce seriali sono una delle prime interfacce create per la comunicazione fra computer, e prevedono appunto una comunicazione via cavo estremamente semplice (e di breve distanza) basata sull'invio dei dati lungo una linea di trasmissione, in cui un dato viene trasmesso appunto come una sequenza di singoli segnali che traducono direttamente il valore dello stesso espresso come sequenza di bit.

Le seriali sono presenti da sempre sui PC, le configurazioni standard di molte schede madri prevedono due interfacce, che possono essere estese a 4, anche se negli ultimi tempi (per la presenza di nuove interfacce più veloci come l'USB) tendono ad essere meno presenti. Ciò non di meno esse restano le interfacce tipiche con cui vengono gestiti i modem,⁸² ed una delle interfacce più semplici per collegarsi a dispositivi esterni come router o switch programmabili.

Data la loro presenza fin dalle origini dell'architettura dei PC, l'assegnazione delle risorse delle porte seriali è predefinita, e sia le porte di I/O che gli interrupt utilizzati per le 4 porte previste dall'architettura sono riportati in tab. 5.16, insieme ai file di dispositivo con cui si può accedere ad essi.

Interrupt	Porta I/O	Dispositivo
3	0x3F8	/dev/ttyS0.
4	0x2F8	/dev/ttyS1.
3	0x3E8	/dev/ttyS2.
5	0x2E8	/dev/ttyS3.

Tabella 5.16: Risorse e file di dispositivo usati dalle porte seriali.

A causa delle limitazioni dell'architettura originale dei PC, l'uso contemporaneo di due porte seriali crea problemi se queste condividono la stessa linea di interrupt. Per questo se è necessario un numero maggiore di porte seriali occorre utilizzare delle schede di estensione apposite (esistono ad esempio delle schede dedicate ad alta velocità, usate per lo più dai provider per gestire i modem), per le quali il kernel è in grado di supportare la condivisione degli interrupt. Un caso di conflitto più comune è invece quello in cui, come parte di una scheda ISDN o di un modem-fax interno, viene installata una nuova interfaccia seriale ed in tal caso occorrerà verificare questa non sia sulla stessa linea di interrupt di una porta seriale che è già in uso.

Uno dei problemi più che si possono avere utilizzando altre porte oltre quelle standard è che queste possono non essere configurate correttamente. Per ovviare a questo problema si può utilizzare il programma di configurazione delle interfacce seriali `setserial`. Questo prende come primo argomento il file di dispositivo da configurare (o controllare), se utilizzato senza opzioni viene stampato un breve riassunto delle principali caratteristiche dello stesso:

```
monk:/home/piccardi/truedoc/corso# setserial /dev/ttyS1
/dev/ttyS1, UART: 16550A, Port: 0x02f8, IRQ: 3
```

dove sono mostrate rispettivamente il tipo di chip della porta seriale (nel caso un 16550A) la porta di I/O e la linea di interrupt utilizzate.

Quando il comando viene utilizzato con l'opzione `-g` gli argomenti vengono interpretati come un lista di dispositivi di cui stampare le proprietà; l'output delle informazioni riportate dal

⁸²quelli esterni, ma pure alcuni di quelli interni, sia che siano modem reali che modem finti, come i winmodem.

comando è controllato dalle ulteriori opzioni **-a**, che fa stampare tutte le informazioni possibili, e **-b** che riporta solo un riassunto della configurazione del dispositivo; l'elenco delle principali opzioni è riportato in tab. 5.17.

Opzione	Significato
-a	stampa tutte le informazioni disponibili.
-b	stampa un riassunto della configurazione.
-g	stampa le informazioni nel formato in cui vengono prese sulla riga di comando.
-z	azzerà tutti i valori prima di fare le impostazioni.

Tabella 5.17: Principali opzioni per il comando **setserial**.

Se invocato senza l'opzione **-g**, gli argomenti successivi al primo permettono di indicare quale parametro del dispositivo deve essere impostato dal comando; in generale soltanto l'amministratore può modificare i parametri dell'interfaccia, ma alcuni di questi possono essere impostati anche da un utente normale.

Opzione	Significato
port 0xHHHH	imposta la porta di I/O.
irq N	imposta il numero della linea di interrupt.
uart type	imposta il tipo di chip, i principali valori sono 8250, 16450, 16550, 16550A, ecc. (per la lista si faccia riferimento alla pagina di manuale); con none si disabilita la porta.
autoconfig	fa eseguire una autoconfigurazione al kernel (la porta di I/O deve essere già impostata) con cui determinare la UART e se auto_irq anche la linea di interrupt.
auto_irq	richiede di autoconfigurare anche la linea di interrupt.
baud_base X	imposta la frequenza delle operazioni di base, in bit per secondo, in genere vale 115200, che è il massimo raggiungibile da una seriale standard.
spd_hi	richiede l'uso di una velocità di 57600bps (bit per secondo).
spd_vhi	richiede l'uso di una velocità di 115200bps (bit per secondo).
spd_shi	richiede l'uso di una velocità di 230400bps (bit per secondo).
spd_warp	richiede l'uso di una velocità di 460800bps (bit per secondo).
spd_normal	richiede l'uso di una velocità di 38400bps (bit per secondo).
spd_cust	richiede una velocità personalizzata pari a valore di baud_base diviso per quello di divisor .
divisor N	imposta il divisore con cui calcolare una velocità personalizzata.

Tabella 5.18: Principali direttive di impostazione del comando **setserial**.

I due parametri principali impostabili dal comando sono **port** e **irq**, il cui significato è ovvio e che prendono rispettivamente come ulteriore argomento il numero di porta e di interrupt. Si può inoltre impostare a mano (qualora non venisse riconosciuta correttamente) il tipo di interfaccia (la UART, *Universal Asynchronous Receiver/Transmitter*) con **uart**; infine si può controllare la velocità della porta seriale con la serie di argomenti **spd_*** il cui significato, insieme a quello degli altri parametri principali, è riportato in tab. 5.18; al solito per una descrizione dettagliata su può fare riferimento alla pagina di manuale di **setserial**.

5.4.4 Gestione delle interfacce USB

L'interfaccia USB (*Universal Serial Bus*) nasce sulla piattaforma PC allo scopo di fornire una interfaccia di comunicazione semplificata e con prestazioni ridotte, ma molto semplice ad realizzare e poco costosa per la realizzazione di dispositivi semplici (ed in genere portabili, è infatti prevista anche la capacità di fornire alimentazione) che non necessitano di tutte le risorse (in termini di velocità di accesso e banda passante nella trasmissione dei dati) fornite dalle usuali interfacce di espansione trattate in sez. 5.4.1.

Un bus USB è costruito in maniera gerarchica ed è controllato da una *unità di controllo* (o *host*), a cui è direttamente collegato un *concentratore* (o *hub*) radice a cui tutti i dispositivi o eventuali altri concentratori secondari vanno ad agganciarsi, secondo la struttura mostrata in fig. 5.7. Il protocollo prevede che le comunicazioni siano tutte controllate dall'unità di controllo, che è l'unica che può iniziare una comunicazione, ed i dispositivi possono solo rispondere all'unità di controllo; non è prevista nessuna possibilità di comunicazione dei singoli dispositivi fra di loro.⁸³

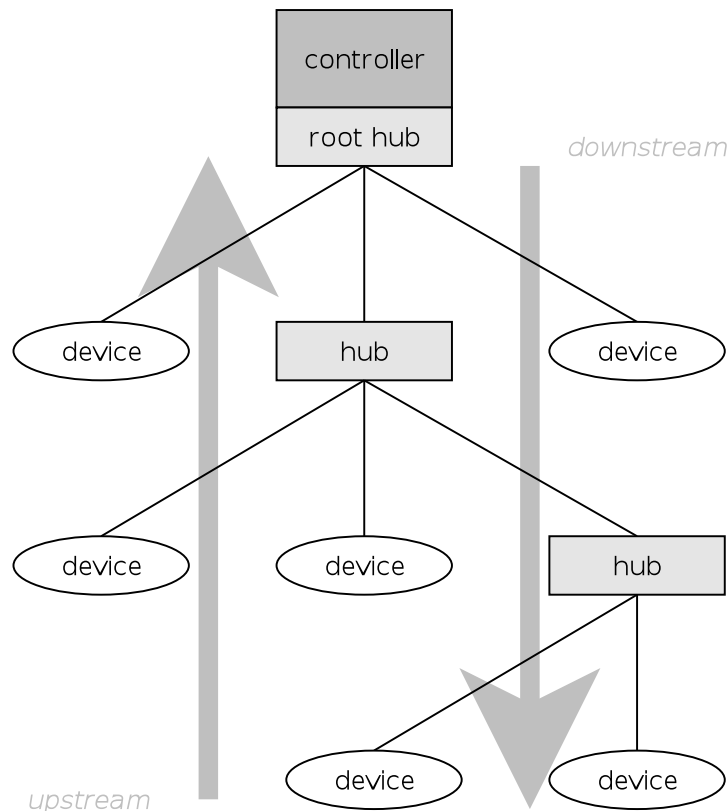


Figura 5.7: Schema della struttura di un bus USB.

In genere su un computer è presente un solo *host* (anche se alcuni ne hanno più di uno, ed è comunque possibile aggiungerne altri usando schede di espansione) cui sono agganciate 2 o 4 porte. A ciascuna porta USB si può agganciare un dispositivo o un *hub*, il quale a sua volta avrà altre porte cui agganciare altri dispositivi ed *hub*. Il bus prevede fino ad un massimo di 127 unità per ciascuna unità di controllo; la presenza di *hub* secondari permette l'utilizzo di un maggior numero di periferiche rispetto alle porte presenti sul PC, ma si tenga conto che anche essi contano come unità sul bus, pertanto le unità disponibili per le periferiche sono in realtà 127 meno il numero di *hub* aggiuntivi.

Come mostrato in fig. 5.7 il bus prevede due direzioni per il flusso dei dati, dall'*host* alla periferica e viceversa, indicati con *downstream* e *upstream*; inoltre vengono specificati quattro tipi di trasferimento:

⁸³altre architetture non hanno queste limitazioni, ma USB è stata progettata con un compromesso fra prestazione ed economicità, per cui si sono evitate funzionalità avanzate come l'*arbitraggio* del bus che comportano una maggiore complessità realizzativa per dispositivi che dovevano essere estremamente semplici

Control transfers

utilizzati per inviare pacchetti di controllo, che devono essere di dimensioni ridotte e trasportati con affidabilità; viene usata per configurare i dispositivi e per supportare i comandi di controllo di base.

Bulk transfers

usata per trasmettere dati alla massima velocità possibile in maniera affidabile; viene usata per i trasferimenti di dati da e verso i dispositivi.

Interrupt transfers

simili ai precedenti, ma ripetuti periodicamente, la richiesta viene ripetuta periodicamente, vengono utilizzati come meccanismo di notifica (dato che non esiste un vero e proprio *interrupt*).

Isochronous transfers

usata per inviare e ricevere dati potendo contare su una quantità di banda garantita, ma senza affidabilità, usata per trasferimenti *real-time* (ad esempio i trasferimenti per video e audio).

La presenza di due direzioni nel flusso di dati si riflette anche sui connettori, che in genere sono classificati in due tipi diversi, a seconda che siano rivolti all'unità di controllo (di *tipo A*) o verso una periferica (di *tipo B*). Inoltre i dispositivi si possono suddividere in autoalimentati, alimentati dal bus,⁸⁴ o entrambi. Un'altra distinzione fra i dispositivi è fra quelli *lenti* (come mouse, tastiere, ecc.) che comunicano al massimo ad 1.5Mbit/s e quelli *veloci* che possono usare potenzialmente fino al 90% della banda passante massima.

La prima versione del protocollo (USB 1.0) consentiva una banda passante massima di 12Mbit/s. Tuttavia l'uso di dispositivi lenti, gli *interrupt* e l'overhead del protocollo non consentono velocità superiori a 8.5Mbit/s anche in condizioni ideali, mentre prestazioni tipiche sono intorno ai 2Mbit/s. La seconda versione del protocollo, la USB2 permette invece di portare la banda passante ad un massimo teorico 480Mbit/s, ampliando notevolmente l'utilizzabilità del bus per l'utilizzo con dispositivi di stoccaggio esterni (in particolare con dischi rimovibili).

Negli ultimi tempi praticamente qualunque PC compatibile viene fornito di interfaccia USB, usualmente fornita direttamente dal *chipset* della piastra madre, anche se sono disponibili schede PCI con a bordo delle unità di controllo. Per quanto riguarda USB 1.0 le unità di controllo sono sostanzialmente di due tipi, o compatibili con le specifiche *Open Host Controller Interface* (o OHCI) della Compaq (usata anche nel mondo Mac) o con le specifiche *Universal Host Controller Interface* della Intel. Le funzionalità sono le stesse, ma la seconda specifica richiede hardware più semplice e quindi è meno costosa, ma richiede un driver più complesso e quindi un maggior carico sulla CPU.

Il supporto per USB è stato introdotto per la prima volta a partire dal kernel 2.2.7; ma nella serie 2.2.x il supporto è molto grezzo, almeno fino al 2.2.18 in cui sono state portate indietro parecchie funzionalità del 2.4; per un supporto pieno delle funzionalità del bus (ed in particolare per utilizzare dischi su USB) occorre comunque usare un kernel della serie 2.4.x o successivo. In ogni caso Linux supporta entrambi i tipi di unità di controllo (UHCI o OHCI), utilizzando rispettivamente i moduli `usb-uhci` e `usb-ohci` (da selezionare nella relativa sezione di configurazione del kernel, vedi sez. 5.1.3).

In genere è estremamente facile riconoscere quale delle due interfacce viene utilizzata, basta infatti usare `lspci` per verificare il tipo di unità di controllo; avremo allora, a seconda dei casi, per una macchina che ha una unità UHCI:

⁸⁴che fornisce un massimo di 400 mA, quindi attenzione a non sovraccaricare, pena il rischio di non far funzionare nulla per mancanza di potenza.

```

holland:~# lspci -v
...
00:04.2 USB Controller: VIA Technologies, Inc. VT82xxxxx UHCI USB 1.1 Controller
(r rev 10) (prog-if 00 [UHCI])
    Subsystem: VIA Technologies, Inc. (Wrong ID) USB Controller
    Flags: bus master, medium devsel, latency 32, IRQ 9
    I/O ports at b400 [size=32]
    Capabilities: [80] Power Management version 2
...

```

mentre per una unità OHCI si avrà:

```

davis:~# lspci -v
...
00:0f.2 USB Controller: ServerWorks CSB6 OHCI USB Controller (rev 05) (prog-if
10 [OHCI])
    Subsystem: ServerWorks: Unknown device 0220
    Flags: bus master, medium devsel, latency 32, IRQ 5
    Memory at fe120000 (32-bit, non-prefetchable) [size=4K]
...

```

Invece se si dispone di una unità USB 2, le specifiche utilizzate sono quelle della *Extended Host Controller Interface*, identificata con la sigla EHCI, mentre in questo caso il modulo da utilizzare è **ehci-hcd**. In generale la scelta di quale modulo deve essere fatta in sede di installazione, e può essere fissata usando **modules.conf** (vedi sez. 5.1.4).

Una delle caratteristiche del bus USB è che (a differenza ad esempio di quanto avviene per il bus SCSI) la numerazione delle periferiche presenti non deve essere effettuata a mano, in quanto questo compito è eseguito dal bus stesso. L'altra caratteristica interessante del bus è che esso supporta sempre la possibilità di disconnettere a caldo i dispositivi (anche se questo non può avere conseguenze poco piacevoli se fatto senza accortezza). Per questo motivo in genere il sistema è in grado di usare un demone dedicato come **hotplug**,⁸⁵ che è in grado di rilevare la connessione e la disconnessione di nuove periferiche caricando (e rimuovendo) automaticamente i relativi moduli.

Il bus fornisce inoltre al kernel la possibilità di rilevare tutta una serie di informazioni dai singoli dispositivi (il protocollo infatti prevede che questi forniscano i dati relativi a loro stessi); in genere questa informazione è disponibile sotto **/proc/bus/usb** una volta che si sia montato il filesystem virtuale *usbdevfs*,⁸⁶ cosa che si può fare automaticamente all'avvio aggiungendo una riga del tipo:

```
none    /proc/bus/usb  usbdevfs      defaults      0          0
```

a **/etc/fstab**.

In questo caso sotto la directory **/proc/bus/usb** compariranno tante directory quanti sono i bus disponibili (numerate in ordine crescente), ciascuna delle quali conterrà un file con nome il numero associato a ciascun dispositivo presente su detto bus, contenente i relativi dati. Un sommario con una presentazione più leggibile degli stessi dati è fornito invece dal file **/proc/bus/usb/devices** che riporta l'elenco di tutti i dispositivi presenti con una contenuto del tipo:

⁸⁵inizialmente la gestione automatica veniva effettuata tramite il demone **usbmgr**, questo è ormai obsoleto e sostituito da **hotplug** che fornisce lo stesso tipo di funzionalità in maniera generica per tutte le periferiche (PCMCIA, SCSI, PCI, USB) che possono essere connesse e disconnesse a caldo.

⁸⁶si ricordi di abilitarne il supporto in caso di ricompilazione del kernel, altrimenti gran parte dei programmi di visualizzazione dei dati del bus non funzioneranno.

```

T: Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#= 1 Spd=480 MxCh= 6
B: Alloc= 0/800 us ( 0%), #Int= 0, #Iso= 0
D: Ver= 2.00 Cls=09(hub ) Sub=00 Prot=01 MxPS= 8 #Cfgs= 1
P: Vendor=0000 ProdID=0000 Rev= 2.06
S: Manufacturer=Linux 2.6.6 ehci_hcd
S: Product=VIA Technologies, Inc. USB 2.0
S: SerialNumber=0000:00:10.3
C:* #Ifs= 1 Cfg#= 1 Atr=40 MxPwr= 0mA
I: If#= 0 Alt= 0 #EPs= 1 Cls=09(hub ) Sub=00 Prot=00 Driver=hub
E: Ad=81(I) Atr=03(Int.) MxPS= 2 Iv1=256ms

T: Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#= 2 Spd=480 MxCh= 0
D: Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs= 1
P: Vendor=04b4 ProdID=6830 Rev= 0.01
S: Manufacturer=Cypress Semiconductor
S: Product=USB2.0 Storage Device
S: SerialNumber=600000001814
C:* #Ifs= 1 Cfg#= 1 Atr=c0 MxPwr= 0mA
I: If#= 0 Alt= 0 #EPs= 2 Cls=08(stor.) Sub=06 Prot=50 Driver=usb-storage
E: Ad=02(0) Atr=02(Bulk) MxPS= 512 Iv1=0ms
E: Ad=88(I) Atr=02(Bulk) MxPS= 512 Iv1=0ms

```

Il principale utilizzo di questo file è per verificare se il riconoscimento di una periferica e la relativa numerazione è stata eseguito correttamente, infatti per ciascun dispositivo presente sul bus viene riportata ne file una diversa sezione con tutti i dati ad esso relativi. Ogni sezione è introdotta da una linea che inizia con la lettera T e che indica il posizionamento dello dispositivo all'interno del bus, dove **Bus** indica il numero progressivo del bus, **Lev** il livello (in termini di passaggi per eventuali *hub* secondari) **Dev** il numero progressivo assegnato. Nelle righe successive sono poi riportate altre informazioni relative al dispositivo, sono particolarmente importanti, come riportati nella riga iniziante per P, gli identificatori del venditore e del prodotto che vengono utilizzati per selezionare automaticamente⁸⁷ il modulo da utilizzare per il supporto del dispositivo, ma che in caso di fallimento possono essere utilizzati per cercare informazioni su internet; informazioni utili relative al dispositivo sono quelle delle stringe fornite dal produttore e mostrate nelle righe inizianti per S. Per il resto della sintassi si può fare riferimento alla *Linux USB Guide* disponibile su <http://www.linux-usb.org/USB-guide/book1.html>

Per una visualizzazione più intelligibile di queste informazioni si può usare i programmi del pacchetto `usbutils`, in particolare il programma `lsusb` permette di avere una lista molto semplice dei dispositivi presenti nella forma:

```

piccardi@monk:~/truedoc/corso$ lsusb
Bus 001 Device 002: ID 04b4:6830 Cypress Semiconductor Corp.
Bus 001 Device 001: ID 0000:0000

```

Il programma può comunque stampare la lista anche in forma gerarchica usando l'opzione `-t`, mentre si può ottenere informazioni molto più dettagliate con l'opzione `-v`. Si possono poi usare le opzioni `-s` e `-d` in maniera analoga a come si fa con `lspci` per selezionare un singolo dispositivo sulla base rispettivamente del suo bus e numero identificativo (con la sintassi `bus:num`) o del produttore (usando i rispettivi identificatori cui abbiamo accennato in precedenza). Al solito si faccia riferimento alla pagina di manuale per una descrizione completa.

⁸⁷se si è installato il demone **hotplug**, che si cura di identificare, tramite questi numeri ed un opportuno database di corrispondenze, ogni nuovo dispositivo attaccato ad una porta USB e di caricare in memoria il relativo modulo.

Capitolo 6

Amministrazione avanzata del sistema

6.1 L'utilizzo del RAID

Tratteremo in questa prima sezione l'utilizzo dei sistemi RAID, una tecnologia che permette di utilizzare i dischi a gruppi (detti *array*) per ottenere migliori prestazioni o una maggiore affidabilità; partiremo con una introduzione teorica a questa tecnologia per poi vedere le varie modalità con cui essa viene realizzata e può essere utilizzata con Linux.

6.1.1 Introduzione

La tecnologia RAID (acronimo che sta per *Redundant Arrays of Inexpensive Disks*) nasce con un articolo pubblicato nel 1987 a Berkley, dove si descrivevano varie tipologie di *disk array*. L'idea di base è quella di combinare più dischi indipendenti di piccole dimensioni per ottenere o prestazioni, o affidabilità o quantità di spazio disco superiori a quelle ottenibili con un qualunque disco SLED (*Single Large Expensive Drive*).

L'articolo presentava cinque diversi tipi di architetture RAID (da RAID-1 a RAID-5) ciascuna delle quali prevedeva diversi gradi di ridondanza per fornire tolleranza alla rottura e diversi gradi di prestazioni. A queste si è aggiunta poi la notazione RAID-0 per indicare una configurazione senza ridondanza.

Oggigiorno alcune delle configurazioni previste nell'articolo non sono più supportate (se non in sistemi specializzati), e non le prenderemo neanche in considerazione, ne sono poi emerse altre come la combinazione del RAID-0 con il RAID-1 (spesso indicata come RAID-10) o quella definita *linear*.¹ In generale comunque ci si suole riferire a queste configurazioni con il nome di *livelli*, quelli che prenderemo in esame sono pertanto:

linear In questo livello due o più dischi vengono combinati in un singolo dispositivo. I dischi sono semplicemente accodati uno sull'altro, così che scrivendo in maniera lineare prima verrà riempito il primo e poi i successivi. La dimensione dei singoli dischi in questo caso è del tutto irrilevante.

Questo livello non assicura nessun tipo di ridondanza, e se uno dei dischi si rompe si perderanno probabilmente tutti i dati. Data la modalità di scrittura comunque si perderanno solo i dati presenti nel disco rotto, e qualche forma di recupero potrebbe comunque essere possibile.

¹con il kernel 2.6.x è stata introdotta pure una tecnologia RAID-6 analoga alla RAID-5 che però usa due dischi invece di uno per mantenere la ridondanza e pertanto assicura una maggiore affidabilità.

Questa configurazione non migliora le prestazioni di lettura o scrittura sulla singola operazione, ma quando più utenti accedono all'array si può una distribuzione del carico sui vari dischi, qualora per un caso fortuito venga eseguiti degli accessi a dati posti su dischi diversi.

RAID-0 Questo livello è anche chiamato *stripe mode*; in questo caso dischi dovrebbero avere la stessa dimensione (anche se non è strettamente necessario). In questo caso le operazioni sull'array vengono distribuite sui singoli dischi. Se si scrive un blocco di dati questo verrà suddiviso in *strisce* (le *stripes*, appunto) di dimensione predefinita² che vengono scritte in sequenza su ciascuno dei vari dischi che compongono l'array: una striscia sul primo, poi sul secondo, ecc. fino all'ultimo per poi ricominciare con il primo. In questo modo la capacità totale resta la somma di quella dei singoli dischi.

In questo modo le prestazioni di lettura e scrittura possono essere notevolmente aumentate, in quanto vengono eseguite in parallelo su più dischi, e se i dischi sono veloci ed in numero sufficiente si può facilmente saturare la capacità del bus, o se quest'ultimo è veloce si può ottenere una banda passante totale pari alla somma di quella dei singoli dischi.

Se è presente un disco di dimensioni maggiori si potrà continuare ad accedere operando solo sullo spazio presente sulla parte finale dello stesso, ma in questo caso si opererà su un solo disco, e non si avrà quindi nessun miglioramento delle prestazioni per quanto riguarda i dati che vanno a finire su di esso.

Come nel caso precedente questo livello non fornisce nessuna ridondanza. La rottura di un disco comporta la perdita totale di tutti i dati, compresi pure quelli sugli altri dischi, dato che in questo caso non esistono, a differenza di *linear*, blocchi di dati continui più lunghi della dimensione di una striscia. In questa modalità l'MTBF (*Mean Time Between Failure* però diminuisce proporzionalmente al numero di dischi dell'array in quanto la probabilità di fallimento dell'array è equivalente a quella di un singolo disco moltiplicata per il numero dei dischi che compongono l'array.

RAID-1 Questo è il primo livello che fornisce della ridondanza. Richiede almeno due dischi per l'array e l'eventuale presenza di dischi di riserva. Inoltre i dischi devono essere tutti uguali, se sono diversi la dimensione dell'array sarà quella del più piccolo dei dischi. I dati vengono scritti in parallelo su tutti i dischi che compongono l'array, pertanto basti che sopravviva anche un solo disco dell'array perché i dati siano intatti, in quanto ciascun disco ne ha una copia completa. Gli eventuali dischi di riserva vengono utilizzati in caso di fallimento di un disco dell'array, per dare inizio ad una ricostruzione immediata dell'array stesso.

Con questo livello di RAID si aumenta l'affidabilità in quanto la probabilità di fallimento dell'array è equivalente a quella di un singolo disco divisa per il numero dei dischi dell'array. In realtà poi è ancora minore, in quanto la probabilità che i dischi falliscano tutti insieme è ancora minore, per cui basta sostituire un disco rotto per recuperare l'intera funzionalità dell'array.

Come controparte per l'affidabilità le prestazioni di scrittura di un RAID-1 sono in genere peggiori di quelle del singolo disco, in quanto i dati devono essere scritti in contemporanea su più dischi, e se questi sono molti (e si usa una implementazione software) facendo passare tante copie si può superare il limite di banda del bus su cui sono posti i dischi abbastanza facilmente. In questo caso i controller hardware

²questa viene di norma definita in fase di creazione dell'array, ed è uno dei parametri fondamentali per le prestazioni dello stesso.

hanno il vantaggio di gestire internamente la replicazione dei dati, richiedendo l'invio su bus di una sola copia.

Le prestazioni in lettura sono invece migliori, non tanto sulla singola lettura, ma quando ci sono molte letture contemporanee e spostamenti sui dati. In tal caso infatti i dati possono essere letti in parallelo da dischi diversi, e si può approfittare del diverso posizionamento delle testine sui dischi per leggere i dati da quello che deve compiere un minore spostamento delle testine.³

RAID-4 Benché disponibile è senz'altro il meno usato. Necessita di tre o più dischi. Mantiene delle informazioni di recupero su un disco di *parità*, ed utilizza gli altri in RAID-0. I dischi devono essere di dimensioni identiche, altrimenti l'array avrà la dimensione del più piccolo di essi. Se uno dei dischi in RAID-0 si rompe è possibile utilizzare le informazioni di recupero del disco di *parità* per ricostruire i dati in maniera completa. Se si rompono due dischi si perderanno invece tutti i dati.

Questo livello cerca di bilanciare i vantaggi di affidabilità del RAID-1 con le prestazioni del RAID-0, ma non viene usato molto spesso perché il disco di parità viene a costituire un collo di bottiglia in quanto comunque tutte le informazioni devono esservi replicate, l'unico caso di impiego reale è quello di un disco veloce affiancato ad un gruppo di dischi più lenti.

RAID-5 Come per il RAID-4 anche in questo caso si cercano di ottenere sia vantaggi di affidabilità che quelli di prestazioni, ed è il più usato quando si hanno più di due dischi da mettere in RAID. Anche per il RAID-5 sono necessari almeno tre dischi, ma in questo caso, per evitare il problema del collo di bottiglia, le informazioni di *parità*, queste vengono distribuite su tutti i dischi che fanno parte dell'array. In questo modo anche se uno dei dischi si rompe le informazioni di parità presenti sugli altri permettono di mantenere intatti i dati. Inoltre se si sono inseriti dei dischi di riserva la ricostruzione del disco rotto o indisponibile viene fatta partire immediatamente. Se però si rompono due dischi i dati sono persi.

Il RAID-5 presenta inoltre, dato che dati sono comunque distribuiti su più dischi, gli stessi vantaggi nella lettura presentati dal RAID-0. In scrittura invece le prestazioni sono meno predicibili, e le operazioni possono essere anche molto costose, quando richiedono pure le letture necessarie al calcolo delle informazioni di parità, o dell'ordine di quelle del RAID-1. In generale comunque le prestazioni aumentano sia in lettura che in scrittura. Inoltre nel caso di RAID software è richiesto un discreto consumo di risorse (sia di memoria che di CPU) per il calcolo delle informazioni di parità.

RAID-10 Si suole definire così una configurazione in cui si effettua un RAID-0 di più array configurati in RAID-1. In questo modo si ottiene sia la ridondanza del RAID-1 che l'aumento di prestazioni del RAID-0. Rispetto al RAID-5 ha il vantaggio che possono rompersi anche più dischi, ma fintanto che almeno ne resta almeno uno attivo nei due RAID-1 le informazioni saranno integre. Il tutto al prezzo di un numero molto maggiore di dischi per ottenere la stessa capacità.

Uno degli scopi più comuni di un RAID è quello di proteggersi dal fallimento di un disco (per questo il RAID-0 non deve mai essere usato in un sistema di produzione, il rischio di fallimento hardware si moltiplica per il numero di dischi); si tenga comunque presente che l'utilizzo di un livello di RAID che assicuri la ridondanza non è mai un sostituto di una buona politica di

³è caratteristica comune degli hard disk avere tempi di risposta nettamente diversi per la lettura sequenziale di dati, rispetto al posizionamento (il cosiddetto *seek time*).

backup; un RAID infatti può proteggere dal fallimento dell'hardware di un disco, ma non può nulla contro la corruzione di un filesystem, un utente che cancella i dati dal disco, o un qualunque incidente che distrugge il vostro array.

6.1.2 Il RAID su Linux

In generale per poter creare un array di dischi in RAID ci sono due opzioni fondamentali: hardware o software. Nel primo caso occorre poter disporre di un supporto hardware, cioè di un controller per i dischi che esegue le operazioni necessarie al suo interno, e presenta al sistema operativo l'insieme dei dischi come un dispositivo unico.

In teoria un RAID hardware è l'opzione migliore sul piano delle prestazioni in quanto non necessita di utilizzare dalle risorse (CPU e memoria) della macchina, ma in genere chiede parecchio (specie per gli SCSI) da quelle del portafogli. Inoltre la superiorità di prestazioni è spesso solo teorica, molti controller hardware (specie quelli IDE) si sono rivelati essere molto più lenti dell'implementazione software del kernel Linux per la scarsa potenza dei processori utilizzati. Questo, unito al fatto che in genere per molti server la CPU è una risorsa che conta assai poco in confronto all'importanza dell'I/O su disco, fa sì che l'opzione del RAID software nella maggior parte dei casi si riveli estremamente competitiva.

In ogni caso Linux supporta vari controller RAID hardware, sia SCSI che IDE. Nel primo caso (i controller SCSI) i vari driver sono disponibili nella sezione **Block Devices** della configurazione del kernel. Essi sono in genere accessibili attraverso i file di dispositivo posti in opportune sottodirectory di `/dev` a seconda del controller (ad esempio per il DAC Mylex si usa `rd` mentre per gli array Compaq si usa `ida`). I singoli array vengono identificati per “canale” (spesso i controller supportano più di una singola catena SCSI) e per “disco”, un dispositivo tipico di un array è sempre nella forma `/dev/rd/c0d0` che indica il primo array sul primo canale. Una volta partizionato l'array (che a questo punto viene visto come un altro disco) le partizioni saranno accessibili con nomi del tipo `/dev/ida/c0d0p1`.

Nel caso di RAID su IDE non ci sono canali, ed i dispositivi sono accessibili, una volta attivato il relativo supporto, che è nella sezione **ATA/IDE/MFM/RLL support**, attraverso i file di dispositivo generici disposti nella directory `/dev/ataraid`, i cui nomi sono analoghi ai precedenti, soltanto che non presentano un numero di canale; si avrà cioè ad esempio qualcosa del tipo di `/dev/ataraid/d0p1`.

Per quando non si dispone di un supporto hardware Linux fornisce un supporto software, attivabile nella sezione **Multi-device support**, che supporta la creazione di RAID-0, RAID-1, RAID-4, RAID-5 e *linear*. È cioè possibile far costruire al kernel stesso degli array utilizzando qualunque tipo di dispositivo a blocchi; dato che gli stessi RAID software sono a loro volta dispositivi a blocchi, è possibile riutilizzarli per metterli in RAID fra di loro (pertanto è possibile creare un RAID-10 mettendo semplicemente in RAID-0 un precedente array software creato in RAID-1). Una volta costruiti gli array si potrà accedere ad essi con i file di dispositivo `/dev/mdX`.

6.1.3 Il RAID software

A parte abilitare il supporto nel kernel (cui abbiamo accennato in sez. 5.1.3), per l'utilizzo del RAID software è necessario utilizzare anche gli opportuni programmi in user space; di questi esistono sostanzialmente due versioni, i `raidtools` (disponibili nell'omonimo pacchetto) e il programma `mdadm` che invece è un singolo programma che raccoglie al suo interno tutte le funzionalità divise nei vari componenti dei `raidtools`, e può funzionare anche senza l'ausilio di un file di configurazione (`/etc/raidtab`) che invece è necessario per i `raidtools`. Nel nostro caso ci concentreremo comunque su questi ultimi.

Come accennato i `raidtools` utilizzano nelle loro operazioni un apposito file di configurazione, `/etc/raidtab`, che contiene le definizioni di tutti gli array. Il file è diviso in sezioni che

definiscono i vari array, identificati dalla direttiva **raiddev**, che prende come parametro il nome del dispositivo da utilizzare per accedere ad esso. Una volta che si è definito un array lo si può riutilizzare (in successive direttive **raiddev**. Un esempio di questo file, preso dalla pagina di manuale, è il seguente:

```
#
# sample raiddev configuration file
# 'old' RAID0 array created with mdtools.
#
raiddev /dev/md0
    raid-level          0
    nr-raid-disks       2
    persistent-superblock 0
    chunk-size          8
    device              /dev/hda1
    raid-disk           0
    device              /dev/hdb1
    raid-disk           1
raiddev /dev/md1
    raid-level          5
    nr-raid-disks       3
    nr-spare-disks      1
    persistent-superblock 1
    parity-algorithm     left-symmetric
    device              /dev/sda1
    raid-disk           0
    device              /dev/sdb1
    raid-disk           1
    device              /dev/sdc1
    raid-disk           2
    device              /dev/sdd1
    spare-disk          0
```

La sintassi del file è molto semplice, righe vuote o inizianti per “#” vengono ignorate, ogni riga valida contiene una direttiva seguita da un parametro che ne specifica il valore. Alla definizione di una sezione con **raiddev** devono seguire le altre direttive che specificano di che tipo di array di tratta. La principale è **raid-level**, che prende come parametro il numero di livello: i valori possibili sono 0, 1, 4 o 5) oppure **linear**. A questa segue di solito la direttiva **nr-raid-disks** che indica il numero di dischi dell’array. Per i livelli che supportano dei dischi di riserva (vale a dire RAID-5 e RAID-1) se ne può specificare il numero con **nr-spare-disks**.

Per ciascuna componente dell’array si deve poi specificare con la direttiva **device** qual’è il file di dispositivo con cui vi si accede. Questo può essere un qualunque dispositivo a blocchi, ivi compreso un altro RAID, purché questo sia già stato definito in una precedente sezione di **/etc/raidtab**. Ad una direttiva **device** deve seguire una direttiva **raid-disk** che ne specifica la posizione nell’array (in ordine progressivo, a partire da 0).

Gli eventuali dischi di riserva devono essere anch’essi dichiarati con **device**, e poi specificati con la direttiva **spare-disk**. Nel caso di RAID-4 esiste anche la direttiva di specificazione **parity-disk** che identifica il disco con le informazioni di parità. Devono essere specificate tante direttive **device** quanti sono i dischi che si è indicato (con **nr-raid-disks** e **nr-spare-disks**) fare parte dell’array.

Oltre a queste che sono fondamentali per definire la struttura di una array, esistono una serie di direttive opzionali che permettono di configurare altre caratteristiche del RAID. Ad esempio l’uso della direttiva **persistent-superblock** permette di salvare le informazioni relative alla configurazione di un array in tutte le partizioni che ne fanno parte, in un apposito blocco di dati, il *persistent superblock* appunto, detto così in quanto in questo modo le informazioni sull’array sono sempre disponibili, anche quando non si può accedere ad **/etc/raidtab** (ad esempio quando

si ha la partizione radice sul RAID). I valori possibili sono 1 e 0 che rispettivamente attivano e disattivano l'uso del *persistent superblock*.

Questa funzionalità è essenziale, se si è compilato il relativo supporto del kernel, ad attivare anche l'autorilevamento del RAID all'avvio del kernel, occorre però anche aver marcato le partizioni che fanno parte di un array come di tipo `0xFD`; in tal caso infatti il kernel riconosce la partizione come componente di un RAID, e legge dal *persistent superblock* tutte le informazioni necessarie per attivarlo.

Un'altra direttiva importante è **chunk-size** che dice le dimensioni, in kilobyte, delle *stripe* in cui vengono divisi i dati scritti sull'array per i livelli che supportano lo *striping*. Questa non ha nessun significato né per il RAID-1 né per il linear, perché in entrambi i casi i dati vengono comunque scritti (rispettivamente su entrambi o su uno dei dischi) qualunque sia la loro dimensione, ma negli altri casi questo dice la dimensione delle *strisce* in cui vengono divisi i dati inviati su dischi consecutivi.

Infine è molto utile la direttiva **failed-disk** che permette di creare un array inserendoci un dispositivo senza che questo venga effettivamente utilizzato; in sostanza questa direttiva marca il dispositivo suddetto come *rotto*, e così diventa possibile ad esempio utilizzare il suddetto dispositivo ed al contempo essere in grado anche di montare il dispositivo RAID che userà i restanti dischi. È in questo modo che di solito si effettua una installazione che usi come directory radice un array: prima si installa su un disco normale, poi si crea l'array (di cui fa parte anche detto disco, ma come **failed-disk**) e si copia su di esso tutto il sistema. Riavviando ed usando l'array come radice si potrà successivamente aggiungere all'array il disco usato per l'installazione, facendo fare la sincronizzazione dei dischi al kernel.

Le altre principali direttive di `/etc/raidtab` sono riassunte in tab. 6.1; al solito l'elenco completo è nella pagina di manuale, accessibile con `man raidtab`.

Direttiva	Significato
raiddev	introduce la sezione che definisce un array, prende come parametro il relativo file di dispositivo.
raid-level	indica il tipo di RAID da utilizzare.
nr-raid-disks	indica il numero di dischi che costituiscono direttamente l'array.
nr-spare-disks	indica il numero di dischi di riserva inseriti nell'array.
persistent-superblock	registra le informazioni sull'array in un apposito settore sui dispositivi sottostanti.
chunk-size	dimensione delle <i>strisce</i> (per le modalità RAID che le supportano).
device	indica uno dei dispositivi costituenti l'array.
raid-disk	indica la posizione nell'array del precedente dispositivo indicato con device .
spare-disk	indica la posizione nell'array del precedente dispositivo indicato con device come disco di riserva.
failed-disk	indica la posizione nell'array del precedente dispositivo indicato con device come disco rotto.

Tabella 6.1: Principali direttive di configurazione di `/etc/raidtab`.

Una volta definite le proprietà dei vari RAID in `/etc/raidtab` si può inizializzare ed attivare ciascuno di essi con il comando `mkraid`, che prende come parametro il file di dispositivo `/dev/mdX` che identifica ciascun array. Con l'opzione `-c` si può specificare in file di configurazione alternativo. Si tenga presente che l'operazione distrugge il contenuto dei dischi, a meno che questi non siano marcati come **failed-disk**; in genere comunque il comando rileva la presenza di dati e si rifiuta di eseguire l'inizializzazione a meno che non la si forzi con l'uso dell'opzione `-f`. Il comando rileva anche se i dispositivi scelti sono già stati utilizzati per un altro RAID, nel qual caso per forzare l'inizializzazione occorrerà usare l'opzione `-R`.

Una volta creato un array lo si può attivare con **raidstart** o si lo può disattivare⁴ con **raidstop**; entrambi i comandi vogliono come argomento il dispositivo da attivare (come identificato in `/etc/raidtab`) oppure si può usare l'opzione **-a** per operare su tutti gli array definiti. Anche per questi comandi si può utilizzare un file di configurazione alternativo specificabile con l'opzione **-c**; per tutti i dettagli si faccia al solito riferimento alla pagina di manuale.

Una volta che si è creato ed attivato un array se ne può controllare lo stato sul file `/proc/mdstat` che riporta tutte le informazioni relative ai dispositivi RAID presenti nel sistema; un esempio di questo file è il seguente:

```
davis:~# cat /proc/mdstat
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 hdc1[1] hda1[0]
      116214080 blocks [2/2] [UU]

unused devices: <none>
```

in cui è presente un solo array in RAID-1, sincronizzato e attivo; quando invece si ha il caso di un array in cui alcuni dischi non sono attivi e devono essere sincronizzati si avrà un risultato del tipo:

```
server:~# cat /proc/mdstat
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 hdc1[0] hda1[1]
      5855552 blocks [2/2] [UU]

md1 : active raid1 hdc2[2] hda2[1]
      69336448 blocks [2/1] [_U]
[>.....] recovery = 0.8% (581632/69336448) finish=183.7m
in speed=6232K/sec
md2 : active raid1 hdc3[2] hda3[1]
      4490048 blocks [2/1] [_U]
```

dove si hanno tre array RAID-1 costruiti con le prime tre partizioni di due dischi, in cui è attivo solo il secondo disco (`/dev/hda`), e si vede come il primo array sia integro, il sistema stia provvedendo alla ricostruzione del secondo, ed il terzo abbia un solo disco attivo.

Una delle funzionalità più interessanti dell'uso di tecnologie RAID che supportano la ridondanza (come RAID-1 e RAID-5) è quella per cui è possibile continuare ad utilizzare il sistema anche senza utilizzare uno dei dischi. In genere un disco diventa inutilizzabile in caso di problemi hardware, però fintanto che fa parte dell'array sarà possibile sostituirlo. Per questo esiste il comando **raidhotremove** che permette di forzare l'uscita di un disco dall'array (ovviamente il comando funziona solo se i restanti dischi sono sufficienti a mantenere attivo l'array). Il comando prende come argomenti il dispositivo dell'array seguito da quello del disco da rimuovere.

L'uso di questo comando può servire ad esempio quando si vuole sostituire un disco, una volta tolto il disco dall'array infatti lo si potrà rimuovere (anche senza riavviare, se il dispositivo è *hot-pluggable*) senza problemi, e senza dover fermare l'array (cosa che ad esempio non sarebbe possibile se su di esso si è posta la directory radice). In sostanza l'effetto è lo stesso che si sarebbe ottenuto creando l'array con la direttiva **failed-disk**, ma può essere ottenuto in qualunque momento.

Così come si può rimuovere un disco da un array, è possibile aggiungerne uno usando il comando **raidhotadd**; in questo caso un disco non attivo (perché rimosso in precedenza o perché marcato **failed-disk** quando si è creato l'array) può essere inserito a caldo nell'array, questo

⁴ovviamente l'array non deve essere in uso.

permetterà ad esempio di sostituire un disco mettendone uno nuovo, e non appena esso sarà reinserito nell'array il kernel provvederà automaticamente ad effettuare la sincronizzazione del contenuto dello stesso ricostruendo l'integrità dell'array, così come mostrato nel secondo esempio del contenuto di `/proc/mdstat`.

6.2 Il sistema del *Logical Volume Manager*

Tratteremo in questa sezione il sistema del *Logical Volume Manager*, uno speciale supporto nel kernel che permette di organizzare in maniera flessibile lo spazio disco per superare le difficoltà di modificare le dimensioni di una partizione qualora il filesystem in essa contenuto non abbia più spazio sufficiente. L'uso del *Logical Volume Manager* permette cioè di evitare alla radice tutta le problematiche, evidenziate in sez. 5.2.2, relative alle diverse possibili strategie di partizionamento.

6.2.1 Introduzione

Uno dei problemi che la struttura di un filesystem unix-like presenta è che quando si riempie un dispositivo montato su una directory non se ne può aumentare la dimensione direttamente semplicemente aggiungendo un altro disco perché questo andrebbe montato altrove. Pertanto in casi come questi si è spesso costretti a complesse operazioni di backup per sostituire il disco riempitosi o a creare link simbolici spostando su altri dischi il contenuto di alcune delle directory un disco riempitosi.

Per superare alla radice questo problema è stato creato il sistema del *Logical Volume Manager*, grazie al quale è possibile creare un filesystem invece che su un disco singolo o su una partizione, all'interno di un “*volume logico*”. Quest'ultimo poi non è altro che un dispositivo virtuale creato a partire da un insieme di *volumi fisici*, che vengono opportunamente *riuniti* dal kernel in modo da presentarsi come un unico disco.

Il vantaggio è che i volumi logici possono essere ridimensionati a piacere senza necessità di toccare le partizioni o i dischi che stanno al di sotto, per cui usando i programmi per il ridimensionamento dei filesystem si possono allargare o restringere questi ultimi in maniera molto comoda (e con certi filesystem alcune operazioni possono essere eseguite a “caldo”, con il filesystem montato). In questo modo se lo spazio disco su un volume logico si esaurisce basterà allargarlo, e se lo spazio disco sottostante non è sufficiente basterà comprare un nuovo disco ed agganciarlo al volume logico, per espanderne le dimensioni senza doversi preoccupare di ripartizionare, montare, fare link e spostare contenuti da un disco ad un altro.

Lo schema del funzionamento del *Logical Volume Manager* (da qui in breve LVM) è riportato in fig. 6.1, il sistema consiste nell'introdurre un livello intermedio, quello del *gruppo di volumi* (indicato in figura con la notazione VG, da *volume group*), che viene a costituire l'interfaccia fra i volumi logici (indicati con LV da *logical volume*), che da esso si estraggono, ed i volumi fisici (indicati con PV, da *physical volume*), che lo vanno a costituire. In sostanza il kernel si cura di suddividere un disco fisico in tante piccole sezioni (dette *physical extent*, e indicate in figura con PE) che poi vengono rimappate⁵ nei corrispondenti *logical extent* (indicati con LE) che vanno a costituire un volume logico.

Una delle funzionalità più interessanti di LVM è quella di poter utilizzare il sistema per poter creare degli *snapshot*, cioè poter creare una copia dello stato di un volume logico ad un certo istante utilizzando lo spazio libero nel *gruppo di volumi* di cui questo fa parte. Tutto ciò viene realizzato allocando dello spazio libero per contenere tutte le modifiche eseguite al volume originario. La versione corrente del filesystem sul volume logico originale registrerà tutte le

⁵da cui il nome di *device mapper* che il sistema ha assunto a partire dal kernel 2.6.x, dove è stato riscritto da zero per migliorare stabilità e prestazioni.

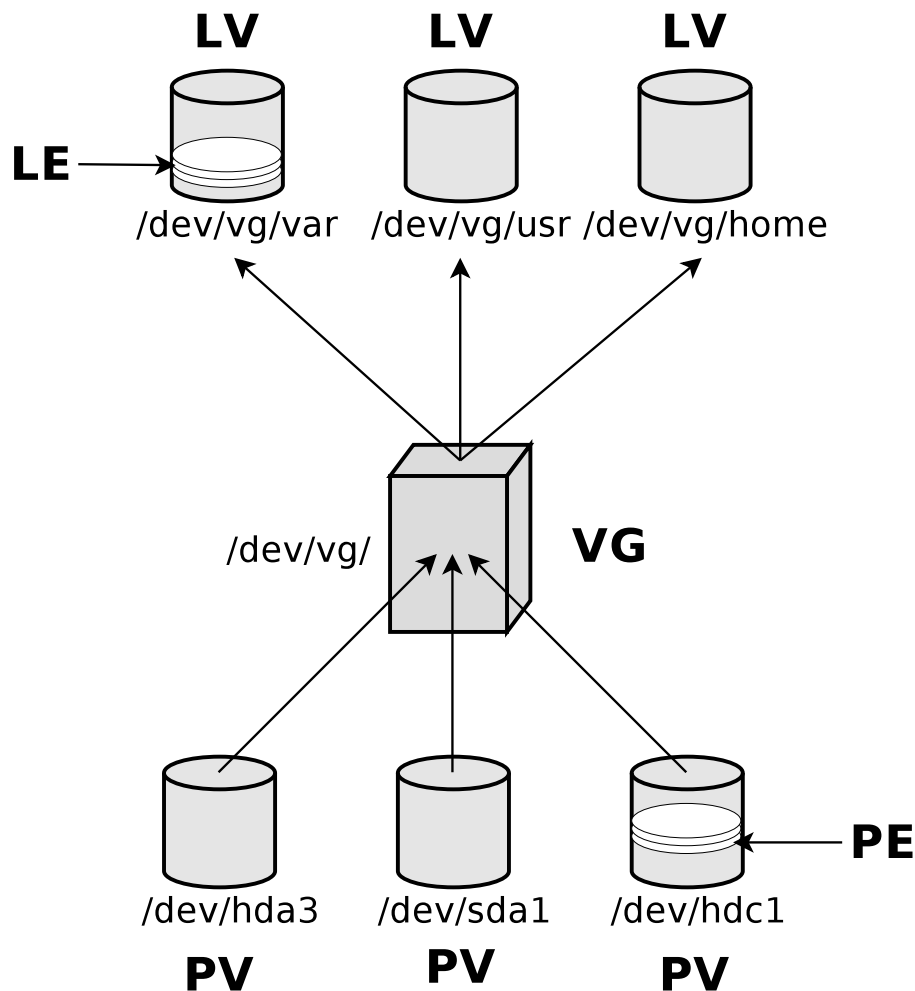


Figura 6.1: Strutturazione del Logical Volume Manager.

modifiche nel nuovo spazio allocato, mentre il volume creato come *snapshot* conterrà i dati così come erano al momento della creazione.

Questo è molto comodo per poter eseguire dei backup per volumi di grosse dimensioni senza doversi preoccupare del fatto che durante l'esecuzione del backup i dati vengano modificati; basta assicurarsi che nel breve tempo necessario a creare lo *snapshot* non ci siano attività, dopo di che si potrà eseguire il backup con tutta calma su di esso mentre le operazioni continuano regolarmente sul volume originale. Una volta completato il backup si potrà rimuovere il volume logico usato per lo *snapshot* ed il volume logico originario manterrà in maniera trasparente tutte le modifiche eseguite nel frattempo; l'unico aspetto critico è quello di assicurarsi di avere allocato per lo *snapshot* spazio sufficiente a contenere tutte le modifiche, se infatti questo dovesse esaurirsi il volume logico originale diverrebbe inusabile.

Per poter utilizzare LVM occorre abilitare il relativo supporto nel kernel (vedi sez. 5.1.3), ed avere installato gli opportuni programmi in user-space. Nel passaggio dal kernel 2.4.x al 2.6.x il sistema è stato completamente riscritto, pertanto si potranno avere delle differenze a seconda che si usi la prima versione (detta anche LVM1) o la seconda (detta LVM2). In ogni caso la nuova versione è in grado di riconoscere la presenza di volumi logici creati con la versione precedente e di riutilizzarli senza problemi ed è in genere più performante, anche se alcune funzionalità sono tutt'ora in fase di adeguamento.

6.2.2 La gestione dei volumi fisici

Il primo passo nell'uso di LVM è quello della inizializzazione dei volumi fisici; per farlo prima occorre prima marcare la partizione che useremo (vedi sez. 5.2.2) con il codice 0x8E per indicare che verrà utilizzata come volume fisico e poi inizializzarla con il comando `pvccreate`. Questo prende come argomento il file di dispositivo che identifica il volume fisico (se ne possono specificare anche più di uno) sul quale crea l'infrastruttura per mantenere le informazioni relative a LVM (come il *gruppo di volumi* di cui andrà a fare parte o l'identificatore univoco assegnato al volume).

Il comando `pvccreate` può essere applicato anche ad un intero disco, nel qual caso però occorre prima cancellare la tabella delle partizioni (se questa viene rilevata il comando segnala un errore). Questo si può fare facilmente con un comando di cancellazione come:

```
dd if=/dev/zero of=/dev/hdX bs=512 count=1
```

Il comando prende come opzioni specifiche `-u`, che permette di specificare uno UUID⁶) a mano invece di farlo generare automaticamente al comando e `-f` che forza la creazione del volume senza richiedere conferma (distruggendo eventuali dati presenti). Infine (ma solo con la versione di LVM2) con `-M` si può indicare quale versione usare per il formato meta-dati (specificato con il numero con `lvm1` o `lvm2`). Le altre opzioni riguardano aspetti specifici della gestione interna ed al solito rimandiamo alla pagina di manuale per una trattazione più dettagliata.

Con il comando `pvscan`, si può eseguire una scansione dei dischi presenti per rilevare al loro interno la presenza di volumi fisici usati da LVM, il comando si invoca senza argomenti, e si avrà un risultato del tipo:

```
monk:/home/piccardi/Truelite/documentazione/corso# pvscan
pvscan -- reading all physical volumes (this may take a while...)
pvscan -- ACTIVE   PV "/dev/hda3" of VG "vg" [54.51 GB / 19.51 GB free]
pvscan -- total: 1 [54.52 GB] / in use: 1 [54.52 GB] / in no VG: 0 [0]
```

che mostra la presenza di un unico volume fisico completamente utilizzato.

Le opzioni del comando controllano sostanzialmente l'informazione che viene stampata a video, con `-s` si riduce la stessa al minimo, con `-u` si fanno stampare gli UUID presenti, con `-n` si stampano solo i volumi fisici che non appartengono ad un volume logico, per le altre opzioni si faccia riferimento alla pagina di manuale.

Un secondo comando diagnostico è `pvdisplay` che stampa a video le caratteristiche di uno più volumi fisici (specificati come argomenti), nel nostro caso avendo visto che abbiamo un volume fisico su `/dev/hda3` potremo ottenere:

```
monk:/home/piccardi/Truelite/documentazione/corso# pvdisplay /dev/hda3
--- Physical volume ---
PV Name           /dev/hda3
VG Name           vg
PV Size           54.52 GB [114334605 secs] / NOT usable 4.19 MB [LVM: 182 KB]
PV#               1
PV Status         available
Allocatable       yes
Cur LV           3
PE Size (KByte)   4096
Total PE          13955
Free PE           4995
Allocated PE      8960
PV UUID           ZeyhF7-CK5M-HXup-rc0f-Ji1U-UKqq-3rr4yE
```

⁶lo *Universal Unique IDentifier* è un numero a 128 bit utilizzato per identificare univocamente un oggetto, è composto di varie sezioni, fra cui un riferimento ad un *MAC address* per il computer locale, una marca temporale e una parte casuale per garantire unicità.

che ci mostra tutte le caratteristiche di quest'ultimo.

Anche per questo comando le opzioni servono a controllare l'informazione riportata; le principali sono: **-s** che fa stampare solo la dimensione totale del volume, **-c** che produce un output coi dati separati da un carattere ":" così da poterlo usare facilmente all'interno di script. Per un elenco completo e i dettagli si può fare riferimento alla pagina di manuale.

Infine un comando estremamente utile è **pvmove** che permette di spostare tutti i dati mantenuti su un certo volume fisico su di un altro (così da poterlo rimuovere dal gruppo di volumi di cui fa parte). Il comando prende come argomenti il volume fisico sorgente da cui rimuovere i dati e quello di destinazione su cui inviarli; ovviamente la destinazione deve avere spazio libero sufficiente a contenerli. Si può anche evitare di specificare un volume fisico di destinazione, nel qual caso lo spazio sarà redistribuito automaticamente sui volumi appartenenti allo stesso gruppo.

Il comando ha tre opzioni principali, la prima è **-i** che fa stampare una percentuale di completamento delle operazioni tutte le volte che è trascorso il numero di secondi passato come parametro; con **-n** invece si può restringere lo spostamento ai dati relativi al solo volume logico il cui nome si è passato come parametro. Infine con **--abort** si può abortire un precedente tentativo di spostamento non completato; se infatti il comando viene interrotto durante l'esecuzione non c'è nessun problema, e basterà reinvocarlo senza parametri per fare ripartire lo spostamento dal punto un cui era stato interrotto, a meno appunto di invocare **pvmove --abort**.

6.2.3 La gestione dei gruppi di volumi

Una volta inizializzati tutti i volumi fisici necessari il secondo passo è combinarli in un *gruppo di volumi*, per questo si deve usare il comando **vgcreate** che crea un nuovo gruppo di volumi assegnandogli il nome specificato come primo argomento, ed inserendoci tutti i volumi fisici specificati negli argomenti successivi; perché il comando abbia successo tutti i volumi fisici devono essere stati preventivamente inizializzati con **pvcreate**. Una volta creato un gruppo di volumi con nome **vgname** si avrà a disposizione una corrispondente directory **/dev/vgname/** in cui si troveranno tutti i file (in particolare i file relativi ai volumi logici, che tratteremo in sez. 6.2.4).

L'opzione **-s** permette di definire le dimensioni dei *physical extent* in cui saranno divisi i volumi fisici, da passare come argomento numerico seguito dalle estensioni standard per indicare kilo, mega e gigabyte. La dimensione standard è di 4Mb, ed i valori possono spaziare da 4kb a 16Gb in potenze di due, si tenga presente che un volume logico può mantenere un massimo di 64k *physical extent* che comporta una dimensione massima di 256Gb, se si ha a disposizione più spazio disco occorrerà usare delle dimensioni più ampie per i *physical extent*.

L'opzione **-l** permette di definire un numero massimo di volumi logici che possono essere estratti dal gruppo; il valore massimo assoluto è di 256, con **-p** invece si imposta il numero massimo di volumi fisici che possono essere inclusi, ed il massimo assoluto è sempre lo stesso. Per le ulteriori opzioni si può fare riferimento alla pagina di manuale, un elenco di opzioni comuni a buona parte dei comandi che operano con i gruppi ed i volumi logici si trova in tab. 6.2, l'elenco completo invece è riportato nella pagina di manuale dedicata a LVM, accessibile con **man lvm**.

Una volta creato un gruppo di volumi questo può essere espanso aggiungendovi altri volumi fisici con **vgextend**, che di nuovo prende come primo argomento il nome del gruppo (che stavolta deve esistere) seguito dai volumi fisici che si vogliono aggiungere (al solito specificati tramite il relativo file di dispositivo). Per i dettagli sulle opzioni (alcune delle quali sono riportate in tab. 6.2) si può fare riferimento alla pagina di manuale.

Il lavoro opposto, quello di rimuovere un volume da un gruppo, viene invece eseguito da **vgreduce** che al solito prende come primo argomento il nome del gruppo, seguito dai volumi che si vogliono rimuovere. Ovviamente perché il comando abbia successo tutti i volumi che si vogliono rimuovere dal gruppo non devono essere utilizzati; con l'opzione **-a** non è necessario

Opzione	Significato
-v	imposta il livello di prolissità, può essere ripetuto fino a tre volte per aumentare le informazioni stampate a video.
-d	importa il livello di debug, può essere ripetuto fino a sei volte per aumentare le informazioni stampate a video.
-t	esegue il comando in modalità di test, i dati non vengono modificati.
-h	stampa un messaggio di aiuto.
-M	imposta il formato dei meta-dati, prende come valore <code>lvm1</code> o <code>lvm2</code> (o semplicemente il numero), questa opzione è disponibile solo su LVM2.
-A	esegue automaticamente il backup dei metadati quando essi vengono cambiati, prende come valore <code>y</code> o <code>n</code> , (il default è il primo e non è il caso di cambiarlo).
-P	cerca di accedere al meglio ai dati quando i gruppi di volumi sono solo parzialmente accessibili.

Tabella 6.2: Principali opzioni comuni dei comandi di gestione di LVM.

specificare quali volumi rimuovere, dato che in tal caso si rimuoveranno tutti quelli non utilizzati. Al solito per i dettagli sul comando e le altre opzioni supportate si faccia riferimento alla pagina di manuale.

La creazione di un gruppo di volumi è comunque solo un primo passo, prima di poterlo utilizzare infatti questo deve essere *attivato* in modo da poter essere utilizzato. Per questo si usa in genere il comando `vgchange` il cui scopo è quello di modificare le proprietà di un gruppo di volumi. Il comando al solito prende come argomento il nome del gruppo di volumi, ma se non se ne specifica uno si applica a tutti quelli presenti.

L'opzione che ci interessa è `-a` che è quella che permette di attivare o disattivare il gruppo, rispettivamente con i parametri `"y"` e `"n"`; in genere gli script di avvio di LVM eseguono sempre un comando del tipo `vgchange -a y`. Altre opzioni sono `-l` che modifica il numero massimo di volumi logici, passato come parametro (deve essere eseguita su un gruppo inattivo) e `-x` che attiva o disattiva (sempre con i parametri `"y"` e `"n"`) la possibilità di estendere o ridurre il gruppo togliendo o aggiungendo volumi fisici.

Un secondo comando usato all'avvio del sistema (ma solo per LVM1, in LVM2 la cosa è automatica) è `vgscan` che esegue una scansione dei dischi presenti per rilevare la presenza di gruppi di volumi (analogo a quanto fa `pvscan` per i volumi fisici), in genere lo si lancia prima di `vgchange`, e l'unica opzione significativa (per le altre si consulti la pagina di manuale) è `--mknodes` che crea in `/dev` i file speciali necessari all'uso di dei dispositivi. Un esempio di questo comando è:

```
monk:~# vgscan
vgscan -- reading all physical volumes (this may take a while...)
vgscan -- found active volume group "vg"
vgscan -- "/etc/lvmtab" and "/etc/lvmtab.d" successfully created
vgscan -- WARNING: This program does not do a VGDA backup of your volume group
```

Esiste poi un comando analogo di `pvdisplay` da usare per i gruppi di volumi; con `vgdisplay` si possono elencare le caratteristiche del gruppo passato come argomento, o di tutti quelli presenti nel sistema se non si specifica nulla. Le opzioni sono le stesse già viste in sez. 6.2.2 per `pvdisplay`; un esempio di questo comando è:


```

monk:~# vgdisplay
--- Volume group ---
VG Name                vg
VG Access               read/write
VG Status               available/resizable
VG #                   0
MAX LV                 256
Cur LV                 3
Open LV                 3
MAX LV Size             255.99 GB
Max PV                 256
Cur PV                 1
Act PV                 1
VG Size                 54.51 GB
PE Size                 4 MB
Total PE                13955
Alloc PE / Size         8960 / 35 GB
Free PE / Size          4995 / 19.51 GB
VG UUID                 ePD2qd-BolC-6hOT-Omm5-Th01-Aqbl-BNbds0

```

Quando si vogliono riunire in uno solo due gruppi di volumi si può utilizzare il comando **vgmerge**, questo prende come primo argomento il gruppo di destinazione, che diventerà l'unione dei due, e come secondo il gruppo da inglobare nel primo, che deve essere disattivo. Il comando funziona soltanto se i due volumi sono stati creati con parametri compatibili (stessa versione dei metadata e stessa dimensione dei *physical extent*) e se l'unione dei due rispetta i limiti presenti sul gruppo di destinazione. Per le opzioni (che sono sostanzialmente quelle comuni di tab. 6.2) e per gli altri dettagli si faccia riferimento alla pagina di manuale.

Se invece si vuole spezzare in due un gruppo di volumi si può usare il comando **vgsplit**, questo prende come primo argomento il nome del gruppo da cui si parte e come secondo quello del nuovo gruppo che si vuole creare, seguito dalla lista dei volumi fisici da inserire in quest'ultimo. Dato che i volumi logici non possono essere suddivisi fra gruppi diversi occorrerà che ciascun volume logico esistente sia compreso interamente nei volumi fisici che restano in uno dei due gruppi. Al solito per le opzioni (sono solo quelle comuni) ed i dettagli si può fare riferimento alla pagina di manuale.

6.2.4 La gestione dei volumi logici

Una volta creato un *gruppo di volumi* sarà possibile estrarre da esso lo spazio necessario a creare un *volume logico*. Il comando principale per eseguire questo compito è **lvcreate** che crea il volume logico allocando i *logical extent* necessari dalla riserva di *physical extent* disponibili.

Il comando richiede come argomento obbligatorio il nome del gruppo di volumi da utilizzare, inoltre deve essere sempre specificata la dimensione del volume fisico che si va a creare; questo può essere fatto in due modi, o con l'opzione **-l**, seguita dal numero di *logical extent* da utilizzare,⁷ o con l'opzione **-L** seguita dalla dimensione in byte; l'opzione supporta anche la specificazione di altre unità di misura come kilobyte, megabyte, gigabyte o terabyte se si aggiunge al numero passato come parametro l'iniziale (va bene sia maiuscola che minuscola) della misura utilizzata.

Se non si specifica altro il comando crea un volume logico con un nome assegnato automaticamente al valore **lvolN** dove **N** è il numero progressivo usato internamente dal sistema; si può però specificare un nome qualunque con l'opzione **-n lvname**; in questo caso si potrà accedere al nuovo volume logico usando come file di dispositivo **/dev/vgname/lvname**. Nei kernel della serie 2.6, dato che LVM è stato riscritto appoggiandosi direttamente al sistema del *device mapper*, si

⁷le dimensioni sono le stesse dei *physical extent* sottostanti, definite in fase di creazione del gruppo.

potranno trovare i volumi logici anche direttamente sotto la directory `/dev/mapper`, accessibili direttamente con un nome del tipo `vgname-lvname`.

Una delle funzionalità di LVM è che permette di suddividere i dati su più volumi fisici in maniera analoga al RAID-0, definendo con l'opzione `-i` un numero di *stripes* su cui distribuire i dati. Valgono in questo caso le stesse considerazioni viste in sez. 6.1.1 per il RAID-0.⁸ Se si utilizza questa funzionalità si può anche specificare la dimensione delle *stripes* con l'opzione `-I`.

Se invece di un normale volume logico si vuole creare uno *snapshot* occorre usare modalità completamente diversa di invocazione del comando. In tal caso occorre usare l'opzione `-s` e specificare un nome con l'opzione `-n`; inoltre gli argomenti obbligatori diventano due: il nome del volume logico da usare come *snapshot* e quello del volume logico cui fare da *snapshot*. Un elenco delle opzioni principali del comando `lvcreate` è riportato in tab. 6.3, il comando utilizza anche molte delle opzioni generali accennate in tab. 6.2; per i dettagli al solito si può fare riferimento alla pagina di manuale.

Opzione	Significato
<code>-i</code>	indica di utilizzare il numero di <i>stripe</i> passato come parametro per distribuire i dati sui volumi fisici sottostanti.
<code>-I</code>	imposta la dimensione di una <i>stripe</i> nella distribuzione dei dati su più volumi fisici (da specificare con un numero <code>n</code> da 2 a 9 per indicare 2 ⁿ kilobyte).
<code>-l</code>	imposta la dimensione di un volume fisico per numero (passato come parametro) di <i>logical extent</i> .
<code>-L</code>	imposta la dimensione di un volume fisico specificata in byte, kilobyte, megabyte, gigabyte o terabyte.
<code>-n</code>	imposta il nome del volume logico, passato come parametro.
<code>-p</code>	importa i permessi sul volume logico, prende il valore <code>r</code> per indicare sola lettura o <code>rw</code> per indicare lettura e scrittura.
<code>-r</code>	importa il numero di settori da leggere in anticipo per migliorare le prestazioni (solo per LVM1).
<code>-s</code>	indica la richiesta di creazione di un volume di <i>snapshot</i> .

Tabella 6.3: Principali opzioni del comando `lvcreate`.

Come per i volumi fisici ed i gruppi di volumi sono presenti due comandi, `lvscan` e `lvdisplay`, che possono essere usati rispettivamente per eseguire una scansione dei volumi logici presenti e per ottenere un sommario delle relative proprietà. Anche le opzioni sono analoghe e non staremo a ripeterle.

La comodità di LVM consiste nel fatto che una volta che si è creato un volume logico si potrà accedere a quest'ultimo con il relativo file di dispositivo e trattarlo come un qualunque disco; il vantaggio è che qualora si avesse la necessità di ampliarne le dimensioni si potrà ricorrere al comando `lvextend`, dopo di che non resterà che ridimensionare il filesystem presente sul volume logico (vedi sez. 6.2.5) per avere a disposizione lo spazio aggiunto.

Il comando richiede obbligatoriamente come argomento il volume da estendere e la dimensione finale dello stesso che deve essere specificata o con l'opzione `-l` in numero di *logical extent* o direttamente con l'opzione `-L` (con la stessa convenzione di `lvcreate`). Si possono però usare le stesse opzioni per specificare la dimensione dell'estensione al posto di quella totale, apponendo al valore specificato il carattere `+`.

Il comando supporta anche l'uso delle *stripe* (con le stesse opzioni di tab. 6.3) ed inoltre si può specificare come argomenti ulteriori una lista di volumi fisici sui quali si vuole che sia

⁸in particolare si avrà un enorme degrado di prestazioni qualora si usassero partizioni diverse dello stesso disco come volumi fisici su cui distribuire le *stripes*.

compiuta l'estensione del volume logico. Il comando permette anche di estendere le dimensioni dei volumi di *snapshot*.

Se invece di aumentare le dimensioni di un volume logico le si vogliono ridurre si deve usare **lvreduce**. Il comando prende gli stessi argomenti ed opzioni di **lvextend** (esclusi quelli relativi alle *stripe*), solo che in questo caso la dimensione può specificata apponendovi un segno “-” per richiedere una riduzione. Si tenga presente che i dati presenti nella parte tolta con la riduzione vengono persi, pertanto è fondamentale ridurre le dimensioni di un eventuale filesystem presente sul volume logico *prima* di eseguirne una riduzione.

Per completare i comandi di ridimensionamento si deve citare il generico **lvresize** che unisce le funzionalità dei precedenti **lvreduce** e **lvextend** permettendo ridimensionamenti in qualunque direzione (sia crescenti che decrescenti). Di nuovo le opzioni sono le stesse, e stavolta sono permessi per le dimensioni sia il segno “+” che quello “-”.

Qualora un volume logico non sia più utilizzato lo si può eliminare usando **lvremove**, il comando prende come argomento uno o più volumi da rimuovere, l'unica opzione specifica supportata è **-f** che permette di sopprimere la richiesta di conferma della rimozione dallo standard input.

6.2.5 Il ridimensionamento dei filesystem

Benché non direttamente attinente ad LVM conviene trattare qui i comandi per il ridimensionamento dei filesystem dato che il loro uso più comune è appunto in combinazione con i comandi per ridurre o aumentare le dimensioni dei volumi logici di LVM.⁹

La possibilità di ridimensionare un filesystem dipende dalla presenza di un opportuno programma specifico per ciascun tipo di filesystem. Correntemente la gran parte dei filesystem presenti su Linux (ext2/ext3, reiserfs, XFS,¹⁰ JFS) è dotata di questa funzionalità. In generale questa operazione può essere eseguita solo a filesystem inattivo (cioè senza che questo sia montato), il che comporta degli ovvi problemi se si vuole ridimensionare la directory radice. Alcuni filesystem però supportano anche il ridimensionamento *a caldo* (cioè senza smontare il filesystem) se questo avviene per accrescimento.

Come accennato ogni filesystem ha normalmente un suo programma per il ridimensionamento, fa eccezione JFS, che si ridimensiona con una opzione di **mount**, per ridimensionare un filesystem JFS infatti basterà rimontarlo con un comando del tipo:

```
mount -t jfs -o remount,resize=10G /dev/vg/home /home
```

dove se non si specifica nessun parametro per l'opzione **resize** il ridimensionamento viene fatto alla dimensione totale del dispositivo sottostante.

Il comando che permette di ridimensionare un filesystem ext2 (o ext3, la struttura dei due è identica) è **resize2fs**; il programma richiede che il filesystem sia smontato (anche se il supporto per le operazioni a caldo è disponibile come patch) e prende come primo argomento il dispositivo su cui si trova il filesystem da ridimensionare. Se non viene specificato un secondo argomento indicante la dimensione da usare il ridimensionamento avviene alla dimensione corrente del dispositivo.¹¹ La dimensione può essere indicata con la convenzione sulle unità di misura già illustrata in precedenza che prevede che queste siano specificate posponendo al numero la relativa iniziale.

Si tenga presente che il comando non tiene assolutamente in considerazione le dimensioni del dispositivo sottostante, per cui se si usa una dimensione troppo grande si avranno problemi,

⁹questi comandi sono comunque validi qualunque sia il dispositivo su cui è mantenuto il filesystem, e possono anche essere utilizzati quando si ridimensiona una partizione su un disco con **parted**.

¹⁰che però supporta solo *accrescimenti*.

¹¹questo significa che l'argomento non è necessario in caso di espansione e lo è in caso di riduzione.

inoltre se si ridimensiona una partizione per poi ridimensionare il filesystem occorre avere cura che essa continui ad avere lo stesso punto di inizio, altrimenti il programma non potrà funzionare.

Il comando prevede l'opzione **-d** per abilitare il debug (che prende come parametro una bit-mask indicante quali informazioni stampare), **-p** che stampa una percentuale di completamento durante le operazioni, **-f** che forza le operazioni escludendo alcuni controlli di sicurezza e **-F** che scarica i dati in cache prima di iniziare.

Per ridimensionare un filesystem di tipo `reiserfs` si deve invece usare il comando `resize_reiserfs`, passando come argomento il dispositivo contenente il filesystem da ridimensionare. La nuova dimensione si specifica con l'opzione **-s** seguita dal valore (con la solita convenzione per le unità di misura). L'opzione supporta anche l'uso dei segni “+” e “-” per indicare dei ridimensionamenti relativi (accrescimenti o riduzioni); se non si specifica nulla il comando userà la dimensione del dispositivo sottostante.

Il comando permette anche di specificare con l'opzione **-j** un dispositivo alternativo su cui mantenere il giornale, e di forzare le operazioni senza eseguire controlli con **-f**, per avere più informazioni nel corso delle operazioni si deve invece usare **-v**.

In sez. 5.2.2 abbiamo visto come `fdisk` ed analoghi consentono soltanto la manipolazione del contenuto della tabella delle partizioni. Quando su un disco sono già presenti dei dati questo in genere significa che al più si potranno cancellare alcune delle partizioni per poi riutilizzare lo spazio ottenuto o al più allargare o stringere una partizione esistente, posto che nel primo caso si sia opportunamente ristretto il filesystem in esso contenuto, e che nel secondo sia disponibile dello spazio libero in coda alla partizione e si allarghi poi il filesystem.

Un comando che consente di unire le funzionalità dei comandi per il partizionamento e di quelli per il ridimensionamento dei file¹² è **parted**, che consente non solo la creazione e la cancellazione delle partizioni, ma anche il loro ridimensionamento e spostamento senza perderne il contenuto. Una caratteristica interessante del programma è che ne è disponibile una versione su disco di avvio che consente di ripartizionare un sistema prima di avviarlo.

Il programma opera normalmente in modalità interattiva, e lo si può lanciare direttamente ottenendo un prompt da cui invocare i comandi interni, fra questi c'è **help** che ne stampa un elenco corredato dei relativi argomenti e del significato degli stessi.

6.3 Le quote disco

Una delle caratteristiche avanzate della gestione dei dischi presenti fin dal kernel 2.0 è quella delle quote disco, un meccanismo che consente di limitare la quantità delle risorse su disco che i singoli utenti (o gruppi) possono usare. Tratteremo in questa sezione il loro utilizzo e le relative modalità di gestione.

6.3.1 Visione generale

Una delle risorse principali che gli amministratori di sistema devono gestire è quella dello spazio disco; uno dei problemi più comuni è infatti quello del riempimento del disco, e questo avviene sovente perché alcuni utenti non si curano di controllare le risorse che usano, e finiscono per riempire tutto lo spazio disponibile, a scapito pure degli altri utenti.

Per questo motivo una delle caratteristiche presenti da sempre in sistemi multiutente è quella di poter imporre delle restrizioni allo spazio disco utilizzato dai singoli utenti. Nel caso specifico di un sistema unix-like questo è sempre stato realizzato attraverso le cosiddette *quote disco*, un meccanismo, realizzato nel kernel, che pone delle restrizioni alle risorse che un singolo utente può utilizzare.

¹²ed anche di quelli di controllo e riparazione.

L'idea è che in questo modo esiste un limite di occupazione delle risorse oltre il quale ogni utente non può andare. La struttura del meccanismo di controllo opera sia sulla quantità di spazio disco (cioè sul numero di *blocchi*) che sul numero di file (cioè sul numero di *inode*) da mettere a disposizione; le due risorse sono completamente indipendenti per cui è possibile superare il limite sul numero di file occupati pur avendo ancora spazio disco disponibile.

Le quote sono sempre calcolate per ogni singolo filesystem (che deve supportare il meccanismo¹³), pertanto è usuale attivarle sul filesystem su cui si sono piazzate le home degli utenti. Si tenga presente comunque che se gli utenti possono scrivere su più filesystem le quote opereranno in maniera indipendente su ciascuno di essi, e dovranno perciò essere impostate separatamente.

Un'ultima caratteristica generale delle quote disco è che il controllo sulla occupazione delle risorse può avvenire a sia livello di un singolo utente (cioè in base all'UID), che di un gruppo (cioè in base al GID), che di entrambi. In pratica, si avrà la possibilità di utilizzare il disco per aggiungere un certo file se questo è consentito dalla quota dell'utente cui apparterrà il file in questione o se è consentito dalla quota di gruppo (sempre per il gruppo di appartenenza del file).

6.3.2 Configurazione del sistema delle quote

Per poter utilizzare le quote disco occorre anzitutto abilitare il relativo supporto nel kernel nella sezione **File systems** della configurazione. Del sistema esistono due versioni, entrambe supportate per compatibilità, anche se nei nuovi sistemi è preferibile usare la più recente. Si tenga presente che se si vogliono usare le quote con NFS occorre che sia stato opportunamente avviato il relativo demone per la gestione (vedi sez. 8.4).

Se si vuole utilizzare il controllo delle quote il secondo passo è segnalarne la presenza in fase di montaggio del filesystem perché il kernel sia successivamente in grado di mantenere i dati relativi all'uso del disco ed applicare le restrizioni, questo viene fatto utilizzando due opzioni specifiche per `/etc/fstab`: per le quote gli utenti si usa `usrquota` mentre per quelle dei gruppi `grpquota`, un esempio di utilizzo di queste opzioni è il seguente:

```
/dev/vg/home    /home    ext3    defaults,usrquota,grpquota    0    1
```

Una volta abilitato il supporto nel kernel e montato opportunamente il filesystem il passo successivo è quello di avere presenti¹⁴ nella radice del relativo filesystem (la directory corrispondente al *mount point*) dei due file `quota.user` e `quota.group` se si usa la versione 1 del supporto o `aquota.user` e `aquota.group` se si usa la versione 2. E in questi file che sono mantenuti i dati relativi all'uso del disco da parte di utenti e gruppi rispettivamente.

In generale non è necessario creare questi file a mano,¹⁵ se sono assenti infatti la prima volta che si usa il comando `quotacheck` verranno creati automaticamente. Questo comando deve essere sempre eseguito su un filesystem montato con le opzioni per le quote prima di abilitare le stesse (con un successivo `quotaon`) in modo da ricostruire lo stato dell'occupazione delle risorse sul disco; se infatti le informazioni di `aquota.user` e `aquota.group` non sono aggiornate il meccanismo delle quote non funziona in maniera corretta.

Il comando prende come argomento il file di dispositivo su cui è posto il filesystem da controllare o il rispettivo *mount point* presente in `/etc/fstab`, ma questo argomento può essere tralasciato con l'uso dell'opzione `-a` che controlla tutti i filesystem montati (tranne quelli montati via NFS).

¹³con gli ultimi kernel il supporto è presente per ext2, ext3, e reiserfs, mentre per XFS la caratteristica deve essere esplicitamente abilitata dato che è gestita in modo diverso.

¹⁴questo non è vero per le quote su XFS, in detto filesystem infatti le quote sono mantenute nei metadati e non su file visibili.

¹⁵alcuni HOWTO indicano di crearli vuoti con il comando `touch` se non sono presenti.

Opzione	Significato
-a	controlla tutti i filesystem montati.
-R	se specificata con -a non controlla la radice.
-F	controlla le quote nel formato specificato, prende i valori: vfsold (per la versione 1), vfsv0 (per la versione 2), rpc (per le quote su NFS) e xfs (per le quote su XFS).
-u	richiede l'esecuzione del controllo per le sole quote degli utenti.
-g	richiede l'esecuzione del controllo per le sole quote dei gruppi.
-c	non legge i file con i dati delle quote esistenti.
-v	riporta lo stato di avanzamento delle operazioni.
-m	forza il controllo delle quote senza rimontare il filesystem in sola lettura.
-M	esegue il controllo anche con il filesystem montato in scrittura se il tentativo di rimontarlo in sola lettura fallisce.

Tabella 6.4: Principali opzioni del comando `quotacheck`.

Il comando costruisce una tabella con dell'uso corrente del disco e la confronta, a meno di non usare l'opzione `-c` (nel qual caso detti file non vengono letti) con i dati memorizzati su `aquota.user` e `aquota.group`; se trova delle incoerenze aggiorna le informazioni. In genere il comando necessita di montare il filesystem in sola lettura, onde evitare delle variazioni dello spazio occupato durante il calcolo; si può forzare l'esecuzione se il filesystem non è montabile in sola lettura usando l'opzione `-m`.

Di default il comando controlla solo le quote utente, se si vogliono controllare solo le quote gruppo occorre specificarlo esplicitamente usando l'opzione `-g`, se le si vogliono controllare entrambe va specificata esplicitamente anche l'opzione `-u`. Le altre opzioni principali sono riportate in tab. 6.4, per l'elenco completo ed i dettagli al solito si faccia riferimento alla pagina di manuale.

Una volta che si è montata una partizione con il controllo delle quote perché questo sia applicato è necessaria una attivazione esplicita con il comando `quotaon`; come per `quotacheck` il comando richiede come argomento uno (o più) file di dispositivo su cui è situato il filesystem con le quote o l'opzione `-a` per attivarli tutti. Il comando prende le opzioni `-u` e `-g` con lo stesso significato di `quotacheck` per attivare le quote su utenti e gruppi; al solito di default sono attivate soltanto quelle sugli utenti. L'opzione `-v` aumenta la prolissità del comando stampando ogni filesystem per il quali si sono attivate le quote, mentre con `-p` si stampa solo lo stato delle stesse invece di attivarle. Per le altre opzioni ed i dettagli di funzionamento al solito si può fare riferimento alla pagina di manuale.

In sostanza per poter utilizzare le quote, una volta modificato `/etc/fstab` per indicare i filesystem su cui usarle, si dovrà provvedere a lanciare negli script di avvio (subito dopo il montaggio dei filesystem) un opportuno script che invochi prima `quotacheck` e poi `quotaon`; gran parte delle distribuzioni installano automaticamente uno script di questo tipo con l'installazione dei programmi di controllo delle quote.¹⁶

Il sistema del controllo delle quote può anche essere disabilitato utilizzando `quotaoff`. Il comando (che è equivalente a chiamare `quotaon` con l'opzione `-f`) e notifica al kernel la richiesta di non effettuare più controlli sul superamento delle quote. Il comando prende gli stessi argomenti di `quotaon` (in realtà si tratta dello stesso eseguibile, solo invocato con un nome diverso), e le due opzioni `-v` e `-p` hanno lo stesso effetto descritto in precedenza.

¹⁶nel caso di Debian lo script è `/etc/init.d/quota` e fa parte del pacchetto omonimo.

6.3.3 Gestione delle quote di utenti e gruppi

Una volta attivato il configurato il supporto delle quote, e attivato lo stesso con i passi appena visti, il passo successivo è quello di impostare quali restrizioni (in termini di spazio disco e di inode disponibili) si vogliono imporre ai vari utenti (e gruppi).

Come accennato in sez. 6.3.1 le quote si applicano in maniera indipendente per ciascun filesystem, inoltre se si ricorda quanto illustrato in sez. 5.2.3, un filesystem unix-like ha due tipi di risorse, i *blocchi* e gli *inode*, i primi sono relativi allo spazio disco utilizzabile, i secondi al numero di file, e su entrambi possono essere applicate delle restrizioni in maniera indipendente.

La struttura del meccanismo delle quote disco prevede la presenza di due limiti, uno *morbido* (detto appunto *soft limit*) che può essere superato per brevi periodi di tempo, ed uno *duro* (detto *hard limit*) che non può mai venir superato. Il breve periodo di tempo durante il quale è possibile superare il primo limite è detto *periodo di grazia* (*grace period*), durante detto periodo è ancora possibile creare nuovi file o usare spazio disco superando il limite *morbido*, ma comunque mai al di là del limite *duro*.

Il programma che permette di impostare le quote (ovviamente solo all'amministratore) è *edquota*, e nella sua forma più semplice prende come argomento l'utente di cui si vogliono modificare le quote, il comando infatti per default agisce sulle quote utente, se si vuole operare su un gruppo occorre specificare l'opzione *-g*. Il comando crea un file di testo temporaneo con la rappresentazione delle quote correnti per l'utente o il gruppo scelto, e lancia l'editor di default (in genere *vi* o quello stabilito dalla variabile di ambiente *EDITOR*), all'interno del quale apparirà un contenuto del tipo:

```
Quotas for user piccardi:
/dev/hda3: blocks in use: 2594, limits (soft = 5000, hard = 6500)
          inodes in use: 356, limits (soft = 1000, hard = 1500)
```

le informazioni riguardo l'uso corrente sono riportate solo per riferimento, e una loro modifica verrà ignorata, modificando invece i valori dei limiti fra parentesi, salvando il file ed uscendo dall'editor questi verranno utilizzati come nuovi limiti; un limite può essere annullato usando il valore 0.

In realtà è possibile specificare come argomenti più utenti (o gruppi) ed utilizzare l'opzione *-p* per specificare un utente (o gruppo) di riferimento le cui impostazioni dovranno essere utilizzate per tutti gli altri; in questo modo è possibile effettuare una impostazione generale per un singolo utente senza doverla poi ripetere a mano per tutti gli altri.

Se non si specifica nulla il comando imposta le quote per tutti i filesystem che le supportano; si può delimitare l'azione del programma ad un solo filesystem specificando quest'ultimo (indicato per file di dispositivo) con l'opzione *-f*.

Infine con l'opzione *-t* si può impostare il *periodo di grazia*, questa è una proprietà globale e può essere impostata solo a livello di tutti gli utenti (o tutti i gruppi) di un singolo filesystem; in tal caso di nuovo sarà aperto un file temporaneo all'interno dell'editor di default, il cui contenuto sarà qualcosa del tipo:

```
Time units may be: days, hours, minutes, or seconds
Grace period before enforcing soft limits for users:
/dev/hda2: block grace period: 0 days, file grace period: 0 days
```

Con la prima versione della gestione delle quote specificare un valore nullo per il periodo di grazia significava utilizzare il valore del periodo di default (una settimana) con le nuove versioni è necessario impostare esplicitamente un limite.

Un elenco delle principali opzioni del comando *edquota* è riportato in tab. 6.5, per i dettagli e l'elenco completo si faccia al solito riferimento alla pagina di manuale.

Opzione	Significato
-f	limita le impostazioni filesystem specificato.
-u	richiede l'impostazione delle quote per un utente.
-g	richiede l'impostazione delle quote per un gruppo.
-t	imposta il <i>periodo di grazia</i> .
-p	specifica un utente di riferimento.

Tabella 6.5: Principali opzioni del comando `edquota`.

Una volta attivare le quote si possono ci sono due comandi che permettono di verificarne lo stato, il primo è `quota` che permette ad un utente di verificare le condizioni delle sue quote disco, il comando con l'opzione `-u` (il default, può essere omessa) stampa un resoconto delle quote utente, se si vuole controllare le quote di gruppo occorre usare l'opzione `-g`.

L'amministratore (e solo lui) può anche richiedere il resoconto delle quote di un altro utente o gruppo specificando quest'ultimo come argomento del comando. Le altre opzioni principali di `quota` sono riportate in tab. 6.6, per i dettagli e l'elenco completo si faccia al solito riferimento alla pagina di manuale.

Opzione	Significato
-f	limita le impostazioni filesystem specificato.
-u	richiede i dati delle quote utente.
-g	richiede i dati delle quote gruppo.
-l	riporta solo le quote su filesystem locali.
-q	stampa le informazioni in modalità <i>succinta</i> .

Tabella 6.6: Principali opzioni del comando `quota`.

Un secondo comando per il controllo dello stato delle quote è `repquota` che però è ad uso esclusivo dell'amministratore, in quanto serve a fornire dei rapporti globali sullo stato delle quote nel sistema, un esempio del suo risultato potrebbe essere il seguente:

```
root@monk# repquota -a
```

			Block limits			File limits			
User		used	soft	hard	grace	used	soft	hard	grace
root	--	175419	0	0		14679	0	0	
bin	--	18000	0	0		735	0	0	
uucp	--	729	0	0		23	0	0	
man	--	57	0	0		10	0	0	
user1	--	13046	15360	19200		806	1500	2250	
user2	--	2838	5120	6400		377	1000	1500	

Il comando non prevede argomenti, e le opzioni servono per lo più a controllare le informazioni che vengono stampate, un sommario delle principali opzioni è riportato in tab. 6.7, al solito per l'elenco completo ed i dettagli di funzionamento del comando si può fare riferimento alla pagina di manuale.

Opzione	Significato
-a	elenca i dati per tutti i filesystem in cui <code>/etc/mtab</code> riporta l'uso di quote.
-u	elenca i dati delle quote utente.
-g	elenca i dati delle quote gruppo.
-n	non risolve i nomi di utenti e gruppi.
-q	stampa le informazioni in modalità <i>succinta</i> .
-s	adatta le unità di misura per spazio disco e inode.
-v	riporta maggiori informazioni e stampa anche i dati delle quote anche quando non sono usate.

Tabella 6.7: Principali opzioni del comando `repquota`.

Capitolo 7

L'amministrazione di base delle reti

7.1 Un'introduzione ai concetti fondamentali delle reti.

Il campo della comunicazione via rete è uno dei più vasti e complessi di tutta l'informatica. Nel corso degli anni sono stati sviluppati molti mezzi (cavo, fibra, radio), e molti protocolli (TCP/IP, AppleTalk, IPX, ecc.) che permettessero di far comunicare fra loro computer diversi.

In generale con il termine di rete si identifica quella serie di meccanismi e metodi di collegamento tramite i quali tanti singoli computer, chiamati *nodi* o *stazioni* (in inglese *host*) vengono messi in comunicazione fra di loro in modo da potersi scambiare dati.

In questa sezione introduttiva esamineremo in breve alcuni concetti di base relativi alle reti, a partire dai diversi criteri che vengono usati come base per le loro classificazione, quale l'estensione, la *topologia* con cui sono realizzate ed i protocolli di comunicazione usati.

7.1.1 L'estensione

Una delle forme di classificazione di una rete più elementari, ed una delle meno precise, è quella per estensione o *area*. In origine, quando le reti erano disponibili solo nei grandi centri di ricerca o nelle grandi imprese, la classificazione era molto semplice e prevedeva soltanto le due tipologie:

LAN *Local Area Network*, che indica le reti locali, realizzate su brevi distanze, all'interno di uno stesso edificio o gruppo limitrofo di edifici (in genere una università o la sede di una grande impresa) che di norma sono possedute e gestite da una sola organizzazione.

WAN *Wide Area Network*, che indica in generale una rete di grande estensione, e a cui in genere si fa riferimento per indicare la struttura che connette varie reti locali, estendendosi potenzialmente su tutto il mondo (*Internet* è un esempio di WAN). In questo caso la rete non è proprietà di una entità singola, ma viene gestita in comune da più enti o organizzazioni.

Con il diffondersi dei computer e delle tecnologie di comunicazione, anche questa classificazione delle reti si è evoluta, e la suddivisione per area di estensione si è diversificata notevolmente, tanto che attualmente oltre alle due precedenti sono state introdotte una lunga serie di altre tipologie, come ad esempio:

MAN *Metropolitan Area Network*, che indica in genere una rete cittadina, di dimensione intermedia fra LAN e WAN, in cui varie reti locali vengono integrate preliminarmente fra di loro a livello di area metropolitana, grazie a delle infrastrutture dedicate. Viene di norma gestita da entità legate al governo della città.

- SAN** *Storage Area Network*, che fa riferimento alle reti dedicate a fornire un sistema di stoccaggio dati comune ad un insieme di computer. In generale sono usate all'interno di singole organizzazioni e non sono da considerarsi propriamente delle reti di comunicazione.
- PAN** *Personal Area Network*, che indica in genere quelle reti di comunicazione usate per connettere computer e dispositivi di uso personale (ad esempio il telefonino), su distanze di pochi metri, come il *bluetooth*.
- DAN** *Desktop Area Network*, che in genere fa riferimento alle reti di comunicazione usate per connettere vari dispositivi periferici ad un computer a brevissima distanza (la scrivania appunto) come le reti ad infrarossi IRDA.

In generale comunque, la sola classificazione rilevante è la prima, le altre si sovrappongono spesso fra di loro (come per DAN e PAN) o non hanno a che fare, come la SAN (e le tecnologie di comunicazione usate per i cluster, o all'interno dei computer per le varie CPU) con quello che tratteremo in queste dispense.

7.1.2 La topologia

Una classificazione generale applicabile a qualunque rete è quella basata sulla sua topologia. La topologia (dal greco *topos*) è una branca della geometria che studia le proprietà delle connessioni fra oggetti geometrici, e si applica pertanto anche alla struttura delle reti.

La classificazione riportata prevede solo alcune topologie di base, in genere poi le reti sono costruite con l'interconnessione di reti diverse e possono assumere delle topologie ibride rispetto a quelle qui elencate. Le topologie fondamentali sono:

- bus** È una rete che utilizza un canale centrale (detto *backbone*) per connettere fra loro tutti i dispositivi. Il canale funziona come collettore cui ogni stazione si collega con un connettore inviandovi tutti i messaggi. Ogni stazione può osservare tutto il traffico del canale, estraendone i messaggi a lei indirizzati.
- È la topologia classica delle vecchie reti Ethernet su BNC. In genere questo tipo di rete ha il vantaggio della facilità di installazione, ma è efficace solo per un numero limitato di stazioni, in quanto il canale deve distribuire ad ogni nodo tutto il traffico, compreso quello destinato agli altri, e può essere facilmente saturato. Inoltre in caso di rottura del canale tutto il traffico di rete è totalmente bloccato e l'intera rete non è usabile.
- ring** È una rete in cui ogni stazione ha due stazioni vicine, a loro volta collegate ad un'altra stazione, fino a formare un anello. Ciascuna stazione comunica con le stazioni limitrofe, ed i messaggi vengono inoltrati lungo l'anello per essere ricevuti dalla stazione di destinazione. È potenzialmente più efficiente della struttura a *bus* in quanto non è necessario che il messaggio attraversi tutto l'anello, ma solo la parte necessaria a raggiungere la destinazione. Anche in questo caso però la rottura di un cavo o di una stazione comporta l'inutilizzabilità dell'intera rete.
- star** È una rete in cui esiste uno snodo centrale (un *hub* o uno *switch*) cui vengono connesse le varie stazioni. Richiede normalmente una maggiore estensione della cablatura, ma la rottura di un cavo non interrompe tutta la rete. Inoltre usando uno *switch* i pacchetti vengono smistati direttamente dalla stazione di origine alla destinazione, e non si soffre del problema del collo di bottiglia dovuto alla necessità di trasmettere tutti i dati ad ogni stazione collegata.

- tree** È una rete che integra in forma gerarchica più reti a stella. Viene in genere realizzata connettendo su una *backbone* o su uno switch centrale diversi *hub* o *switch* periferici. In questo modo si possono aumentare le stazioni collegate superando i limiti sul numero di dispositivi collegati ad un singolo switch e limitare la quantità di traffico che deve passare per la *backbone*.
- mesh** È una rete che comporta la presenza di diversi percorsi possibili per la comunicazione. È in genere la modalità in cui vengono create le reti WAN, o i cosiddetti *fabric switch*, in cui, per ottimizzare la velocità di trasmissione, si effettuano connessioni incrociate fra più switch, così da avere strade diverse per il flusso dei dati, con un traffico più distribuito, che permette di comunicare con latenze inferiori.

7.1.3 I protocolli

Come abbiamo accennato parlare di reti significa parlare di un insieme molto vasto ed eterogeneo di mezzi di comunicazione che vanno dal cavo telefonico, alla fibra ottica, alle comunicazioni via satellite o via radio; per rendere possibile la comunicazione attraverso un così variegato insieme di mezzi sono stati adottati una serie di protocolli, il più famoso dei quali, quello alla base del funzionamento di internet, è il protocollo TCP/IP.

Una caratteristica comune dei protocolli di rete è il loro essere strutturati in livelli sovrapposti; in questo modo ogni protocollo di un certo livello realizza le sue funzionalità basandosi su un protocollo del livello sottostante. Questo modello di funzionamento è stato standardizzato dalla *International Standards Organization* (ISO) che ha preparato fin dal 1984 il Modello di Riferimento *Open Systems Interconnection* (OSI), strutturato in sette livelli, secondo quanto riportato in tab. 7.1.

Livello	Nome	
Livello 7	<i>Application</i>	Applicazione
Livello 6	<i>Presentation</i>	Presentazione
Livello 5	<i>Session</i>	Sessione
Livello 4	<i>Transport</i>	Trasporto
Livello 3	<i>Network</i>	Rete
Livello 2	<i>DataLink</i>	Collegamento Dati
Livello 1	<i>Physical</i>	Connessione Fisica

Tabella 7.1: I sette livelli del protocollo ISO/OSI.

La definizione di ciascuno di questi livelli è la seguente:

Applicazione

Il livello di applicazione indica il livello finale in cui un utente interagisce con l'applicazione. Si indicano come esempi di questo livello le applicazioni per l'uso di telnet e FTP.

Presentazione

Il livello di presentazione fornisce le funzionalità di codifica e conversione dei dati usate dal successivo livello di applicazione, come la rappresentazione dei caratteri, i formati dei dati (audio, video, immagini), gli schemi di compressione, la cifratura.

Sessione

Il livello di sessione gestisce, crea e termina sessioni di comunicazione; è a questo livello che sono impostati, definiti e sincronizzati gli scambi di dati. Di norma è qui che sono definiti i protocolli di funzionamento dei singoli servizi (telnet, FTP, e-mail) che vengono offerti sulla rete.

Trasporto

Il livello di trasporto fornisce la comunicazione tra le applicazioni che girano sulle due stazioni terminali; è sostanzialmente equivalente all'omonimo livello del modello TCP/IP illustrato più avanti.

Rete Il livello di rete si occupa dello smistamento dei singoli pacchetti su una rete interconnessa, ed è a questo livello che si definiscono gli indirizzi di rete che identificano macchine non in diretto collegamento fisico; è sostanzialmente equivalente all'omonimo livello del modello TCP/IP illustrato più avanti.

Collegamento Dati

Il livello di collegamento dati è quello che fornisce una trasmissione affidabile dei dati su una connessione fisica. A questo livello si definiscono caratteristiche come l'indirizzamento dei dispositivi, la topologia della rete, la sequenza dei pacchetti, il controllo del flusso e la notifica degli errori sul livello di connessione fisica (cioè delle singole stazioni collegate direttamente fra loro da una connessione fisica, e non attraverso i livelli superiori del protocollo).

La IEEE ha diviso questo livello in due ulteriori parti, il *Logical Link Control* (LLC), definito nello standard IEEE 802.2, che gestisce le comunicazioni fra dispositivi all'interno di un singolo collegamento in una rete, ed il *Media Access Control* (MAC) che gestisce l'accesso al mezzo fisico, e consente a dispositivi diversi di identificarsi univocamente in una rete: a questo livello sono definiti gli indirizzi fisici, detti appunto *MAC address*, delle schede di rete.

Connessione fisica

Il livello di connessione fisica si occupa delle caratteristiche materiali (elettriche, fisiche, meccaniche) e funzionali per attivare, disattivare e mantenere il collegamento fisico fra diverse reti di comunicazione. Sono definiti a questo livello caratteristiche come l'ampiezza e la temporizzazione dei segnali, il tipo dei connettori, la massima capacità di trasmissione, le distanze raggiungibili.

Il modello ISO/OSI è stato sviluppato in corrispondenza alla definizione della serie di protocolli X.25 per la commutazione di pacchetto. Nel frattempo però era stato sviluppato un altro protocollo di comunicazione, il TCP/IP (su cui si basa internet) che è diventato uno standard de facto. Il modello di quest'ultimo, più semplice, viene chiamato anche modello DoD (sigla che sta per *Department of Defense*), dato che fu sviluppato dall'agenzia ARPA per il Dipartimento della Difesa Americano.

Così come ISO/OSI anche il modello del TCP/IP è stato strutturato in livelli (riassunti in tab. 7.2); un confronto fra i due modelli è riportato in fig. 7.1, dove viene evidenziata anche la corrispondenza fra i rispettivi livelli (che comunque è approssimativa). Si è indicato in figura anche fra quali livelli, nel sistema operativo, viene inserita l'interfaccia di programmazione (i cosiddetti *socket*) per l'accesso alla rete.

Livello	Nome		Esempi
Livello 4	<i>Application</i>	<i>Applicazione</i>	Telnet, FTP, etc.
Livello 3	<i>Transport</i>	<i>Trasporto</i>	TCP, UDP
Livello 2	<i>Network</i>	<i>Rete</i>	IP, (ICMP, IGMP)
Livello 1	<i>Link</i>	<i>Collegamento</i>	ethernet, PPP, SLIP

Tabella 7.2: I quattro livelli del protocollo TCP/IP.

Il nome del modello deriva dai due principali protocolli su cui è basata internet, il TCP (*Transmission Control Protocol*) che copre il livello 3, e l'IP (*Internet Protocol*) che copre il livello 2. Le funzioni dei vari livelli sono le seguenti:

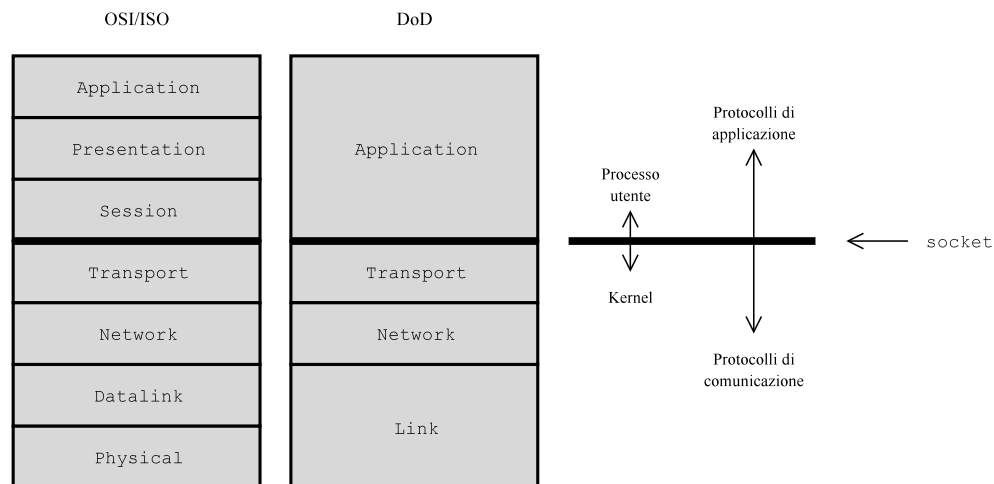


Figura 7.1: Confronto fra il modello OSI ed il modello TCP/IP nella loro relazione con il sistema.

Applicazione È relativo ai programmi di interfaccia con la rete, in genere questi vengono realizzati secondo il modello client-server, realizzando una comunicazione secondo un protocollo che è specifico di ciascuna applicazione.

Trasporto Fornisce la comunicazione tra applicazioni che girano sulle stazioni terminali su cui girano gli applicativi, regola il flusso delle informazioni, può fornire un trasporto affidabile, cioè con recupero degli errori o inaffidabile. I protocolli principali di questo livello sono il TCP e l'UDP.

Rete Si occupa dello smistamento dei singoli pacchetti su una rete complessa e interconnessa, a questo stesso livello operano i protocolli per il reperimento delle informazioni necessarie allo smistamento, per lo scambio di messaggi di controllo e per il monitoraggio della rete. Il protocollo su cui si basa questo livello è IP (sia nella attuale versione, IPv4, che nella nuova versione, IPv6).

Collegamento

È responsabile per l'interfacciamento al dispositivo elettronico che effettua la comunicazione fisica, gestendo l'invio e la ricezione dei pacchetti da e verso l'hardware.

Quale dei due modelli usare è spesso una questione di gusti; il modello ISO/OSI è più teorico e generale, il modello TCP/IP è più legato alla struttura con cui la gestione della rete è implementata in un sistema GNU/Linux. Per questo motivo, e data la sua maggiore semplicità, nel resto delle dispense faremo riferimento solo a quest'ultimo.

Per cercare di capire meglio le ragioni di tutta questa suddivisione in livelli consideriamo un'analogia con quanto avviene nella spedizione di una lettera per posta aerea in America. Voi scrivete la vostra bella lettera su un foglio, che mettete in una busta con l'indirizzo che poi imbucate. Il postino la raccoglierà dalla buca delle lettere per portarla al centro di smistamento, dove in base alla sua destinazione sarà impacchettata insieme a tutte quelle che devono essere inviate per posta aerea, e mandata al successivo centro di raccolta. Qui sarà reimpacchettata insieme a quelle provenienti dagli altri centri di smistamento ed imbarcata sull'aereo. Una volta scaricate in America le varie lettere saranno spaccettate e reimpacchettate per lo smistamento verso la destinazione finale. Qui un altro postino prenderà la vostra e la metterà nella cassetta della posta del vostro destinatario, il quale la aprirà e leggerà quello che gli avete scritto.

Questo è molto simile a quello che succede quando volete spedire dei dati via rete, secondo il procedimento che è illustrato in fig. 7.2. Ad ogni passaggio attraverso un livello del protocollo al

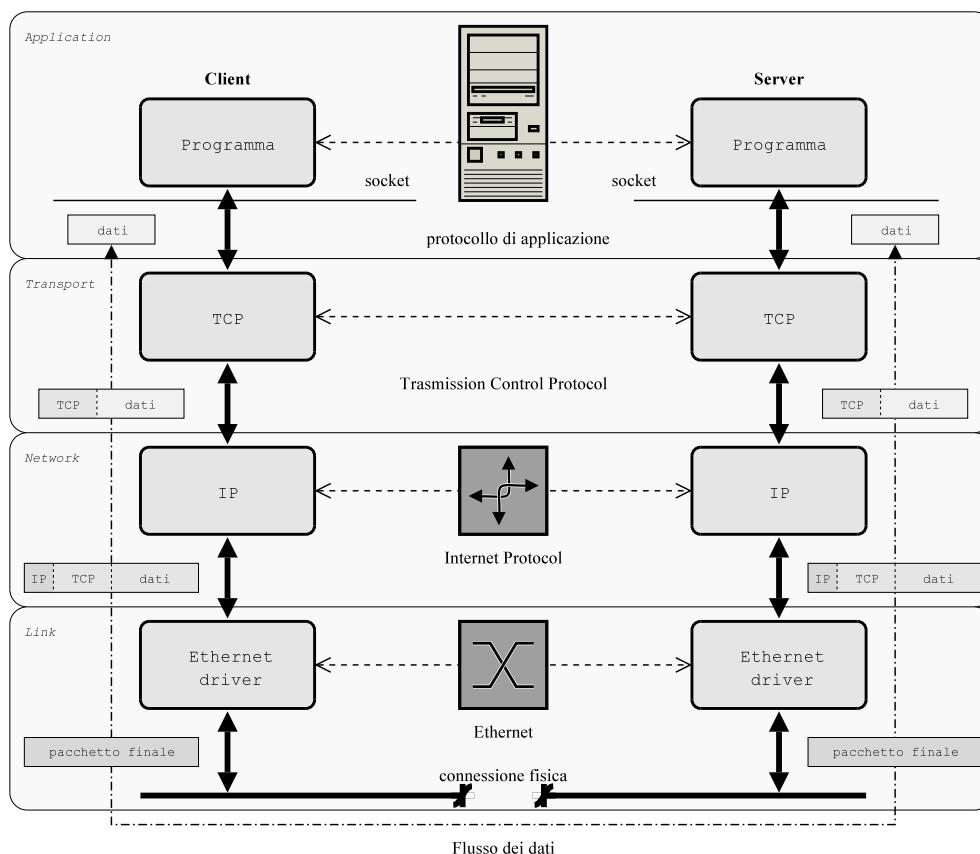


Figura 7.2: Strutturazione del flusso dei dati nella comunicazione fra due applicazioni attraverso i protocolli della suite TCP/IP.

nostro pacchetto di dati viene aggiunta una intestazione relativa al protocollo utilizzato in quel livello, si dice cioè che un pacchetto di un protocollo viene “imbustato” nel protocollo successivo. A differenza della posta in questo caso i pacchetti con i dati non vengono disfatti ad ogni livello per passare in un pacchetto diverso, ma vengono reimbastati pari pari al livello successivo, i vari “pacchetti” vengono disfatti (e la relativa intestazione rimossa) solo quando si torna indietro ad un livello superiore.

Questo meccanismo fa sì che il pacchetto ricevuto ad un livello n dalla stazione di destinazione sia esattamente lo stesso spedito dal livello n dalla stazione sorgente. Tutto ciò rende facile il progettare il software in maniera modulare facendo riferimento unicamente a quanto necessario ad un singolo livello, con la confidenza che questo poi sarà trattato uniformemente da tutti i nodi della rete.

7.2 Il TCP/IP.

Come accennato in sez. 7.1.3 negli anni sono stati creati molti tipi diversi di protocolli per la comunicazione via rete, in queste dispense però prenderemo in esame soltanto il caso di reti basate su TCP/IP, il protocollo¹ più diffuso, quello su cui si basa “Internet”, e che ormai è diventato uno standard universale e disponibile su qualunque sistema operativo. In questa sezione ci limiteremo ad introdurre i concetti fondamentali delle reti IP, la notazione e le terminologie principali.

¹in realtà non si tratta di un solo protocollo, ma di una *suite* in cui sono inseriti vari protocolli, in modo da coprire in maniera sostanzialmente completa le più varie esigenze di comunicazione.

7.2.1 Introduzione.

Dato che il protocollo è nato su macchine Unix, il TCP/IP è la modalità di comunicazione nativa di GNU/Linux, e benché per compatibilità siano stati implementati nel kernel anche parecchi altri protocolli di comunicazione (come DECnet, AppleTalk, IPX), questo resta il principale ed il più usato.

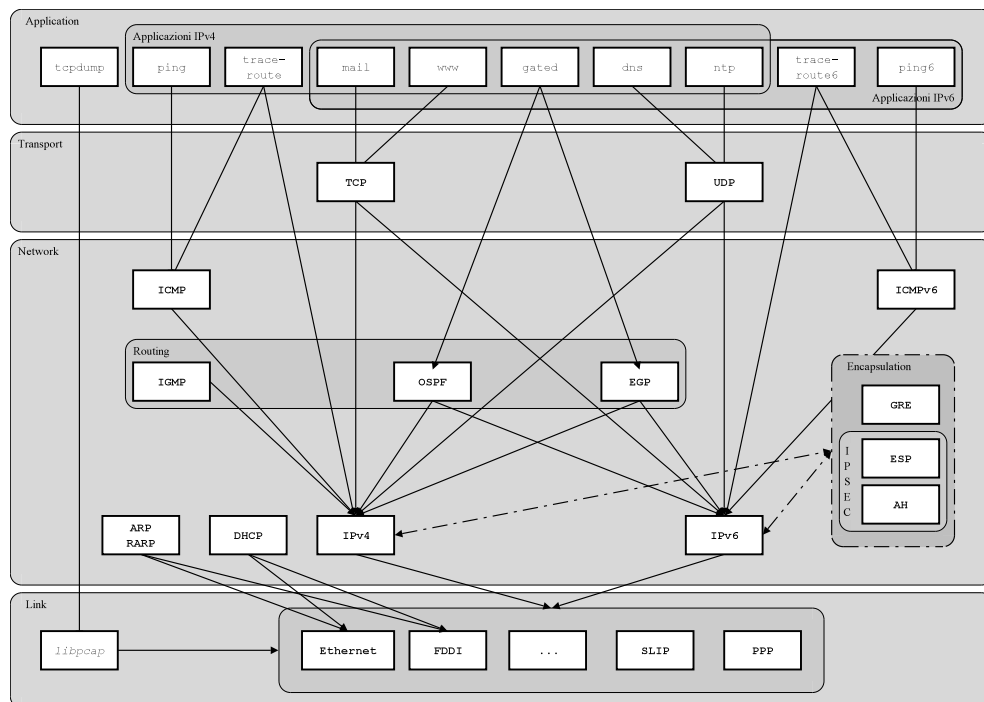


Figura 7.3: Panoramica sui vari protocolli che compongono la suite TCP/IP.

Come illustrato in sez. 7.1.3 l'insieme di protocolli che costituisce il TCP/IP è strutturato, secondo l'omonimo modello, su 4 livelli; a ciascuno di essi corrisponde un particolare compito, svolto da uno (o più) protocolli specifici. In fig. 7.3 si è riportata una panoramica dei principali protocolli che costituiscono il TCP/IP e come questi vengono suddivisi nei quattro livelli illustrati in precedenza.

L'interfaccia fondamentale usata dal sistema operativo per la comunicazione in rete è quella dei cosiddetti *socket*, nata nel 1983 a Berkley e resa pubblica con il rilascio di BSD 4.2. La flessibilità e la genericità dell'interfaccia consente di utilizzare i socket con i più disparati meccanismi di comunicazione, e non solo con l'insieme dei protocolli TCP/IP.

Con i socket infatti si può creare un canale di comunicazione fra due stazioni remote, sul quale inviare i dati direttamente, senza doversi preoccupare di tutto il procedimento di passaggio da un livello all'altro che viene eseguito dal kernel. Tutte le applicazioni che forniscono i principali servizi su internet (pagine WEB, posta elettronica, connessione remota) sono realizzati a livello di applicazione usando questa interfaccia. Solo per applicazioni specialistiche per il controllo della rete il kernel mette a disposizione delle ulteriori interfacce che permettono di accedere direttamente ai livelli inferiori del protocollo.

Come mostrato in fig. 7.2 i *socket* fanno da ponte fra il livello di applicazione e quello di trasporto; una volta inviati su un socket i dati vengono passati (dal kernel) ad uno dei protocolli del livello di trasporto, che sono quelli che si curano di trasmettere i dati dall'origine alla destinazione, secondo le modalità specificate dal tipo di socket scelto. Così se si è usato uno *stream socket* basato sul TCP quest'ultimo si curerà di stabilire la connessione con la macchina

remota e garantire l'affidabilità della comunicazione controllando eventuali errori, ritrasmettendo i pacchetti persi e scartando quelli duplicati.²

Al livello successivo sarà poi IP che curerà l'invio dei singoli pacchetti sulla rete, ed è su questo livello che operano i vari dispositivi che su internet consentono lo smistamento dei pacchetti fino alla loro destinazione finale. È a questo livello che sono definiti gli indirizzi IP che identificano ogni macchina sulla rete.

Il livello finale è quello dell'interfaccia di rete usata per trasmettere i pacchetti verso l'esterno, che a seconda dei casi può essere una scheda ethernet o l'interfaccia associata al modem; questo si incaricherà di trasmettere i dati sulla rete fisica che connette la macchina all'esterno (sia questa una rete locale o internet).

In fig. 7.3 si sono riportati i protocolli principali che costituiscono il TCP/IP. In queste dispense copriremo solo quelli di utilità più comune; una breve descrizione di quelli riportati in figura è comunque la seguente:

- IPv4** *Internet Protocol version 4.* È quello che comunemente si chiama IP. Ha origine negli anni '80 e da allora è la base su cui è costruita internet. Usa indirizzi a 32 bit, e mantiene tutte le informazioni di instradamento e controllo per la trasmissione dei pacchetti sulla rete; tutti gli altri protocolli della suite (eccetto ARP e RARP, e quelli specifici di IPv6) vengono trasmessi attraverso di esso.
- IPv6** *Internet Protocol version 6.* È stato progettato a metà degli anni '90 per rimpiazzare IPv4. Ha uno spazio di indirizzi ampliato a 128 bit che consente più gerarchie di indirizzi, l'autoconfigurazione, ed un nuovo tipo di indirizzi, gli *anycast*, che consentono di inviare un pacchetto ad una stazione su un certo gruppo. Effettua lo stesso servizio di trasmissione dei pacchetti di IPv4 di cui vuole essere un sostituto.
- TCP** *Transmission Control Protocol.* È un protocollo orientato alla connessione che provvede un trasporto affidabile per un flusso di dati bidirezionale fra due stazioni remote. Il protocollo ha cura di tutti gli aspetti del trasporto, come l'acknowledgment, i timeout, la ritrasmissione, ecc. È usato dalla maggior parte delle applicazioni.
- UDP** *User Datagram Protocol.* È un protocollo senza connessione, per l'invio di dati a pacchetti. Contrariamente al TCP il protocollo non è affidabile e non c'è garanzia che i pacchetti raggiungano la loro destinazione, si perdano, vengano duplicati, o abbiano un particolare ordine di arrivo.
- ICMP** *Internet Control Message Protocol.* È il protocollo usato a livello di rete per gestire gli errori e trasportare le informazioni di controllo fra *stazioni remote* e *instradatori* (cioè fra *host* e *router*). I messaggi sono normalmente generati dal software del kernel che gestisce la comunicazione TCP/IP, anche se ICMP può venire usato direttamente da alcuni programmi come *ping*. A volte ci si riferisce ad esso come ICMPv4 per distinguerlo da ICMPv6.
- IGMP** *Internet Group Management Protocol.* È un protocollo di livello di rete usato per il *multicasting*. Permette alle stazioni remote di notificare ai *router* che supportano questa comunicazione a quale gruppo esse appartengono. Come ICMP viene implementato direttamente sopra IP.
- ARP** *Address Resolution Protocol.* È il protocollo che mappa un indirizzo IP in un indirizzo hardware sulla rete locale. È usato in reti di tipo *broadcast* come Ethernet, Token

²in questo caso dal punto di vista dell'utente la trasmissione dei dati è più simile ad un collegamento telefonico che ad un invio di lettere. Useremo questa ed altre analogie nel seguito, ma si deve essere consapevoli che nessuna di esse è mai perfettamente congruente con il comportamento effettivo della rete.

Ring o FDDI che hanno associato un indirizzo fisico (il cosiddetto *MAC address*) alla interfaccia, ma non serve in connessioni punto-punto.

- RARP** *Reverse Address Resolution Protocol*. È il protocollo che esegue l'operazione inversa rispetto ad ARP (da cui il nome) mappando un indirizzo hardware in un indirizzo IP. Veniva usato durante l'avvio per assegnare un indirizzo IP ad una macchina.
- ICMPv6** *Internet Control Message Protocol, version 6*. Combina per IPv6 le funzionalità di ICMPv4, IGMP e ARP.
- EGP** *Exterior Gateway Protocol*. È un protocollo di instradamento usato per comunicare lo stato fra *gateway* vicini a livello di *sistemi autonomi*³, con meccanismi che permettono di identificare i vicini, controllarne la raggiungibilità e scambiare informazioni sullo stato della rete. Viene implementato direttamente sopra IP.
- OSPF** *Open Shortest Path First*. È in protocollo di instradamento per *router* su reti interne, che permette a questi ultimi di scambiarsi informazioni sullo stato delle connessioni e dei legami che ciascuno ha con gli altri. Viene implementato direttamente sopra IP.
- GRE** *Generic Routing Encapsulation*. È un protocollo generico di incapsulamento che permette di incapsulare un qualunque altro protocollo all'interno di IP.
- AH** *Authentication Header*. Fornisce l'autenticazione dell'integrità e dell'origine di un pacchetto. È una opzione nativa in IPv6 e viene implementato come protocollo a sé su IPv4. Fa parte della suite di IPSEC che provvede la trasmissione cifrata ed autenticata a livello IP.
- ESP** *Encapsulating Security Payload*. Provvede la cifratura insieme all'autenticazione dell'integrità e dell'origine di un pacchetto. Come per AH è opzione nativa in IPv6 e viene implementato come protocollo a sé su IPv4. Anch'esso fa parte di IPSEC.
- PPP** *Point-to-Point Protocol*. È un protocollo di livello di collegamento progettato per lo scambio di pacchetti su connessioni punto-punto. Viene usato per configurare i collegamenti, definire i protocolli di rete usati ed incapsulare i pacchetti di dati. È un protocollo complesso con varie componenti.
- SLIP** *Serial Line over IP*. È un protocollo di livello di collegamento che permette di trasmettere un pacchetto IP attraverso una linea seriale.
- DHCP** *Dinamic Host Configuration Protocol*. È un protocollo di livello di collegamento che permette di inviare informazioni di inizializzazione alle stazioni presenti in una LAN, come indirizzo IP, indirizzi di *gateway* e server DNS, file per il boot via rete, ecc.

7.2.2 Gli indirizzi IP

Per poter comunicare fra loro due computer in rete devono potersi in qualche modo riconoscere. Gli indirizzi IP, definiti dall'Internet Protocol visto in sez. 7.1.3, svolgono esattamente questo ruolo, che è analogo a quello del numero di telefono o dell'indirizzo di una casa. Essi devono identificare univocamente un nodo della rete.

L'analogia più diretta è quella con il telefono; per poter telefonare occorre avere un numero di telefono e conoscere quello di chi si vuole chiamare. L'indirizzo IP è l'equivalente del numero di telefono, solo che invece che di un numero decimale composto di un numero variabile di cifre

³vengono chiamati *autonomous systems* i raggruppamenti al livello più alto della rete.

è un numero binario (quindi espresso con soli 0 e 1) ed ha una dimensione fissa di quattro byte. Come per i telefoni ad un numero può corrispondere solo un telefono, ma normalmente ad un telefono non possono essere associati più numeri, vedremo più avanti invece come ad uno stesso computer (o a una stessa interfaccia di rete) possono essere assegnati più indirizzi IP.

Di solito, visto che scrivere i numeri in formato binario è poco comprensibile, si è soliti esprimere il numero che costituisce l'indirizzo IP usando una apposita notazione che viene chiamata *dotted decimal*, usando un numero decimale per ciascuno dei quattro byte che compongono l'indirizzo, separati da un punto; un esempio di questa notazione potrebbe essere qualcosa del tipo di 192.168.111.11. È attraverso questo numero il vostro computer viene identificato univocamente su Internet.

Come per il telefono di casa ogni computer connesso ad Internet viene sempre considerato come facente parte di una rete; la cosa è vera anche per le reti private non connesse direttamente ad Internet (come quelle che collegano i computer di un ufficio). Per proseguire nell'analogia si pensi alle linee telefoniche interne di una ditta, usate per parlare all'interno degli uffici.

Dato che Internet, come dice il nome, è un insieme di reti, anche queste devono venire identificate. Questo è fatto attraverso altrettanti indirizzi IP, che corrispondono alla parte *comune* di tutti gli indirizzi delle macchine sulla stessa rete. Per proseguire nell'analogia con il telefono si può pensare all'indirizzo di rete come al prefisso che serve per parlare con un'altra città o un altro stato, e che è lo stesso per tutti i telefoni di quell'area.

La differenza con i prefissi è che un indirizzo di rete IP, quando lo si scrive, deve essere completato da tanti zeri quanti sono necessari a raggiungere la dimensione di 32 bit degli indirizzi normali; per riprendere il precedente esempio di indirizzo IP, un possibile indirizzo di rete ad esso relativo potrebbe essere 192.168.111.0.

Questo meccanismo significa in realtà che ogni indirizzo su Internet, pur essendo espresso sempre come un singolo numero, nei fatti è composto da due parti, l'*indirizzo di rete*, che prende la parte superiore dell'indirizzo e identifica la particolare sezione di internet su cui si trova la vostra rete e l'*indirizzo della stazione* (il cosiddetto *host*), che prende la parte inferiore del numero e che identifica la macchina all'interno della vostra rete.

La situazione dunque è ancora analoga a quella di un numero di telefono che è diviso in prefisso e numero locale. Un prefisso è l'equivalente dell'indirizzo di rete, l'indirizzo IP completo è quello che identifica il singolo telefono, solo che in questo caso il numero di cifre (binarie) che si usano per il prefisso non è fisso, e può anche essere cambiato a seconda dei casi.

Per questo motivo, quando si configura una macchina, ad ogni indirizzo IP si associa sempre anche quella che viene chiamata una *netmask*: una maschera binaria che permette di dire quali bit dell'indirizzo sono usati per identificare la rete e quali per il nodo. Questa viene espressa con la solita notazione dotted decimal, usando il numero binario costruito mettendo un 1 ad ogni bit dell'indirizzo corrispondente alla rete e uno zero a quello corrispondente alla stazione: nel caso dell'indirizzo in esempio si avrebbe allora una netmask uguale a 255.255.255.0).

L'assegnazione degli indirizzi IP è gestita a livello internazionale dalla IANA (*Internet Assigned Number Authority*), che ha delegato la gestione di parte delle assegnazioni ad altre organizzazioni regionali (come INTERNIC, RIPE NCC e APNIC). Originariamente, per venire incontro alle diverse esigenze, gli indirizzi di rete erano stati organizzati in *classi*, (riportate tab. 7.3), per consentire dispiegamenti di reti di dimensioni diverse.

Classe	Intervallo	Netmask
A	0.0.0.0 — 127.255.255.255	255.0.0.0
B	128.0.0.0 — 191.255.255.255	255.255.0.0
C	192.0.0.0 — 223.255.255.255	255.255.255.0
D	224.0.0.0 — 239.255.255.255	240.0.0.0
E	240.0.0.0 — 247.255.255.255	

Tabella 7.3: Le classi di indirizzi IP.

Le classi usate per il dispiegamento delle reti di cui è attualmente composta Internet sono le prime tre; la classe D è destinata all'ancora non molto usato *multicast*,⁴ mentre la classe E è riservata per usi sperimentali e non viene impiegata. Oggigiorno questa divisione in classi non è più molto usata (perché come vedremo fra poco è inefficiente), ma la si trova riportata spesso, ed alcuni programmi cercano di calcolare automaticamente la netmask a seconda dell'IP che gli date, seguendo queste tabelle.⁵

Una rete di classe A è una rete che comprende 16777216 (cioè 2^{24}) indirizzi di singoli computer ed ha una netmask pari a 255.0.0.0, una rete di classe B comprende 65536 (cioè 2^{16}) indirizzi ed ha una netmask pari a 255.255.0.0 e una rete di classe C comprende 256 (cioè 2^8) indirizzi ed ha una netmask pari a 255.255.255.0.



Tabella 7.4: Uno esempio di indirizzamento CIDR.

La suddivisione riportata in tab. 7.3 è largamente inefficiente in quanto se un utente necessita di anche solo un indirizzo in più dei 256 disponibili⁶ con una classe A occorre passare a una classe B, con un conseguente enorme spreco di numeri (si passerebbe da 256 a 65536).

Per questo nel 1992 è stato introdotto un indirizzamento senza classi (detto CIDR) in cui il limite fra i bit destinati a indicare l'indirizzo di rete e quelli destinati a indicare l'host finale può essere piazzato in qualunque punto dei 32 bit totali (vedi tab. 7.4), permettendo così di accorpare più classi C su un'unica rete o suddividere una classe B. È stata così introdotta anche una nuova notazione, che permette di indicare la parte di rete appendendo all'indirizzo l'indicazione del numero di bit riservati alla rete; nel caso in esempio si avrebbe allora 192.168.111.11/24.

Oltre alla comunicazione fra due singoli nodi della rete, quella descritta finora con l'analogia telefonica, su internet è possibile un'altra modalità di comunicazione, detta *broadcast*. In questo caso è un singolo nodo che può inviare pacchetti che saranno ricevuti da tutti gli altri nodi presenti nella sua stessa rete; questa modalità di comunicazione è analoga alla trasmissione via radio, da cui deriva il nome (in inglese).

Per permettere questo tipo di comunicazione su ogni rete, oltre agli indirizzi dei singoli nodi ed a quello della rete stessa, è necessario disporre di un indirizzo riservato, che viene chiamato appunto indirizzo di *broadcast*. Questo per convenzione è sempre ottenuto mettendo ad 1 tutti i bit della porzione dell'indirizzo IP riservata all'host. È questo indirizzo che deve essere utilizzato da una singola stazione quando vuole inviare un messaggio contemporaneamente a tutti gli altri nodi presenti su quella rete.⁷

La presenza di un indirizzo di *broadcast* permette il funzionamento di una serie di protocolli ausiliari del TCP/IP che devono poter scambiare e ricevere informazione con tutti i computer presenti (ad esempio quelli che permettono di scoprire quali sono gli indirizzi realmente attivi), senza doversi indirizzare a ciascuno di essi individualmente.

In tab. 7.5 si è riassunta la struttura generica di un indirizzo IP all'interno di una rete, con tutte le tipologie di indirizzi, ed i vari valori numerici che possono essere richiesti nella configurazione di una interfaccia di rete per l'uso di un indirizzo IP.

⁴il *multicast* è una modalità di comunicazione per consentire la spedizione simultanea di uno stesso pacchetto IP da una stazione singola, che lo emette una volta sola, verso più stazioni riceventi; ottimizzando quindi la trasmissione dati da uno verso molti; per utilizzare questa comunicazione occorre operare su uno di questi indirizzi.

⁵e questo alle volte può creare problemi, dato che non è detto che la rete in questione sia ancora classificabile in questo modo.

⁶in realtà gli indirizzi disponibili con una classe A sono 254, questo perché l'indirizzo .0 è riservato per indicare la rete, mentre l'indirizzo .255, come vedremo fra poco, è quello di *broadcast*; questi indirizzi sono riservati per ogni rete e non possono essere usati per un singolo host.

⁷questo permette di usare direttamente le capacità di alcune interfacce di rete che supportano questa modalità di comunicazione.

Indirizzo	Esempio
Indirizzo completo	192.168.111.11
Maschera di rete	255.255.255.0
Porzione di rete	192.168.111.
Porzione del nodo	.11
Indirizzo di rete	192.168.111.0
Indirizzo broadcast	192.168.111.255

Tabella 7.5: Specchietto riassuntivo della struttura degli indirizzi IP.

Nella assegnazione di un indirizzo IP occorre tenere conto che alcuni indirizzi sono riservati e non possono essere utilizzati su internet. Ad esempio dalla classe A è stata rimossa l'intera rete 127.0.0.0 che viene associata alla speciale interfaccia di rete detta *loopback*.⁸ Si tratta di una interfaccia assolutamente virtuale che effettua la comunicazione localmente facendo passare i pacchetti attraverso il kernel per farli comparire sulla stessa interfaccia (da cui il suo nome); è convenzione seguita universalmente associare a questa interfaccia l'indirizzo 127.0.0.1, il *localhost*, che viene utilizzato per comunicare, senza usare fisicamente la rete, quando un programma che usa i socket TCP/IP deve fare riferimento ad un altro programma che gira sulla stessa macchina.

Un altro indirizzo speciale è l'indirizzo nullo, 0.0.0.0, che viene usato per indicare un indirizzo generico non specificato. Lo si usa anche per indicare l'insieme di tutti gli indirizzi possibili,⁹ e per questo viene anche usato per indicare la cosiddetta *default route*, cioè la destinazione verso cui inviare tutti i pacchetti con una destinazione al di fuori della rete locale.

Classe	Intervallo
A	10.0.0.0 — 10.255.255.255
B	172.16.0.0 — 172.31.255.255
C	192.168.0.0 — 192.168.255.255

Tabella 7.6: Le classi di indirizzi IP riservate per le reti private.

Infine l'RFC 1918 riserva una serie di indirizzi per le reti private, cioè per le reti interne che non devono mai essere connesse direttamente ad internet; si tratta di una rete di classe A, 16 reti di classe B e 256 reti di classe C; i loro indirizzi sono riportati in tab. 7.6.

7.2.3 L'instradamento o *routing*

L'identificazione di un nodo sulla rete effettuata tramite l'indirizzo IP è solo il primo passo per poter stabilire effettivamente una comunicazione. Una volta che si sappia la destinazione finale infatti, occorre sapere anche come poterci arrivare. Il lavoro di smistamento e reindirizzamento verso la loro destinazione finale dei pacchetti di dati che vengono trasmessi via rete è quello che in termini tecnici viene chiamato *instradamento*, in inglese *routing*.

Il *routing* è uno degli argomenti più complessi delle tecnologie di rete, ma qui lo tratteremo soltanto nei suoi aspetti più elementari. In generale il problema si può dividere in due: nella parte relativa alla propria rete locale, ed in quello che succede una volta usciti da essa.

Il funzionamento del *routing* per la trasmissione dei pacchetti su Internet comporta una serie di problematiche estremamente complesse, per cui ci limiteremo ad una semplice descrizione funzionale, ma dal punto di vista di una rete locale tutto si riduce al problema di come fare arrivare all'esterno i pacchetti che escono dai singoli computer in essa presenti.

In questo caso si può ancora fare riferimento all'analogia telefonica: si può pensare alla propria rete locale come alla rete telefonica interna di una ditta; per poter uscire e telefonare

⁸è quella indicata dalla sigla **lo**, interna al singolo nodo, che serve per spedire pacchetti a se stessi.

⁹ed in effetti se lo si prende come indirizzo di rete corrisponde appunto alla rete che ha come host tutti i possibili indirizzi, e cioè Internet.

all'esterno occorre in qualche modo passare dal centralino. Su internet le macchine che fanno da "centralino" e permettono il collegamento di reti diverse smistando fra di esse i pacchetti loro destinati sono dette *router*.

In una rete locale il ruolo del centralino è svolto dal cosiddetto *default gateway*, questa è la macchina che nella vostra rete fa da ponte verso l'esterno. Tutte le telefonate dirette fuori (cioè tutti i pacchetti di dati che devono uscire dalla rete locale per andare su internet) devono passare da questa macchina; per questo quando installate una macchina in una rete locale dovete sempre sapere l'indirizzo del *default gateway*.

La differenza coi numeri telefonici sta però nel fatto che l'indicazione di usare un centralino non si può inserire all'interno del numero che si compone (ad esempio mettendo un 0 o un 1 all'inizio dello stesso), ma in questo caso deve essere proprio specificato il numero completo della macchina che fa da ponte, che anch'essa avrà un suo indirizzo IP. Un'altra differenza è che con il TCP/IP quando si deve far arrivare un pacchetto da una rete ad un'altra, anche se non si dovesse uscire su internet (ad esempio due reti interne diverse), deve essere sempre utilizzato un *gateway*, cioè una macchina che possa fare da "cancello di ingresso" per i pacchetti destinati a quella rete.

Una volta usciti dalla rete locale la situazione si complica molto, ed in questo caso l'analogia telefonica non ci aiuta, perché il collegamento telefonico nasce a *commutazione di linea*, cioè per realizzarlo si mettevano effettivamente in collegamento elettrico i due telefoni con una serie di selettori, mentre internet è progettata per funzionare a *commutazione di pacchetto*, cioè non esiste mai una connessione diretta fra due nodi, anche se poi i programmi usano delle funzionalità che permettono di lavorare come se le cose fossero effettivamente così. In questo caso infatti i dati inviati vengono spezzati in pacchetti e passati da un nodo della rete all'altro fino ad arrivare alla loro destinazione finale, dove vengono riassemblati.

Per questo, come dice poi lo stesso nome di *instradamento*, un'analogia che illustra più chiaramente i concetti del *routing* è quella delle reti stradali. Ad esempio quando lavoravo per l'INFN mi capitava spesso di dover andare al CERN. Per farlo prendevo l'autostrada a Firenze Sud, a Firenze Nord cambiavo sulla Firenze Mare, uscendo a Lucca per fare il raccordo per prendere l'autostrada per Genova, da Genova proseguivo per Alessandria, cambiavo di nuovo per Torino, dove fatta la circonvallazione prendevo l'autostrada per il traforo del Monte Bianco. Da lì il raccordo porta sull'autostrada per Ginevra, nella cui periferia sono i laboratori del CERN.

Come vedete si tratta di un bel percorso complicato, che comporta il passaggio da diversi caselli; ora quando inviate un pacchetto su internet succede qualcosa di simile, e anche lui deve passare attraverso dei "caselli". Quello che succede ad esempio quando vi collegate con un modem e iniziate a "chattare" con qualcun'altro, o spedite un pacchetto fuori dal *gateway* della vostra rete locale, è che i pacchetti che escono dal vostro computer vengono inviati al *router* (l'equivalente del casello) del vostro provider; da lì prenderanno la strada opportuna per arrivare al *router* del provider a cui è collegato il computer del vostro interlocutore, che li manderà a lui.

In tutto questo percorso i pacchetti passeranno per una serie di altri *router* che sanno che strada devono prendere i pacchetti per poter arrivare alla destinazione finale. La differenza fra i caselli ed i *router* è che questi ultimi sanno indicare da soli ai pacchetti la strada su cui devono andare per arrivare a destinazione. In realtà sono ancora più intelligenti, e sono in grado di far prendere ai pacchetti la strada più veloce, tenendo conto di eventuali ingorghi, incidenti, interruzioni del traffico ecc. Così se il tunnel del Monte Bianco viene chiuso, quando arrivate a Torino il *router* vi farà dirottare per il Frejus.

7.2.4 I servizi e le porte.

Finora abbiamo parlato quasi esclusivamente di indirizzi IP; come accennato nell'introduzione (si ricordi quanto detto in sez. 7.1.3) *Internet Protocol* è solo uno dei protocolli di internet, e copre soltanto il *livello di rete*. Abbiamo già visto che per poter effettuare delle comunicazioni fra loro

i programmi devono poter creare delle connessioni, e per far questo si deve usare l'interfaccia dei socket, che sono basati sui protocolli del *livello di trasporto*, come TCP ed UDP.

Occorre perciò introdurre un'altra delle caratteristiche del protocollo TCP/IP, relativa stavolta al *livello di trasporto*, senza comprendere la quale mancherebbero le basi per poter spiegare il funzionamento di quest'ultimo: quella delle *porte*. Anche in questo caso l'analogia telefonica ci viene, sia pure in maniera molto parziale, in aiuto. Finora infatti abbiamo parlato degli indirizzi IP come se fossero dei numeri di telefono, ma questo riguarda solo la parte del protocollo che viene usato per effettuare la trasmissione fra due computer, e cioè il protocollo IP.

Allora come su un numero di telefono può rispondere una persona (se solleva la cornetta), una segreteria telefonica, un fax, o un altro computer (se c'è attaccato un modem), lo stesso accade anche per internet; su un indirizzo IP possono in realtà rispondere diversi *servizi*, corrispondenti a forme di comunicazione diversa.

L'analogia usata è molto debole perché di solito per fare ognuno di questi compiti ci vogliono apparecchi diversi (anche se talvolta si trovano oggetti che assommano più di uno di essi). Per questo in realtà si potrebbe pensare alle *porte* come ai canali della filodiffusione,¹⁰ cioè a delle specie di "*frequenze*" diverse su cui sintonizzate il vostro telefono, sulle quali trovate i contenuti più diversi.

In realtà non è neanche così, perché nel caso della filodiffusione il segnale non viene da un altro telefono, ma dal fornitore del servizio telefonico, potete solo ascoltare, ed un canale alla volta, mentre con internet potete sia ascoltare che trasmettere, da e verso qualunque altro telefono e su quanti canali volete¹¹ in contemporanea.

Questo avviene perché, come spiegato, TCP/IP è un insieme di protocolli, ed IP (quello degli indirizzi) serve solo a gestire la trasmissione dei pacchetti attraverso una rete. Per poter effettuare uno scambio di dati fra due programmi che comunicano via rete, occorre una modalità per stabilire un canale di comunicazione che permetta di andare più in là di quanto si fa con IP, che serve solo ad inviare pacchetti da un computer all'altro, e non da una applicazione ad un'altra.

È per questo motivo che nei protocolli del livello di *trasporto*, come UDP e TCP, è stato introdotto il concetto delle *porte*, cioè un numero ulteriore (oltre a quello dell'indirizzo) che permetta di identificare le due applicazioni che stanno usando il protocollo. In questo modo diventa possibile gestire più connessioni¹² contemporanee tra le stesse macchine (destinate a servizi diversi o provenienti da applicazioni diverse) e tenere separati i pacchetti ad esse relative.

Così quando si vuole inviare della posta elettronica si comunicherà utilizzando una di queste porte, mentre quando si vuole leggere una pagina web se ne userà un'altra, usando uno scambio di dati specifico che va a costituire l'ultimo livello (quello di *applicazione*) della struttura mostrata in fig. 7.1.

Si tenga conto però che il concetto di porta è spesso fuorviante, in quanto con porta si intende una qualche forma di accesso permanente che può essere aperto o chiuso. In realtà non esiste nessuna forma di accesso permanente e lo scambio di dati avviene solo se si hanno da ambo le parti gli strumenti per effettuarlo¹³ (il server ed il client); per questo sarebbe più chiaro parlare di frequenza, su cui si può trasmettere o ascoltare, ed in cui ciascuno può essere la trasmittente (il server) o il ricevente (il client) o anche entrambi allo stesso tempo (ad esempio nei sistemi *peer to peer*).

Bussando ad una porta (o sintonizzandosi su quella frequenza a seconda dell'analogia che

¹⁰per chi non ha idea di che cosa sia, si tratta di una specie di radio via telefono, usata per trasmettere musica quando le radio avevano una pessima qualità, ma che oggi non esiste praticamente più.

¹¹in realtà lo si può fare fino ad un numero massimo di 65535 porte, pari a $2^{16} - 1$.

¹²per UDP è più corretto parlare di canali di comunicazione, in quanto non c'è una connessione come nel caso del TCP, il quale, oltre all'invio dei pacchetti, permette anche di assicurare una comunicazione affidabile.

¹³per questo non sarà mai possibile sfondare una "*porta*" sul vostro computer, se su di essa non c'è un server a ricevere i dati, così come non possono spaccarvi i timpani urlando alla radio, se è spenta.

si preferisce) si potranno scambiare, attraverso l'opportuno protocollo di applicazione, i dati relativi al servizio associato. Dal punto di vista del TCP/IP si potrebbe usare un numero di porta qualsiasi, ma la standardizzazione ha portato ad associare alcuni numeri a dei servizi specifici (la porta 25 alla posta elettronica, la porta 80 al web, ecc.).

In un sistema Unix le prime 1024 porte sono dette *riservate* in quanto solo l'amministratore può installarci sopra dei servizi; la corrispondenza fra queste porte ed i servizi che ci devono essere installati è regolata a livello internazionale: nessuno vi obbliga a rispettare la convenzione, ma se mettete la posta elettronica sulla porta 80 e il web sulla 25 avrete certamente delle grosse difficoltà a comunicare con gli altri, dato che in genere i browser cercano i siti sulla porta 80, la posta viene inviata usando la porta 25.

Al di sopra della porta 1024 qualunque utente può mettere un suo servizio, avviando un demone che si metta in ascolto su una porta qualunque. Alcune di queste però sono state usate tradizionalmente da servizi molto diffusi, per cui in certi casi si potrebbero avere dei conflitti; come vedremo in sez. 7.4.4 c'è un elenco che associa i numeri di porta ai servizi che li utilizzano, che viene mantenuto nel file `/etc/services`.

Inoltre si ricordi che come c'è una porta per contattare il server sulla macchina di destinazione, si deve avere anche una porta che identifichi il programma client che ha eseguito la connessione sulla macchina sorgente. Questo perché se si lanciasse due volte un browser per leggere lo stesso sito si avrebbe una situazione in cui si contatta un server web a partire dallo stesso IP sorgente e verso lo stesso IP e la stessa porta di destinazione; ma il kernel deve essere in grado di poter separare i pacchetti di risposta alle due istanze del browser (che potranno leggere pagine diverse) e per questo dovranno essere usate da ciascuna istanza delle porte *sorgenti* diverse.

Questo significa che nella comunicazione sul livello di trasporto (sia con TCP che con UDP) ogni client invierà i suoi pacchetti di richiesta e riceverà i pacchetti di risposta dal server utilizzando una porta specifica, che viene appunto chiamata *porta sorgente*. La differenza fra questa porta e quelle usate per i servizi noti su cui ascoltano i server (che nella comunicazione sono identificate come *porta destinazione*) è che in genere una porta sorgente viene assegnata automaticamente dal kernel quando il client crea la connessione; per questo motivo viene chiamata anche *porta effimera* (o *ephemeral port*) proprio in quanto non necessita di avere un valore predeterminato, ed il suo utilizzo è limitato alla durata di una connessione.

7.3 La configurazione di base

La configurazione della rete è una materia alquanto complessa e di una vastità impressionante. In particolare essendo numerosissimi, e spesso molto complessi, i servizi di rete, sono altrettanto numerose e complesse le configurazioni che si possono affrontare. In questa sezione però affronteremo solo la configurazione di base della rete, e non dei vari servizi che possono essere realizzati su di essa.

In particolare vedremo come effettuare le varie impostazioni relative all'uso e alla gestione dei tre livelli più bassi del protocollo TCP/IP, che sono la base su cui è costruito tutto il resto, ed esamineremo sia i comandi con i quali si effettua la configurazione manuale della rete, che i file di configurazione usati nel procedimento di avvio per la configurazione automatica delle interfacce.

7.3.1 L'assegnazione degli indirizzi ed il comando `ifconfig`

Il primo passo per poter utilizzare la rete è quello di assegnare ad una interfaccia di rete il suo indirizzo IP. Il comando che consente di fare questo è `ifconfig`, che permette anche di impostare

le varie caratteristiche delle interfacce di rete. Quello che serve nella maggior parte dei casi sono soltanto le opzioni che permettono di attivare e disattivare una interfaccia.

Se usato senza opzioni e senza specificare una interfaccia il comando mostra lo stato di tutte le interfacce attive. Questo è il primo passo da fare sempre prima di qualunque configurazione per vedere lo stato del sistema (ed anche dopo per controllare che sia tutto a posto). Un risultato possibile è il seguente:

```
[root@havernor root]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:01:02:2F:BC:40
          inet addr:192.168.0.234  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:82075264 errors:0 dropped:0 overruns:0 frame:0
          TX packets:51585638 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:2858378779 (2.6 GiB)  TX bytes:2524425895 (2.3 GiB)
          Interrupt:10 Base address:0x8800

eth0:0    Link encap:Ethernet  HWaddr 00:01:02:2F:BC:40
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:10 Base address:0x8800

eth1      Link encap:Ethernet  HWaddr 00:E0:7D:81:9C:08
          inet addr:192.168.168.1  Bcast:192.168.168.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:9 Base address:0x6000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:10226970 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10226970 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1385547296 (1.2 GiB)  TX bytes:1385547296 (1.2 GiB)
```

Se si specifica come argomento il nome di una interfaccia il comando mostra solo lo stato dell'interfaccia specificata, se invece si vuole lo stato di tutte le interfacce esistenti, comprese quelle non attive, occorre usare l'opzione `-a`.

Come si vede nell'esempio sulla macchina sono presenti 3 interfacce di rete; due di esse (`eth0` e `eth1`) corrispondono a due schede di rete ethernet, la terza (`lo`) è una interfaccia logica, la cosiddetta interfaccia di *loopback* che viene usata per le comunicazioni locali, e che deve essere sempre attivata anche per i computer non connessi in rete.

Valore	Tipo indirizzo
UP	l'interfaccia è attiva.
NOARP	non è attivato il protocollo ARP per l'interfaccia.
RUNNING	l'interfaccia sta funzionando.
MULTICAST	sull'interfaccia è attivato il <i>multicast</i> .
BROADCAST	sull'interfaccia è attivato il <i>broadcast</i> .
POINTOPOINT	l'interfaccia è in modalità punto-punto.
LOOPBACK	l'interfaccia è in modalità <i>loopback</i> .

Tabella 7.7: Stati riportati nella terza riga del comando `ifconfig` e relativo significato.

Si può notare come il comando riporti le varie caratteristiche delle singole interfacce: nella

prima riga viene scritto il tipo di collegamento fisico, e, se presente per quel tipo di interfaccia, anche l'indirizzo fisico. Nella seconda riga viene riportato il tipo ed il valore dell'indirizzo associato all'interfaccia, quindi segue una riga che indica lo stato corrente della stessa, in cui sono elencate una serie di valori il cui elenco e relativo significato è riportato in tab. 7.7. Infine sulle righe seguenti vengono stampate una serie di altre informazioni statistiche relative al traffico sostenuto dall'interfaccia.

Valore	Tipo indirizzo
inet	Indirizzo IPv4.
inet6	Indirizzo IPv6.
ax25	Indirizzo AX25.
ddp	Indirizzo AppleTalk.
ipx	Indirizzo Novell IPX.

Tabella 7.8: Valori del parametro **aftype** del comando **ifconfig**.

A parte il caso in cui si legge la configurazione generica, il comando richiede sempre come primo argomento il nome dell'interfaccia su cui operare, ad esso può seguire un argomento opzionale **aftype** che serve a specificare il tipo di indirizzo che si vuole associare all'interfaccia. I valori possibili sono riportati in tab. 7.8, il valore di default, sottinteso quando non si specifica nulla, è **inet**, che indica il protocollo TCP/IP. Normalmente questo argomento non viene mai specificato in quanto il caso più comune di utilizzo del comando **ifconfig** è quello dell'impostazione di un indirizzo IP.

Il risultato del comando riportato a pag. 274 ci mostra anche che l'interfaccia **eth0** è *multihomed*; su di essa cioè sono presenti due indirizzi diversi, riportati nelle due sezioni separate introdotte dalle stringhe **eth0** ed **eth0:0**. Si noti comunque come le statistiche siano riportate solo per la prima istanza, in quanto esse non possono che essere uniche per ciascuna interfaccia.

È sempre possibile assegnare più indirizzi ad una stessa interfaccia, purché sia abilitato il relativo supporto nel kernel; l'opzione veniva indicata come **IP aliasing** nella sezione **Networking options**, ma nei kernel recenti il supporto è attivo di default, e l'opzione non compare più. L'assegnazione di ulteriori indirizzi su una qualunque interfaccia di rete si può fare utilizzando delle interfacce "virtuali" nella forma **eth0:N** dove N è un numero crescente a partire da 0.

Prima di configurare una interfaccia è opportuno verificare che essa non sia già attiva, perché quando si assegna un indirizzo IP un valore precedente sarà sovrascritto. Il comando **ifconfig iface** (dove **iface** è il nome della singola interfaccia che si vuole controllare, ad esempio **eth0**) ci mostrerà lo stato dell'interfaccia, dando un errore nel caso il supporto nel kernel non sia attivato o l'interfaccia non esista. Qualora invece voglia disattivare una interfaccia attiva si potrà usare il comando:

```
[root@havernor root]# ifconfig eth0 down
```

specificando l'opzione **down**, mentre per riattivarla, mantenendo le impostazioni che c'erano in precedenza, si potrà utilizzare il comando:

```
[root@havernor root]# ifconfig eth0 up
```

infine quando si vuole assegnare un indirizzo ed allo stesso tempo attivarla si potrà utilizzare il comando:

```
[root@havernor root]# ifconfig eth0 192.168.1.100
```

in cui è sottintesa l'opzione **up** poiché si dà per scontato che assegnare un indirizzo IP ad una interfaccia implichi anche la volontà di utilizzarla.

Si ricordi che ad ogni indirizzo IP è sempre associata una rete, nell'esempio precedente però essa sembra non comparire; questo è dovuto al fatto che, se non viene specificato esplicitamente, il comando `ifconfig` assegna automaticamente all'interfaccia la `netmask` corrispondente alla classe a cui appartiene l'indirizzo IP, secondo la suddivisione tradizionale di tab. 7.3 (nel caso sarebbe pari a 255.255.255.0, dato che l'indirizzo è di classe C).

Quando questa scelta non è corretta si può specificare la rete su cui si affaccia l'interfaccia indicando esplicitamente la `netmask` da utilizzare con un comando del tipo:

```
[root@hawnor root]# ifconfig eth0 192.168.1.100 netmask 255.255.0.0
```

Questo ultimo esempio ci mostra la sintassi generica dell'invocazione del comando `ifconfig`; esso prevede la specificazione nei primi due argomenti dell'interfaccia di rete e di un indirizzo, mentre negli argomenti seguenti possono essere passate una serie di ulteriori direttive e relativi parametri che consentono di impostare le varie caratteristiche dell'interfaccia.

Valore	Significato
<code>[-]arp</code>	attiva e disattiva (con <code>-</code>) l'uso del protocollo ARP per l'interfaccia. Se non specificato il default è <code>arp</code> .
<code>media type</code>	seleziona il tipo di mezzo (in genere il cavo), tramite il parametro <code>type</code> (che può assumere ad esempio valori come <code>10base2</code> , <code>10baseT</code> , ecc.); di solito viene impostato automaticamente usando il default che è <code>auto</code> .
<code>multicast</code>	abilita il <i>multicast</i> sull'interfaccia, normalmente è selezionato automaticamente quando si attiva l'interfaccia.
<code>[-]promisc</code>	attiva e disattiva (con <code>-</code>) il "modo promiscuo" ¹⁴ per l'interfaccia.
<code>netmask mask</code>	imposta la maschera di rete che deve essere specificata in notazione <i>dotted decimal</i> (si ricordi quanto illustrato in sez. 7.2.2).
<code>broadcast addr</code>	imposta l'indirizzo di <i>broadcast</i> che deve essere specificato in notazione <i>dotted decimal</i> .
<code>up</code>	attiva l'interfaccia.
<code>down</code>	disattiva l'interfaccia.

Tabella 7.9: Possibili direttive del comando `ifconfig`.

In tab. 7.9 si sono elencate le principali direttive di configurazione disponibili per `ifconfig`, con il formato e la presenza di un eventuale parametro. Per un elenco completo di tutte le altre, che normalmente vengono lasciate al valore di default, si rimanda alla lettura della pagina di manuale, accessibile con `man ifconfig`.

7.3.2 L'impostazione dell'instradamento ed il comando `route`

Avere attivato una interfaccia di rete ed averle assegnato un indirizzo IP è solo il primo passo; perché sia possibile utilizzare la rete occorre anche impostare l'*instradamento* dei pacchetti. Per questo si usa il comando `route`, che si chiama così proprio perché serve a specificare le strade che essi possono prendere per arrivare a destinazione.

Il comando permette di manipolare la *tabella di instradamento* (o *routing table*) del protocollo IP: questa è una tabella usata e mantenuta dal kernel per decidere come smistare i pacchetti in uscita e in transito. In sostanza la tabella contiene le associazioni fra le possibili destinazioni di un pacchetto e l'interfaccia che deve essere usata perché questo possa raggiungerle.

Il comando prende come argomenti delle direttive che ne specificano le operazioni; le direttive fondamentali sono due: `add`, che permette di aggiungere una voce alla tabella, e `del` che ne permette la cancellazione. Se non si specifica nessuna direttiva il comando viene usato per mostrare lo stato corrente della tabella, e le opzioni utilizzabili sono riportate in tab. 7.10.

¹⁴si chiama così una modalità di funzionamento di una interfaccia di rete per cui essa riceve tutti i pacchetti che vede passare e non soltanto quelli diretti agli indirizzi ad essa assegnati.

Opzione	Descrizione
-C	opera sulla cache di instradamento del kernel.
-e	usa il formato di <code>netstat</code> (vedi sez. 7.5.3), ripetuto due volte stampa più informazioni.
-F	opera direttamente sulla tabella di instradamento (il default).
-n	non effettua la risoluzione degli indirizzi.
-A	usa gli indirizzi della famiglia passata come parametro. Il default è <code>inet</code> per gli indirizzi IP; prende gli stessi valori usati da <code>ifconfig</code> riportati in tab. 7.8.
-v	esegue le operazioni in maniera <i>prolissa</i> .

Tabella 7.10: Opzioni a riga di comando per l'uso di `route`.

Il concetto fondamentale del routing è che ciascun nodo sulla rete deve sapere a quale nodo limitrofo deve rivolgersi per inviare un pacchetto verso una certa destinazione. In sostanza dovete avere una segnaletica stradale che vi dice da quale parte svoltare per arrivare a destinazione. In realtà quando si ha a che fare con una rete locale le cose sono molto più semplici, e, tornando all'analogia telefonica, tutto quello che dovete fare è specificare qual'è il numero del centralino e quali sono i numeri diretti.

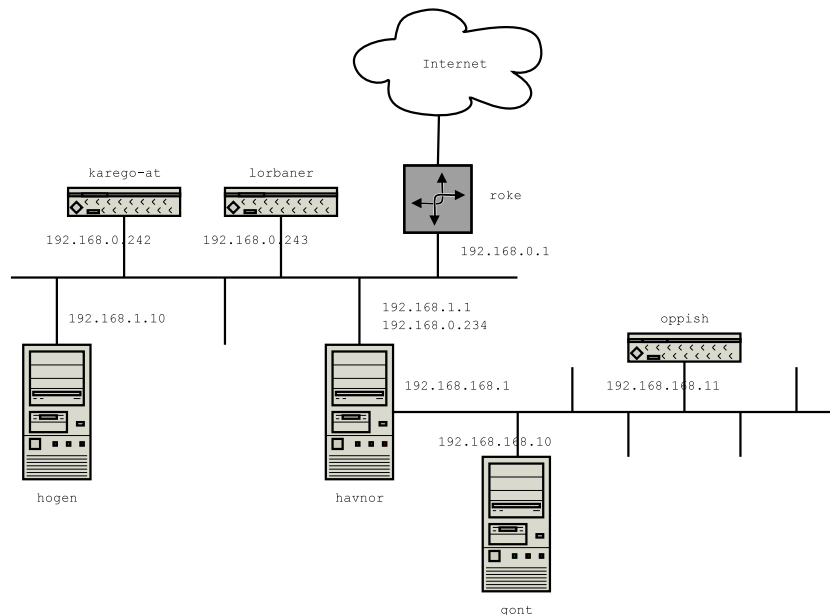


Figura 7.4: Schema di una rete di prova.

Supponiamo di avere la rete schematizzata in fig. 7.4, che prenderemo come riferimento per i nostri esempi, e vediamo come deve essere configurata la tabella di instradamento per le varie macchine di cui la rete è composta. In questo caso si vede che un ruolo particolare è rivestito dalla macchina `hawnor`, che è posta a cavallo fra due reti diverse; nel nostro esempio questa dovrà fare da ponte¹⁵ fra le due reti, passando i pacchetti da una interfaccia ad un'altra.

Perché questo funzioni però deve essere stato abilitato il cosiddetto *IP forwarding*, che di default è disabilitato; questo si può fare attraverso l'interfaccia del `sysctl`, o usando direttamente

¹⁵tecnicamente si parla di un *bridge* quando si ha a che fare con un apparato che fa da *ponte* fra due sottoreti fisiche passando i pacchetti dall'una all'altra a livello di *datalink*, in maniera completamente trasparente ai livelli superiori. Linux può essere utilizzato anche in questo modo, abilitando l'opportuno supporto nel kernel, qui però stiamo parlando semplicemente del passaggio di un pacchetto da una interfaccia ad un'altra a livello di rete, che invece è il lavoro svolto dai *router*.

i file con cui questa viene rappresentata nel filesystem `/proc`; per cui abilitare l'*IP forwarding* basterà eseguire il comando:

```
[root@havnor root]# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Vediamo allora come configurare le rotte della rete in fig. 7.4 usando `route`. Il primo passo è sempre quello di controllare la situazione corrente. Quando è invocato senza parametri il comando vi mostra il contenuto corrente della *tabella di instradamento*. Ad esempio, nel caso illustrato in fig. 7.4, andando su `havnor` avremo:

```
[root@havnor root]# route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
192.168.0.0      0.0.0.0         255.255.255.0   U        0      0      0 eth0
192.168.1.0      0.0.0.0         255.255.255.0   U        0      0      0 eth0
192.168.168.0    0.0.0.0         255.255.255.0   U        0      0      0 eth1
0.0.0.0          192.168.0.1     0.0.0.0         UG       0      0      0 eth0
```

dove si è usata l'opzione `-n` per avere una stampa con i valori numerici per gli indirizzi.

L'uscita del comando mostra le voci presenti nella tabella di instradamento, che usualmente vengono chiamate anche *rotte*; in generale le rotte impostate con `route` vengono dette *rotte statiche* (in quanto una volta impostate non vengono più cambiate), in contrapposizione alle *rotte dinamiche* che vengono impostate dai *demoni di routing*,¹⁶ che modificano continuamente le rotte presenti nella tabella di instradamento per tenere conto delle condizioni della rete.

Nome	Descrizione
Destination	rete o nodo di destinazione.
Gateway	indirizzo del <i>gateway</i> (0.0.0.0 se non impostato).
Genmask	<i>netmask</i> della rete di destinazione
Flags	flag associati alla rotta (vedi tab. 7.12).
Metric	metrica della rotta (distanza dalla destinazione in numero di salti).
Ref	numero di riferimenti nella tabella di instradamento (non usato nel kernel).
Use	numero di verifiche sulla rotta.
Iface	interfaccia verso cui sono inviati i pacchetti.
MSS	default per l'MMS (<i>Maximum Segment Size</i>) delle connessioni TCP su questa rotta.
Window	default per la <i>window size</i> delle connessioni TCP su questa rotta.
irtt	valore iniziale dell'RTT (<i>Round Trip Time</i>) per il protocollo TCP.
HH	numero di voci ARP e rotte in cache che fanno riferimento allo stessa intestazione hardware.
Arp	flag che indica se il l'indirizzo hardware per la rotta in cache è aggiornato.

Tabella 7.11: Nomi della colonna e relativo significato per le varie informazioni riportate nell'uscita del comando `route`.

Se non specificato altrimenti il comando tenta anche di risolvere (vedi sez. 7.4) i nomi delle macchine e delle reti. L'output usa la formattazione appena mostrata in cui la prima colonna identifica la destinazione, la seconda il *gateway* (nella nostra analogia il centralino per quella destinazione), la terza la sottorete coperta dalla destinazione, la quarta lo stato della rotta e

¹⁶abbiamo accennato in sez. 7.1.3 alla presenza di protocolli usati dai *router* per scambio delle informazioni, i *demoni di routing* sono programmi che implementano questi protocolli e aggiornano automaticamente la tabella di instradamento con le informazione ottenute.

l'ultima l'interfaccia usata per l'invio dei pacchetti; il significato delle altre colonne è associato agli aspetti più sofisticati del routing, gestiti di norma dai demoni di routing; uno specchietto delle informazioni mostrate dal comando è in tab. 7.11, maggiori dettagli sono nella pagina di manuale.

Nell'esempio mostrato in precedenza si può notare come ci siano tre diverse destinazioni associate a tre diverse sottoreti, due delle quali fanno capo alla stessa interfaccia (quella che in sez. 7.3.1 abbiamo visto essere *multihomed*). Per tutte queste sottoreti, essendo esse accessibili direttamente da una interfaccia locale, non esiste un *gateway* (è come per le telefonate all'interno dell'ufficio, non c'è bisogno di usare il centralino) e questo è indicato dall'uso dell'indirizzo generico in seconda colonna.

L'ultima riga indica invece il cosiddetto *default gateway*, cioè l'indirizzo cui devono essere inviati i pacchetti che non hanno una rotta specificata altrimenti; in tal caso la destinazione è indicata dall'indirizzo nullo (che fa le veci della wildcard).¹⁷

In realtà se avete una sola interfaccia, e non uscite dalla rete locale, non c'è bisogno di chiamare esplicitamente **route**; infatti tutte le volte che assegnate un indirizzo ad una interfaccia, la rotta per la rete a cui detto indirizzo è associato viene inserita automaticamente. Il problema si pone quando la struttura della rete è più complicata, e la rete è suddivisa in parti diverse.

Simbolo	Significato
U	la rotta è attiva.
H	la destinazione è una macchina singola.
G	usa un <i>gateway</i> .
R	rotta reintegrata da un instradamento dinamico.
D	installata dinamicamente da un demone o da una redirectione.
M	modificata da un demone o una redirectione.
A	installata da addrconf .
C	voce nella cache.
!	rotta bloccata (impedisce l'instradamento per la destinazione specificata).

Tabella 7.12: Significato dei simboli utilizzati nella colonna **Status** del comando **route**.

Nel caso di un computer con una sola interfaccia di rete inserito in una LAN, una volta assegnato l'indirizzo con **ifconfig** tutto quello che resta da fare è specificare qual'è la *default gateway*, cioè l'indirizzo della macchina (di solito un *router*) che si usa per uscire in internet; questo si fa con il comando:

```
[root@havnor root]# route add default gw 192.168.0.1
```

che ovviamente deve essere eseguito dall'amministratore, in quanto cambiare i contenuti della tabella di instradamento è una operazione privilegiata. Il comando è mostrato per **havnor**, ma dovrà essere ripetuto per tutte le macchine mostrate in fig. 7.4.

Per le macchine sulla rete 192.168.168.0 c'è però il problema che esse non possono vedere direttamente il *router*, che è posto sulla rete 192.168.0.0; lo stesso vale per le macchine di quella rete, che non possono accedere direttamente alla rete 192.168.168.0. In questo caso infatti i pacchetti, per passare da una rete all'altra, devono attraversare **havnor**, che, con le sue due interfacce di rete, fa da *router* fra i due tratti.

In tutti i casi in l'accesso ad una rete è condizionato al passaggio da una macchina specifica che fa da *router*, si deve specificare una rotta statica che indichi anche quest'ultima come *gateway*

¹⁷come abbiamo già accennato in sez. 7.2.2 l'indirizzo di rete nullo con netmask nulla corrisponde ad una rete in cui tutti e 32 i bit disponibili sono utilizzati come indirizzi delle macchine al suo interno, e cioè appunto all'intera Internet.

per quella rete. Così se `lorbaner` vuole accedere a `gont`, si dovrà impostare una rotta statica con:

```
[root@lorbaner root]# route add -net 192.168.168.0 netmask 255.255.255.0 \
gw 192.168.0.234
```

che dice ai pacchetti destinati alla rete `192.168.168.0` di usare come gateway la macchina `192.168.0.234`. Allo stesso modo perché `oppish` possa accedere ad `hogen` (e ad internet attraverso il router `roke`) si dovrà impostare una rotta statica con:

```
[root@oppish root]# route add -net 192.168.0.0 netmask 255.255.255.0 \
gw 192.168.168.1
```

Il kernel ordina le rotte nella tabella di instradamento in ordine di dimensione della rete di destinazione a partire da quella più specifica per arrivare alla più generica (di norma quella associata al *default gateway* che ha per destinazione tutta internet), ed invia un pacchetto in uscita sulla prima rotta che comprende nella sua destinazione l'indirizzo cui esso è destinato. Così nella rete di esempio un pacchetto destinato ad `oppish` verrà instradato (correttamente) su `havnor` e non su `roke`, perché la rotta per la rete `192.168.168.0` è più specifica di quella di default.

Il comando `route`, quando si aggiunge una nuova rotta alla tabella di instradamento, permette di impostare oltre alla destinazione e all'eventuale *gateway*, anche una serie di altre caratteristiche. In generale la direttiva `add` prevede sempre una fra le opzioni `-net`, per indicare una voce riferita ad una rete, e `-host` per una stazione singola. A queste deve seguire l'indirizzo (di rete o di nodo) della destinazione (sia in forma numerica che simbolica).

Si deve inoltre specificare, nel caso si sia indicata una rete, anche la relativa *netmask* con la direttiva `netmask` ed eventualmente l'interfaccia da utilizzare con la direttiva `dev`. Le altre principali direttive del comando sono riportate in tab. 7.13; in genere queste sono opzionali o inutilizzate per le rotte statiche, e di norma vengono impostate automaticamente agli opportuni default corrispondenti all'interfaccia cui la rotta fa riferimento.

Le opzioni di base (`-net` e `-host`) valgono anche per la direttiva `del`, la differenza fra è che mentre con `add` devono essere specificate interamente le caratteristiche della rotta, in questo caso basta identificare univocamente la voce che si vuole cancellare perché il comando abbia effetto.

Direttiva	Parametro	Descrizione
<code>netmask</code>	Nm	imposta la maschera per la rete.
<code>gw</code>	Gw	imposta l'indirizzo del gateway.
<code>reject</code>	-	installa una rotta bloccata.
<code>metric</code>	M	imposta il valore del campo Metric .
<code>mss</code>	M	imposta la <i>Maximum Segment Size</i> per i pacchetti TCP che usano la rotta.
<code>window</code>	W	imposta la dimensione della <i>advertizing window</i> del TCP per la rotta.
<code>irtt</code>	I	imposta il <i>Round Trip Time</i> iniziale per la rotta.
<code>mod,dyn,reinstate</code>		flag diagnostici usati dai demoni di routing.
<code>dev</code>	If	imposta l'interfaccia usata per raggiungere la destinazione.

Tabella 7.13: Direttive del comando `route`.

7.3.3 La configurazione automatica all'avvio del sistema.

La configurazione della rete all'avvio del sistema viene fatta da degli opportuni script di inizializzazione, la cui collocazione dipende dalla distribuzione; in tab. 7.14 si sono riportati quelli

delle principali distribuzioni.¹⁸ In genere, per le distribuzioni che supportano i run level in stile System V, per far partire o fermare la rete è sufficiente lanciare uno di questi script rispettivamente con il parametro **start** o **stop**. Gli script usano al loro interno i comandi visti in sez. 7.3, per impostare gli indirizzi IP sulle interfacce ed il default gateway (o eventuali rotte statiche).

Distribuzione	IP e routing	Servizi
Debian	/etc/init.d/networking	/etc/rc2.d/...
Slackware	/etc/rc.d/rc.inet1	/etc/rc.d/rc.inet2
RedHat	/etc/rc.d/init.d/network	/etc/rc.d/rc3.d/...
Suse	/etc/rc.d/network	/etc/rc.d/rc3.d/...

Tabella 7.14: Gli script di inizializzazione della rete per varie distribuzioni.

Di solito la configurazione dei dati permanenti della rete viene effettuata una volta per tutte in fase di installazione, dove una opportuna applicazione vi richiederà tutte le informazioni necessarie che verranno memorizzate negli opportuni file di configurazione (vedi sez. 7.3.3). Di solito i casi più comuni che si presentano sono due:

- Il computer di casa, che si collega ad internet attraverso un provider (connessione via modem analogico, ISDN, ADSL).
- Un computer connesso in rete locale (con una scheda di rete).

Nel primo caso vi verranno chiesti i dati necessari alla connessione, che vi devono essere forniti dal provider (ad esempio nel caso del modem numero telefonico, username e password del vostro account, modalità di autenticazione) ed eventualmente quelli relativi al vostro modem (anche se ormai tutte le distribuzioni sono in grado di eseguire il riconoscimento automatico), dopo di che sarà il programma di connessione che si occuperà di eseguire le relative operazioni per attivare la connessione.

In questo caso non dovete preoccuparvi dell'impostazione dell'indirizzo IP in quanto ci penserà il programma di connessione (che in genere è una qualche forma di front-end per **pppd**, che tratteremo in sez. 7.7.2) a utilizzare opportunamente le informazioni che gli vengono fornite dal provider per eseguire le configurazioni opportune. Al più ci potrà essere da configurare a mano, per quei pochi provider che non forniscono l'informazione, gli indirizzi dei DNS (torneremo su questo in sez. 7.4.2).

Nel secondo caso invece è molto probabile che dobbiate eseguire voi l'impostazione dell'IP chiedendo all'amministratore di fornirvi una serie di informazioni. In particolare vi occorrerà l'indirizzo IP da assegnare alla vostra macchina, la netmask, e l'indirizzo del gateway (oltre alla informazione sul DNS da usare). Se invece nella rete è disponibile un server DHCP l'indirizzo non dovrà essere specificato a mano, ma potrà impostato automaticamente grazie ad esso. Tutti i programmi di configurazione prevedono questa possibilità, nel qual caso non dovrete fornire nessuna informazione specifica, se non quella di usare DHCP.

Questo è un passo necessario anche se volete creare la vostra rete interna, in questo caso l'amministratore di rete siete voi, e i numeri li dovete decidere da soli; vi consiglio caldamente di usare le varie classi riservate per le reti locali di tab. 7.6, che la IANA ha destinato appositamente a questo uso, e che non vengono mai usati per macchine pubbliche su Internet.¹⁹

¹⁸le directory in cui effettivamente si trovano gli script possono variare, ma in genere il *Filesystem Hierarchy Standard* richiede che tutti gli script di avvio vadano messi in **/etc/init.d**.

¹⁹se con le vostre macchine non accedete mai ad internet potreste anche pensare di usare altri indirizzi; questo purtroppo è un errore sciocco, che va interamente a vostro scapito. Una volta infatti che uno di questi computer dovesse accedere ad internet, ad esempio attraverso un modem, i siti su internet che hanno i numeri che voi avete assegnato alle vostre macchine interne non sarebbero più raggiungibili.

Quasi tutte le distribuzioni hanno dei programmi per configurare la rete locale,²⁰ che permettono di impostare questi valori in maniera semplice, in genere attraverso una interfaccia a finestre e campi (che può essere grafica o testuale). I comandi più comuni sono riportati in tab. 7.15, ed in genere si tratta di soltanto di specificare i valori richiesti.

Distribuzione	Comando
Debian	<code>dpkg-reconfigure etherconf</code>
RedHat	<code>netcfg, netconfig</code>
Slackware	<code>netconfig</code>

Tabella 7.15: Comandi di configurazione della rete.

Come accennato tutte le distribuzioni attivano la rete tramite gli opportuni script di avvio; uno specchio degli script usati è riportato nella seconda colonna di tab. 7.14. In genere questi script non fanno altro che andare a leggere degli opportuni file di testo che contengono le informazioni necessarie (quante interfacce ci sono, quali IP devono essere assegnati, qual'è il default gateway, ecc.) ed eseguono poi i comandi necessari ad attivare le interfacce ed impostare le rotte statiche.

L'uso manuale dei singoli comandi per impostare la rete lo abbiamo già discusso nelle sezioni precedenti, ma se si modifica la configurazione a mano, al successivo riavvio tutti i cambiamenti saranno perduti, per cui se vogliamo effettuare una impostazione permanente dobbiamo andare a modificare i file in cui sono memorizzate le informazioni usate dagli script di avvio della rete.

Anche questi file variano da distribuzione a distribuzione, così come può essere diverso il loro formato; prenderemo in esame due dei casi più comuni, Debian e RedHat (Mandrake usa gli stessi file di RedHat). In genere i programmi di configurazione automatica (o i vari programmi grafici per la configurazione) non fanno altro che leggere e modificare i valori che stanno su questi file; farlo a mano può servire quando non avete la grafica a disposizione; essendo file di testo basta usare un editor qualunque.

Debian Il file di configurazione delle interfacce²¹ è `/etc/network/interfaces`, il cui formato è descritto in dettaglio dalla omonima pagina di manuale. Per l'uso normale è sufficiente specificare i dati con un contenuto del tipo:

```
auto lo eth0
iface lo inet loopback
iface eth0 inet static
    address 194.177.127.234
    netmask 255.255.255.0
    gateway 194.177.127.1
```

la prima riga, introdotta dalla parola chiave `auto`, dice quali sono le interfacce da attivare automaticamente all'avvio del sistema, che possono essere specificate sia insieme, come nell'esempio, che in altrettante righe distinte.

Le due righe seguenti, introdotte dalla parola chiave `iface`, servono a impostare i parametri di ciascuna interfaccia. Per ciascuna interfaccia deve essere fornita una riga in cui specificarne il nome, la famiglia di protocolli usata,²² e le modalità della stessa; queste sono sostanzialmente tre:

²⁰nel caso di Debian non c'è un programma specifico, ma si può usare il pacchetto `etherconf`, che usa il sistema standard di `debconf` per effettuare la riconfigurazione.

²¹in realtà si tratta del file di configurazione usato dai comandi `ifup` e `ifdown` che sono quelli usati da Debian per gestire attivazione e disattivazione delle interfacce.

²²`inet` indica l'usuale TCP/IP, ma sono possibili anche `ipx` per IPX, e `inet6` per IPv6.

loopback si usa per l'interfaccia di loopback.
dhcp si richiedono i dati di configurazione ad un server DHCP.
static si assegnano i valori secondo i parametri specificati nelle righe seguenti.

per i primi due non occorre altro, ma se si specifica **static** si devono impostare i relativi parametri nelle righe seguenti, che di solito si indentano per maggiore chiarezza, come nell'esempio.

Quando si usa la parola chiave **iface** eventuali righe successive specificano le ulteriori opzioni. Nel caso in esempio si sono indicati l'indirizzo, la maschera di rete e l'indirizzo del *default gateway*. Quest'ultimo deve essere specificato una volta sola, per l'interfaccia da cui è raggiungibile. Possono inoltre essere specificati dei comandi da chiamare contestualmente all'attivazione e alla disattivazione dell'interfaccia, usando le parole chiave **up**, **pre-up**, **down**, **post-down** seguite dal comando, che verrà eseguito rispettivamente dopo e prima dell'attivazione e prima e dopo la disattivazione.

Si tenga presente che in generale le interfacce temporanee relative all'uso di PPP non vengono menzionate in questo file, ma sono configurate a parte dagli script di avvio della connessione con **pppd** (vedi sez. 7.7.2).

RedHat i file per la configurazione all'avvio delle interfacce di rete sono mantenuti nella directory `/etc/sysconfig/networking/devices`, e ce n'è uno per interfaccia con un nome del tipo `ifcfg-iface`, dove `iface` è il nome dell'interfaccia da configurare. Ciascuno di questi contiene poi i relativi parametri. Così ad esempio avremo un file `ifcfg-eth0` relativo alla prima interfaccia ethernet, il cui contenuto sarà:

```
DEVICE=eth0
BOOTPROTO=dhcp
IPADDR=
NETMASK=255.255.255.0
GATEWAY=192.168.0.254
BROADCAST=192.168.0.255
NETWORK=192.168.0.0
USERCTL=no
ONBOOT=yes
```

Il contenuto di ciascun file è sempre nella forma di assegnazione di un valore ad una variabile²³ il cui significato è espresso in maniera evidente dal relativo nome per gran parte di esse. Si noti che nel caso si è scelto, usando `BOOTPROTO=dhcp` di assegnare l'IP attraverso un server DHCP, ed `IPADDR` non viene definita, se invece si fosse voluto assegnare un IP fisso si sarebbe dovuto utilizzare il valore `BOOTPROTO=static` ed impostare il relativo indirizzo IP definendo `IPADDR`.

7.4 Il sistema della risoluzione dei nomi

Nella sezione precedente abbiamo trattato delle configurazioni di base della rete, direttamente legate alla gestione dei protocolli di rete da parte del kernel. Una delle funzionalità di base della rete però è quella che consente di individuare un nodo sulla base di un nome simbolico invece che tramite il suo indirizzo IP. Esamineremo in questa sezione la configurazione del servizio di risoluzione dei nomi, che è quello che consente di fare tutto ciò. Vedremo come questa

²³in effetti si tratta proprio di questo, il file viene letto dallo script di avvio che usa queste variabili di shell per effettuare la configurazione.

funzionalità sia fornita direttamente dalle librerie del sistema e quali sono i file di configurazione che ne controllano il comportamento.

7.4.1 Introduzione al *resolver*

Quando si intende utilizzare un servizio su Internet è piuttosto raro che si debba fornire il valore numerico di un indirizzo IP, questo perché uno dei servizi di base per il funzionamento di Internet è quello della risoluzione dei nomi, grazie al quale potete identificare un sito o una macchina sulla rete attraverso un nome simbolico che poi sarà *risolto* nel relativo indirizzo IP.

Su Internet la risoluzione dei nomi viene realizzata attraverso un apposito servizio, il cosiddetto *Domain Name Service* (da qui in avanti DNS) che tratteremo in cap. 9. Questo servizio è in sostanza un enorme database distribuito, interrogabile via rete, che associa un nome simbolico (quello dei *nomi a dominio*, che identifica i siti internet) al corrispondente indirizzo IP. Qualora sia necessaria una risoluzione si può interrogare il DNS; ogni programma che vuole accedere ad un nodo attraverso il suo nome invece che con il numero IP è quindi un client per questo servizio.

Dato che la risoluzione dei nomi è un servizio usato da un gran numero di programmi diversi, in un sistema unix-like questo viene effettuato tramite una serie di funzioni, inserite direttamente nelle librerie standard del C, che vanno a costituire quell'insieme di funzionalità che viene chiamato *resolver*, e che permette di ottenere le associazioni fra nodi della rete ed i nomi identificativi associati agli stessi. Si tenga presente che tutto ciò non riguarda soltanto l'associazione fra nome a dominio e numero IP, ma anche quella fra numero di porta e nome del corrispondente servizio e molto altro.

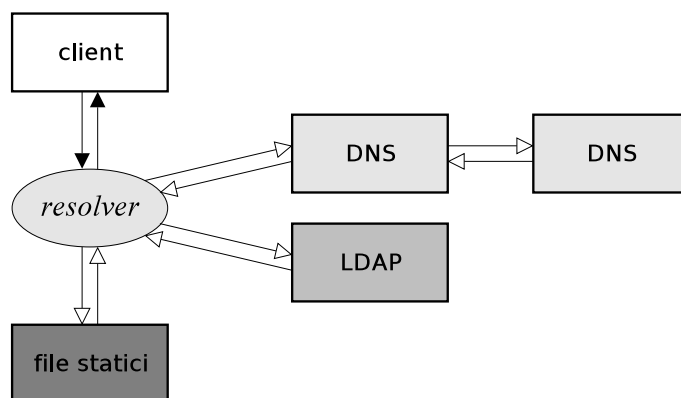


Figura 7.5: Schema di funzionamento delle funzioni del *resolver*.

Per questo prima di trattare del DNS in quanto servizio fornito sulla rete, dovremo prendere in considerazione i file di configurazione usati dalle funzioni di libreria del *resolver*, che sono quelle che svolgono in maniera generica il servizio di risoluzione dei nomi, e che non necessariamente eseguono questo compito rivolgendosi ad un server DNS. Le librerie infatti consentono di mantenere le informazioni relative alla risoluzione su diversi supporti, definendo una interfaccia astratta che viene usata dai programmi, secondo l'architettura illustrata in fig. 7.5.

Se restiamo nella nostra analogia telefonica, il sistema del *resolver* è in sostanza il meccanismo che ci porta dal sapere il nome del destinatario all'avere il numero di telefono. Come per il telefono questo può avvenire in più modi, si può andare a cercare sulla rubrica, che nel caso è realizzata dal file `/etc/hosts` (vedi sez. 7.4.3), si può cercare nell'elenco del telefono, oppure ci si può rivolgere ad un server DNS, che è più simile al servizio "12".²⁴ L'uso del DNS ha anche il vantaggio (come il servizio 12 rispetto ad una rubrica) che qualora una associazione fra nome e indirizzo IP dovesse cambiare questa viene aggiornata automaticamente.

²⁴anche se oggi non esiste più, continueremo a fare riferimento al vecchio numero, così da evitare pubblicità occulte.

Per questo motivo una delle informazioni che di norma vengono richieste nella configurazione manuale della rete²⁵ è l'indirizzo IP di un server DNS (ogni provider ne mette a disposizione almeno uno) da contattare per la risoluzione dei nomi quando navigate su Internet.²⁶

7.4.2 I file di configurazione del *resolver*

Come abbiamo già visto in sez. 3.1.3, in un sistema unix-like oltre alla risoluzione dei nomi a dominio esiste la necessità di poter effettuare altre associazioni fra valori simbolici e numerici, come quelle fra numero di porta e nome del servizio, fra user-id e nome di login dell'utente, ecc. Siccome questo tipo di corrispondenze possono essere mantenute in diversi modi, le librerie del C GNU prevedono una modularizzazione di questi servizi, attraverso il sistema del *Name Service Switch*.

Nel nostro caso ci interessano solo le classi di informazioni relative alla rete; un estratto della sezione del file `/etc/nsswitch.conf` attinente ad esse è il seguente:

```
...

hosts:      files dns
networks:   files
protocols:  db files
services:   db files
ethers:     db files
rpc:        db files

...
```

Nel caso di risoluzione dei nomi a dominio la riga che interessa è quella relativa a **hosts**. L'impostazione riportata nel precedente esempio ci dice che prima si dovrà andare a risolvere i nomi usando i file locali standard (cioè `/etc/hosts`) e poi il DNS. Se si volessero mettere le corrispondenze fra macchine locali e nomi su un server LDAP si potrebbe modificare questa linea come:

```
hosts:      files ldap dns
```

nel qual caso, prima di interrogare il DNS, il *Name Service Switch* provvederebbe ad effettuare una ricerca su LDAP. A meno di non avere una esigenza specifica di questo tipo, in genere non ci sono motivi per modificare il contenuto di questo file.

Il principale file di configurazione del *resolver* è `/etc/resolv.conf`, su cui vengono mantenute le informazioni di base necessarie al funzionamento del sistema. Il file contiene la lista degli indirizzi IP dei server DNS a cui ci si rivolge quando si vuole effettuare una risoluzione ed l'indicazione del dominio locale. Il formato del file è molto semplice, una serie di righe nella forma **direttiva valore** con le righe vuote e quelle che iniziano per **#** che vengono ignorate. Un possibile esempio del suo contenuto è il seguente:

```
search fiorenze.linux.it
nameserver 127.0.0.1
nameserver 62.48.34.105
```

La direttiva **nameserver** serve ad indicare l'indirizzo IP di un server DNS a cui rivolgersi per la risoluzione dei nomi a dominio; se ne possono specificare più di uno e la risoluzione di un nome verrà eseguita interrogandoli in sequenza, nell'ordine in cui essi sono stati specificati.

²⁵ogni distribuzione prevede un suo sistema, e molte volte queste informazioni sono richieste anche in fase di installazione.

²⁶che ovviamente dovrà essere numerico, dato che ancora non si può risolverlo.

Normalmente si possono indicare fino ad un massimo di 3 diversi server, l'indirizzo deve sempre essere fornito in forma numerica.²⁷

La direttiva **search** specifica una lista (separata da spazi) di domini in cui cercare i nomi delle macchine quando sono indicati in forma non qualificata;²⁸ di norma si specifica come argomento solo il dominio locale, così quando si effettua la ricerca su un nome, gli viene aggiunto automaticamente quel dominio come suffisso. In questo modo si può specificare solo l'*hostname* di un computer sulla rete locale, e la ricerca verrà eseguita automaticamente come se lo si fosse scritto con il nome completo. Il tutto, se se ne è indicato più d'uno, verrà ripetuto per ciascuno dei domini presenti nella lista.

Un elenco delle principali direttive usate nel file è riportato in tab. 7.16; una descrizione più dettagliata e completa si può trovare nella pagina di manuale, ottenibile con **man resolv.conf**.

Direttiva	Significato
nameserver	definisce l'IP di un server DNS da utilizzare per la successiva risoluzione dei nomi.
search domain	lista dei domini su cui effettuare una ricerca preventiva. nome del dominio locale, se assente viene determinato sulla base di quanto specificato con /etc/hostname .

Tabella 7.16: Direttive del file di con **resolv.conf**.

In genere le informazioni mantenute su **/etc/resolv.conf**, come il dominio locale e l'IP dei server DNS, vengono fornite in fase di installazione o di configurazione iniziale della rete. In sostanza è in questo file che dovete dire qual'è il numero di telefono del servizio "12" a cui vi rivolgete; ogni provider ne mette a disposizione almeno uno, e come vedremo in sez. 9 è possibile anche creare un DNS locale.

Per una macchina posta in una rete locale questi valori di norma devono essere impostati manualmente.²⁹ In genere questo viene fatto dal programma di configurazione della rete, che, quando vi chiede dominio e DNS, va a creare automaticamente questo file, inserendovi le direttive **search** e **nameserver** con i valori da voi forniti.

Se invece usate un collegamento punto-punto con un modem o una ADSL sono i programmi che lanciano la connessione (cioè **pppd** o l'eventuale interfaccia a quest'ultimo, vedi sez. 7.7.2) che usano le informazioni ottenute dal provider durante la fase di negoziazione del collegamento, e riscrivono al volo questo file; in questo caso vi può servire di aggiungerci al volo in un secondo tempo qualche altro DNS (ce ne sono di pubblici) qualora quello del vostro provider avesse problemi.

Il secondo file di configurazione principale del *resolver* è **/etc/host.conf**, che controlla le modalità di funzionamento delle funzioni che eseguono la risoluzione dei nomi. Al solito le righe vuote od inizianti per **#** vengono ignorate, le altre devono contenere una parola chiave che esprime una direttiva, che può essere seguita o meno da un valore. Un esempio comune del contenuto di questo file è il seguente:

```
order hosts,bind
multi on
```

La direttiva **order** controlla la sequenza in cui viene effettuata la risoluzione dei nomi, nell'esempio si dice che deve prima essere usato il file **/etc/hosts** (vedi sez. 7.4.3), e poi possono essere eseguite eventuali interrogazioni ai DNS esterni. La direttiva **multi** permette di ottenere come risposta tutti gli indirizzi validi di una stazione che compare più volte in **/etc/hosts**, invece di avere solo il primo.

²⁷ anche qualora si disponesse di una risoluzione statica su **/etc/hosts**, dato che il *resolver* non può risolvere un nome senza aver prima interpretato correttamente il contenuto di questo file.

²⁸ per forma qualificata di un nome a dominio si intende un nome completo del tipo **freedom.softwarelibero.it**,

Direttiva	Significato
nospoof	attiva il controllo antispoofing, chiedendo la risoluzione inversa dell'IP ricevuto e fallendo in caso di mancata corrispondenza.
spoofalert	se nospoof è attivo inserisce un avviso degli errori rilevati nei log di sistema.
reorder	riordina gli indirizzi in modo da restituire per primi quelli locali.

Tabella 7.17: Principali direttive del file `host.conf`.

Oltre a queste due le altri principali direttive (ed il relativo significato) è riportato in tab. 7.17. Tutte prendono come argomenti i valori **on** ed **off**, che attivano e disattivano il comportamento richiesto. Quando non vengono esplicitamente utilizzate, il comportamento di default è equivalente ad **off**. Per l'elenco completo ed i dettagli su tutte le direttive si può consultare la pagina di manuale con `man host.conf`.

7.4.3 La gestione locale dei nomi

Normalmente, a meno di una diversa impostazione su `/etc/host.conf`, il primo metodo utilizzato dal *resolver* per effettuare una corrispondenza fra nomi simbolici e indirizzi IP è utilizzare il contenuto di `/etc/hosts`. Questo file contiene un elenco di nomi di macchine, associati al relativo indirizzo IP. Lo si usa quindi come un'agenda del telefono per specificare gli indirizzi delle macchine per le quali si ha una mappa *statica* degli indirizzi, ad esempio le macchine di una rete privata che non vanno su internet, ma che volete risolvere col nome che gli avete assegnato.

Il formato del file è molto semplice, le righe vuote od inizianti per una **#** sono ignorate, le altre righe devono contenere, separati da spazi o caratteri di tabulazione, l'indirizzo IP, il nome completo (in termini di dominio, quello che viene detto FQDN, *Fully Qualified Domain Name*) ed una eventuale lista di nomi alternativi. Un possibile esempio di questo file è il seguente:

```
# private nets
192.168.168.10 gont.gnulinix.it      gont
192.168.168.11 oppish.gnulinix.it   oppish
```

che associa agli IP delle macchine che in fig. 7.4 sono sulla rete secondaria di **havnor** i relativi nomi.

Ovviamente usare questo file è la maniera più semplice per identificare una macchina su una rete locale, lo svantaggio è che deve essere presente su ogni macchina, e quando il numero di nodi coinvolti aumenta, diventa sempre più complicato il lavoro di tenerli tutti aggiornati e coerenti fra loro. Per questo vedremo in sez. 9.2.4 come fare lo stesso lavoro attraverso un DNS locale.

Benché il file faccia riferimento a delle caratteristiche tipiche dell'uso della rete, esso deve essere presente anche quando la macchina non è fisicamente in rete, in quanto esiste l'interfaccia di *loopback*, che viene utilizzata da molti programmi, che fanno riferimento a questo file per la risoluzione del nome `localhost`, ad essa associato, all'indirizzo `127.0.0.1`.

Un caso particolare di gestione dei nomi è quello che riguarda il nome che contraddistingue la macchina stessa, comunemente detto *hostname*. Benché l'*hostname* abbia di solito più significato quando si è collegati in rete, il nome di una macchina è una proprietà del tutto indipendente dalla presenza in rete della stessa, e viene riportato anche da vari comandi di sistema attinenti alla rete, come `uname`, il prompt della shell o le schermate di avvio.

Di norma buona parte delle distribuzioni chiedono il nome della macchina in fase di installazione e lo memorizzano nel file `/etc/hostname`.³⁰ Questo file viene usato negli script di avvio per leggere il nome della macchina quando questo viene impostato con il comando `hostname`.

usare una forma non qualificata all'interno del dominio `softwarelibero.it` sarebbe usare semplicemente `freedom`.

²⁹a meno di non usare il servizio DHCP, vedi sez. 8.2.2.

³⁰questa è la scelta di Debian, altre distribuzioni come RedHat invece usano `/etc/HOSTNAME`.

Perciò si deve essere consapevoli che cambiare il contenuto di questo file non cambia il nome della macchina fintanto che non si chiama direttamente anche `hostname` o si rieseguono gli script di avvio.

Se invocato direttamente il comando `hostname` si limita a stampare il nome della macchina, mentre può impostare un nome diverso se quest'ultimo viene passato come argomento, se invece si usa l'opzione `-F` il comando legge il nome da un file (è con questa opzione che gli script di avvio usano il contenuto di `/etc/hostname`).

Il comando può essere invocato anche come `dnsdomainname` che invece permette di ottenere il nome a dominio completo. In questo caso però non è possibile modificarne il valore, in quanto questo viene determinato tramite *resolver* sulla base della configurazione di quest'ultimo. Per le altre opzioni ed per una descrizione completa si può fare riferimento alla pagina di manuale, accessibile con `man hostname`.

7.4.4 La gestione degli altri nomi di rete

Come accennato in sez. 7.4.2 oltre alle associazioni fra indirizzo IP e nome simbolico di una macchina, esistono una serie di altre informazioni (si faccia riferimento a quanto riportato in tab. 3.1) relative a protocolli e servizi di rete, di norma identificate da valori numerici, ai quali poi viene assegnato un corrispondente nome simbolico come facilitazione mnemonica.

Queste corrispondenze oggi sono gestite tutte tramite il *Name Service Switch*, ma quasi sempre, come risulta dall'estratto di configurazione citato in sez. 7.4.2, vengono utilizzati i file statici su cui tradizionalmente³¹ esse venivano mantenute, per questo motivo nel resto della sezione tratteremo soltanto del formato di questi file.

In sez. 7.2.4 abbiamo già accennato come la corrispondenza fra i servizi di rete ed i numeri di porta loro assegnati venga mantenuta nel file `/etc/services`; è al contenuto di questo file che fanno riferimento tutti i programmi che vogliono utilizzare il nome simbolico di un servizio al posto del valore numerico della porta ad esso associato.

Di norma questo non è un file che sia necessario modificare, ma è utile conoscerne il contenuto. Il formato del file è molto semplice, un elenco di numeri di porta a ciascuno dei quali è associato un nome simbolico che individua il servizio ad esso associato dalle convenzioni internazionali o dall'uso comune. Un estratto di questo file, preso da una Debian Sarge, è:

```
...
ftp-data      20/tcp
ftp           21/tcp
fsp           21/udp      fspd
ssh           22/tcp      # SSH Remote Login Protocol
ssh           22/udp      # SSH Remote Login Protocol
telnet        23/tcp
# 24 - private
smtp          25/tcp      mail
# 26 - unassigned
time          37/tcp      timserver
time          37/udp      timserver
whois         43/tcp      nicname
re-mail-ck    50/tcp      # Remote Mail Checking Protocol
re-mail-ck    50/udp      # Remote Mail Checking Protocol
domain        53/tcp      nameserver  # name-domain server
domain        53/udp      nameserver
mtp           57/tcp      # deprecated
bootps        67/tcp      # BOOTP server
bootps        67/udp
bootpc        68/tcp      # BOOTP client
```

³¹cioè prima che si passasse all'uso del *Name Service Switch*, quando le informazioni erano appunto ricavate direttamente dalla scansione di questi file.


```

bootpc      68/udp
tftp        69/udp
gopher      70/tcp          # Internet Gopher
gopher      70/udp
rje         77/tcp          netrjs
finger      79/tcp
www         80/tcp          http      # WorldWideWeb HTTP
www         80/udp          # HyperText Transfer Protocol
...

```

Come per buona parte dei file di configurazione, righe vuote e tutto quello che segue un **#** viene considerato un commento ed ignorato; ogni riga ha il formato:

```
nome          numero/protocollo    alias
```

dove **nome** è l'identificativo simbolico del servizio, **numero** è il numero di porta ad esso assegnato, **protocollo** indica se il servizio usa UDP o TCP, mentre **alias** è una lista (separata da spazi) di eventuali altri nomi associati allo stesso servizio.

Un secondo file che mantiene corrispondenze fra valori numerici e nomi è **/etc/networks**, che è l'analogo di **/etc/hosts** per quanto riguarda le reti. Anche queste, come le singole stazioni, possono essere identificate da un nome, e di nuovo le corrispondenze statiche fra nome e indirizzo IP della rete vengono mantenute in questo file, il cui formato, come descritto dalla pagina di manuale accessibile con **man networks**, è composto da tre campi separati da spazi.

Il primo campo indica il nome simbolico della rete, il secondo campo il suo indirizzo, nella notazione *dotted decimal* (tralasciando opzionalmente gli eventuali .0 finali), ed il terzo campo un eventuale alias (questo campo è opzionale). Al solito le righe vuote e tutto quello che segue un **#** viene ignorato. Un esempio del contenuto di questo file potrebbe essere:

```
localnet 192.168.1.0
```

il contenuto di questo file viene utilizzato da comandi come **netstat** o **route** per mostrare i nomi simbolici al posto dei valori numerici; si tenga conto però che con questo file viene supportata solo la corrispondenza con reti espresse nella notazione tradizionale per classi di tipo A, B o C, e che la notazione in formato CIDR non funziona.

Un altro file usato per le corrispondenze fra valori numerici e nomi simbolici è **/etc/protocols**, che associa il numero usato per indicare all'interno dei pacchetti IP il protocollo di trasporto usato nello strato successivo (si ricordi quanto detto in sez. 7.1.3) ad un nome identificativo del protocollo stesso (come TCP e UDP).

Il formato di questo file è identico a quello di **/etc/networks**; sono supportati tre campi divisi da spazi, in cui il primo identifica il nome simbolico, il secondo il valore numerico che identifica il protocollo ed il terzo un alias. Un esempio di questo file è il seguente (l'estratto è preso da una Debian):

```

ip      0      IP      # internet protocol, pseudo protocol number
icmp    1      ICMP    # internet control message protocol
igmp    2      IGMP    # Internet Group Management
ggp     3      GGP     # gateway-gateway protocol
ipencap 4      IP-ENCAP # IP encapsulated in IP (officially "IP")
st      5      ST      # ST datagram mode
tcp     6      TCP     # transmission control protocol
egp     8      EGP     # exterior gateway protocol
pup     12     PUP     # PARC universal packet protocol
udp     17     UDP     # user datagram protocol
hmp     20     HMP     # host monitoring protocol
xns-idp 22     XNS-IDP  # Xerox NS IDP
rdp     27     RDP     # "reliable datagram" protocol
iso-tp4 29     ISO-TP4  # ISO Transport Protocol class 4

```

xtp	36	XTP	# Xpress Transfer Protocol
ddp	37	DDP	# Datagram Delivery Protocol
idpr-cmtp	38	IDPR-CMTP	# IDPR Control Message Transport
ipv6	41	IPv6	# Internet Protocol, version 6
ipv6-route	43	IPv6-Route	# Routing Header for IPv6
ipv6-frag	44	IPv6-Frag	# Fragment Header for IPv6
idrp	45	IDRP	# Inter-Domain Routing Protocol
rsvp	46	RSVP	# Reservation Protocol
gre	47	GRE	# General Routing Encapsulation
esp	50	IPSEC-ESP	# Encap Security Payload
ah	51	IPSEC-AH	# Authentication Header
skip	57	SKIP	# SKIP
...			

Al solito le righe vuote e tutto quello che segue un # viene ignorato; come per tutti gli altri file analoghi la descrizione completa del formato è riportata nella rispettiva pagina di manuale, accessibile con `man protocols`.

7.5 I comandi diagnostici

Una volta eseguita la configurazione di base della rete in genere è opportuno controllarne il funzionamento, cosa che può essere fatta utilizzando un qualunque client (il modo più comune è lanciare un browser). In caso di malfunzionamento, o comunque per eseguire controlli più approfonditi, sono disponibili la serie di comandi che tratteremo in questa sezione, che possono essere utilizzati per verificare il buon funzionamento della rete.

7.5.1 Il comando ping

Una volta configurate le interfacce di rete in genere il primo controllo che si va ad effettuare per vedere se la rete funziona è quello di “pingare”³² un'altra macchina. Nella nostra analogia telefonica questo equivale a telefonare ad un numero per sentire se dà la linea.

Il comando `ping` permette di inviare un pacchetto ICMP (abbiamo accennato a questo protocollo in sez. 7.2.1) di tipo *echo request*; il protocollo prevede che la macchina che riceve un pacchetto di questo tipo debba rispondere con l'emissione di un altro pacchetto ICMP, di tipo *echo reply* che `ping` si incaricherà di ricevere, mostrando anche il tempo intercorso fra l'invio e la risposta.

Il comando si invoca passando come argomento l'indirizzo della macchina bersaglio, si può usare anche il nome simbolico, ma in tal caso oltre al collegamento verso la macchina bersaglio deve funzionare quello con il DNS,³³ perché se per qualche motivo non riuscite a risolvere il nome non otterreste nulla anche se il resto della rete fosse a posto. Un esempio di uso del comando è il seguente:

```
[piccardi@havnor corso]$ ping 192.168.168.10
PING 192.168.168.20 (192.168.168.20): 56 data bytes
64 bytes from 192.168.168.20: icmp_seq=0 ttl=255 time=0.7 ms
64 bytes from 192.168.168.20: icmp_seq=1 ttl=255 time=0.3 ms
64 bytes from 192.168.168.20: icmp_seq=2 ttl=255 time=0.3 ms

--- 192.168.168.20 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.4/0.7 ms
```

³²il nome e la terminologia derivano dall'analogia con gli impulsi usati dai *sonar*.

³³in realtà si può usare anche il nome pure in assenza di DNS, ma esso deve essere scritto in `/etc/hosts` (vedi sez. 7.4.3).

Il comando invia un pacchetto al secondo; nell'esempio riceve sempre risposta, e riporta quindi il tempo che intercorso fra l'invio e la ricezione. Quando viene fermato (nel caso dell'esempio con **C-c**) stampa anche una statistica riassuntiva. L'esempio ci mostra che la macchina con IP 192.168.168.10 è attiva ed è raggiungibile. Il comando **ping** prende una serie di opzioni, le principali delle quali, insieme al loro significato, sono riportate in tab. 7.18, per un elenco completo si può fare riferimento alla pagina di manuale.

Opzione	Significato
-c count	invia solo count pacchetti e poi esce stampando la statistica senza bisogno di interruzione esplicita.
-f	invia i pacchetti alla velocità con cui tornano indietro, o 100 al secondo, stampando un "." per ogni pacchetto inviato ed un backspace per ogni pacchetto ricevuto, così da visualizzare le perdite. Solo l'amministratore può usare questa opzione che carica pesantemente la rete.
-i wait	aspetta wait secondi invece di uno fra l'invio di un pacchetto ed il successivo.
-n	non effettua la risoluzione di nomi e indirizzi.
-p pattern	si possono specificare fino a 16 byte con cui riempire il carico del pacchetto. Il valore pattern deve essere specificato come numero esadecimale. Si usa questa opzione per diagnosticare eventuali problemi di corruzione dei dati.
-q	sopprime tutte le stampe eccetto le statistiche finali.
-s size	invia pacchetti di dimensione size invece dei 56 byte usati pre default.

Tabella 7.18: Principali opzioni del comando **ping**.

Si tenga presente che quello che si può effettuare con **ping** è solo un controllo preliminare; se non si riceve risposta il motivo può dipendere da molti fattori: da un errore di configurazione all'aver usato un indirizzo IP sbagliato, compreso il fatto che la macchina che volete controllare può essere spenta, che la rete può essere scollegata o che uno dei router che devono inoltrare i pacchetti per raggiungerla può avere un malfunzionamento. Se però va tutto bene potete se non altro concludere che la rete è attiva e funzionante ed evitare di mettervi a controllare se avete attaccato il cavo.

7.5.2 I comandi **traceroute** ed **mtr**

Un secondo comando che permette di controllare il funzionamento di un collegamento è **traceroute**, che, come dice il nome, serve a tracciare la strada che fanno i pacchetti per arrivare alla destinazione indicata.

Il comando sfrutta una caratteristica del protocollo IP che prevede nelle informazioni associate a ciascun pacchetto un campo chiamato TTL, che viene decrementato ogni volta che il pacchetto attraversa un router, in quello che in gergo viene chiamato un *hop*. Quando il valore si annulla il protocollo richiede³⁴ che il router scarti il pacchetto ed invii un messaggio ICMP (di tipo *time exceeded*) al mittente.

Il comando **traceroute** invia una serie di pacchetti UDP con TTL crescente a partire da 1, così da ricevere un ICMP *time exceeded* da ogni router attraversato per giungere a destinazione. In questo modo si può tracciare tutta la strada fatta da un pacchetto. Un esempio del funzionamento del comando è il seguente:

³⁴questo viene fatto per evitare che i pacchetti persi nei *routing loop* continuino a circolare sulla rete, con spreco di banda.

```

traceroute to picard.linux.it (62.177.1.107), 30 hops max, 38 byte packets
 1 s35-swa-in.net.playnet.it (62.48.32.2)  0.614 ms  0.575 ms  0.464 ms
 2 r71-bgpactive-in.net.playnet.it (62.48.45.18)  1.068 ms  0.689 ms  0.714 ms
 3 r37-wind-out.net.playnet.it (62.48.45.129)  0.701 ms  1.264 ms  0.712 ms
 4 151.5.148.85 (151.5.148.85)  1.576 ms  1.489 ms  2.087 ms
 5 FIAR-B01-Ge10-0.80.wind.it (151.6.69.225)  1.827 ms  1.628 ms  2.209 ms
 6 151.6.0.117 (151.6.0.117)  7.949 ms  7.373 ms  7.297 ms
 7 MIOT-N01-MIOT-T01-Ge0-0-0.wind.it (151.6.0.121)  7.446 ms  7.497 ms  7.455 ms
 8 * * *
 9 81.208.50.58 (81.208.50.58)  7.892 ms  7.968 ms  7.828 ms
10 81.208.50.211 (81.208.50.211)  7.943 ms  8.369 ms  8.454 ms
11 213.140.31.130 (213.140.31.130)  8.447 ms  8.749 ms  7.831 ms
12 213.156.61.18 (213.156.61.18)  13.066 ms  12.633 ms  12.830 ms
13 sircore1-ext.publinet.it (62.177.0.1)  14.936 ms  13.421 ms  13.553 ms
14 picard.linux.it (62.177.1.107)  14.630 ms  15.402 ms  14.452 ms

```

Così se per un qualche motivo non riuscite a raggiungere il vostro indirizzo di destinazione potete verificare se questo è dovuto al fatto che la strada che prendono i vostri pacchetti è interrotta da qualche parte,³⁵ visualizzando dove si fermano sulla strada verso la destinazione.

Opzione	Significato
-l	stampa anche il TTL dei pacchetti ricevuti come risposta, utile per verificare la presenza di un routing asimmetrico.
-f first	imposta il valore del TTL del primo pacchetto.
-i iface	invia i pacchetti con l'indirizzo dell'interfaccia iface ; ha senso solo quando ci sono più indirizzi sulla stessa macchina.
-n	non effettua le risoluzioni di nomi e indirizzi.
-I	invia pacchetti ICMP invece che UDP.
-s source	usa l'indirizzo source come indirizzo sorgente dei pacchetti inviati.
-m	imposta il valore massimo del TTL usato (il default è di 30 hop).
-w time	imposta il numero di secondi time da attendere per ricevere una risposta.

Tabella 7.19: Opzioni principali del comando **traceroute**.

Il comando prende come argomento l'indirizzo (numerico o simbolico) di cui si vuole tracciare la rotta, ed opzionalmente la dimensione dei pacchetti da usare. Il comando supporta anche numerose opzioni, che permettono di impostare altre caratteristiche dei pacchetti inviati. Al solito si sono riportate quelle principali in tab. 7.19; per le restanti si faccia riferimento alla pagina di manuale.

Un programma analogo a **traceroute**, e disponibile anche in versione con interfaccia grafica, è **mtr**, che mostra dinamicamente lo stato della strada percorsa dai pacchetti, insieme con una serie di statistiche relative al loro inoltro, unificando le funzionalità dei due comandi **traceroute** e **ping**.

Il comando prende come argomento l'indirizzo IP o simbolico della macchina da tracciare, ed invia una serie di pacchetti ICMP di tipo *echo request* con TTL crescente, così può misurare sia il tempo di percorrenza con la ricezione degli *echo reply*, che la strada seguita con la ricezione dei *time exceeded*. Inoltre il comando, a meno di non averne preimpostato un numero specifico con l'opzione **-c**, esegue indefinitamente la ripetizione dell'invio dei pacchetti, raccogliendo e stampando in tempo reale una serie di statistiche relative alla rotta da esso tracciata. In questo caso si potrà terminare il programma con la solita interruzione (con **C-c**) o premendo il tasto "q".

³⁵sempre che qualche amministratore di rete troppo zelante non si sia messo a filtrare anche i pacchetti UDP usati dal protocollo.

```

Truelite
File Edit View Terminal Tabs Help

My traceroute [v0.69]
monk.truelite.it (0.0.0.0) (tos=0x0 psize=64 bitpattern=Fri Feb 24 18:05:25 2006
Keys: Help Display mode Restart statistics Order of fields quit

          Packets
Host      Loss%  Snt   Last   Avg   Best  Wrst StDev
1. pat.truelite.it      0.0%   78    0.2    0.4    0.1   7.9   1.3
2. 62.94.58.63          0.0%   78   51.0  158.8   48.3 648.6 163.6
3. al-0-10.mil5.eb.eutelia.it 0.0%   78  108.0   85.2   74.0 130.3  15.0
4. uli.mix-it.net       0.0%   78   77.7   81.2   74.8 121.5   9.7
5. f2-0.gw-border.uli.it 0.0%   78   78.2   91.2   74.9 440.3  53.7
6. f0-1.gw-brain-border.uli.it 0.0%   78   76.1   95.4   76.1 255.8  43.8
7. r76-1.bgpactive-out.net.playnet. 0.0%   78   97.1   93.2   82.6 271.8  27.5
8. dodds.truelite.it    0.0%   78   85.9   88.9   83.3 143.9   8.1

```

Figura 7.6: Schermata dei risultati del comando `mtr`.

Un esempio dell'uscita del programma, invocato in modalità testo con l'opzione `-t`, è riportato in fig. 7.6, le colonne mostrano le macchine attraversate (la cui risoluzione avviene in tempo reale), la percentuale di perdita dei pacchetti su ciascun tratto, e le statistiche sui tempi di trasmissione rilevati.

Opzione	Significato
<code>-a addr</code>	usa l'indirizzo <code>addr</code> per forzare l'invio dei pacchetti in uscita sulla relativa interfaccia.
<code>-c count</code>	ripete <code>count</code> cicli di invio e poi termina automaticamente. ³⁶
<code>-r</code>	abbinato a <code>-c</code> stampa la statistica finale dei risultati ottenuti alla fine dei cicli.
<code>-n</code>	non effettua la risoluzioni di nomi e indirizzi.
<code>-t</code>	usa l'interfaccia testuale.
<code>-i time</code>	imposta un numero di secondi <code>time</code> da far passare fra un invio ed il successivo (il default è 1).

Tabella 7.20: Opzioni principali del comando `mtr`.

Un altro vantaggio di `mtr` nei confronti di `traceroute` è che usando pacchetti ICMP esso riceve risposta anche dai router che filtrano i pacchetti UDP in presenza dei quali `traceroute` si ferma, considerandoli come irraggiungibili. Le opzioni principali di `mtr` sono riportate in tab. 7.20, per l'elenco completo si consulti la pagina di manuale.

7.5.3 Il comando `netstat`

Un altro comando diagnostico molto utile che permette di visualizzare una grande quantità di informazioni relative alla rete, è `netstat`. Il comando è piuttosto complesso dato che permette di ottenere informazioni riguardo a tutte le funzionalità del sistema concernenti la rete, anche se lo scopo per cui viene usato più spesso è quello di visualizzare tutte le connessioni attive su una macchina (un po' come potrebbe fare il quadro di un centralino che mostra tutti i collegamenti in corso).

³⁶si tenga conto che usando questa opzione senza chiamare anche `-r` la schermata con i risultati viene visualizzata solo durante l'esecuzione del comando, e cancellata alla sua terminazione.

Il comando prevede che la prima opzione indichi il tipo di informazione da mostrare, l'elenco di quelle disponibili è riportato in tab. 7.21, ma se non se ne indica nessuna viene assunto il comportamento di default che è quello di mostrare la lista di tutti i socket aperti nel sistema.

Opzione	Significato
-r	mostra le informazioni relative alla tabella di instradamento (equivalente all'uso di <code>route</code> senza argomenti).
-g	mostra le informazioni relative ai gruppi di <i>multicast</i> di cui si fa parte. ³⁷
-i	mostra le informazioni relative alle interfacce di rete (equivalente all'uso di <code>ifconfig</code> senza argomenti).
-M	mostra le informazioni relative alle connessioni mascherate dal firewall (funziona solo con i kernel della serie 2.2.x).
-s	mostra delle statistiche sommarie sull'uso dei vari protocolli di rete.

Tabella 7.21: Opzioni del comando `netstat` per il controllo del tipo di informazioni da visualizzare.

Quando viene usato con le opzioni di default il comando mostra le informazioni riguardo a *tutti* i socket aperti. La lista comprende anche i socket locali, che sono usati da vari programmi come meccanismo di intercomunicazione, e che non hanno nulla a che fare con la rete. Per questo motivo è opportuno specificare le opzioni `-t` per richiedere di visualizzare solo i socket TCP o `-u` per vedere quelli UDP, che sono quelli che riguardano le connessioni con la rete esterna. Un possibile esempio del risultato di `netstat` è allora il seguente:

```
[piccardi@gont piccardi]$ netstat -t
tcp        0      0 ppp-42-241-98-62.:32798 serverone.firenze:imaps ESTABLISHED
```

che mostra una connessione attiva verso un server di posta.

Invocato in questa maniera il comando riporta solo i dati relativi ai socket attivi, in realtà oltre a questi saranno presenti anche i socket per i quali non ci sono connessioni in corso, ma che sono in ascolto, in attesa di riceverne. Questi vengono mostrati soltanto se si usa l'opzione `-a`; nel qual caso il precedente risultato sarebbe stato:

```
[piccardi@gont piccardi]$ netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:printer               *:*                     LISTEN
tcp        0      0 *:5865                   *:*                     LISTEN
tcp        0      0 *:webcache               *:*                     LISTEN
tcp        0      0 *:tproxy                 *:*                     LISTEN
tcp        0      0 gont.earthsea.ea:domain *:*                     LISTEN
tcp        0      0 localhost:domain        *:*                     LISTEN
tcp        0      0 *:ssh                    *:*                     LISTEN
tcp        0      0 *:ipp                     *:*                     LISTEN
tcp        0      0 *:nntp                   *:*                     LISTEN
tcp        0      0 *:smtp                   *:*                     LISTEN
tcp        0      0 ppp-42-241-98-62.:32798 serverone.firenze:imaps ESTABLISHED
```

Infine usando l'opzione `-n` i risultati verranno stampati con tutti gli indirizzi ed i numeri delle porte espressi in forma numerica, senza che ne sia effettuata la *risoluzione* (vedi sez. 7.4) nei rispettivi nomi simbolici. Le altre principali opzioni di controllo, non rientranti in quelle già illustrate in tab. 7.21, sono riportate in tab. 7.22.

Quando si usa `netstat` per verificare lo stato delle connessioni di rete, il comando genera una tabella con una voce per ciascuna di esse, come illustrato negli esempi precedenti. Il campo

³⁷ per poter utilizzare una comunicazione in *multicast* una macchina deve registrarsi presso il router per segnalare appunto a quale *gruppo di multicast* intende aderire; in questo modo il router potrà reinviarle i pacchetti di *multicast* ricevuti per esso.

Opzione	Significato
-t	mostra i dati relativi ai socket TCP.
-u	mostra i dati relativi ai socket UDP.
-n	non esegue la risoluzione di indirizzi e porte.
-c	ripete continuamente il comando ogni secondo.
-e	stampa informazioni aggiuntive, può essere ripetuto due volte per avere ancora più informazioni.
-l	mostra i dati relativi ai socket in ascolto (di default non vengono mostrati).
-a	mostra i dati relativi sia ai socket attivi che a quelli in ascolto.

Tabella 7.22: Opzioni principali del comando `netstat`.

Proto riporta il protocollo della connessione. I campi **Local Address** e **Foreign Address** indicano gli indirizzi locale e remoto della stessa, che nel caso di socket su internet sono nella forma:

`indirizzo:porta`

dove un asterisco indica un indirizzo o una porta qualunque. Il campo **State** indica lo stato della connessione. Una spiegazione dettagliata del significato dei valori di questo campo va di nuovo al di là delle finalità di questo testo, e richiede una trattazione approfondita del protocollo TCP/IP,³⁸ per cui ci limiteremo ad alcune note panoramiche.

Delle varie righe quelle che meritano attenzione sono quelle relative agli stati **LISTEN** ed **ESTABLISHED**. Lo stato **LISTEN** indica la presenza di un programma in ascolto sulla vostra macchina in attesa di connessione, nell'esempio precedente ce ne sono vari, corrispondenti a servizi come la posta, le news, il DNS, la stampa via rete. Gli indirizzi di norma non sono specificati in quanto la connessione può essere effettuata su uno qualunque degli indirizzi disponibili sulle interfacce locali, e a partire da un qualunque indirizzo esterno. Lo stato **ESTABLISHED** indica le connessioni stabilite ed attive, e riporta nei campi degli indirizzi i numeri di IP e porta dei due capi della connessione. Altri stati che possono essere riportati dal comando sono **FIN_WAIT**, **TIME_WAIT**, che si riferiscono a connessioni che si stanno chiudendo.

Si noti anche che quando si esegue `netstat` con l'opzione `-u` per rilevare lo stato dei socket UDP, nella tabella risultante il campo **State** è vuoto anche quando ci sono servizi in ascolto, in quanto lo stato è definito soltanto per i socket TCP.

7.5.4 Il protocollo ARP ed il comando `arp`

Come accennato in sez. 7.2.1 il protocollo ARP viene usato (dal kernel) per associare ad un indirizzo fisico presente sulla rete locale (quello che per gran parte delle reti è il cosiddetto *MAC address*) al corrispondente indirizzo IP, cosicché il kernel possa sapere a quale scheda (ethernet nel caso più comune) mandare i pacchetti.

In realtà il protocollo viene utilizzato tutte le volte che il kernel deve inviare un pacchetto IP verso l'esterno su una rete di tipo ethernet (o con funzionalità equivalenti, come il token ring o l'FDDI). Le possibilità sono sempre due, o si deve comunicare con un indirizzo IP nella stessa sottorete, nel qual caso si invierà il pacchetto direttamente al destinatario tramite l'indirizzo fisico della sua scheda di rete, o si deve inviare il pacchetto altrove passando attraverso un gateway, nel qual caso dalla tabella di routing si otterrà l'indirizzo IP di quest'ultimo che sarà usato per ricavare il MAC address della scheda a cui inviare il pacchetto.

Il protocollo viene usato per mandare delle richieste in *broadcast*, cioè richieste che vengono ricevute da tutte le schede su una stessa LAN; queste hanno la tipica forma “*devi dire chi è X.X.X.X a Y.Y.Y.Y*” dove *X.X.X.X* è l'indirizzo IP che si vuole risolvere, ed *Y.Y.Y.Y* è quello del richiedente. Quest'ultimo viene automaticamente identificato dato che il suo *MAC address*

³⁸una trattazione completa è in [4], una sintesi si può trovare nelle appendici di [1].

è riportato come indirizzo sorgente nel pacchetto di richiesta, così che la macchina la cui interfaccia ha l'IP `X.X.X.X` può rispondere direttamente con un messaggio del tipo "`X.X.X.X è NN:NN:NN:NN:NN:NN`" trasmettendo il suo *MAC address* direttamente al richiedente.

In questo modo una macchina può interrogare le sue vicine sulla stessa LAN e costruire un elenco di corrispondenze. Per evitare di oberare la rete di richieste le corrispondenze trovate vengono mantenute per un certo tempo in quella che viene chiamata la *ARP cache* del kernel, e rinnovate solo dopo che sono scadute.

In certi casi può essere utile esaminare e modificare la *ARP cache*. Ad esempio una tecnica che impiega questo protocollo è quella del cosiddetto *proxy ARP*, usata quando si divide in due reti separate una rete che prima era unica, introducendo un router a separare i due nuovi rami.

Dato che un router non trasmette i pacchetti a livello di collegamento fisico, impostando su di esso un *proxy ARP* si fa sì che alle richieste ARP per gli IP posti sul nuovo tratto di rete al di là del router venga comunque risposto, fornendo l'indirizzo fisico della scheda del router. In questo modo, anche se non si è impostato su tutte le macchine una opportuna rotta statica per la nuova configurazione della rete, il router riceverà i pacchetti destinati al nuovo tratto, e potrà inoltrarli. Ovviamente questa è una soluzione provvisoria, che come tutte le soluzioni provvisorie ha la brutta abitudine di diventare spesso definitiva.

Il comando che permette di esaminare e modificare la *ARP cache* del kernel è `arp`. Se chiamato senza opzioni il comando mostra il contenuto della cache, ad esempio se eseguiamo il comando da `gont` avremo:

[root@gont corso]# arp					
Address	HWtype	HWaddress	Flags	Mask	Iface
oppish.earthsea.ea	ether	00:48:54:3A:9A:20	C		eth0
havnor.earthsea.ea	ether	00:48:54:6A:4E:FB	C		eth0

Le due operazioni fondamentali del comando `arp` sono la cancellazione (con l'opzione `-d`) e l'inserimento (con l'opzione `-s`) di una voce nella cache, ad esempio possiamo cancellare la voce relativa ad `oppish` con il comando:

```
arp -d oppish
```

mentre si può inserire una voce in più con il comando:

```
arp -s lorbaner 00:48:54:AA:9A:20
```

le altre opzioni principali sono riportate in tab. 7.23, al solito per una spiegazione completa si può fare ricorso alla pagina di manuale con `man arp`.

Opzione	Significato
<code>-a host</code>	mostra le voci relative al nodo <code>host</code> , se non specificato mostra tutte le voci.
<code>-f file</code>	legge i valori da immettere nella cache dal file <code>file</code> .
<code>-n</code>	non effettua la risoluzioni di nomi e indirizzi.
<code>-s host val</code>	inserisce una voce nella cache, associando al nodo <code>host</code> l'indirizzo <code>val</code> .
<code>-d host</code>	cancella la voce relativa al nodo <code>host</code> .

Tabella 7.23: Principali opzioni del comando `arp`.

Come per la risoluzione dei nomi (vedi ad esempio sez. 7.4.4) è possibile anche specificare un certo numero di corrispondenze manualmente, usando un opportuno file di configurazione. Nel caso specifico questo file è `/etc/ethers`, che contiene delle corrispondenze nella forma:

```
MAC-address indirizzo-IP
```


ed al posto dell'indirizzo IP si può anche specificare un nome simbolico, posto che questo sia risolvibile. Il formato dei *MAC address* è quello solito di sei byte esadecimali separati dal carattere “:”, il contenuto tipico di */etc/ethers* sarà cioè qualcosa del tipo:

```
08:00:20:00:61:CA 129.168.1.245
```

7.5.5 I servizi RPC

In sez. 7.2.4 abbiamo visto come normalmente i servizi di rete vengano forniti utilizzando una porta, il cui valore li identifica univocamente. Con il crescere del numero di servizi che potevano essere resi disponibili questa assegnazione statica ha iniziato ad essere percepita come problematica, in quanto le porte possibili sono solo 65535, numero che si potrebbe esaurire con una certa facilità se se ne dovesse assegnarne una ad ogni servizio possibile.

Ad una prima impressione questo può sembrare poco realistico, ma quando si inizia a pensare di rendere disponibile sulla rete una qualunque funzione di sistema attraverso dei servizi appositi (che per questo vengono chiamati *Remote Procedure Call* o RPC), il problema può diventare reale. Per questo motivo si è passati a progettare un meccanismo che ne consentisse l'allocazione dinamica.

Il concetto che sta dietro le *Remote Procedure Call* è quello di un meccanismo di intercomunicazione generico basato sui protocolli di trasporto (sia TCP che UDP), che permetta ad un processo che gira su una certa macchina di accedere a dei servizi (nella forma di chiamate a funzioni esterne) in maniera trasparente rispetto alla rete. In sostanza una sorta di libreria eseguita via rete, in cui si demanda l'elaborazione dei dati ad una macchina remota, che restituirà i risultati attraverso la rete.

Ciascun servizio RPC è fornito da un apposito demone, in ascolto su una porta non definita a priori, in modo da poterne definire quanti se ne vuole, ma di impiegare delle porte solo per quelli effettivamente utilizzati. Un programma che voglia utilizzare un servizio RPC deve prima rivolgersi ad un servizio speciale, il *portmapper*, che registra i servizi attivi ed è in grado di indicare la porta su cui sono forniti.

Il servizio del *portmapper* è l'unico a cui viene assegnata una porta fissa, la 111, che corrisponde al nome *sunrpc*.³⁹ Tutti gli altri servizi RPC (in sez.8.4 vedremo che NFS utilizza questa infrastruttura) quando si avviano si registrano presso il *portmapper* utilizzando un numero identificativo, ed indicano la porta su cui sono in ascolto.

In sostanza nel caso di servizi RPC si fa effettuare al *portmapper* una risoluzione dinamica fra un servizio che è stato registrato dal rispettivo server ed il numero di porta che questo sta utilizzando. In questo modo è possibile richiedere il servizio usando il numero identificativo, che è completamente indipendente dal numero di porta e non soggetto ai limiti di quest'ultimo.

La lista dei servizi RPC viene mantenuta in un apposito file⁴⁰ */etc/rpc* che è l'analogo di */etc/services*. Il formato del file prevede tre campi separati da spazi; il primo indica il nome ufficiale del servizio RPC, il secondo il numero identificativo ad esso associato, il terzo la lista (separata da spazi) degli alias del nome ufficiale. Un esempio di questo file, preso da una Debian Sarge, è:

```
# This file contains user readable names that can be used in place of rpc
# program numbers.

portmapper      100000  portmap sunrpc
rstatd          100001  rstat rstat_svc rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
```

³⁹i servizi RPC sono stati introdotti per la prima volta da Sun per i suoi Unix, da cui questo nome per il servizio associato al *portmapper*.

⁴⁰in realtà anche in questo caso si usa il *Name Service Switch*.

```

ypserv      100004  ypprog
mountd      100005  mount showmount
ypbind      100007
walld       100008  rwall shutdown
ypasswdd    100009  yppasswd
etherstatd  100010  etherstat
rquotad     100011  rquotaprogram quota rquota
...

```

Per poter usufruire dei servizi RPC è pertanto necessario attivare il *portmapper*; questo viene usualmente fatto direttamente dagli script di avvio, attraverso l'uso dal demone **portmap**. Dato che questo demone viene è essenziale al funzionamento dei servizi RPC, esso viene installato di default da quasi tutte le distribuzioni.

Per una maggiore sicurezza, se viene usato solo per servizi utilizzati localmente, sarebbe opportuno lanciarlo con l'opzione **-i** che permette di indicare un indirizzo specifico su cui porsi in ascolto, facendogli utilizzare esclusivamente il **localhost**. Per le altre opzioni si faccia riferimento come al solito alla pagina di manuale.

Se il *portmapper* è attivo si può verificare il funzionamento dei servizi RPC con il comando **rpcinfo**; questo, se invocato con l'opzione **-p**, prende come argomento l'indirizzo di un nodo (se non si indica nulla usa la macchina locale) e stampa una lista di tutti i servizi registrati; un esempio potrebbe essere:

```

piccardi@monk:~/truedoc/corso$ rpcinfo -p
program vers proto  port
 100000    2   tcp    111  portmapper
 100000    2   udp    111  portmapper
 100024    1   udp    787  status
 100024    1   tcp    790  status

```

Il comando consente anche, con l'opzione **-u** di interrogare un nodo per la presenza di uno specifico servizio (passato come secondo argomento) e di inviare richieste in *broadcast* sulla rete per rilevare la presenza di altre macchine che forniscono servizi RPC (nel qual caso occorrerà passare come argomenti il numero del servizio e quello di versione). Per maggiori dettagli e l'elenco completo delle opzioni si faccia riferimento alla pagina di manuale.

7.6 I client dei servizi di base

Una volta che se ne sia completata la configurazione e verificato il funzionamento, l'utilizzo della rete avviene attraverso l'uso dei servizi che su di essa vengono forniti. In generale i servizi di base sono forniti tutti secondo un'architettura client-server; affronteremo più avanti la configurazione del lato server di alcuni di essi (la maggior parte dei servizi corrispondenti ai client qui descritti viene fornita attraverso uno dei *superdemoni* descritti in sez. 8.1), qui ci limiteremo a descrivere il funzionamento dei programmi client usati per usufruire del servizio.

7.6.1 I comandi telnet e netcat

Uno dei servizi di base un tempo più utilizzato sulla rete è quello del *telnet*, nato per eseguire delle connessioni con cui poter operare via rete su un terminale su una macchina remota. Dato che tutti i dati vengono trasmessi in chiaro sulla rete, password comprese, questo uso è assolutamente da evitare, ed oggi è essenzialmente sostituito dal comando **ssh** che vedremo in sez. 8.3.2.

In ogni caso il comando **telnet** continua ad esistere, e può essere usato su una rete locale sicura o quando non esiste una versione di SSH per la macchina in questione (ad esempio se si ha a che fare con un vecchio VAX). Un suo uso più interessante però è quello diagnostico, per verificare la funzionalità dei servizi.

Il comando prende come argomento la macchina cui collegarsi ed opzionalmente la porta (specificata per numero o per nome del servizio); non specificando nessuna porta il comando si collega sulla porta 23, che corrisponde al servizio standard *telnet*, mostrando (qualora sia attivo il relativo server) una schermata di login.

Il comando supporta varie opzioni, per lo più relative alle modalità di gestione della sessione di terminale remota. Dato che questo uso è assolutamente sconsigliabile non le riportiamo neanche, i più curiosi possono consultare la pagina di manuale accessibile con `man telnet`. Di norma si usa il comando indicando semplicemente un indirizzo ed una porta, con qualcosa del tipo di:

```
piccardi@oppish:~$ telnet localhost 25
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 oppish.earthsea.ea ESMTP Postfix (Debian/GNU)
```

in questo caso lo si è usato per connettersi alla porta 25 della propria macchina, per verificare che il relativo servizio SMTP (cioè il server per la ricezione e la trasmissione della posta) sia attivo: si noti che si è poi anche ottenuta la risposta del server che nel caso in questione è *Postfix*.

Con la stessa sintassi si può usare il comando per verificare la presenza e l'attività dei servizi sulle relative porte. Anche se non si riceve nessuna risposta (ad esempio perché il servizio interrogato necessita di una richiesta prima di rispondere), la presenza della riga **Connected** indicherà che la connessione è riuscita ed è attiva.

Se invece tentiamo di collegarci ad un servizio su cui non è attivo nessun server otterremo qualcosa del tipo:

```
piccardi@oppish:~$ telnet localhost 80
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
```

con un ritorno pressoché immediato alla riga di comando. Questo è il comportamento normale, ma quando la macchina contattata viene protetta da un firewall, è configurazione comune bloccare direttamente tutti i pacchetti in ingresso se non sui servizi consentiti; questo comporta anche che il tentativo di connessione su una porta senza servizi venga completamente ignorato. In questa evenienza, non ricevendo nessuna risposta⁴¹ il comando `telnet` resterebbe bloccato in attesa di una risposta fino alla scadenza del timeout (il default è circa 30 secondi), riportando poi un errore diverso.

Un secondo comando che si può utilizzare per effettuare una verifica sui servizi è `netcat`, che può essere invocato anche in forma più compatta con `nc`. Il grande vantaggio rispetto a `telnet` è che mentre quest'ultimo funziona soltanto su TCP, `netcat` può essere anche usato su UDP, e permette quindi di controllare anche i servizi che usano questo protocollo.

Inoltre `netcat`, come suggerisce il nome stesso, si limita a leggere e scrivere i dati su delle connessioni di rete associando *standard input* e *standard output* alla connessione su cui lavora, supporta quindi la redirectione ed il *pipelining*, ed è particolarmente adatto ad essere utilizzato anche all'interno di script. Questo non è possibile con `telnet`, che associa alla connessione un terminale, ed è quindi soggetto alla *disciplina di linea*⁴² di quest'ultimo. Inoltre `telnet`

⁴¹in caso di un tentativo di connessione su una porta su cui non ci sono servizi, il protocollo TCP prevede, per notificare l'errore, l'invio di un pacchetto di risposta al client che ha effettuato il tentativo; alla ricezione di questo pacchetto `telnet` riporta l'errore di connessione rifiutata, come mostrato nell'esempio precedente.

⁴²si chiama così le modalità con cui il dispositivo di terminale gestisce l'I/O, ad esempio il fatto che l'input è bufferizzato linea per linea.

interpreta alcuni caratteri come caratteri di controllo (eliminandoli dal flusso dei dati) e scrive alcuni dei suoi messaggi interni sullo *standard output*.

Al contrario **nc** preserva rigorosamente il flusso dei dati, e se da una parte questo lo rende meno adatto all'amministrazione via rete (ma lo sarebbe comunque, non essendo prevista la cifratura della connessione), dall'altra lo rende uno strumento molto flessibile⁴³ per inviare e ricevere dati dalla rete, e manipolarli usando la riga di comando.

La forma più elementare di invocazione del comando è identica a quella di **telnet**, in cui si specifica il nome di una macchina (o il suo indirizzo) e la porta che si desidera contattare. Ripetendo l'esempio precedente avremo:

```
piccardi@monk:~$ nc localhost 25
220 monk.truelite.it ESMTP Postfix (Debian/GNU)
```

e come si può notare in questo caso non viene scritto niente che non sia quanto stato ricevuto dalla rete, se si vogliono avere i messaggi diagnostici occorre utilizzare l'opzione **-v**, che li stamperà rigorosamente sullo *standard error* (ripetendo due volte l'opzione il comando diventerà più prolisso). Se il servizio che si vuole contattare è su UDP basterà utilizzare l'opzione **-u**.

Un'altra caratteristica interessante del programma è che si possono specificare più porte di destinazione, ed i dati verranno inviati e letti su tutte quante. Questo permette anche di eseguire una scansione elementare per la presenza di servizi attivi, che potrebbe essere realizzata con qualcosa del tipo:

```
echo QUIT | nc -w 1 localhost 25 1-1024 6000-7000
```

dove si è usata l'opzione **-w** che consente di specificare un tempo massimo di durata, per chiudere le connessioni che altrimenti resterebbero aperte.

Un'altra caratteristica interessante di **netcat** è che può essere utilizzato in *listen mode*, cioè anche come "server", usando l'opzione **-l**, che mette il programma in ascolto sulla porta specificata con l'opzione **-p**. In tal caso sarà possibile collegarsi da remoto al programma, che stamperà quanto ricevuto sulla connessione. In questo modo ad esempio è possibile trasferire un file via rete reindirizzando lo standard output su un file locale, utilizzando **nc** sull'altro capo della connessione per inviarlo. Infine usando anche l'opzione **-e** è possibile associare *standard input* e *standard output* di questo alla connessione e trasformare **nc** (se ad esempio si fa eseguire la shell) in un server analogo al servizio *telnet*.

Opzione	Significato
-e cmd	esegue il file specificato, collegando il relativo <i>standard input</i> e <i>standard output</i> alla connessione.
-l	si pone in ascolto sulla porta specificata con -p .
-p port	indica la porta (o le porte) su cui mettersi in ascolto, da usare insieme a -p .
-q secs	attende secs secondi dopo la chiusura dello standard input.
-s addr	specifica l'indirizzo sorgente locale.
-u	esegue le connessioni su UDP.
-v	modalità <i>prolissa</i> , stampa informazioni sullo <i>standard error</i> .
-w secs	aspetta secs per il timeout.

Tabella 7.24: Principali opzioni del comando **netcat**.

In tab. 7.24 si sono riportate le principali opzioni del comando, al solito per l'elenco completo si può fare riferimento alla pagina di manuale.

⁴³tanto che la pagina di manuale lo classifica come "coltellino svizzero del TCP/IP".

7.6.2 Il comando ftp

Il protocollo FTP è uno dei più vecchi protocolli che consentono lo scambio di file su internet. Il protocollo permette di prelevare od immettere file su un server FTP, previa autenticazione analoga a quella del login o dell'accesso con telnet. Una sua forma particolare è il cosiddetto *FTP anonimo* in cui il servizio viene utilizzato per distribuire i file posti in un server (in questo caso il servizio non richiede autenticazione, e di norma consente solo il prelievo dei file).

Dato che il protocollo, come *telnet*, non prevede alcuna cifratura dei dati, è meglio non usarlo in quanto l'autenticazione verrebbe eseguita in chiaro; unica eccezione è la modalità anonima in cui non c'è autenticazione che è tutt'oggi una delle modalità più usate per distribuire pubblicamente file.⁴⁴ Qualora si necessiti di un meccanismo per lo scambio di file via rete fra i vari utenti sono disponibili alternative come l'uso di **scp** o **sftp**⁴⁵ che tratteremo in sez. 8.3.

Oggi esistono molti client per il protocollo FTP, sia grafici che testuali; ma il comando originale usato per lanciare il client testuale è semplicemente **ftp**. Un esempio di uso generico del comando è:

```
piccardi@oppish:~$ ftp ftp.linux.it
Connected to vlad-tepes.bofh.it.
220 (vsFTPd 1.1.3)
Name (ftp.linux.it:piccardi): anonymous
331 Please specify the password.
Password:
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

eventuali opzioni, ed anche la stessa risposta, dipendono in genere dalla versione del comando che si è installata,⁴⁶ e vanno verificate facendo riferimento alla relativa pagina di manuale.

Comando	Significato
ls	stampa la lista dei file (sul server).
cd	cambia directory (sul server).
lcd	cambia directory (sul client).
pwd	stampa la directory corrente (sul server).
ascii	modalità di trasferimento di file ascii.
binary	modalità di trasferimento di file binari.
pasv	abilita i trasferimenti in modo passivo.
get file	scarica il file file dal server.
put file	invia il file file sul server.
mget files*	scarica i file che corrispondono alla wildcard files* .
mput files*	invia i file che corrispondono alla wildcard files* .
open host	apre una connessione verso il server host .
quit	chiude la sessione.

Tabella 7.25: Comandi del protocollo FTP.

Si noti come nell'uso generico sia sufficiente indicare la macchina cui ci si vuole collegare. Nel caso ci viene notificato l'*hostname* effettivo del server e ci viene richiesto un nome di login

⁴⁴ anche questo utilizzo è ampiamente discutibile, in quanto sarebbe comunque più efficiente distribuire i suddetti dati via web usando HTTP, che sovraccarica meno la rete e non presenta tutte le problematiche di funzionamento (in particolare quando si devono attraversare dei firewall) che ha FTP.

⁴⁵ **sftp** è un programma che ha esattamente la stessa sintassi di **ftp**, ma consente l'uso di una connessione cifrata attraverso il protocollo SSH.

⁴⁶ questa è la risposta data dal comando **ftp** che si trova su Debian Woody, derivato dall'originale programma nato con BSD, alternative possono essere client più evoluti come **lftp** o **ncftp** che supportano la storia dei comandi, il completamento dei nomi, e capacità aggiuntive.

che di default corrisponde al nostro username. Avendo contattato un FTP anonimo l'utente da usare è **anonymous** specificato il quale ci viene richiesta una password, che nel caso è ininfluente (qualunque cosa si scriva ci sarà garantito l'accesso).

Fatto questo ci si ritrova con un prompt **ftp>** dal quale sarà possibile inviare i vari comandi del protocollo; i più importanti di questi si sono riportati in tab. 7.25. La lista completa è al solito disponibile nella pagina di manuale.

Si tenga inoltre presente che non tutti i comandi possono essere supportati dal server. Ad esempio il comando **pasv**, che abilita il modo passivo,⁴⁷ è eseguibile solo se il server supporta questa modalità di operazione, mentre i comandi **ascii** e **binary** non hanno nessun significato su sistemi unix-like, ma possono averlo per altri sistemi come il VMS o il DOS che introducono una distinzione fra questi tipi di file.

7.6.3 I comandi finger e whois

Il comando **finger** viene usato fin dalle origini di Unix, dove agiva solo sulla macchina locale, per riportare informazioni relative agli utenti di un sistema. In questo modo si poteva consentire agli altri utenti di sapere chi era collegato alla macchina. Il servizio è stato trasferito sulla rete, ed il client funziona sia in locale che in remoto (nel qual caso si eseguirà una richiesta con un parametro del tipo **nomehost**).

Un possibile esempio di uso del comando, fatto in locale, è il seguente:

```
[piccardi@gont piccardi]$ finger franci
Login: franci                      Name: Francesco Piccardi
Directory: /home/franci           Shell: /bin/bash
Home Phone: 0XX-XX000XX
Last login Tue Jun 17 21:22 (CEST) on :0
No mail.
No Plan.
```

e come si vede il programma mostra una serie di informazioni riguardo l'utente **franci**; questi può aggiungervi ulteriori informazioni creando un file **.plan** nella sua home directory, che verrà mostrato all'esecuzione del comando.

Il comando prende come argomento un nome utente che viene confrontato sia con lo username che con il nome reale dello stesso memorizzato su **/etc/passwd**; qualora si voglia eliminare quest'ultimo confronto si può usare l'opzione **-m**. Se invocato senza argomenti mostra l'elenco degli utenti collegati e relativo terminale come:

```
[piccardi@gont corso]$ finger
Login      Name           Tty      Idle  Login Time   Office   Office Phone
piccardi   Simone Piccardi *:0              Jul 21 21:55
```

altre due opzioni utili sono **-s** che mostra una lista semplice come questa e **-l** che mostra una lista lunga come la precedente, per i dettagli si può fare riferimento alla pagina di manuale.

Dato che il comando fornisce delle informazioni che potrebbero essere utili per attività non troppo benevole o per violare la privacy degli utenti (ad esempio l'uso dell'indirizzo di posta elettronica per inviare dello spam), oggi si tende sempre di più a non attivare questo servizio sulla rete. Talvolta però esso viene utilizzato per pubblicare delle informazioni, non legate alla presenza di specifici utenti sulla macchina, come eventuali recapiti e contatti, oppure per distribuire le chiavi GPG degli utenti (usando un file **.pgpkey** nella home directory degli stessi).

⁴⁷il protocollo FTP prevede l'uso di due porte, la connessione avviene sempre da parte del client sulla porta 21 del server, ma su di essa vengono solo inviati i comandi, quando si richiede l'invio di file questo viene effettuato dal server con una seconda connessione che contatta il client sulla porta 20. Dato che normalmente i firewall bloccano le connessioni entranti, il modo passivo fa sì che sia sempre il client a creare la nuova connessione anche per i dati.

Il comando `whois` viene usato per contattare i server che forniscono il servizio *whois*. Questo servizio permette di accedere alle informazioni presenti nel database dei titolari dei domini su internet, che vengono mantenute dagli enti che sono responsabili della registrazione degli stessi (in Italia il NIC). In genere lo si utilizza quando si cercano informazioni di natura amministrativa relativa ai domini, ad esempio per avere un indirizzo di contatto dei relativi proprietari.

Il comando prende come argomento una stringa di ricerca, normalmente un nome a dominio, e cerca di determinare automaticamente qual'è il server più opportuno a cui rivolgersi.⁴⁸, il quale restituisce le informazioni che verranno stampate a video, un esempio di uso del comando è:

```
[piccardi@gont piccardi]$ whois truelite.it
*****
* Please note that any results obtained are a      *
* subgroup of the data contained in the database *
*
* The full objects' data can be visualised at:    *
* http://www.nic.it/RA/database/index.html      *
*****

domain:      truelite.it
org:         Truelite srl
admin-c:     CS3862-ITNIC
tech-c:      EDM149-ITNIC
postmaster:  CS3862-ITNIC
zone-c:      EDM149-ITNIC
nserver:     62.48.34.25 NS.TRUELITE.IT
nserver:     62.94.0.2 DNS2.EDISONTEL.IT
remarks:     On going transfer 20041125
mnt-by:      EUTELIA-MNT
created:     20020819
expire:      20061209
source:      IT-NIC

person:      Christian Surchi
address:     TRUELITE SRL
address:     VIA MONFERRATO, 6
address:     50142 Firenze (FI)
nic-hdl:     CS3862-ITNIC
source:      IT-NIC

person:      Eutelia Domain Managing
address:     Eutelia S.p.A.
address:     Via P.Calamandrei 173
address:     52100 Arezzo
nic-hdl:     EDM149-ITNIC
source:      IT-NIC
```

e come si può notare si ottengono tutta una serie di informazioni relative a chi gestisce (come il titolare) un nome a dominio.

Si può richiedere al comando di contattare un server specifico, usando l'opzione `-h`, mentre con `-H` si indica al comando di non mostrare le note legali che molti server inviano. Per i dettagli sulle altre opzioni si consulti la pagina di manuale o si invochi il comando senza argomenti.

7.7 Le connessioni di rete con PPP

Finora abbiamo trattato delle configurazioni di rete facendo riferimento alla casistica tipica di una rete locale; in questa sezione esamineremo invece la configurazione della rete per connessioni

⁴⁸nel caso non riesca a determinarne uno più specifico, viene contattato il server di riferimento internazionale che è `whois.networksolutions.com`.

punto-punto, come quelle che si ottengono tutte le volte che ci si connette ad internet tramite un modem, ed in particolare tratteremo il protocollo PPP, che consente di realizzare questo tipo di connessioni, e dei programmi e relative configurazioni che ci consentono di usarlo.

7.7.1 Cenni sul protocollo PPP

Le configurazioni di base illustrate in sez. 7.3 abbiamo sempre fatto riferimento al caso di una macchina inserita all'interno di un tratto di rete (il caso più comune è l'uso di *ethernet*), per cui ad una interfaccia non viene associato solo un indirizzo IP, ma anche una maschera di rete.

In generale però questo non è vero, ed è infatti è pratica comune collegarsi ad internet attraverso un modem (analogico o ADSL che sia). In tal caso la comunicazione verso l'esterno non avviene attraverso il passaggio per una rete locale, ma direttamente con una macchina posta all'altro capo della connessione, con quello che si chiama un collegamento *punto-punto*.

In sez. 7.2.1 abbiamo visto che esiste un apposito protocollo del livello di collegamento, il *Point to Point Protocol* (comunemente detto PPP) su cui è possibile innestare tutti i protocolli dei livelli successivi. Nel caso specifico PPP è un protocollo generico che permette di inviare dati su diverse tipologie di collegamenti fisici, mantenendo una stessa struttura generale, che consente di creare, inviare e gestire connessioni punto-punto.

Il protocollo si compone di tre parti, un metodo generico per incapsulare dati su un collegamento fisico, un protocollo estensibile per il controllo del collegamento fisico (il *Link Control Protocol* o LCP) ed una famiglia di protocolli (il *Network Control Protocols* o NCP) per stabilire e configurare le connessioni coi protocolli di livello superiore.

In generale PPP fa riferimento ad un qualche meccanismo sottostante di trasmissione (che sia un modem analogico, ISDN o ADSL) sopra il quale costruisce uno strato ulteriore che gli permette di incapsulare i protocolli di livello superiore. L'uso più tipico di questo protocollo è comunque quello che viene fatto con i modem analogici o ADSL, per i quali permette di gestire la creazione della connessione verso il fornitore del servizio di connessione e la configurazione della relativa interfaccia.

7.7.2 Il demone pppd

Delle tre parti, illustrate in sez. 7.7.1, di cui è composto il protocollo PPP, l'incapsulazione dei dati è gestita direttamente dal kernel; il relativo supporto è abilitato di default in tutte le distribuzioni, ed in genere da questo punto di vista non c'è da fare niente, se non assicurarsi, qualora si sia ricompilato il proprio kernel (vedi sez. 5.1.3) di aver incluso quanto necessario nella sezione *Network device support*.

Tutto il resto è affidato ad un apposito demone, **pppd**, che si cura di fornire il controllo del collegamento, l'autenticazione, ed la parte dell'NCP che riguarda la configurazione e la gestione del protocollo IP su PPP.

La sintassi generica con cui si può chiamare il programma **pppd**, come riportata dalla pagina di manuale, è la seguente:

```
pppd [ ttyname ] [ speed ] [ options ]
```

dove **ttyname** specifica il dispositivo da usare per la comunicazione, **speed** la velocità di comunicazione dello stesso in *baud* e **options** indica tutte le altre opzioni. Ad esempio, per far partire la connessione con un modem attaccato alla prima seriale, si potrebbe usare il comando:

```
pppd /dev/ttyS0 115200 connect "/usr/sbin/chat -v -f /etc/chatscripts/provider"
```

dove l'opzione **connect** viene usata per specificare il comando da usare per far impostare correttamente la linea seriale prima di far partire la connessione.

In realtà non è necessario specificare nessuna opzione a riga di comando, esse di norma vengono lette all'avvio del demone dal file `/etc/ppp/options`, o, per quelle che sono impostabili da un normale utente, dal file `.pppdrc` nella home dello stesso. Se si è specificato a riga di comando un particolare dispositivo (come `/dev/ttyS0` nell'esempio), le opzioni verranno prese, se esiste, dal file `/etc/ppp/options.ttyNAME` (nel caso `options.ttyS0`).

Una forma diversa di chiamata del demone può essere attraverso l'opzione `call`, che permette di personalizzare le opzioni in base al fornitore di servizi (il *provider*) che si vuole utilizzare; in questo infatti caso le opzioni saranno lette da un file posto nella directory `/etc/ppp/peers`, così se `/etc/ppp/peers/adsl` contiene le opzioni per chiamare il provider con un modem ADSL, potremo attivare la connessione con un comando del tipo di:

```
pppd call adsl
```

Le opzioni più comuni si sono riportate in tab. 7.26, insieme ad una loro breve descrizione, la descrizione e l'elenco completo delle varie opzioni sono disponibili nelle pagine di manuale accessibili con `man pppd`.

Direttiva	Significato
<code>call name</code>	usa le opzioni contenute nel file <code>name</code> in <code>/etc/ppp/peers</code> .
<code>connect script</code>	usa lo script <code>script</code> (di norma <code>chat</code>) per preimpostare la linea.
<code>crtstcts</code>	attiva il controllo di flusso hardware sulle porte seriali.
<code>defaultroute</code>	aggiunge una rotta di default alla tabella di instradamento usando l'indirizzo IP ricevuto sulla connessione.
<code>lock</code>	crea un file di lock per la seriale.
<code>demand</code>	crea la connessione a richiesta quando vede del traffico.
<code>usepeerdns</code>	chiede alla connessione gli indirizzi di due server DNS, che vengono passati allo script <code>/etc/ppp/ip-up</code> nelle variabili di ambiente <code>DNS1</code> e <code>DNS2</code> ed usati per la creazione di un file <code>resolv.conf</code> che li usi come server DNS.
<code>debug</code>	abilita il debugging, è utile per avere più informazioni quando la connessione non funziona.

Tabella 7.26: Principali opzioni per il demone `pppd`.

Una volta lanciato il demone stabilisce una connessione sulla linea indicata tramite le opzioni. Di norma perché questo avvenga è necessario che la linea venga opportunamente preparata; ad esempio se si ha un modem occorrerà che questo effettui la chiamata telefonica ed attivi la connessione. Di solito questo si fa attraverso l'uso del comando `chat`. Questo ultimo viene lanciato da `pppd` con l'opzione `connect`, che esegue un qualunque programma o script da usare per inizializzare la connessione.

Nel nostro caso `chat` verrà solo usato per dare le istruzioni al modem e verificare la riuscita della chiamata. Le opzioni più significative sono `-f` che permette di indicare un file per lo script di chat, e `-v` che permette di registrare i risultati della comunicazione a scopo di controllo. Le restanti opzioni possono essere trovate nella pagina di manuale accessibile con `man chat`.

Il comando usa uno script di chat con il quale gestisce la comunicazione iniziale con il modem, esso è composto da coppie di valori, in cui il primo indica quello che ci si attende di sentire dal dispositivo ed il secondo quanto scrivere in risposta. Un esempio di file di chat potrebbe essere il seguente:

```
ABORT BUSY
"" ATZ
OK ATDT055507979
CONNECT \d\c
```

in cui le direttive `ABORT` definiscono le condizioni in cui viene abortita la connessione (possono essere `BUSY`, `VOICE`, `NO DIALTONE` ecc.). Si noti come si inizi non aspettandosi niente (questo il

significato della stringa vuota "") cui si risponde con il comando **ATZ** del modem. Alla risposta **OK** si esegue il comando **ATDT055507979** che esegue la telefonata. Alla risposta **CONNECT** lo script finisce e il controllo passa direttamente a **pppd** che a questo punto si suppone sia in grado di parlare con il corrispettivo dall'altra parte.

Si tenga presente che in certi casi è necessaria una procedura di login prima che l'altro capo attivi il suo demone **pppd** sulla connessione, il che può comportare ad esempio la presenza di ulteriori linee del tipo di:

```
login: username
ssword: password
```

o quel che sarà necessario a seconda della risposta dell'altro server (di norma appunto un **login:** cui va risposto con l'username, cui seguirà **password:** a cui rispondere con la password), tenendo conto che basta una corrispondenza parziale della risposta, come nell'esempio.

Se la connessione viene stabilita regolarmente, il demone si incarica di creare una nuova interfaccia di comunicazione **ppp0** (in generale **pppN** a seconda di quante connessioni PPP si sono stabilite) sulla quale passerà il relativo traffico. Inoltre viene lanciato lo script **/etc/ppp/ip-up** che si cura eseguire una serie di azioni programmate relative alla presenza di una nuova connessione (come far scaricare la posta e le news). In generale questo script si limita ad eseguire tutti gli script posti in **/etc/ppp/ip-up.d**.

Allo stesso modo, quando la connessione viene chiusa viene lanciato lo script **/etc/ppp/ip-down** che esegue quelli presenti in **/etc/ppp/ip-down.d**, dopo di che l'interfaccia di rete viene disattivata.

7.7.3 I meccanismi di autenticazione

Al di là della possibile necessità di autenticarsi per l'accesso alla linea seriale (come mostrato nell'esempio riguardante **chat**) il demone **pppd** prevede una sua procedura di autenticazione diretta. Questa può avvenire secondo due protocolli, il *Password Authentication Protocol* (noto come PAP) ed il *Challenge Handshake Authentication Protocol* (noto come CHAP).

Il primo è più elementare e prevede che il client invii un username ed una password in chiaro sulla linea; si è così esposti ad una eventuale intercettazione telefonica. Il secondo procedimento invece prevede l'invio da parte del server di una *sfida* nella forma di un pacchetto che contiene il nome del server, così che il client può rispondere con un pacchetto che contenga un *hash* crittografico del pacchetto di sfida cui si è aggiunto il valore di un segreto condiviso (la password) così da comprovare la conoscenza del segreto senza inviarlo sulla linea.

Le informazioni di autenticazione usate dal demone vengono mantenute rispettivamente in due file: **/etc/ppp/pap-secrets** ed **/etc/ppp/chap-secrets** che hanno lo stesso formato. Un esempio di uno questi file è il seguente:

```
# INBOUND connections
# UserIDs that cannot use PPP at all. Check your /etc/passwd and add any
# other accounts that should not be able to use pppd!
guest   frijole "*"      -
master  frijole "*"      -
root    frijole "*"      -
support frijole "*"      -
stats   frijole "*"      -
# OUTBOUND connections
piccardi *          password
```

Il file prevede quattro campi separati da spazi; il primo campo indica il l'username usato dal client, il secondo il nome del server, il terzo il segreto condiviso, il quarto, opzionale, una lista degli IP da cui è possibile effettuare il collegamento (un "-" indica nessun IP), si possono indicare delle sottoreti in notazione CIDR ed usare un "!" iniziale per negare la selezione.

I nomi di client e server possono contenere caratteri qualunque, ma gli spazi e gli asterischi devono essere protetti scrivendoli fra virgolette; il valore * indica un nome qualsiasi e fa da wildcard. Se il campo della password password inizia con il carattere “@” si indica che quest’ultima deve essere letta dal file il cui nome è specificato dal resto della stringa.

Dato che il collegamento è punto-punto il procedimento di autenticazione è simmetrico: ogni demone può chiedere all’altro di autenticarsi; per questo i file contengono sia le informazioni per essere autenticati presso gli altri che per autenticarli. Di norma però, a meno di non gestire un provider, si deve solo essere autenticati, per questo il default è che `pppd` non esegue richieste di autenticazione, ma si limita a rispondere a quelle che gli vengono fatte.

Capitolo 8

La gestione dei servizi di base

8.1 I programmi di ausilio alla gestione dei servizi di base

Tratteremo in questa sezione alcuni programmi di ausilio per la gestione dei servizi di rete più semplici, ed in particolare l'uso di un cosiddetto “*super-demone*” per gestire servizi usati occasionalmente, e l'uso dei *TCP wrappers* per fornire un meccanismo elementare di controllo degli accessi anche a quei programmi che non ne hanno uno al loro interno.

8.1.1 I servizi elementari e i super-demoni

Spesso su una macchina in rete si trovano installati una serie di piccoli servizi elementari,¹ usati normalmente a scopo uso di test, come `echo` o `discard`, o una serie di servizi utilizzati in maniera occasionale.

Nelle prime versioni di Unix mantenere perennemente in esecuzione dei singoli demoni per fornire dei servizi utilizzati solo in maniera occasionale (come potevano essere `telnet` o `ftp`) veniva a costituire uno spreco di risorse non accettabile. Per questo motivo venne introdotto il concetto di *super-demone*, un demone speciale che facesse da intermediario,² mettendosi in ascolto sulle varie porte interessate, ed in grado di smistare le richieste ai demoni che implementano il servizio richiesto.

In sostanza un *super-demone* svolge quello che è il lavoro di un centralinista, che riceve le telefonate sul centralino e poi passa ciascuna di esse all'ufficio competente; nel caso specifico il programma riceve le connessioni su una certa porta e poi lancia automaticamente il relativo programma di risposta. In questo modo non c'è bisogno di mantenere sempre in esecuzione tanti programmi diversi, ma li si possono lanciare solo quando servono.

Inoltre (come avviene per il centralinista che ti può dire direttamente l'orario degli uffici) un *super-demone* è in genere in grado di rispondere direttamente per alcuni servizi elementari (come `echo` o `discard`), per i quali non necessita di invocare nessun programma esterno.

Infine una caratteristica comune dei *super-demoni* è che sono in grado di associare alla connessione ad una porta l'esecuzione di un programma qualunque. È possibile cioè, in occasione di una connessione ad una porta, lanciare un programma collegando il socket della connessione allo *standard input* e allo *standard output* del programma stesso. Così ad esempio si potrà attivare il servizio di rete `netstat` sulla porta 15, usando l'omonimo programma visto in sez. 7.5.3.

Di norma i servizi più importanti, come il web, la posta, il DNS, o SSH non vengono utilizzati in questo modo, perché questa *interposizione* di un altro programma avrebbe effetti negativi sulle prestazioni, anche se in molti casi l'importanza del servizio (ad esempio quella di un server web

¹oggi è sempre meno comune trovare questi servizi attivi, in quanto una buona norma di sicurezza prevede che è meglio non attivare un servizio fino a che questo non venga utilizzato.

²cioè con *super* inteso come qualcosa che sta sopra.

o di posta elettronica) va commisurata con l'uso che se ne fa: per una macchina client tenere sempre attivo Apache per fornire delle pagine locali consultate ogni tanto può effettivamente essere eccessivo.

Tradizionalmente il programma utilizzato per svolgere questo compito è `inetd`, che fino a qualche anno fa veniva installato di default da tutte le distribuzioni; `inetd` però presenta numerosi limiti, ed oggi tende ad essere sostituito con il più recente `xinetd`, se non ad essere eliminato del tutto. In generale infatti l'utilità di un *super-demone* sta venendo meno, considerato che il costo sempre minore delle risorse ne rende meno vantaggioso l'utilizzo.

8.1.2 Il super-demone `inetd`

Come appena accennato la gestione dei servizi di base veniva effettuata tramite il programma `inetd`, che ancor oggi viene usato da molte distribuzioni. Il programma viene lanciato dagli script di avvio, ma solito si può avviare e fermare il demone manualmente con il relativo script, che di norma è `/etc/init.d/inetd`.

In caso di necessità si può eseguire `inetd` anche direttamente a riga di comando, nel qual caso si può usare l'opzione `-d` per farlo partire in modalità di debug (in cui non si distacca dal terminale). Il comando prende come argomento un file di configurazione alternativo, altrimenti utilizza il default che è `/etc/inetd.conf`. Per le altre opzioni ed i dettagli del funzionamento si faccia riferimento come al solito alla pagina di manuale.

Il file di configurazione di default di `inetd` è `/etc/inetd.conf`, ed è qui che si specifica su quali porte il demone deve porsi in attesa e quali programmi lanciare. Un esempio di questo file è il seguente:

```
# /etc/inetd.conf:  see inetd(8) for further informations.
#
# Internet server configuration database
#
# <service_name> <sock_type> <proto> <flags> <user> <server_path> <args>
#
#:INTERNAL: Internal services
#echo          stream  tcp    nowait  root    internal
#echo          dgram   udp    wait    root    internal
#chargen       stream  tcp    nowait  root    internal
#chargen       dgram   udp    wait    root    internal
#discard       stream  tcp    nowait  root    internal
#discard       dgram   udp    wait    root    internal
#daytime       stream  tcp    nowait  root    internal
#daytime       dgram   udp    wait    root    internal
#time          stream  tcp    nowait  root    internal
#time          dgram   udp    wait    root    internal
#:MAIL: Mail, news and uucp services.
nntp           stream  tcp    nowait  news    /usr/sbin/tcpd  /usr/sbin/leafnode
#:INFO: Info services
ident          stream  tcp    wait    identd  /usr/sbin/identd  identd
```

come si vede il formato è relativamente semplice: una tabella in cui ogni riga è relativa ad un servizio da lanciare, ed i cui campi sono separati da spazi o tabulatori. Al solito righe vuote e inizianti con `#` sono ignorate.

Il servizio da fornire, o meglio la porta su cui porsi in ascolto, è identificato dal primo campo tramite il nome simbolico di `/etc/services`. Il successivo campo indica il tipo di socket: i due tipi più comuni sono `stream` o `dgram`, ma si possono usare tutti i tipi di socket supportati con Linux, (come `raw`, `rdm`, o `seqpacket`). Il terzo campo indica il protocollo usato: di norma si tratta di `tcp` o `udp`,³ che corrispondono rispettivamente ai tipi `stream` e `dgram`.

³in realtà si può specificare un qualunque altro protocollo col nome riportato in `/etc/protocols`, ma benché

Il quarto campo è applicabile solo quando si è usato un socket **dgram**, e indica se il server è in grado di trattare altri pacchetti⁴ mentre sta rispondendo o no. Qualora non lo sia si dovrà aspettare la fine del processo corrente prima di rilanciare un'altra istanza del programma, e questo si fa utilizzando il valore **wait**. Se questo non è necessario (e per gli tutti gli altri socket) deve essere usato il valore **nowait**. Il quinto campo indica l'utente per conto del quale viene eseguito il programma; se si vuole specificare anche un gruppo diverso dal gruppo principale dell'utente, lo si può fare scrivendo il campo nella forma **user.group**.

Il sesto campo deve indicare il pathname completo del comando che verrà eseguito in caso di connessione, o la parola chiave **internal** se quel servizio è fornito direttamente da **inetd**.⁵ Eventuali argomenti dovranno essere specificati di seguito; dato però che **inetd** usa direttamente il resto della linea per invocare la funzione **exec**, questi non potranno essere specificati normalmente riprendendo quanto si scriverebbe sulla shell, ma dovranno essere preceduti dall'argomento iniziale⁶ che indica il nome del programma lanciato.

Come si può notare nell'esempio sono abilitati solo due servizi, il primo è un server news gestito tramite il programma **leafnode**, il secondo è il servizio **ident**, definito dall'RFC 1413, che permette di identificare l'utente proprietario del processo che ha creato una certa connessione. Si noti anche come nel caso di **leafnode** questo non sia stato lanciato direttamente ma invocato attraverso **tcpd**, un utile programma di sicurezza che permette di controllare gli accessi ai servizi di rete, come vedremo in sez. 8.1.4. Se vogliamo provare ad abilitare un nuovo servizio **netstat** possiamo aggiungere al file una riga del tipo:

```
netstat      stream  tcp      nowait  nobody      /bin/netstat netstat -ant
```

e si noti come si sia ripetuto, prima dei parametri **-ant**, il nome del programma **netstat**; senza questa ripetizione il risultato sarebbe stato che all'esecuzione di **/bin/netstat** il nome del processo sarebbe risultato **-ant**,⁷ mentre le opzioni sarebbero state perse.

A questo punto, dopo aver riavviato **inetd** per fargli prendere le modifiche effettuate alla configurazione, si può verificare il funzionamento del nostro nuovo servizio usando **telnet**; avremo allora che:

```
[piccardi@gont corso]$ telnet localhost netstat
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:15              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 192.168.1.1:53          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:631             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN
tcp        0      0 192.168.1.1:32777       195.110.124.18:993      ESTABLISHED
tcp        0      0 127.0.0.1:32944         127.0.0.1:15            ESTABLISHED
tcp        0      0 192.168.1.1:32778       62.177.1.107:5223      ESTABLISHED
tcp        0      0 192.168.1.1:32772       192.168.1.168:5901      ESTABLISHED
```

sia possibile usare il demone con qualunque tipo di socket e protocollo di rete, in pratica sono questi due gli unici che vengono utilizzati.

⁴per un socket di tipo **dgram** non esiste il concetto di connessione, ed in genere i servizi che usano questo tipo di socket rispondono a ciascun pacchetto che arriva.

⁵come accennato **inetd** è in grado di fornire direttamente alcuni servizi elementari come **echo**, **discard** o **daytime**, per i quali non è necessario lanciare nessun programma a parte.

⁶la shell costruisce automaticamente questo argomento (che negli script è **\$0**), per cui non viene mai specificato.

⁷come potreste osservare anche direttamente eseguendo **ps**, posto che siate così veloci (e fortunati) da riuscire a lanciarlo mentre **netstat** sta girando.

```

tcp      0      0 127.0.0.1:15      127.0.0.1:32944    ESTABLISHED
Connection closed by foreign host.

```

e come si può notare si ottiene il risultato del comando **netstat**, eseguito come se fossimo su una macchina remota, in cui compare, nell'ultima riga, anche la connessione su cui lo stiamo leggendo.

Il precedente esempio funziona perché quando lancia un programma **inetd** fa sì che *standard input*, *standard output* e *standard error* siano associati al socket su cui è stata aperta la connessione. Si può così eseguire un programma qualsiasi, e questo accetterà l'input dal socket e su di esso scriverà il suo output, ed in generale sarà compito del programma messo in esecuzione gestire opportunamente la connessione di rete, fornendo il servizio richiesto.

Nell'esempio appena mostrato quello che è successo è che il programma **netstat** è stato eseguito sul server remoto,⁸ e ha prodotto la lista dei socket attivi scrivendola sullo standard output, per poi terminare. Questo ha fatto sì che noi la ricevessimo all'altro capo del socket creato dalla connessione effettuata con **telnet**, che è stato automaticamente chiuso alla terminazione del processo. La stessa procedura è applicabile in generale a qualunque comando di shell, per cui potremmo definire un servizio di rete **ps** (assegnandogli una porta in **/etc/services**) che ci fornisca l'elenco dei processi, ed in generale potremmo anche crearci un servizio ad hoc usando degli script.

8.1.3 Il super-demone xinetd

Il programma **xinetd** nasce come riscrittura di **inetd** per eseguire lo stesso compito: far partire i server appropriati in caso di connessione al relativo servizio, evitando di lanciare e tenere in memoria dei programmi che resterebbero dormienti per la gran parte del tempo.

Rispetto ad **inetd** esso supporta nativamente il controllo di accesso con i *TCP wrappers* (vedi sez. 8.1.4), ma oltre a questo ha una serie di funzionalità ulteriori come la possibilità di mettere a disposizione i servizi in orari determinati, dei meccanismi di redirectione delle connessioni, delle capacità di registrazione degli eventi più estese, dei meccanismi di protezione nei confronti dei portscan, e la capacità di limitare il numero di istanze del server lanciate, per resistere agli attacchi di *denial of service*.

Le maggiori funzionalità comportano ovviamente il prezzo di una maggiore complessità di configurazione, comunque il comando supporta una opzione **-inetd_compat** che gli permette di operare in modalità di compatibilità con **inetd**. In tal caso infatti il programma prima legge i suoi file di configurazione, e poi **/etc/inetd.conf** facendo partire i servizi definiti in quest'ultimo (questa è la configurazione standard di Debian).

Il file principale di configurazione di **xinetd** è **/etc/xinetd.conf**; un esempio del suo contenuto, che illustra le principali direttive generiche, è il seguente:

```

# Simple configuration file for xinetd
defaults
{
    # The maximum number of requests a particular service may handle
    # at once.
    instances    = 25

    # The type of logging.  This logs to a file that is specified.
    # Another option is: FILE /var/log/servicelog
    log_type     = SYSLOG auth

    # What to log when the connection succeeds.
    # PID logs the pid of the server processing the request.
    # HOST logs the remote host's ip address.

```

⁸il fatto che si sia usato **localhost** è stato solo per comodità.


```

# USERID logs the remote user (using RFC 1413)
# EXIT logs the exit status of the server.
# DURATION logs the duration of the session.
log_on_success = HOST PID

# What to log when the connection fails. Same options as above
log_on_failure = HOST RECORD

# The maximum number of connections a specific IP address can
# have to a specific service.
per_source = 5
}
includedir /etc/xinetd.d

```

in questo caso ci si è limitati a dichiarare la sezione speciale **defaults**, che contiene i valori di default delle opzioni, da applicare per tutti i servizi per i quali essi non sono stati esplicitamente specificati.

La riga finale con la riga **includedir** ci dice poi di includere automaticamente nella configurazione il contenuto di tutti i file contenuti nella directory specificata (di norma, come nel caso, si usa **/etc/xinetd.d**), il che permette di poter attivare in maniera indipendente ciascun servizio con la semplice installazione di un file in tale directory.

Al solito le righe vuote e il cui primo carattere non di spaziatura è **#** vengono ignorate; per il resto per ciascun servizio che si vuole attivare è necessario specificare una voce (nel caso basterà scrivere un file che la contenga in **/etc/xinetd.d**) che inizia con la direttiva **service**; la sintassi generica di tale voce è del tipo:

```

service <nome_servizio>
{
    <attributo> <operatore> <valore> <valore> ...
    ...
}

```

dove **<nome_servizio>** indica il servizio che si vuole fornire, e gli attributi permettono di specificarne le caratteristiche, con un operatore di assegnazione che nella gran parte dei casi è **=**, con significato ovvio, ma che può essere anche **+=** o **-=** rispettivamente per aggiungere o togliere valori (solo alcuni attributi lo supportano).

Come mostrato nell'esempio precedente, anche la direttiva **default** prende degli attributi; in genere questi sono attributi generali che possono a loro volta essere rispecificati in maniera diversa per i singoli servizi. Nel nostro caso il primo attributo è **instances** che permette di porre un limite massimo al numero di istanze di uno stesso server che il programma può lanciare.

Il secondo attributo è **log_type**, che permette di specificare le modalità con cui viene effettuato la registrazione dei dati delle connessioni; queste possono essere **FILE** per specificare a seguire un file su cui salvare direttamente i dati, o **SYSLOG** per specificare l'uso del *syslog*, indicando poi con l'ulteriore parametro **auth** quale *facility* utilizzare (è possibile anche specificare di seguito una *priority*, se diversa dal default **info**).

I due attributi successivi, **log_on_success** e **log_on_failure**, permettono di specificare cosa scrivere nei log in caso rispettivamente di successo e fallimento di una connessione. Le indicazioni **HOST** e **USERID** sono comuni ad entrambi e permettono di registrare rispettivamente l'IP e l'utente (se è disponibile il servizio **identd** secondo l'RFC 1413) relativi alla connessione da remoto; **log_on_success** permette anche di registrare lo stato di uscita del server (con **EXIT**), la durata della connessione (con **DURATION**) ed il numero identificativo del processo (con **PID**).

Infine l'ultimo attributo, **per_source**, permette di stabilire un numero massimo per le connessioni da un singolo IP (una forma per limitare eventuali *denial of service*). Un elenco dei

principali attributi che possono essere specificati nella sezione **defaults**, con la relativa descrizione, è riportato in tab. 8.1, l'elenco completo può essere trovato nelle pagine di manuale, accessibili con `man xinetd.conf`.

Attributo	Descrizione
instances	numero massimo di processi lanciati per ogni servizio; prende un valore intero.
log_type	metodologia di registrazione dei log, su file o tramite <i>syslog</i> ; prende i valori FILE (seguito dal nome del file su vengono aggiunti i messaggi) o SYSLOG seguito dalla <i>facility</i> da usare ed opzionalmente dalla priorità.
log_on_success	cosa registrare per le connessioni riuscite; prende uno o più fra PID , HOST , USERID , EXIT , DURATION .
log_on_failure	cosa registrare per le connessioni fallite, prende uno o più fra HOST , USERID , ATTEMPT .
per_source	numero massimo di connessioni per singolo IP sorgente; prende un valore intero.
only_from	lista delle macchine da cui è possibile accedere; prende indirizzi IP in forma dotted decimal, CIDR o simbolica.
no_access	lista delle macchine da cui è impossibile accedere; prende indirizzi IP in forma dotted decimal, CIDR o simbolica.
access_times	orario giornaliero nel quale è possibile accedere ai servizi, prende un intervallo temporale specificato nella forma HH:MM-HH:MM .
cps	rate massimo di accesso; prende un valore decimale per indicare il limite sulle connessioni al secondo, ed un valore intero per specificare il numero di secondi da aspettare prima di accettare nuove connessioni una volta superato il limite.
nice	valore di <i>nice</i> (vedi sez. 1.3.3) da applicare ai demoni lanciati; prende un valore intero.
max_load	carico massimo oltre il quale la macchina smette di accettare connessioni; prende un valore decimale.

Tabella 8.1: Attributi specificabili in generale per tutti i servizi gestiti attraverso il *super-demone* **xinetd**.

Come accennato il programma è in grado sia di implementare direttamente dei controlli di accesso, che di utilizzare quelli eventualmente specificati tramite i file di controllo dei *TCP wrappers*⁹ (vedi sez. 8.1.4). I due attributi **only_from** e **no_access** sono in grado di specificare direttamente le stesse condizioni all'interno del file di configurazione di **xinetd**. In questo caso le sintassi supportate sono sia le forme dotted decimal (interpretando gli zeri finali come indirizzi di rete), che quelle CIDR, sia quelle espresse con indirizzi simbolici.

Come già accennato ciascun servizio che si vuol lanciare con **xinetd** deve essere specificato con la direttiva **service** seguita dal nome dello stesso; un esempio possibile è il seguente:

```
service time
{
    disable = yes
    type      = INTERNAL
    id        = time-stream
    socket_type = stream
    protocol  = tcp
    user      = root
    wait      = no
}
service time
{
```

⁹si tenga presente che l'uso dei *TCP wrappers* ha di norma la precedenza sul controllo interno, per cui se si abilita l'accesso con queste direttive, ma esso è negato dai TCP wrappers, questi ultimi avranno la meglio.

```

        disable      = yes
        type         = INTERNAL
        id           = time-dgram
        socket_type   = dgram
        protocol     = udp
        user         = root
        wait         = yes
    }
    service nntp
    {
        socket_type = stream
        wait        = no
        user        = news
        server       = /usr/sbin/leafnode
        server_args = -v
        only_from   = 192.168.0.0/24
        access_times = 08:00-17:00
    }
}

```

In questo caso si sono definiti due servizi gestiti direttamente da `xinetd`, e cioè il servizio *time* su TCP e UDP, entrambi sono disabilitati, avendo `disable` impostato su `yes`. Si noti come siano stati differenziati attraverso la presenza di un argomento `id`. Inoltre con `socket_type` si è specificato il tipo di socket da usare, e con `protocol` il relativo protocollo. Il campo `type` dice che il servizio è fornito internamente, mentre `user` indica che verrà eseguito per conto dell'utente `root`. Infine il campo `wait` indica se il servizio può essere fornito in maniera concorrente (con molte connessioni contemporanee) senza attendere la conclusione di una connessione, o no.

Oltre ai due servizi interni si è abilitato anche il servizio di `news`; in questo caso restano specificati con lo stesso significato precedente gli argomenti `wait`, `socket_type` e `user` (anche se per quest'ultimo si è usato l'utente `news`), mentre non esistendo il servizio `news` su UDP non è stato necessario importare `protocol`. Si è invece specificato il programma da usare come server con `server`, passando i relativi argomenti con `server_args`. Inoltre nel caso si è ristretto l'accesso al servizio alle macchine della sottorete `192.168.0.0/24` con `only_from` e negli orari di ufficio usando `access_times`.

Oltre a quelli appena illustrati, si sono riportati i principali attributi utilizzabili in tab. 8.2. Si ricordi che anche i precedenti attributi di tab. 8.1 possono essere utilizzati, e saranno applicati solo al servizio in questione. Al solito nella pagina di manuale di `xinetd.conf` è riportato un elenco completo di tutti gli argomenti presenti e la descrizione dettagliata di ciascuno di essi.

Anche con `xinetd` è possibile creare un proprio servizio usando i comandi di shell; ripeteremo allora quanto visto con `inetd` definendo un nuovo servizio `netstat`. Dato che il servizio è previsto in `/etc/services` solo per TCP possiamo attivarlo creando in `/etc/xinetd.d` un nuovo file con un contenuto del tipo di:

```

service netstat
{
    socket_type      = stream
    wait            = no
    user            = root
    server          = /bin/netstat
    server_args     = -ant
}

```

e, una volta riavviato `xinetd`, potremo verificarne come prima il funzionamento, con un `telnet` sulla porta 15.

Attributo	Descrizione
<code>socket_type</code>	tipo di socket; prende gli stessi valori (<code>stream</code> , <code>dgram</code> , ecc.) dell'analogo parametro in <code>inetd.conf</code> .
<code>user</code>	utente per conto del quale è lanciato il servizio; deve essere presente in <code>/etc/passwd</code> , si può specificare un eventuale gruppo con l'attributo <code>group</code> .
<code>server</code>	pathname del programma server da lanciare; va sempre specificato se il servizio non è gestito internamente.
<code>wait</code>	indica se si deve attendere o meno la conclusione del server per lanciare un'altra istanza; analogo dello stesso parametro in <code>inetd.conf</code> ; prende i valori <code>yes</code> e <code>no</code> .
<code>protocol</code>	protocollo usato (analogo di <code>tcp</code> ed <code>udp</code> per <code>inetd.conf</code>); deve essere un nome valido in <code>/etc/protocols</code> .
<code>port</code>	porta su cui ascoltare le connessioni; deve essere specificata se si è specificato un servizio non riportato in <code>/etc/services</code> .
<code>type</code>	tipo di servizio; può assumere i valori <code>INTERNAL</code> (per servizi gestiti internamente), <code>RPC</code> per indicare che si userà un servizio RPC (vedi sez. 7.5.5), <code>UNLISTED</code> per servizi non presenti in <code>/etc/services</code> o <code>/etc/rpc</code> .
<code>server_args</code>	eventuali argomenti da passare al programma server quando viene lanciato; non necessita di specificare l'argomento iniziale come per <code>inetd</code> .
<code>disable</code>	indica se attivare il servizio, il default è disattivo; può assumere i valori <code>yes</code> e <code>no</code> .
<code>id</code>	identificatore aggiuntivo qualora si tratti di servizi diversi con lo stesso nome.
<code>bind</code>	consente di specificare l'interfaccia su cui fornire il servizio; prende l'indirizzo IP ad essa associato.
<code>redirect</code>	consente di redirigere il servizio ad un'altra macchina; prende l'indirizzo IP e la porta verso quale redirigere tutto il traffico.
<code>umask</code>	imposta la <code>umask</code> ereditata dal processo mandato in esecuzione.

Tabella 8.2: Attributi specificabili per un servizio gestito attraverso il super-demone `xinetd`.

8.1.4 I TCP wrappers

Per molto tempo gran parte dei demoni di rete, nati in un periodo in cui internet era una rete costituita principalmente da istituzioni ed enti di ricerca, ed in cui il numero delle persone che potevano utilizzarla era molto limitato e facilmente controllabile, non sono stati forniti di nessun tipo di controllo degli accessi, permettendo a chiunque di collegarsi ad essi, qualunque fosse la sua macchina o il suo indirizzo IP.

Con l'espandersi della rete e la possibilità di accessi non voluti o *maliziosi*, è diventato sempre più importante poter effettuare un controllo degli accessi. Per questo Wietse Wenema, un esperto di sicurezza, ha creato un insieme di librerie e programmi, chiamati *TCP wrappers*, che consentono di realizzare un controllo degli accessi permettendo il collegamento ai servizi da essi protetti solo da parte di certe stazioni o reti. Riprendendo l'analogia telefonica si tratta di una specie di filtro che impedisce che ci possano chiamare da certi numeri.

Le modalità con cui si possono utilizzare le capacità di filtraggio dei *TCP wrappers* sono sostanzialmente due. La prima è quella più semplice che vede l'uso del programma `tcpd`, che fa da "involucro" (in inglese *wrapper*, da cui il nome) all'esecuzione di altri programmi. Esso viene di norma utilizzato attraverso il demone `inetd` come illustrato in sez. 8.1.2 per lanciare altri programmi, effettua un controllo e se l'accesso è consentito lancia il demone specificato come argomento.

La seconda modalità è quella che prevede l'uso, direttamente all'interno del demone che

fornisce il servizio, delle librerie di controllo dei *TCP wrappers*, in modo da poterne usare le funzionalità senza dover ricorrere ad un programma esterno. Sono esempi di questa modalità servizi come SSH, LDAP o NFS.

In entrambi i casi le connessioni ai servizi controllati con i *TCP wrappers* vengono registrate sul sistema del *syslog*, in modo da lasciare traccia di eventuali tentativi di accesso non autorizzati, dopo di che vengono eseguiti i vari controlli che sono stati richiesti.

Il controllo di accesso implementato dai *TCP wrappers* è gestito attraverso due file, `hosts.allow` e `hosts.deny`, che contengono le regole di accesso, secondo una sintassi specifica che è riportata per esteso nella pagina di manuale accessibile con `man 5 hosts_access`. Il primo file, come suggerisce il nome, elenca le regole che negano l'accesso, il secondo quelle che lo consentono.

Si tenga presente che il funzionamento dei *TCP wrappers* è tale che prima viene controllato `hosts.allow`, e se una regola corrisponde l'accesso è garantito e la procedura di controllo finisce immediatamente; altrimenti viene controllato `hosts.deny` e se una regola corrisponde l'accesso è negato. Se entrambi i file sono vuoti quindi, l'accesso è consentito.

In genere allora quello che si fa è di negare tutti gli accessi in `hosts.deny` e consentire poi solo quelli voluti in `hosts.allow`. Per questo l'esempio tipico di `hosts.deny` è il seguente:

```
# /etc/hosts.deny: list of hosts that are _not_ allowed to access the system.
#                               See the manual pages hosts_access(5), hosts_options(5)
#                               and /usr/doc/netbase/portmapper.txt.gz
#
ALL: ALL
```

la sintassi delle regole si intravede già in questo esempio; al solito righe vuote e tutto quello che segue un `#` viene ignorato, ogni riga poi ha la forma:

```
lista dei server: lista dei client : comando shell
```

con tre campi separati dal carattere “:”, dei quali il terzo è opzionale e può essere omesso (in effetti è scarsamente utilizzato).

Un secondo esempio, più significativo del precedente, è quello del contenuto di `hosts.allow`, riportato di seguito:

```
# /etc/hosts.allow: list of hosts that are allowed to access the system.
#                               See the manual pages hosts_access(5), hosts_options(5)
#                               and /usr/doc/netbase/portmapper.txt.gz
#
sshd:      ALL
leafnode:  127.0.0.1
portmap:   127.0.0.1 192.168.1.
mountd:    127.0.0.1 192.168.1.
statd:     127.0.0.1 192.168.1.
lockd:     127.0.0.1 192.168.1.
rquotad:   127.0.0.1 192.168.1.
```

Nell'esempio indicato si sono indicati tre servizi, il primo è la *secure shell* (vedi sez. 8.3), che permette un collegamento sicuro da remoto, che è gestita come server dal programma `sshd`; il secondo è il servizio che permette di tenere un server di news in locale (a scopo di caching), gli altri sono i vari server necessari al funzionamento di NFS (si veda sez. 8.4.1). L'esempio consente l'accesso ad `ssh`, l'accesso a NFS per la rete locale, e l'uso del server di news solo tramite il localhost.

In generale per lista dei server si intende il nome (o i nomi, se se ne vuole indicare più di uno) del programma che gestisce lo specifico servizio. Occorre fare attenzione, perché il nome è quello del programma che fornisce il servizio, non quello del servizio indicato in `inetd.conf`.

Nella lista dei client si indicano invece gli indirizzi IP o le reti a cui si vuole consentire l'accesso. Per le reti in genere si usa la notazione sia numerica che alfabetica, e si può usare il

carattere “*” come wildcard per raggruppare indirizzi; si può anche specificare una maschera di rete con un indirizzo del tipo:

```
131.155.72.0/255.255.254.0
```

e specificare la lista degli indirizzi usando un file dando il pathname assoluto dello stesso.

Il formato e la lista completa delle funzionalità che sono controllabili tramite questi file è riportato nella pagina di manuale ad essi associate, accessibili con `man 5 hosts_access`. Si tenga comunque conto che alcuni servizi (NFS per esempio) supportano solo un sottoinsieme delle funzionalità definite in generale.

Per una migliore gestione dei TCP wrapper il relativo pacchetto fornisce anche dei programmi di utilità che permettono di verificare la configurazione effettuata ed effettuare dei controlli di accesso.

Il primo programma è `tcpdchk` che esegue un controllo delle regole di accesso impostate con `hosts.allow` e `hosts.deny`, confrontandole anche con i servizi attivati in `inetd.conf`. Il comando riporta tutti gli eventuali problemi rilevati, a partire da un uso scorretto di wildcard e indirizzi, servizi che non sono riconosciuti, argomenti o opzioni non validi, ecc. Così ad esempio potremo avere:

```
[root@gont corso]# tcpdchk
warning: /etc/hosts.allow, line 15: apt-proxy: no such process name in /etc/inetd.conf
```

in corrispondenza ad una regola di accesso rimasta aperta per un servizio che in seguito è stato rimosso.

Il comando permette di controllare i file `hosts.allow` e `hosts.deny` nella directory corrente invece che sotto `/etc` usando l'opzione `-d`, mentre si può specificare un diverso file per `inetd.conf` con l'opzione `-i`, la documentazione completa è al solito disponibile con `man tcpdchk`.

Il secondo comando di controllo è `tcpdmatch` che permette di verificare il comportamento dei TCP wrapper per una specifica richiesta da un servizio. Il comando richiede due parametri, il primo che specifichi il servizio che si vuole controllare ed il secondo la stazione da cui si intende effettuare l'accesso. Il comando eseguirà una scansione delle regole e riporterà i risultati. Ad esempio potremo richiedere:

```
[root@gont corso]# tcpdmatch sshd oppish
warning: sshd: no such process name in /etc/inetd.conf
warning: oppish: hostname alias
warning: (official name: oppish.earthsea.ea)
client:  hostname oppish.earthsea.ea
client:  address  192.168.1.168
server:   process  sshd
matched:  /etc/hosts.allow line 14
access:   granted
```

che controlla l'accesso al servizio SSH (si noti che si deve specificare il nome del programma che esegue il servizio) da parte della macchina `oppish`, trovando che questo è consentito dalla riga 14 del file `hosts.allow`. Il comando rileva anche che il servizio non è lanciato attraverso `inetd` e quale è l'IP effettivo della macchina.

8.2 L'assegnazione dinamica degli indirizzi IP

In questa sezione tratteremo vari protocolli (ed i relativi programmi di utilizzo) che vengono usati per l'assegnazione automatica dei numeri IP all'interno di una rete locale. Il primo, il

RARP è stato il primo protocollo usato a questo scopo, ed è sostanzialmente in disuso essendo sostituito dagli altri due. Il secondo, il BOOTP, nasce per distribuire via rete immagini di avvio, mentre l'ultimo, il DHCP, ricomprende tutte queste esigenze all'interno di un unico servizio, e lo tratteremo con maggiori dettagli.

8.2.1 I protocolli RARP, BOOTP e DHCP

Il protocollo RARP (sigla che sta per *Reverse Address Resolution Protocol*), è un protocollo elementare, definito nell'RFC 903 che, come il nome stesso indica, esegue il compito inverso rispetto al protocollo ARP già visto in sez. 7.5.4. Il protocollo è implementato direttamente a livello di collegamento fisico (cioè nella maggior parte dei casi su ethernet) e serve ad ottenere, dato un *MAC address* sulla rete, l'indirizzo IP ad esso associato.

Lo scopo del protocollo era principalmente quello di fornire un meccanismo automatico per l'assegnazione di un numero IP ad una macchina in fase di avvio; questa esegue (posto che il kernel disponga del relativo supporto, che su Linux deve essere opportunamente abilitato) una richiesta RARP su ethernet ed utilizza il numero IP fornito come risposta per configurare l'interfaccia di rete da cui ha eseguito la richiesta; la cosa presuppone ovviamente la presenza nel suddetto tratto di rete di un server che fornisca la risposta.

Tutto ciò oggi, in presenza di altri protocolli più sofisticati in grado di eseguire sia questo che altri compiti, ha solo interesse storico (considerato anche che sono assai pochi ormai i sistemi operativi che lo supportano). Per chi si trovasse in situazioni in cui per compatibilità con vecchi sistemi viene ancora usato questo metodo per l'assegnazione degli indirizzi, ci limitiamo a citare la possibilità compilare il kernel per utilizzare questo supporto.

Il protocollo BOOTP (il nome sta per *Bootstrap Protocol*), è definito nell'RFC 951 e nasce come protocollo per gestire l'avvio automatico delle macchine. Rispetto al precedente RARP è basato su IP e UDP (e non direttamente sul collegamento fisico) ed non si limita alla ricerca di un numero IP da assegnare automaticamente ad una scheda di rete, ma prevede anche l'invio delle informazioni necessarie ad ottenere un file da usare come sistema operativo per l'avvio e l'indirizzo di una macchina a cui richiedere quest'ultimo.

Il protocollo BOOTP nasce principalmente per fornire un meccanismo con cui macchine senza disco possano eseguire un avvio del sistema operativo via rete, ottenendo quest'ultimo¹⁰ da un opportuno server; in genere il trasferimento dei dati avviene tramite il protocollo TFTP,¹¹ ma in teoria possono essere utilizzati anche altri protocolli. In questo caso è compito del BIOS della macchina implementare la parte client del protocollo,¹² in modo da ottenere poi l'immagine del sistema che verrà caricata in memoria ed eseguita.

Come sottoinsieme delle funzionalità del protocollo BOOTP c'è ovviamente anche quella della assegnazione automatica di un numero IP; il trasferimento dell'immagine del sistema avviene sempre con un altro protocollo che necessita di IP; per questo motivo il protocollo può essere usato per distribuire gli indirizzi su una rete. Di nuovo qualora questa funzionalità fosse necessaria per l'avvio, ad esempio per l'avvio di un sistema con la directory radice su NFS, occorrerà compilare il kernel con il relativo supporto, analogamente a quanto si farebbe per l'uso di RARP.

Il supporto per la parte server del protocollo è disponibile con il programma `bootpd` (su Debian il pacchetto è `bootp`) che implementa tutte le funzionalità definite nell'RFC 951 e pure le estensioni dei successivi RFC 1532 e RFC 1533. In genere il servizio viene avviato tramite `inetd`, ma può essere eseguito anche in modalità *standalone*.

Il comando prevede come argomento il file da cui leggere le impostazioni, che può essere

¹⁰ cioè i vari file contenenti i dati necessari, nel caso di Linux l'immagine del kernel ed un eventuale ramdisk.

¹¹ il *Trivial File Transfer Protocol* è un protocollo elementare per il trasferimento di file basato su UDP, che è utilizzabile in programmi estremamente semplificati come quelli che sono contenuti nel BIOS di una macchina.

¹² sia di BOOTP che di TFTP, se poi viene usato quest'ultimo.

omesso, nel qual caso verrà utilizzato il default che è `/etc/bootptab`; quest'ultimo contiene tutte le definizioni dei vari parametri da usare per ciascun client nella forma:

```
hostname:tg=value... :tg=value... :tg=value. ...
```

dove **hostname** indica il nome associato ad un client cui seguono le assegnazioni dei parametri che sono identificati da una etichetta di due caratteri, separata con l'uso del carattere ":".

Tag	Descrizione
bf	file di avvio.
dn	nome del dominio.
ds	lista dei server DNS.
gw	lista dei gateway.
ha	indirizzo fisico (<i>MAC address</i>) della macchina.
hd	home directory per il file di avvio.
hn	hostname da inviare al client.
ip	indirizzo IP da assegnare.
rp	pathname della directory da montare come radice.
sa	indirizzo del server TFTP da usare.
sm	maschera di rete della macchina.
td	directory radice per il server TFTP.

Tabella 8.3: Etichette dei parametri assegnabili in `bootptab`.

Un elenco delle etichette dei principali parametri usati da `bootptab` è riportato in tab. 8.3, al solito l'elenco completo e tutti i dettagli sono nella relativa pagina di manuale, accessibile con `man bootptab`.

Il protocollo più utilizzato per la configurazione automatica della rete è il DHCP, sigla che sta per *Dynamic Host Configuration Protocol*, che al giorno d'oggi ha soppiantato in maniera praticamente completa sia RARP che BOOTP. Oltre ai servizi previsti dal precedente BOOTP, del quale implementa, in maniera compatibile all'indietro, tutte le funzionalità, il protocollo supporta l'assegnazione dinamica degli indirizzi. La descrizione completa è disponibile nell'RFC 2131 che ne definisce tutte le caratteristiche.

A differenza di BOOTP lo scopo principale di DHCP, come espresso dal nome stesso, è quello della configurazione dinamica delle macchine in una rete locale, uno dei vantaggi di questo protocollo è che è in grado di fornire parecchie informazioni in più rispetto a quelle elementari fornite da BOOTP, rendendo possibile la configurazione automatica di tutte le funzionalità della rete per le macchine di una LAN.

La peculiarità di DHCP è comunque il supporto per la gestione dinamica degli indirizzi. Con BOOTP infatti l'assegnazione di un indirizzo IP è fatta staticamente sul server, per ogni client deve essere configurato un indirizzo; il client si limita a fare una richiesta e configurare la rete in base alla risposta (o fallisce se il server non risponde). Non viene gestito il caso di una assegnazione temporanea all'interno di un pool di indirizzi, cioè il caso tipico di quando al computer di un ospite deve essere assegnato un indirizzo temporaneo all'interno della propria rete.

Una assegnazione di questo tipo comporta ovviamente una serie di problemi in più rispetto all'assegnazione statica, ed in particolare l'assegnazione degli indirizzi si scontra con le problematiche relative alla presenza discontinua od occasionale di macchine all'interno della rete, alla necessità di dover gestire la riassegnazione di indirizzi usati in precedenza, di accorgersi quando un indirizzo non è più in uso, e di evitare di assegnare due volte lo stesso indirizzo a macchine diverse.

Per risolvere questo tipo di problematiche DHCP prevede una comunicazione non occasionale fra il client, che resta attivo anche dopo la configurazione dell'indirizzo, ed il server e tre diverse modalità di assegnazione degli indirizzi, dette rispettivamente *statica*, *automatica* e *dinamica*.

La configurazione *statica* è sostanzialmente la stessa di BOOTP, in cui si associa staticamente un indirizzo ad una specifica macchina sulla base del *MAC address* della stessa. Con la configurazione *automatica* invece viene assegnato in maniera permanente un indirizzo IP ad ogni macchina che ne fa richiesta, ma non è necessario che l'amministratore imposti la corrispondenza come nel caso precedente. Questo può andare bene nel caso di macchine sempre presenti sulla rete, ma non è ovviamente adatto al caso di presenze occasionali.

Per questo la modalità che alla fine viene utilizzata di più è quella della configurazione *dinamica*, in cui ad ogni assegnazione è associato anche un tempo di permanenza, il cosiddetto *lease*, passato il quale, se il client non ha chiesto un prolungamento, l'assegnazione viene cancellata. In questo modo diventa possibile gestire la presenza occasionale di macchine sulla rete, assegnando un indirizzo libero che tornerà automaticamente disponibile dopo un certo periodo di inutilizzo, o che potrà essere rilasciato esplicitamente quando segnalato dal client stesso.

Il protocollo DHCP prevede che sia sempre il client a contattare il server; all'avvio il client invia dei pacchetti in *broadcast* su UDP,¹³ e alla ricezione di una richiesta il server invia un pacchetto di risposta contenente le varie informazioni di configurazione,¹⁴ che prevedono l'indirizzo IP da assumere, quello del *default gateway*, l'indirizzo di eventuali server DNS ed altre informazioni relative alla rete, ma soprattutto, qualora si usi la configurazione dinamica, il periodo di *lease* per il quale l'assegnazione dell'indirizzo sarà considerata valida.

8.2.2 Uso del servizio DHCP dal lato client

Per poter usufruire dei servizi di un server DHCP esistono vari programmi; i più comuni e diffusi sono **pump** e **dhclient**, di quest'ultimo esistono poi due versioni, a seconda che si usi la versione 2 o la versione 3 del protocollo.

Il programma **pump** è un client sia per DHCP che per BOOTP; il suo uso è immediato, basta invocarlo specificando l'interfaccia che si vuole configurare in maniera automatica con l'opzione **-i**, con qualcosa del tipo:

```
pump -i eth0
```

anche se esso può essere fatto partire automaticamente all'avvio del sistema, quando si configura la rete per eseguire la configurazione tramite DHCP. Il comando prende una serie di altre opzioni che consentono di scartare le informazioni ottenute dal server (come i DNS o il *default gateway*) o controllare lo stato del servizio, al solito si può fare riferimento alla pagina di manuale. Il programma può anche essere controllato tramite un file di configurazione (`/etc/pump.conf` il cui formato è sempre descritto nella pagina di manuale, accessibile con **man pump**).

All'invocazione del comando, una volta ricevuta risposta da un server, il comando provvede utilizzare le informazioni ricevute per configurare la rete; queste in genere comprendono, oltre l'indirizzo IP da assegnare, anche il dominio locale, gli indirizzi dei server DNS e quello del *default gateway*, pertanto il comando non si limiterà ad assegnare un indirizzo all'interfaccia specificata, ma creerà anche un opportuno **resolv.conf** ed inserirà la *default route* nella tabella di routing. Qualora il server non dia nessuna risposta invece il comando fallirà, così come la configurazione automatica della rete.

¹³ci si può chiedere come possa essere utilizzato UDP, che presuppone IP, prima che la macchina abbia un indirizzo IP con il quale contattarla; quello che accade è che i pacchetti vengono inviati con IP sorgente nullo (indirizzo 0.0.0.0 e porta 68), ed in *broadcast* come IP di destinazione (indirizzo 255.255.255.255 e porta 67) con al loro interno il *MAC address* del richiedente, che così può essere raggiunto dal server.

¹⁴in realtà il meccanismo è più complesso, il protocollo prevede che il client invii un pacchetto di tipo *dhcp discovery*, che in sostanza chiede se esiste un server DHCP disponibile; a questo i server presenti risponderanno con dei pacchetti di offerta (di tipo *dhcp offer*) con una proposta di indirizzo IP e di *lease*, ed il client potrà scegliere quello che preferisce (in genere in base alla maggior durata del *lease*) rispondendo con un pacchetto di tipo *dhcp request*; la negoziazione si chiude alla risposta del server con un pacchetto *dhcp ack*, senza il quale il procedimento ricomincia da capo.

Il secondo programma che viene utilizzato come client per la configurazione automatica della rete è `dhclient`. Questo è parte della implementazione di riferimento del protocollo DHCP fatta dall'Internet Software Consortium, ed in genere viene utilizzato nella versione 3, anche se alcune distribuzioni (Debian ad esempio) utilizzano anche la versione 2 del programma.

L'invocazione manuale del programma è analoga a quella di `pump`, ma è necessario specificare nessuna opzione per indicare una interfaccia. Se non si passano argomenti il programma esamina le interfacce presente, scarta quelle che non supportano il *broadcast*, e poi cerca di configurarle tutte; altrimenti si può specificare la lista delle interfacce da configurare con qualcosa del tipo:

```
dhclient eth1 eth2
```

il comando supporta tre opzioni, `-p` consente di specificare l'uso di una porta diversa da quella standard, `-d` che esegue il programma in modalità di debug mantenendolo agganciato al terminale,¹⁵ e `-e` che causa l'uscita del programma se questo non riesce a configurare l'interfaccia entro un certo tempo.

Il comportamento di `dhclient` è controllato dal file `/etc/dhclient.conf` (o, se si è installato la versione 3, `/etc/dhcp3/dhclient.conf`) riguardo ad aspetti come le temporizzazioni, le informazioni richieste al server, la presenza di valori che devono sovrascrivere o essere aggiunti a quelli presenti in eventuali risposte o utilizzati come default qualora questo non risponda.

Il formato del file prevede al solito che le righe vuote e tutto quanto segue il carattere “#” venga ignorato. Le direttive sono di due tipi, quelle elementari, che specificano delle caratteristiche singole, hanno la forma di una parola chiave seguita da uno o più valori e sono terminate dal carattere “;”. Quelle complesse possono specificare più caratteristiche e prevedono una parola chiave seguita da un eventuale parametro e da di un blocco di ulteriori direttive semplici (nella forma precedente) che viene delimitato da parentesi graffe. In generale spaziature e ritorni a capo che vengono ignorati, ed i nomi delle direttive sono *case insensitive*.

Come esempio di `dhclient.conf`, preso dalla versione installata su una Debian Sarge (nella versione 3 del client) è il seguente estratto, dove come si può notare la maggior parte delle direttive sono commentate:

```
# Configuration file for /sbin/dhclient, which is included in Debian's
#     dhcp3-client package.
#
# This is a sample configuration file for dhclient. See dhclient.conf's
#     man page for more information about the syntax of this file
#     and a more comprehensive list of the parameters understood by
#     dhclient.
#
# Normally, if the DHCP server provides reasonable information and does
#     not leave anything out (like the domain name, for example), then
#     few changes must be made to this file, if any.
#

#send host-name "andare.fugue.com";
#send dhcp-client-identifier 1:0:a0:24:ab:fb:9c;
#send dhcp-lease-time 3600;
#supersede domain-name "fugue.com home.vix.com";
#prepend domain-name-servers 127.0.0.1;
request subnet-mask, broadcast-address, time-offset, routers,
        domain-name, domain-name-servers, host-name,
        netbios-name-servers, netbios-scope;
#require subnet-mask, domain-name-servers;
#timeout 60;
#retry 60;
#reboot 10;
```

¹⁵si ricordi che anche il client DHCP lavora come demone, dovendo gestire la scadenza dei *lease*.

```
#select-timeout 5;
#initial-interval 2;
#script "/etc/dhcp3/dhclient-script";
#media "-link0 -link1 -link2", "link0 link1";
#reject 192.33.137.209;

...
```

In genere non è necessario modificare questo file, che può anche essere lasciato vuoto. Nell'esempio precedente la sola direttiva utilizzata è **request** che serve ad indicare quale informazioni si richiedono al server DHCP. Un'altra direttiva che può essere utilizzata è **send** che consente di inviare informazioni al server, come ad esempio l'hostname (vedi sez. 7.4.3) che può essere utilizzato da quest'ultimo per aggiornare la lista delle macchine presenti sulla rete. Un elenco delle principali direttive è riportato in tab. 8.4, l'elenco completo è al solito disponibile sulla pagina di manuale.

Direttiva	Significato
request	elenca la lista delle opzioni DHCP (vedi tab. 8.7) che il client richiede al server.
timeout	indica il tempo massimo oltre il quale il client stabilisce che non ci sono server raggiungibili.
send	indica una opzione DHCP (vedi tab. 8.7) ed il relativo valore che il client invia al server.
script	specifica lo script di usato dal client per configurare l'interfaccia, di default viene usato <code>/etc/dhcp3/dhclient-script</code> .
reject	specifica l'indirizzo IP di un server da cui non devono essere accettate risposte.
interface	permette di impostare direttive di configurazione specifiche per una interfaccia; richiede come argomento il nome della interfaccia in questione ed a seguire il blocco delle direttive da applicare.

Tabella 8.4: Principali direttive del file `dhclient.conf`.

Si tenga presente che le operazioni di **dhclient** non sono limitate alla configurazione iniziale della rete; il protocollo infatti prevede che il client debba ricontattare periodicamente il server per prolungare un *lease* che sta scadendo (mantenendo l'associazione con il relativo numero IP) o per riacquisire le informazioni, o per notificare la cessazione dell'uso di un indirizzo IP in caso di uscita.

Per la gestione della assegnazione dinamica degli indirizzi **dhclient** mantiene una lista dei *lease* ottenuti nel file `dhclient.leases` (in genere questo file si trova sotto `/var/lib/dhcp/`), in modo da tenerne traccia anche se si sono effettuati riavvii del sistema o se il server ha avuto un crash.¹⁶ Anche questo file viene letto all'avvio ed i suoi dati vengono utilizzati qualora un server DHCP sia irraggiungibile.

Se questo poi tornasse disponibile i vecchi *lease* che non sono scaduti vengono ricontrollati presso il server e se sono trovati validi vengono riutilizzati. Il client si cura inoltre di aggiornare `dhclient.leases` tutte le volte che ottiene una nuova associazione con un nuovo *lease*, e per evitare che il file cresca indefinitamente crea periodicamente una nuova versione del file contenente solo i *lease* attivi al momento, salvando la vecchia come `dhclient.leases~`.

8.2.3 La configurazione di un server DHCP

Sul lato server il protocollo DHCP viene realizzato tramite un apposito demone, `dhcpd`, che si incarica di ricevere le richieste e fornire le risposte. Il server viene lanciato dal relativo script di

¹⁶il formato del file è descritto nella relativa pagina di manuale, accessibile con `man dhclient.lease`.

avvio (su Debian è `/etc/init.d/dhcp3-server` o `/etc/init.d/dhcpd` a seconda della versione), ma può anche essere avviato a mano. In quel caso si può usare l'opzione `-d` per eseguire il server interattivamente in modalità di debug. Un'altra opzione è `-p` che permette di usare una porta diversa da quella di default (la 67 su UDP); per le altre opzioni si può fare riferimento alla pagina di manuale).

Come per il client il server mantiene una lista delle assegnazioni effettuate e dei relativi *lease* in un apposito file, `dhcpd.leases` (sempre sotto `/var/lib/dhcp/`), così da poter tenere traccia dello stato del sistema anche in caso di riavvio.¹⁷ Il server scrive in questo file ogni nuovo *lease* assegnato e ne cura la rotazione periodica come fa `dhclient` per `dhclient.leases`.

Il funzionamento del server è controllato da un file di configurazione che è `/etc/dhcpd.conf` (anche se per la versione 3 su Debian viene usato `/etc/dhcp3/dhcpd.conf`); un esempio del contenuto di questo file è il seguente:

```
#
# Sample configuration file for ISC dhcpd for Debian
#

# option definitions common to all supported networks...
option domain-name "earthsea.ea";
option domain-name-servers gont.earthsea.ea;

option subnet-mask 255.255.255.0;
default-lease-time 600;
max-lease-time 7200;

subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.32 192.168.1.63;
    option broadcast-address 192.168.1.255;
    option routers gont.earthsea.ea;
}

host oppish {
    hardware ethernet 08:00:07:26:c0:a5;
    fixed-address oppish.earthsea.ea;
}
```

Il formato del file è identico a quello di `dhclient.conf` e come per questo il contenuto è divisibile sommariamente in due categorie di direttive, quelle che prevedono la specificazione di semplici parametri e quelle più complesse, dette dichiarazioni, che contengono blocchi di altre direttive. In generale il file prevede una serie di direttive iniziali, che servono ad impostare i valori che vengono utilizzati come default. Questi stessi valori possono essere impostati all'interno di dichiarazioni più specifiche a valori diversi.

Un esempio di direttive semplici sono le righe iniziali del precedente esempio che nel caso sono utilizzate per impostare alcune opzioni DHCP a livello generale, come il dominio di riferimento o il server DNS, o la maschera di rete. In questo caso si usa la direttiva `option`, seguito dal nome dell'opzione DHCP che si intende impostare (le principali sono riportate in tab. 8.7) ed dal relativo valore. Altre due direttive semplici sono `default-lease-time`, che imposta il valore di default (nel caso 600 secondi) del *lease* delle risposte, e `max-lease-time` che imposta valore massimo che può essere assegnato ad un *lease*. Un elenco delle principali direttive per la dichiarazione di parametri è illustrato in tab. 8.5.

In generale DHCP consente di suddividere gli IP restituiti alle stazioni in sottoreti, nel nostro esempio ne viene usata una soltanto. Questo è fatto tramite la direttiva `subnet`, che serve a dichiarare una sottorete; essa è seguita dall'indirizzo della rete e dalla specificazione della relativa

¹⁷per chi fosse interessato il formato di questo file è descritto nella pagina di manuale accessibile con `man dhcpd.leases`.

Direttiva	Significato
option	imposta il valore di una opzione DHCP prende come argomento il nome dell'opzione (vedi tab. 8.7), seguito dagli eventuali valori.
default-lease-time	imposta il valore di default del tempo di <i>lease</i> inviato a un client se questo non ha fatto una richiesta specifica.
max-lease-time	imposta il valore massimo del tempo di <i>lease</i> che può essere inviato ad un client, indipendentemente da quanto esso possa avere richiesto.
range	imposta un intervallo di indirizzi da assegnare dinamicamente.
fixed-address	imposta un indirizzo (o più) da assegnare staticamente, viene in genere utilizzato all'interno di una dichiarazione host insieme a hardware address .
hardware address	indica l'indirizzo hardware (in genere un <i>MAC address</i>) cui assegnare un indirizzo statico.
include	legge il contenuto di un altro file come se questo fosse stato incluso nella configurazione.
filename	specifica il nome del file da caricare come sistema operativo (deve essere riconosciuto come nome valido dal protocollo, in genere TFTP, usato dal client per caricarlo).
next-server	specifica il server da cui scaricare il file per il boot del sistema (indicato dalla precedente filename).

Tabella 8.5: Principali direttive del file `dhcpd.conf`.

`netmask` (preceduta dalla parola chiave `netmask`). All'interno della dichiarazione si potrà poi indicare l'uso di una assegnazione dinamica specificando l'intervallo di indirizzi con la direttiva `range`, oltre ai tempi di *lease* e alle altre opzioni DHCP da usare per quella sottorete.

Un'altra dichiarazione utile è quella che consente di configurare una stazione singola (analoga dell'assegnazione statica di BOOTP), anch'essa mostrata nell'esempio a pag. 8.2.3. Questa viene fatto tramite la direttiva `host` seguita nome della macchina, quest'ultima sarà identificata dal *MAC address* specificato dalla direttiva `hardware ethernet`, mentre l'indirizzo fisso da assegnare verrà specificato dalla direttiva `fixed-address`).

Direttiva	Significato
subnet	dichiara una sottorete, ad essa deve seguire l'indirizzo IP della sottorete e la relativa maschera introdotta dalla parola chiave <code>netmask</code> ; il corpo della dichiarazione conterrà i parametri da applicare alla suddetta sottorete.
host	dichiara una macchina singola, ad essa deve seguire il nome della stessa; il corpo della dichiarazione conterrà i parametri da applicare alla suddetta macchina.
group	dichiara un gruppo di macchine, ad essa deve seguire il nome del gruppo; il corpo della dichiarazione conterrà i parametri da applicare alle macchine del gruppo identificate attraverso ulteriori dichiarazioni di tipo <code>host</code> .

Tabella 8.6: Principali direttive di dichiarazione del file `dhcpd.conf`.

Si noti come in entrambi questi esempi di dichiarazioni si sono dovute poi specificare i valori dei parametri da applicare con una serie di altre direttive semplici poste all'interno di un blocco delimitato da parentesi graffe. Un elenco delle principali direttive di dichiarazione è illustrato in tab. 8.6. Per una lista completa dei parametri si può fare riferimento alla pagina di manuale del file di configurazione accessibile con `man dhcpd.conf`.

Una parte essenziale della configurazione del server è quella relativa alle opzioni DHCP, che costituiscono il contenuto delle risposte che il server fornisce ai client. È attraverso queste

Opzione	Significato
<code>domain-name</code>	specifica il nome di dominio in cui ci si trova (ad uso dell'impostazione automatica di <code>resolv.conf</code>).
<code>domain-name-servers</code>	specifica una lista di server DNS (anche questo per l'impostazione automatica dei relativi campi in <code>resolv.conf</code>).
<code>subnet-mask</code>	specifica la netmask per la rete associata all'indirizzo assegnato.
<code>broadcast-address</code>	specifica l'indirizzo di <i>broadcast</i> per la rete associata all'indirizzo assegnato.
<code>routers</code>	specifica il <i>default gateway</i> per la rete associata all'indirizzo assegnato.
<code>host-name</code>	specifica il nome della macchina.
<code>netbios-name-servers</code>	specifica l'indirizzo di un server WINS (usato dalle macchine Windows).

Tabella 8.7: Principali opzioni del protocollo DHCP.

opzioni che si inviano ai client le varie informazioni sulla rete che ne permettono la configurazione automatica. Come già visto esse possono essere inserite sia direttamente nel corpo principale di `dhcpd.conf`, dove assumeranno il ruolo di valore di default, che all'interno di singole dichiarazioni per sottoreti o macchine singole.

Le opzioni vengono sempre impostate da una direttiva nella forma `option nome-opzione valore-opzione`; ed una lista dei nomi delle principali opzioni utilizzate è riportata in tab. 8.7; al solito l'elenco completo è disponibile sulla relativa pagina di manuale, accessibile con `man dhcp-options`.

8.3 Il servizio SSH

La sigla SSH sta a significare *Secure SHell*, ma in realtà identifica un protocollo di comunicazione che permette di creare un canale di comunicazione cifrato fra due macchine. Benché sia possibile utilizzare il canale in maniera generica con qualunque tipo di servizio, l'uso principale di SSH è quello di fornire una shell remota per l'amministrazione, da cui il nome del servizio.

8.3.1 Il server sshd

Il servizio SSH viene fornito dal demone `sshd`. Come per gli altri demoni esso di norma viene lanciato automaticamente dagli script di avvio creati in fase di installazione del pacchetto. Il servizio ascolta di default sulla porta 22. In genere i file di configurazione del servizio vengono mantenuti in `/etc/ssh/`; la configurazione del server è data dal file `sshd_config`, un cui estratto è riportato di seguito:

```
# What ports, IPs and protocols we listen for
Port 22
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
# Lifetime and size of ephemeral version 1 server key
KeyRegenerationInterval 3600
ServerKeyBits 768
# Logging
SyslogFacility AUTH
LogLevel INFO
# Authentication:
LoginGraceTime 600
```

```

PermitRootLogin yes
StrictModes yes
RSAAuthentication yes
PubkeyAuthentication yes
# rhosts authentication should not be used
RhostsAuthentication no
# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no
# To enable empty passwords, change to yes (NOT RECOMMENDED)
PermitEmptyPasswords no
# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
# Use PAM authentication via keyboard-interactive so PAM modules can
# properly interface with the user
PAMAuthenticationViaKbdInt yes
X11Forwarding yes
X11DisplayOffset 10
PrintMotd no
KeepAlive yes

Subsystem      sftp      /usr/lib/sftp-server

```

Si tenga presente che nel nostro caso si fa riferimento alla implementazione del protocollo realizzata dal pacchetto *OpenSSH*, il programma *ssh* originale infatti, dopo essere stato rilasciato per un certo tempo con licenza libera, è diventato proprietario, ma un gruppo di programmatori indipendente è riuscito, partendo dalla ultima versione libera disponibile, a creare una implementazione completa del protocollo, che nel frattempo è stato standardizzato.

Le opzioni principali usate nel file di configurazione, ed il relativo significato, sono state riportate in tab. 8.8; si sono descritte solo le opzioni che è più probabile che un amministratore si trovi a dover modificare, i valori delle altre sono di norma impostati in fase di installazione. Al solito per una descrizione completa delle varie opzioni si può fare riferimento alla pagina di manuale disponibile con `man sshd_config`.

Opzione	Significato
Protocol	indica la versione del protocollo, deve essere impostato a 2, in quanto le precedenti versioni sono insicure; è possibile abilitare versioni inferiori solo a fine di compatibilità, ma si consiglia energicamente di aggiornare i client.
PermitRootLogin	permette il login diretto all'amministratore; per default è disabilitato, il che comporta la necessità di collegarsi come utente normale e poi usare <code>su</code> .
X11Forwarding	abilita il <i>forwarding</i> delle sessioni X11, cioè la possibilità di esportare il display di X e le finestre da una macchina all'altra attraverso il canale cifrato.
X11DisplayOffset	assegna al display X fatto passare attraverso il canale cifrato un offset di 10, così da non creare conflitti con gli altri display eventualmente presenti in locale.
Subsystem	permette di abilitare l'accesso alla macchina con una sintassi simile a FTP, tramite il programma <code>sftp</code> .

Tabella 8.8: Principali opzioni di configurazione per il demone `sshd` usate nel file `sshd_config`.

Si tenga presente inoltre che `sshd` onora sia la presenza del file `/etc/nologin` non consentendo, quando esso esiste, la connessione ad utenti che non siano l'amministratore, che l'utilizzo dei *TCP wrappers*, rispondendo alle restrizioni poste nei file `hosts.allow` e `hosts.deny` (vedi sez. 8.1.4).

8.3.2 I comandi ssh ed scp

I due comandi principali di utilizzo di SSH dal lato client sono **ssh** e **scp** utilizzati rispettivamente per il collegamento su una macchina da remoto, e per la copia dei file. Per **scp** è di norma disponibile anche un front-end con comandi simili a quelli di FTP, attraverso l'uso del comando **sftp**.

Prima di entrare nei dettagli dei vari comandi occorre precisare che anche i client sono controllati da un file di configurazione, `/etc/ssh_config`, un estratto del quale è riportato di seguito; di norma, come nell'esempio, non è necessario impostare niente di diverso dai default. L'amministratore può però voler imporre delle restrizioni nell'accesso a certe macchine (in particolare si può disabilitare la possibilità del *forwarding* delle sessioni X).

```
# Host *
#   ForwardAgent no
#   ForwardX11 no
#   RhostsAuthentication no
#   RhostsRSAAuthentication no
#   RSAAuthentication yes
#   PasswordAuthentication yes
#   HostbasedAuthentication no
#   BatchMode no
#   CheckHostIP yes
#   StrictHostKeyChecking ask
#   IdentityFile ~/.ssh/identity
#   IdentityFile ~/.ssh/id_rsa
#   IdentityFile ~/.ssh/id_dsa
#   Port 22
#   Protocol 2,1
#   Cipher 3des
#   Ciphers aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc
#   EscapeChar ~
```

Esempi delle possibili opzioni impostabili sono riportati direttamente nell'esempio; al solito l'elenco completo, la spiegazione delle stesse e i loro possibili valori sono riportati nella pagina di manuale accessibile con `man ssh_config`.

Il comando generico del client è **ssh**. La sintassi ha due forme, la più semplice, che serve per ottenere una shell remota (o eseguire un comando specifico), è:

```
ssh [-l login_name] hostname | user@hostname [command]
```

in cui in sostanza ci si può collegare ad un computer remoto con un username e password (quest'ultima viene richiesta da terminale).

Non specificando **command** si avrà una shell remota, dalla quale inviare comandi come da un qualunque terminale, altrimenti si può specificare un qualunque comando che sarà eseguito sulla macchina remota. Una opzione interessante è la possibilità di specificare l'opzione **-X**, che abilita il forward della sessione X11, permettendo di ricevere sul proprio desktop le finestre delle applicazioni lanciate in remoto. Per le altre opzioni si rimanda di nuovo alla pagina di manuale del comando che è accessibile con `man ssh`.

Con il comando **scp** si può invece copiare un file da una macchina ad un'altra attraverso la rete, usando sempre il canale cifrato, la sintassi è analoga a quella del comando **cp**, solo che un file può essere indicato in forma generica da un identificativo del tipo **utente@macchina:file** dove **utente** è l'username con il quale ci si vuole collegare alla macchina remota, che deve essere specificato, se non corrisponde a quello con cui si sta lavorando, mentre **macchina** è l'indirizzo (numerico o simbolico) di quella macchina, e **file** il pathname (assoluto o relativo alla home dell'utente usato) del file in questione su quella macchina.

Così se si vuole copiare un file da una macchina remota si potrà eseguire un comando del tipo:


```
scp piccardi@firenze.linux.it:netadmin.pdf .
```

che copia il file `netadmin.pdf` dalla mia home sul server alla directory corrente, mentre per effettuare un trasferimento in direzione opposta si potrà usare:

```
scp -r gapil piccardi@firenze.linux.it:public_html/
```

che copia ricorsivamente il contenuto della directory `gapil` nella directory `public_html` della mia home sul server (che è quella pubblicata su web tramite Apache).

Il comando richiede ovviamente l'immissione da terminale della password relative all'utente delle macchine a cui ci si collega, dopo di che effettua la copia; l'opzione `-p` permette di preservare i tempi ed il modo del file originale, mentre l'opzione `-r` esegue una copia ricorsiva di intere directory. Al solito per i dettagli su tutte le altre opzioni si può fare riferimento alla pagina di manuale accessibile con `man scp`.

8.3.3 Autenticazione a chiavi

Una delle caratteristiche più interessanti del protocollo è quella di consentire, rispetto alla classica autenticazione con password identica a quella ottenibile su un terminale classico, la possibilità di utilizzare una autenticazione basata su chiavi crittografiche. Questo permette una serie di semplificazione dell'uso dei comandi, come la possibilità di creare delle sessioni in cui, data una password all'inizio, diventa possibile evitare di riscriverla tutte le volte che si utilizza uno dei comandi `ssh` o `scp`.

La tecnica utilizzata è quella delle chiavi asimmetriche, che possono essere sia di tipo RSA che DSA. L'uso di chiavi asimmetriche permette il riconoscimento univoco di un utente, una volta che questo dimostri di essere in possesso della chiave privata associata alla chiave pubblica usata del server come identificatore dello stesso. La trattazione della crittografia a chiave simmetrica va al di là di quanto sia possibile affrontare in questo contesto, basti sapere che le chiavi asimmetriche sono generate in coppie e che un messaggio cifrato con una delle due chiavi può essere decifrato solo dall'altra. Per questo in genere si distribuisce una delle due chiavi della coppia, la *chiave pubblica*, in modo che solo chi possiede l'altra, la *chiave privata* possa decifrare i messaggi creati con la prima.

Nel caso specifico il protocollo prevede comunque l'uso di chiavi simmetriche in fase di negoziazione della connessione via SSH; ciascun capo della connessione fornisce all'altro la propria chiave pubblica, dopo di che i due si potranno scambiare in maniera cifrata una *chiave di sessione* con cui sarà crittato tutto il successivo scambi di dati. Per maggiore sicurezza questa chiave viene cambiata periodicamente (secondo quanto specificato per il server con il parametro di configurazione `KeyRegenerationInterval`).

Nel caso di autenticazione a chiave quello che succede è che il server invia al client un segreto cifrato con la chiave pubblica dell'utente cui si vuole garantire l'accesso, solo il reinvio del segreto decifrato è garanzia che il client conosce la chiave privata, che è quello che garantisce l'autenticità dell'identità dell'utente.

Per poter utilizzare questa modalità di autenticazione occorre anzitutto generare una coppia di chiavi; il pacchetto *OpenSSH* mette a disposizione un apposito programma per la creazione e la gestione delle chiavi, `ssh-keygen`, la cui sintassi, come risulta dalla pagina di manuale, è la seguente:

```
ssh-keygen [-q] [-b bits] -t type [-N new_passphrase] [-C comment]
            [-f output_keyfile]
ssh-keygen -p [-P old_passphrase] [-N new_passphrase] [-f keyfile]
ssh-keygen -c [-P passphrase] [-C comment] [-f keyfile]
```

Se invocata senza nessuna opzione il comando stampa un messaggio di aiuto, per eseguire la creazione di una nuova coppia di chiavi occorre infatti specificare almeno il tipo di chiave con l'opzione `-t`; i possibili tipi di chiavi sono tre: `rsa` per chiavi RSA, `dsa` per chiavi DSA e `rsa1` per il formato di chiavi RSA usato dal vecchio protocollo (la versione 1, adesso in disuso perché insicura).

Se non si specifica nient'altro il comando chiede il file in cui salvare la chiave, che di default è nella directory `.ssh` nella home directory dell'utente, con un nome che può essere `id_dsa` o `id_rsa` a seconda del tipo della chiave (o `identity` per le chiavi del vecchio protocollo), ed infine una *passphrase* che serve a proteggere l'accesso alla chiave privata, che non può essere letta senza di essa. Il comando crea anche la rispettiva chiave pubblica, con lo stesso nome usato per quella privata ma con un `.pub` terminale. Si può specificare un nome diverso anche a riga di comando usando l'opzione `-f` seguita dal nome del file.

Se si vuole cambiare la passphrase in un secondo tempo si può usare l'opzione `-p`, alla chiave è pure associato un commento, che non ha nessun uso nel protocollo, e serve solo da identificativo della chiave; esso viene di solito inizializzato alla stringa `user@host` e può essere cambiato con l'opzione `-c`. Con l'opzione `-l` invece si può stampare la *impronta digitale* (la cosiddetta *fingerprint*) della chiave a scopo di verifica; infine l'opzione `-b` permette di specificare la lunghezza della chiave (in bit); il valore di default, 1024, è più che adeguato. I dettagli sul funzionamento del comando e le restanti opzioni sono accessibili nella pagina di manuale con `man ssh-keygen`.

Una volta creata la coppia di chiavi diventa possibile utilizzarla per l'autenticazione. Per far questo occorre inserire la chiave pubblica della persona a cui si vuole dare l'accesso nel file `.ssh/authorized_keys` posto nella home dell'utente per conto del quale si accederà. Il file può contenere un numero imprecisato di chiavi pubbliche,¹⁸ per cui si deve effettuare l'aggiunta con un comando del tipo:

```
cat id_dsa.pub >> .ssh/authorized_keys
```

(si suppone di essere nella home dell'utente che dà l'accesso) dove `id_dsa` è la chiave dell'utente che deve poter accedere. Si tenga presente che, come accennato, in questo caso essa viene effettuata solo sulla base della corrispondenza fra una chiave pubblica ed una chiave privata, non è necessario che il nome dell'utente sia lo stesso, tutto quello che serve è la presenza della coppia di chiavi, la pubblica per l'utente che concede l'accesso, e la privata per quello che deve accedere. Il procedimento può essere eseguito direttamente con il comando `ssh-copy-id`, che si cura di aggiungere adeguatamente la chiave nell'`authorized_keys` di una macchina destinazione con:

```
ssh-copy-id piccardi@oppish.truelite.it
```

Dunque la protezione della chiave privata è essenziale, chiunque ne venga in possesso e possa utilizzarla ottiene immediatamente tutti gli accessi a cui essa è abilitata, e questo è il motivo per cui essa viene protetta con una passphrase, che deve essere fornita tutte le volte che la si deve usare. Essa comunque deve essere protetta sia da lettura che da scrittura da parte di estranei. Questo è imposto dallo stesso comando, che si rifiuta di usare (stampando un clamoroso avvertimento) una chiave privata leggibile da altri rispetto al proprietario.

A questo punto però ci si potrebbe chiedere l'utilità di tutto questo armamentario, dato che invece di una password per il login bisogna comunque inserire una passphrase per sbloccare la chiave pubblica; l'utilità sta nel fatto che è possibile usare un altro programma, `ssh-agent`, che permette di sbloccare la chiave una sola volta, all'inizio di una sessione di lavoro, e poi

¹⁸le chiavi sono mantenute una per riga, in formato codificato ASCII, per cui le singole righe sono di norma molto lunghe.

mantiene la chiave privata sbloccata in memoria, permettendone il successivo riutilizzo senza che sia necessario fornire di nuovo la passphrase.

L'idea è che **ssh-agent** dovrebbe essere lanciato all'inizio di una sessione, dopo di che tutti i comandi verranno lanciati come client di esso cosicché questi possano interrogarlo tutte le volte che è necessario compiere una operazione con la chiave privata. Sarà comunque **ssh-agent** ad eseguire le operazioni necessarie, fornendo ai richiedenti i risultati ottenuti, in modo da non dover mai comunicare verso l'esterno la chiave privata.

Di norma **ssh-agent** viene lanciato automaticamente all'avvio delle sessioni grafiche; mentre lo si deve lanciare esplicitamente nel caso di login da terminale. Si tenga presente che all'avvio il programma non ha nessuna chiave privata. Per poterlo usare occorre aggiungere una chiave privata con il comando **ssh-add**, questo chiederà la passphrase della chiave privata e la sbloccherà, inviandola all'agent che da quel momento in poi potrà utilizzarla.

Per poter comunicare con **ssh-agent** i vari programmi usano la variabile di ambiente **SSH_AUTH_SOCK** che identifica il socket locale su cui esso è in ascolto; se questa non è definita anche se il programma è attivo l'autenticazione a chiavi non funzionerà (e si passerà direttamente alla autenticazione normale). La variabile è definita dallo stesso **ssh-agent** quando viene lanciato (che la stampa anche sullo standard output). Si ricordi però che le variabili di ambiente definite da un processo vengono viste soltanto nei processi figli (per questo la shell prevede il comando **export**), pertanto quando si lancia **ssh-agent** in console, questo resterà attivo in background, ma fintanto che non si inserirà manualmente la variabile **SSH_AUTH_SOCK** nell'ambiente, non potrà essere utilizzato.

8.4 Il protocollo NFS

Il protocollo NFS, sigla che sta a significare *Network File System*, è stato creato da Sun per permettere montare dischi che stanno su stazioni remote come se fossero presenti sulla nostra macchina. In questo modo si può assicurare un accesso trasparente ai file, e si può utilizzare un filesystem anche su una macchina senza dischi, passando solo attraverso la rete.

8.4.1 Il server NFS

In genere tutte le distribuzioni prevedono i pacchetti per l'installazione di un server NFS. La sola scelta che si può dover fare è fra l'uso del supporto nel kernel o l'uso di una implementazione realizzata completamente in user-space, che per le sue peggiori prestazioni è comunque da evitare.

Per questo occorrerà avere abilitato il supporto nel kernel, cosa che vale in genere per i kernel standard di tutte le distribuzioni. Qualora si ricompili un kernel da soli occorrerà verificare che siano abilitate in **.config** le seguenti opzioni:

```
CONFIG_NFS_FS=m
CONFIG_NFS_V3=y
CONFIG_NFSD=m
CONFIG_NFSD_V3=y
```

accessibili con **make menuconfig** dal menù **File system** nel sotto menù **Network File System**.

Il protocollo NFS è piuttosto complesso, ed è basato sul sistema delle RPC (vedi sez. 7.5.5); al funzionamento di NFS, oltre a **portmap**, concorrono ben altri cinque demoni:

rpc.nfsd	è il demone che svolge la gran parte del lavoro, realizzando le funzioni di accesso al contenuto dei file. Deve essere lanciato dopo che è stato attivato portmap .
rpc.statd	è il demone che gestisce le caratteristiche dei file come tempi di accesso, permessi, ecc. Deve essere lanciato dopo che è stato attivato portmap .

`rpc.lockd` è il demone che gestisce, quando questo è abilitato, il file locking. Di norma viene lanciato da `rpc.nfsd` in caso di necessità.

`rpc.mountd` è il demone che gestisce le richieste di montaggio del filesystem. Deve essere lanciato dopo che sono stati attivati `rpc.nfsd` e `rpc.statd`.

`rpc.rquotad` è il demone che permette, quando sono attivate, di gestire le quote sul filesystem di rete. Deve essere lanciato dopo che sono stati attivati `rpc.nfsd` e `rpc.statd`.

Benché complicato da descrivere dal punto di vista funzionale un server NFS è in genere molto semplice da configurare. Infatti di norma il servizio viene avviato automaticamente dagli script di avvio installati dai pacchetti, e tutto quello che c'è da configurare è il file `/etc/exports` che definisce su ciascuna macchina quali sono le directory da *esportare* verso l'esterno e chi ha la facoltà di utilizzarle; un esempio del file è il seguente:

```
# /etc/exports: the access control list for filesystems which may be exported
#                to NFS clients.  See exports(5).
/home/piccardi/share  192.168.1.0/24(rw)
```

Il file prende una serie di campi separati da spazi o tabulazioni; il primo campo specifica la directory da condividere, i campi successivi definiscono chi e come può utilizzarla. Si può specificare, come nell'esempio, sia una singola stazione, che una rete (usando la notazione CIDR) si possono usare anche indirizzi simbolici (che in questo caso devono poter essere risolti) nel qual caso si possono usare i caratteri *wildcard* “*” e “?” per esprimere i nomi. Fra parentesi tonde poi si possono specificare le modalità di accesso; il formato generale di una riga è il seguente:

```
/path/to/directory machine1(option11,option12) machine2(option21,option22)
```

dove il separatore è costituito dallo spazio.¹⁹

Le opzioni sono suddivise in due gruppi: quelle generali, relative al funzionamento del server e quelle di mappatura degli utenti. Le opzioni principali riguardanti il comportamento del server sono quelle che permettono di impostare le modalità di accesso ai file, `rw` e `ro`, che indicano rispettivamente lettura/scritture e sola lettura, e quelle che ne governano la risposta in reazione a richieste di scrittura, come `sync` che richiede che la scrittura dei dati su disco sia completata prima di concludere una risposta, ed `async` che permette una risposta immediata con una scrittura dei dati asincrona.²⁰

Quando si esporta una directory si pone il problema dell'accesso ai file, infatti il protocollo associa i file agli utenti ed ai gruppi così come sono impostati sul server, utilizzando rispettivamente i relativi uid e gid, ed utente sul client potrà avere accesso ai file solo qualora questi corrispondano. In alcuni casi però occorrono delle eccezioni, ad esempio non è desiderabile che l'amministratore di un client sia trattato come root anche nell'accesso ai file del server. Per questo motivo di default è attiva l'opzione `root_squash` che rimappa gli uid e gid 0 usati dall'amministratore (`root`) sul valore 65534 (-2), corrispondente rispettivamente a `nobody` e `nogroup`.

I nomi delle opzioni principali ed il relativo significato sono riportati in tab. 8.9, l'elenco completo delle opzioni è riportato nella pagina di manuale accessibile con `man exports`.

Per controllare i filesystem che si sono esportati si può usare il comando `exportfs`. La sintassi del comando, come descritta dalla pagina di manuale è:

¹⁹attenzione quindi a non specificare le opzioni inserendo uno spazio fra il nome della macchina e le parentesi che le delimitano, in tal caso infatti esse verrebbero considerate come un nuovo indirizzo.

²⁰questa opzione è disabilitata di default, in quanto in caso di crash del server le modifiche andrebbero perse. Il comando `exportfs` notifica se nessuna di queste due opzioni è stata impostata, avvisando che sarà usata `sync` di default.

Opzione	Significato
rw	accesso in lettura e scrittura.
ro	accesso in sola lettura.
async	abilita la scrittura asincrona sul server.
sync	esegue le scritture in maniera sincrona.
root_squash	rimappa l'uid ed il gid 0.
no_root_squash	non rimappa l'uid ed il gid 0.
all_squash	mappa tutti gli uid e i gid.
anonuid	imposta l'uid usato per l'accesso anonimo.
anongid	imposta il gid usato per l'accesso anonimo.

Tabella 8.9: Possibili opzioni per le directory esportate tramite NFS nel file `/etc/exports`.

```

/usr/sbin/exportfs [-avi] [-o options,...] [client:/path ...]
/usr/sbin/exportfs -r [-v]
/usr/sbin/exportfs [-av] -u [client:/path ...]
/usr/sbin/exportfs [-v]

```

Di norma vengono esportati tutti i filesystem elencati in `/etc/exports` con il comando `exportfs -a`; qualora si modifichi `/etc/exports` si può usare il comando `exportfs -r` per sincronizzare la tabella dei filesystem esportati con i contenuti del file. Infine con l'opzione `-u` si può rimuovere uno dei filesystem esportati, mentre con `-o` si possono cambiare le opzioni. I dettagli si trovano al solito nella pagina di manuale, accessibile con `man exportfs`.

Si tenga conto infine che i vari demoni che forniscono il servizio hanno il supporto per i TCP wrapper, quindi onorano le restrizioni descritte in sez. 8.1.4. È buona norma, visto la delicatezza del servizio fornito, impostare sempre una politica di accesso negato di default per tutti i demoni, per abilitare in `/etc/hosts.allow` gli accessi alle stesse macchine riportate in `/etc/exports`.

8.4.2 NFS sul lato client

Per poter utilizzare NFS sul lato client occorrerà avere il supporto nel kernel, ed oltre a `portmap` dovranno essere attivi i due demoni `rpc.statd` e `rpc.lockd`. Inoltre si deve avere una versione sufficientemente recente del comando `mount` che sia in grado di montare un filesystem NFS.

Come per il lato server di norma l'utilizzo è molto semplice e si esaurisce nel montare il filesystem remoto sulla directory prescelta, in quanto con l'installazione dei relativi pacchetti tutte le distribuzioni si curano che siano avviati tutti i servizi necessari.

Montare un filesystem NFS è comunque estremamente semplice, invece di indicare un dispositivo basterà indicare l'indirizzo (simbolico o numerico) del server seguito da `:` e dalla directory che si vuole montare (che dovrà essere presente e accessibile nel file `/etc/exports` del server). Se si vuole eseguire il comando a mano basterà eseguire `mount` con una sintassi del tipo:

```
[piccardi@hogen]# mount -t nfs havnor:/home/piccardi/share /mnt/nfs
```

Si può poi definire un *mount point* permanente in `/etc/fstab` aggiungendo una riga del tipo:

```
192.168.1.1:/home/piccardi/temp /mnt/nfs nfs user,exec,noauto 0 0
```

dove la differenza con un filesystem su disco è l'uso di `nfs` come *filesystem type* e l'indirizzo della directory da montare al posto del file di dispositivo.

Per ulteriori dettagli riguardo il protocollo e le opzioni più avanzate si possono consultare le pagine di manuale dei vari comandi e file di configurazione.

Si tenga infine conto di un problema che può sorgere con l'uso di NFS. Il protocollo supporta un filesystem di rete di tipo Unix, prevede pertanto permessi e proprietari dei file, il problema è

che i proprietari dei file sono riconosciuti sulla base degli user-id definiti sul server. Pertanto se sul client l'utente relativo non esiste o ha un user-id diverso (cosa possibile, se non si sono creati gli utenti nella stessa sequenza) la titolarità dei file non corrisponderà, e si avranno problemi. Pertanto l'uso di NFS richiede una certa cura nella gestione di utenti e gruppi.

8.5 La condivisione dei file con Samba

La suite Samba è un insieme di programmi che permette ad una macchina Unix di utilizzare il protocollo di comunicazione SMB (*Service Message Block*) dei server Windows. Originariamente il servizio veniva fornito attraverso il protocollo proprietario *NetBIOS* usato da Windows come protocollo di trasporto; a questo livello il protocollo ormai è in disuso, ed attualmente è implementato direttamente su TCP/IP.

8.5.1 La configurazione di Samba come server

L'implementazione dei servizi SMB è realizzata da Samba attraverso due demoni, `smbd` che implementa la condivisione dei file e delle stampanti e `nmbd` che implementa i servizi NetBIOS (cioè il riconoscimento della presenza di server sulla rete, la risoluzione dei nomi, ecc.).

L'avvio del servizio è generalmente curato dagli opportuni script di avvio creati in fase di installazione del pacchetto, che ciascuna distribuzione inserisce all'interno del proprio meccanismo di boot. Nel caso di Debian ad esempio questo è gestito dallo script `/etc/init.d/samba`.

Entrambi i demoni vengono configurati tramite il file `smb.conf`. La sintassi è quella dei file `.ini` di Windows, ma viene supportata pure la sintassi dei commenti inizianti per `#` di Unix. Un estratto di questo file, preso dalla installazione di una Debian, è il seguente:

```
[global]
# Do something sensible when Samba crashes: mail the admin a backtrace
panic action = /usr/share/samba/panic-action %d
printing = bsd
printcap name = /etc/printcap
load printers = yes
guest account = nobody
invalid users = root
security = user
workgroup = WORKGROUP
server string = %h server (Samba %v)
socket options = IPTOS_LOWDELAY TCP_NODELAY SO_SNDBUF=4096 SO_RCVBUF=4096
encrypt passwords = true
passdb backend = tdbsam unixsam
dns proxy = no
unix password sync = false

[homes]
comment = Home Directories
browseable = no
read only = yes
create mask = 0700
directory mask = 0700

[printers]
comment = All Printers
browseable = no
path = /tmp
printable = yes
public = no
writable = no
create mode = 0700
```

Il file è diviso in sezioni distinte introdotte da un nome in parentesi quadra, ciascuna delle quali descrive una risorsa condivisa attraverso SMB (quello che viene chiamato uno *share*). Sono

però previste tre sezioni speciali, `[global]`, `[homes]`, e `[printers]` che non corrispondono a degli share, ma servono ad impostare alcune funzionalità generiche. All'interno di una sezione i vari parametri sono impostati con direttive del tipo **parola chiave = valore**.

La sezione `[global]` permette di impostare i parametri che si applicano al server nel suo insieme. Le opzioni fondamentali sono **workgroup** che definisce il nome del dominio NT in cui figurerà la macchina e **security** che definisce le modalità di accesso al server (e può assumere i valori **user**, **share**, **server** e **domain**), di norma (a meno di non avere un accesso anonimo, nel qual caso si può usare **share**) si utilizza come valore **user** che richiede che esista un utente Unix per ciascun utente che si collega al server. Gli altri due valori servono quando si vuole dirottare la richiesta di autenticazione.

In tab. 8.10 sono riportati le altre opzioni principali. L'elenco completo di tutti i parametri è descritto in dettaglio nella sezione **PARAMETERS** della pagina di manuale di `smb.conf`, di norma nel file fornito dai pacchetti di installazione, vengono impostati valori ragionevoli adatti agli usi più comuni. Una spiegazione dettagliata va al di là dello scopo di questo corso.

Parametro	Significato
security	stabilisce le modalità di accesso al server, il valore di default è user che richiede un username ed una password per garantire l'accesso; è necessaria la presenza di un corrispondente utente sulla macchina.
workgroup	definisce il gruppo di lavoro di cui il server farà parte (quello che è il nome di dominio NT, detto anche <i>workgroup name</i>).
encrypt passwords	utilizza una autenticazione basata su password cifrate, l'impostazione di default è true in quanto corrispondente al rispettivo default di Windows 98 e NT; se attivato necessita la presenza dell'autenticazione basata sul file smbpasswd .
guest account	specifica l'utente utilizzato per l'accesso <i>ospite</i> (cioè senza autenticazione); di norma si usa nobody che non ha nessun privilegio (in certi casi questo non consente l'uso delle stampanti).
invalid users	una lista di utenti che per conto dei quali non si può utilizzare il servizio; di norma si specifica sempre root .
socket options	specifica delle opzioni relative al comportamento del sistema nei confronti della rete.
passdb backend	specifica la lista dei supporti vengono mantenute le password.
dns proxy	quando il server è usato per la risoluzione dei nomi, passa le richieste non risolte al DNS.
unix password sync	se specificato il server tenta di sincronizzare le password UNIX quando si è cambiata nel file smbpasswd .
wins support	indica se supportare la risoluzione dei nomi <i>NetBIOS</i> di NT.
wins server	indica un server per la risoluzione dei nomi <i>NetBIOS</i> usati dal protocollo.
host allow	indica la lista delle stazioni da cui è possibile connettersi al server.

Tabella 8.10: Significato dei parametri globali per Samba, come impostati nella sezione `[globals]` di `/etc/smb.conf`.

La sezione `[homes]` quando dichiarata permette di creare al volo degli share corrispondenti alle home degli utenti. In caso di connessione prima vengono controllate le altre sezioni definite, e qualora non sia stata trovata nessuna corrispondenza la sezione richiesta viene trattata come un username da controllare sul file delle password locali. Se l'utente esiste ed è stata data una password corretta viene creato al volo il corrispondente *share* con le caratteristiche impostate in questa sezione.

Tutto questo funziona in maniera immediata solo quando si è impostato il parametro **encrypt password** a **false**, altrimenti diventa necessario creare l'opportuno supporto perché Samba possa mantenere gli utenti e relative password. I client infatti (come semplici client Windows) non sono in grado di inviare password con un *hash* in formato UNIX, ed è pertanto impossibile eseguire un confronto diretto con il contenuto di **/etc/passwd**; la cosa diventa possibile solo quando la password viene inviata in chiaro, nel qual caso è il server ad operare il confronto. Torneremo sull'autenticazione degli utenti in sez. 8.5.2.

La sezione **[printers]** ha lo stesso scopo della sezione **[homes]**, ma vale per le stampanti, se non viene trovata una sezione esplicita corrispondente alla stampante cercata, viene esaminato il file **/etc/printcap** per verificare se contiene una stampante con il nome richiesto, ed in caso positivo viene creato al volo il relativo *share*. Viene comunque usata l'autenticazione degli accessi appena esposta.

Per aggiungere uno *share* è pertanto sufficiente definire una nuova sezione; ad esempio si potrà inserire in coda al file mostrato in precedenza una ulteriore sezione del tipo:

```
[foo]
path = /home/bar
browseable = yes
read only = no
```

in questo modo gli utenti potranno accedere ad uno *share* denominato **foo** corrispondente alla directory **/home/bar**. L'accesso richiederà un username ed una password, a meno che non si sia indicata una ulteriore riga del tipo **guest ok = yes**, nel qual caso sarà consentito l'accesso senza password, con l'utente specificato da **guest user**. L'uso della direttiva **browseable = yes** permette di far apparire il nuovo share nella lista pubblica mostrata dal *Network Neighborhood* di Windows.

In ogni caso gli effettivi permessi disponibili su **/home/bar** saranno quelli previsti dal sistema per l'utente con cui ci si è connessi, il server non può in nessun caso garantire maggiori permessi di quelli forniti dal sistema, può invece ridurli, ad esempio impostando **read only = yes**.

Per maggiori informazioni si può consultare il *Samba-HOWTO*, tutti i dettagli della configurazione sono documentati nella pagina di manuale di **smb.conf**. Il pacchetto Samba mette comunque a disposizione alcuni programmi di controllo come **testparms**, che esegue una rapida analisi del file di configurazione per rilevarne eventuali errori. Il comando **smbstatus** inoltre permette di controllare lo stato delle connessioni.

8.5.2 L'impostazione degli utenti

Abbiamo già accennato come uno dei servizi forniti da Samba sia quello della autenticazione degli utenti; questo si sovrappone in maniera spesso non banale con lo stesso servizio fornito dal sistema di autenticazione nativo di Linux, ad esempio abbiamo visto in sez. 8.5.1 riguardo la necessità di inviare in chiaro le password, se si vuole che l'autenticazione degli share creati automaticamente nelle home directory degli utenti sia eseguita contro il file **/etc/passwd**.

Inviare password in chiaro sulla rete non è mai una buona cosa, pertanto è opportuno abilitare sempre l'uso delle password cifrate, questo però comporta che sia il server a dover effettuare da solo l'autenticazione, con un protocollo compatibile con Windows, per questo dovrà essere in grado di mantenere un elenco di utenti e password in maniera indipendente da quello del sistema ospite.

I supporti per l'autenticazione sono vari, il più semplice è l'uso del file **smbpasswd**, che viene abilitato specificando come valore del parametro di configurazione **passwd backend** il valore **smbpasswd**. Questo file contiene gli *hash* in formato Windows delle password di ciascun utente che cerca di collegarsi al server; il formato del file è analogo a quello di **passwd**, una riga per ogni utente con campi separati da ":", il primo campo specifica il nome utente ed il secondo

il suo *uid*, che devono corrispondere agli analoghi in **passwd**, seguono due *hash* crittografici in formato diverso, poi dei flag per l'*account*, identificati da lettere fra parentesi quadra e infine la data di ultima modifica della password. Il formato è descritto in dettaglio nella pagina di manuale accessibile con **man 5 smbpasswd**.

Un secondo supporto possibile è quello di un piccolo database binario, che in macchine con molti utenti permette di velocizzare notevolmente le operazioni di scansione, in tal caso i dati verranno memorizzati nel file **passdb.tdb** e si dovrà indicare per **passdb backend** il valore **tdbsam**. È infine usare come supporto LDAP, nel qual caso il valore da usare è **ldapsam**.

Di norma la creazione del file di supporto per l'autenticazione deve essere eseguita esplicitamente dall'amministratore di sistema, ed i singoli utenti devono essere aggiunti al sistema. Non essendo infatti possibile ricavare la password degli stessi dall'*hash* mantenuto in **/etc/passwd** non esiste una modalità di "replicazione" automatica dei contenuti di quest'ultimo. Pertanto l'amministratore dovrà inserire i singoli utenti a mano con comandi del tipo:

```
smbpasswd -a utente
```

il comando chiederà la nuova password (due volte per evitare errori di battitura) ed aggiungerà l'utente, che deve essere già presente nel sistema, all'interno dell'opportuno supporto. L'opzione **-a** che permette di aggiungere un utente, così come l'opzione **-x** che lo cancella, e **-e** e **-d** che rispettivamente lo abilitano e disabilitano temporaneamente, possono essere usate solo dall'amministratore. Lo stesso vale per **-n** che permette di impostare una password nulla.

L'utente normale può usare lo stesso comando per modificare la sua password, nel qual caso dovrà comunque fornire quella precedente. Al solito l'elenco completo delle opzioni (il comando consente pure di cambiare la propria password su un server remoto e quella sul LDAP) è disponibile nella pagina di manuale, accessibile con **man smbpasswd**.

8.5.3 L'uso di Samba dal lato client

L'uso principale di Samba è permettere la condivisione di file e stampanti su una macchina Unix da parte di client Windows. Da questo punto di vista Samba è completamente trasparente all'utente, che dovrà semplicemente cercare fra le risorse di rete i servizi messi a disposizione dal server Samba.

La suite però comprende anche alcuni programmi per poter utilizzare da GNU/Linux i servizi presenti su un server Windows (o un altro server Samba, anche se in questo caso sarebbe più opportuno utilizzare le applicazioni native Unix).

Il primo programma è **smbclient**, che permette di connettersi ad un server SMB con una interfaccia simile a quella di FTP. La sintassi del comando come riportata dalla pagina di manuale, è:

```
smbclient servicename [ password ] [ -b <buffer size> ] [ -d debuglevel ]  
  [ -D Directory ] [ -U username ] [ -W workgroup ] [ -M <netbios name> ]  
  [ -m maxprotocol ] [ -A authfile ] [ -N ] [ -l logfile ]  
  [ -L <net- bios name> ] [ -I destinationIP ] [ -E ] [ -c <command string> ]  
  [ -i scope ] [ -O <socket options> ] [ -p port ] [ -R <name resolve order> ]  
  [ -s <smb config file> ] [ -T<c|x>IXFqgbNan ]
```

dove in generale **servicename** è specificato nella forma **//server/service** e la password può essere omessa nel qual caso sarà richiesta all'esecuzione del comando.

L'opzione principale è **-U** che permette di specificare il nome utente (relativo al server) con il quale ci si vuole connettere al servizio. Per i dettagli relativi alle altre opzioni si può fare riferimento alla pagina di manuale, accessibile con **man smbclient**, che riporta anche vari esempi.

Un altro comando importante è **smbmount**, che permette di montare uno share di Windows all'interno del filesystem di Linux. Richiede il supporto nel kernel del filesystem SMB (di solito presente in tutte le distribuzioni standard, e comunque attivabile nella sezione **Network filesystem** dei menù di configurazione). La sintassi del comando è:

```
smbmount service mount-point [ -o options ]
```

ma lo si può invocare indirettamente attraverso **mount** specificando l'opzione **-t smbfs**. Un analogo comando **smbumount** permette di smontare un filesystem **smbfs**.

La sintassi è molto semplice, **service** si specifica nella stessa forma **//server/service** usate per **smbclient**, mentre il *mount point* è una qualunque directory locale. Le opzioni sono sempre nella forma **keyword=valore**, l'opzione più importante è **username** che permette di specificare, nella forma **user/work-group%password**, utente, workgroup e relativa password con la quale accedere allo share di Windows. Per i dettagli si può consultare la pagina di manuale accessibile con **man smbmount**.

Capitolo 9

Il servizio DNS

9.1 Il funzionamento del servizio DNS

In questa prima sezione ci occuperemo di introdurre i meccanismi di funzionamento del servizio DNS, descrivendone le caratteristiche generali ed una serie di programmi diagnostici usati appunto per eseguire interrogazioni su questo servizio.

9.1.1 Introduzione

Il DNS è uno dei protocolli fondamentali per il funzionamento di internet, e come già accennato in sez. 7.4 si tratta in realtà di un enorme database, distribuito su un gran numero di *nameserver* (si chiamano così i server che rispondono alle richieste del protocollo).

Alle origini, quando internet era piccola e le macchine erano poche, la risoluzione dei nomi era fatta scaricando su ogni macchina un singolo file che conteneva tutte le corrispondenze fra numeri IP ed nomi. Col crescere della rete questo approccio è rapidamente divenuto insostenibile, non tanto e non solo per le dimensioni crescenti del file, quanto per la quasi impossibilità di mantenerlo aggiornato. Per questo è stato creato il protocollo del DNS, il cui scopo era quello di poter distribuire il compito di associare un indirizzo simbolico ad uno numerico, e di eseguirlo in maniera veloce ed efficiente.

Il funzionamento del DNS è sostanzialmente basato su un meccanismo chiamato *delegazione*: si sfrutta la suddivisione gerarchica dello spazio dei nomi, in domini di primo livello (i `.com`, `.org`, `.it`, ecc.) di secondo livello (`google.com`, `softwarelibero.org`, `truelite.it`) ecc. per distribuire il carico delle richieste di risoluzione, passandole attraverso una gerarchia di server, in cui, in corrispondenza a ciascun livello, un server che non è in grado di rispondere alla richiesta, sa però qual'è il server del livello successivo a cui reinviare la richiesta.

Il meccanismo con il quale viene eseguita la risoluzione di un nome a dominio, ad esempio `sources.truelite.it`, è il seguente: quando ci si rivolge ad un *nameserver* (ad esempio quello del provider) questo controlla anzitutto se ha in cache la risposta, nel qual caso risponde immediatamente, altrimenti va a cercare, sempre nella cache, se ha l'indirizzo di uno dei server che gli può rispondere, salendo lungo la gerarchia dei domini. Nella peggiore delle ipotesi, in cui non sa a chi chiedere né per `.truelite.it` né per `.it` il server si dovrà rivolgere a quelli che sono chiamati i *root DNS*, che permettono di risolvere i domini di primo livello. La lista degli indirizzi IP di questi server è pubblicata ed aggiornata periodicamente ed ogni server DNS deve sempre essere in grado di contattarli.

La ricerca avviene quindi ricorsivamente, con una scansione dell'albero dei domini: alla radice si contatterà un *root DNS* chiedendogli chi è il server responsabile per il dominio di primo livello `.it`. I domini di primo livello sono definiti a livello internazionale dalla cosiddetta *naming authority*, e la lista dei relativi nameserver è mantenuta direttamente nei *root DNS*. A

questo punto la scansione proseguirà ripetendo la richiesta per `.truelite.it` al DNS di primo livello appena trovato; essendo compito di questo server conoscere tutti quelli di secondo livello, sarà in grado di indicarvi qual'è il server DNS responsabile per `.truelite.it` a cui chiedere la risoluzione di `sources.truelite.it`. In tutti questi passaggi il server DNS che avete interrogato memorizzerà le varie informazioni nella sua cache, in modo da evitare una ulteriore richiesta ai *root DNS* se ad esempio volete risolvere `www.softwarelibero.it`.

Questo meccanismo permette di distribuire in maniera efficace il compito di fornire le risposte e di mantenere aggiornata la corrispondenza fra nomi ed indirizzi IP, in quanto alla fine è il responsabile di ciascun dominio finale a dover mantenere un DNS che contenga le informazioni relative alle sue macchine. Inoltre il meccanismo del caching dei risultati permette di aumentare l'efficienza della ricerca, evitando di ripetere più volte le stesse operazioni. Ovviamente questo ha anche un costo, in quanto se viene eseguito un cambiamento in una associazione nome-indirizzo questo non verrà visto dagli altri server fintanto che c'è un'altra associazione valida nella loro cache. Per questo tutte le informazioni del DNS sono corredate da un tempo di scadenza, da impostare opportunamente a seconda delle frequenze con cui esso può cambiare, che permettono, con tempi più o meno rapidi a seconda di quanto impostato, di propagare i cambiamenti su tutta la rete.

Il concetto fondamentale del protocollo del DNS è quello delle cosiddette *zone di autorità*, cioè delle parti dello spazio dei nomi di dominio per i quali un singolo nameserver ha una risposta diretta. In generale i server di livello più alto non conoscono, a meno che questa non sia nella loro cache, la risposta a richieste come `sources.truelite.it`, perché *delegano* l'autorità per le varie zone di cui è composto il dominio ad altri server, che a loro volta possono effettuare ulteriori delegazioni per quanto loro assegnato. Così i *root DNS* delegano l'autorità per i domini di primo livello ai relativi nameserver, e questi a loro volta a quelli di secondo livello e così via. Questo meccanismo introduce anche una distinzione fra le risposte, che sono dette *autoritative* o meno, a seconda che provengano direttamente dal nameserver che ha l'autorità per quella richiesta o dalla cache di un qualche altro server.

Tipo	Descrizione
A	una corrispondenza nome – indirizzo IP
NS	un nameserver per la zona
CNAME	nome alternativo (un alias ad un altro nome)
SOA	inizio zona per il quale si ha autorità
PTR	una corrispondenza indirizzo – nome
MX	un server di posta
TXT	un commento o altro testo
AAAA	una corrispondenza nome – indirizzo IPv6
SRV	la locazione di un servizio noto

Tabella 9.1: Descrizione ed identificativo per alcuni tipi di record usati dal protocollo DNS.

Si tenga presente inoltre che il protocollo DNS permette di inserire in questo database distribuito vari tipi di informazione, come il server che riceve la posta inviata ad un dominio, le informazioni che dicono se esistono altri nameserver per domini di livello inferiore, corrispondenze con indirizzi diversi da quelli IP (ad esempio indirizzi IPv6), ecc. Per questo motivo esistono vari tipi di *record* che possono essere gestiti inseriti in un DNS; un elenco dei principali *tipi* è riportato in tab. 9.1, un elenco completo si può trovare nella pagina di manuale del comando `host`.

9.1.2 I comandi `host` e `dig`

Il comando `host` permette di interrogare un nameserver. La sintassi del comando, così come è riportata nella pagina di manuale è la seguente:

```

host [-v] [-a] [-t querytype] [options] name [server]
host [-v] [-a] [-t querytype] [options] -l zone [server]
host [-v] [options] -H [-D] [-E] [-G] zone
host [-v] [options] -C zone
host [-v] [options] -A host

host [options] -x [name ...]
host [options] -X server [name ...]

```

e come si può notare il comando è in grado di effettuare diversi tipi di interrogazione.

In genere l'uso più comune del comando è per verificare che l'impostazione della configurazione del resolver funzioni davvero. Accade spesso infatti che la rete funzioni, ma non si riesca a fare nulla perché gli indirizzi simbolici non vengono risolti (ad esempio si è sbagliato ad indicare il nameserver in `resolv.conf`); un esempio del comando è:

```

[piccardi@havnor piccardi]$ host www.linux.it
www.linux.it      CNAME  picard.linux.it
picard.linux.it   A       62.177.1.107

```

che ci dice che l'indirizzo `www.linux.it` è risolto come corrispondente all'IP `62.177.1.107`. L'output del comando mostra anche il tipo di record restituito, nel caso un record di tipo `CNAME`, relativo al nome richiesto, che ci reinvia al nome principale della macchina, ed un record di tipo `A` per quest'ultimo che lo associa al relativo indirizzo. Il comando funziona anche alla rovescia, si può cioè trovare l'indirizzo simbolico a partire da quello numerico con:

```

[piccardi@havnor piccardi]$ host 62.177.1.107
Name: picard.linux.it
Address: 62.177.1.107

```

e si noti come in questo caso il nome riportato sia `picard.linux.it`, che è diverso da `www.linux.it`, in quanto riporta una macchina specifica all'interno del dominio. In realtà si possono avere anche più nomi associati allo stesso indirizzo, ad esempio se cerchiamo:

```

[piccardi@havnor piccardi]$ host www.firenze.linux.it
www.firenze.linux.it  CNAME  serverone.firenze.linux.it
serverone.firenze.linux.it  A       195.110.124.18

```

vediamo che l'indirizzo `www.firenze.linux.it` è un *alias* (infatti il tipo di record è `CNAME`) al precedente. Si tenga presente che ad un solo IP possono essere associati diversi nomi relativi a domini completamente scorrelati; ad esempio anche interrogando con:

```

[piccardi@havnor piccardi]$ host www.softwarelibero.it
www.softwarelibero.it  A       195.110.124.18

```

si ottiene di nuovo lo stesso numero, dato che la macchina che ospita il sito dell'Associazione Software Libero è la stessa.

Il comando `host` permette di richiedere esplicitamente anche gli altri tipi di record usando l'opzione `-t querytype` per indicare il tipo di record voluto, in cui per `querytype` si possono usare i tipi mostrati in tab. 9.1. Ad esempio se vogliamo vedere chi riceve la posta nel dominio `firenze.linux.it` possiamo usare:

```

[piccardi@havnor piccardi]$ host -t MX firenze.linux.it
firenze.linux.it      MX       5 mail.firenze.linux.it
firenze.linux.it      MX       666 lorien.prato.linux.it

```

che ci risponde restituendo i record **MX**, che indicano i server di posta del dominio, usando **ANY** come tipo (o direttamente l'opzione **-a**) si avranno tutti i record di quel dominio. Oltre ai domini il programma è in grado di elencare tutti i record relativi ad una determinata zona, i dettagli sono riportati nella pagina di manuale.

Il comando **dig** ha sostanzialmente le stesse funzionalità di **host**, la sintassi, come riportata dalla pagina di manuale è:

```
dig @server name type
```

dove **server** indica il server DNS da interrogare **name** il nome della risorsa che si vuole cercare e **type** il tipo di record da cercare (al solito con i valori di tab. 9.1). Se non si fornisce nessun argomento il comando sottintende che i server sono quelli specificati in **resolv.conf**, il tipo è **A** ed il nome è **.**, per cui restituisce l'elenco dei **root** DNS.

La differenza con **host** è che il comando restituisce l'output completo inviato dal nameserver, e non solo il campo richiesto, inoltre **dig** può essere usato in *batch* (cioè in forma non interattiva) con l'opzione **-f file** per permette di effettuare le ricerche leggendo i comandi da un file.

9.2 La gestione di un server DNS

In questa sezione prenderemo in esame la gestione di un server DNS, concentrandoci in particolare su una implementazione particolare, quella di *bind*, che detiene comunque la quasi totalità delle installazioni dei server DNS su tutta internet.

9.2.1 Il server named

Essendo un protocollo implementato a livello di applicazione, il servizio del DNS viene fornito attraverso un opportuno demone. A livello internazionale (si ricordi quanto detto in sez. 7.2.4) è stato assegnato al protocollo la porta 53 (sia UDP che TCP) per rispondere a delle interrogazioni. Il server DNS più diffuso è il programma **named**. Il programma è parte del pacchetto **bind**, scritto originariamente da Paul Vixie, e mantenuto dall'Internet Software Consortium, è attualmente giunto alla versione 9.

L'installazione del pacchetto **bind** è prevista da tutte le maggiori distribuzioni. Il programma viene lanciato dagli opportuni script di avvio, riportati in tab. 9.2, che possono essere utilizzati per avviare e fermare il servizio con i soliti parametri standard. Di norma viene anche automaticamente inserito in tutti i runlevel e lanciato all'avvio dal sistema di inizializzazione di System V.

Distribuzione	Comando
Debian	/etc/init.d/bind
RedHat	/etc/rc.d/init.d/named
Slackware	/etc/rc.d/rc.inet2

Tabella 9.2: Script di avvio del server DNS nelle varie distribuzioni.

Nei casi in cui, per effettuare delle prove, si vuole lanciare a mano il programma, la sintassi, come riportata dalla pagina di manuale, è la seguente:

```
named [-d debuglevel] [-p port#] [-(b|c) config_file] [-f -q -r -v]
      [-u user_name] [-g group_name] [-t directory] [-w directory]
      [config_file]
```

per una descrizione delle opzioni si può consultare la stessa pagina di manuale, di norma è sufficiente lanciare il programma senza opzioni, nel qual caso saranno usate le configurazioni standard, se si vuole lasciare il programma attivo, si può usare l'opzione **-f** che evita che vada

in background, mentre con `-d` si può impostare il livello di debug; usandole entrambe si avrà tutta la diagnostica a schermo invece che in un file di log.

Di norma il programma viene lanciato usando l'opzione `-u`, che permette di usare un utente dedicato, senza privilegi speciale, dopo aver effettuato l'inizializzazione dello server¹ per le normali operazioni del DNS. L'utente dipende dalla distribuzione e dall'installazione che si è effettuata.

9.2.2 Il file `named.conf`

Il file di configurazione di `named` è `named.conf`, di solito si trova questo file in `/etc/named` o in `/etc/bind/` (se non direttamente sotto `/etc`) a seconda delle distribuzioni. Il file ha una sintassi abbastanza complessa e simile a quella dei programmi in C, il file contiene una serie di comandi, che a loro volta possono contenere blocchi di ulteriori sottocomandi racchiusi fra parentesi graffe; ogni comando è terminato da un `;` che ne indica la conclusione. Anche i commenti possono essere introdotti dalla usuale `#`, ma viene utilizzata anche la sintassi del C e del C++ con blocchi delimitati da `/* */` o inizianti per `//`.

Una lista dei principali comandi che si possono utilizzare all'interno di `named.conf` e del loro significato generico è la seguente:

- include** include il contenuto di un altro file di configurazione, come se questo fosse stato scritto direttamente dentro `named.conf`. In questo modo diventa possibile dividere le varie configurazioni (ad esempio le zone per diversi domini, o altre configurazioni) e mantenere separatamente le varie parti.
- options** permette di impostare alcune proprietà generali del server ed i default per le altre direttive. Conviene inserirla in un file a parte (ad esempio in Debian è mantenuta in `/etc/bind/named.conf.options`) da includere con `include`.
- logging** definisce le modalità con cui le varie informazioni vengono inviate sui file di log o al sistema del syslog.
- zone** viene usata per definire una *zona* del DNS e le modalità con cui vengono utilizzate le relative informazioni. Una *zona* serve ad identificare una parte di nomi di dominio per il quale il server deve eseguire delle azioni specifiche.

la lista completa e i vari dettagli relativi a ciascuna direttiva possono essere trovati nella pagina di manuale accessibile con `man named.conf`; si tenga presente che le due direttive `logging` e `options` possono comparire soltanto una volta.

Un esempio dell'inizio del file `named.conf`, ripreso dal file installato dal pacchetto `bind` su una distribuzione Debian Sid, è riportato di seguito:

```
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind/README.Debian for information on the
// structure of BIND configuration files in Debian for BIND versions 8.2.1
// and later, *BEFORE* you customize this configuration file.
//

include "/etc/bind/named.conf.options";

// reduce log verbosity on issues outside our control
logging {
    category lame-servers { null; };
}
```

¹dovendo porsi in ascolto sulla porta 53, che è una delle porte riservate, è necessario avere i privilegi di amministratore, che nel corso delle ulteriori operazioni non sono più necessari.

```

        category cname { null; };
};
...

```

si noti come in questo caso viene incluso un file a parte che contiene l'impostazione delle opzioni con il comando **options**, e poi viene usato il comando **logging** per bloccare la scrittura nei log di alcuni tipi di messaggi. Vedremo il resto del file in seguito, in relazione a diverse configurazioni.

9.2.3 La configurazione base

Le modalità di utilizzo di un server DNS possono essere le più svariate, affronteremo qui solo quelle relative al suo uso all'interno di una rete locale. La modalità più semplice è quella di un *caching DNS*, cioè di usare il server come un *proxy* per evitare di ripetere ogni volta le richieste di risoluzione dei nomi su internet.

Anzitutto andranno impostate le opzioni generali per il server, questo viene fatto dal comando **options**, si avrà pertanto una sezione del tipo di:

```

options {
    directory "/var/cache/bind";
    ...
};

```

che specifica la directory rispetto nella quale il server cerca i vari file; anche questa può variare a seconda della distribuzione.

Il primo passo da fare è quello di far sapere al server dove può trovare i *root DNS*; questo viene fatto inserendo in **named.conf** una zona apposita, usando una sezione del tipo di:

```

zone "." {
    type hint;
    file "/etc/bind/db.root";
};

```

in questo caso il comando definisce una zona per la radice (che nel sistema dei nomi di dominio è indicata da un **"."**). I file preimpostati nell'installazione standard già prevedono una sezione di questo tipo.

Le zone possono essere di vari tipi, come specificato dal sotto comando **type**, in questo caso si ha una zona di tipo **hint** che dice semplicemente a chi richiedere le informazioni relative a quella zona. La direttiva **file** specifica il file da cui prendere le informazioni, questo può essere indicato in forma assoluta, come nell'esempio, o in forma relativa rispetto alla directory definita dalla direttiva **directory** del comando **options**. In questo caso il file è **/etc/bind/db.root** che contiene l'elenco dei *root DNS*, questo può essere ottenuto con il comando **dig** (vedi sez. 9.1.2) o scaricato direttamente all'indirizzo **ftp://ftp.internic.net/domain/named.root**.

La sintassi generale del comando **zone** prevede che esso sia seguito, fra **"**, dal nome della zona a cui si fa riferimento e da un ulteriore blocco di direttive che specificano le caratteristiche della zona. In generale, oltre alla zona per i *root DNS*, quando si installa il pacchetto di **bind**, i file di configurazione sono preimpostati per la risoluzione del **localhost**. Vedremo i dettagli al riguardo in sez. 9.2.4, quando tratteremo la configurazione del nameserver per la risoluzione di un dominio locale.

Di norma l'installazione standard prevede da sola tutto quello che serve per un *caching DNS*. A questo punto si può provare a verificarne il funzionamento, se da un terminale si esegue una richiesta ad un indirizzo fino ad allora non usato, avremo che:


```
[piccardi@gont piccardi]$ dig www.consumattori.org

; <<>> DiG 9.2.2 <<>> www.consumattori.org
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32914
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 0

;; QUESTION SECTION:
;www.consumattori.org.          IN      A

;; ANSWER SECTION:
www.consumattori.org.  3600    IN      A      80.241.162.128

;; AUTHORITY SECTION:
consumattori.org.      900     IN      NS      ns7.gandi.net.
consumattori.org.      900     IN      NS      custom2.gandi.net.

;; Query time: 698 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Tue Mar 25 21:42:25 2003
;; MSG SIZE rcvd: 103
```

e come si vede la richiesta viene soddisfatta in 698 msec, ma se subito dopo si ripete la richiesta otterremo:

```
[piccardi@gont piccardi]$ dig www.consumattori.org

; <<>> DiG 9.2.2 <<>> www.consumattori.org
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56592
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 0

;; QUESTION SECTION:
;www.consumattori.org.          IN      A

;; ANSWER SECTION:
www.consumattori.org.  3595    IN      A      80.241.162.128

;; AUTHORITY SECTION:
consumattori.org.      895     IN      NS      ns7.gandi.net.
consumattori.org.      895     IN      NS      custom2.gandi.net.

;; Query time: 3 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Tue Mar 25 21:42:30 2003
;; MSG SIZE rcvd: 103
```

con i tempi che si riducono drasticamente a 3 msec.

Questo ci mostra che il nostro caching nameserver sta funzionando. Con questa impostazione però è il nostro nameserver che si incarica di effettuare il procedimento di scansione ricorsiva che abbiamo illustrato in sez. 9.1.1, possiamo limitare ulteriormente le richieste che escono dalla nostra rete usando il nameserver del provider per effettuare la scansione per conto nostro, questo si fa aggiungendo alla sezione `options` le specifiche per l'uso di altri server come *forwarders*, aggiungendo una sezione del tipo:

```
options {
    ...
```

```

        forwarders {
            213.234.128.211;
            213.234.132.130;
        };
};

```

ed in questo modo uscirà solo una richiesta verso i server che sono dichiarati nella direttiva `forwarders`.

9.2.4 La configurazione di un dominio locale.

Vediamo ora come configurare un dominio locale, per la risoluzione dei nomi delle macchine interne di una LAN; per semplicità prenderemo un dominio completamente astratto, `earthsea.ea`. Per far questo dovremo definire la relativa zona, dovremo aggiungere quindi al file di configurazione le due sezioni:

```

//
// Add local zone definitions here.
zone "earthsea.ea" {
    type master;
    file "/etc/bind/db.earthsea";
};
zone "1.168.192.in-addr.arpa" {
    type master;
    file "/etc/bind/db.192.168.1";
};

```

che per una migliore organizzazione può essere sensato mantenere in un file a parte, da includere dal file principale con il solito comando `include`. Si noti come sempre si debbano definire due zone, entrambe di tipo `master`, una per la risoluzione diretta, relativa al nostro dominio, ed una per la risoluzione inversa degli indirizzi privati che stiamo usando (nel nostro caso si suppone di aver usato la rete `192.168.1.0`).

Per la risoluzione inversa si usa sempre il nome di dominio speciale `in-addr.arpa`, riservato a questo scopo, seguito dal numero IP del quale si vuole effettuare la risoluzione, espresso in forma *dotted decimal* in ordine rovesciato. Esso infatti viene interpretato comunque come un nome di dominio separato da punti, per cui il livello più generale di risoluzione è dato dalla parte più significativa del numero.

Inoltre nella installazione standard è prevista la configurazione per la risoluzione in locale del `localhost`, il che di norma comporta la presenza dentro il `named.conf` di default di due zone del tipo:

```

zone "localhost" {
    type master;
    file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
    type master;
    file "/etc/bind/db.127";
};

```

anche in questo caso il tipo di zona è di tipo `master` in quanto è sempre il nostro server ad essere il responsabile della risoluzione di questo nome.

Vediamo allora quale è il formato dei file di configurazione delle singole zone, che contengono i dati, detti *resource record* (in breve RR) che il server deve fornire quando interrogato. Cominciamo col mostrare quello usato per il `localhost`, che illustra il contenuto generico di uno di questi file:

```
$TTL      604800
@         IN      SOA      localhost. root.localhost. (
                        1      ; Serial
                        604800 ; Refresh
                        86400  ; Retry
                        2419200 ; Expire
                        604800 ) ; Negative Cache TTL
;
@         IN      NS       localhost.
@         IN      A        127.0.0.1
```

I file possono contenere alcune direttive generali, scritte all'inizio, che impostano dei valori di default. Normalmente viene usata solo `$TTL`, che permette di impostare, per tutti i record seguenti, il cosiddetto *time to live*, cioè il tempo massimo di validità del record nella cache di un nameserver (si ricordi quanto detto al proposito in sez. 9.1.1); nel nostro esempio il tempo è stato specificato in secondi, e corrisponde ad una settimana; formati alternativi prevedono l'uso dei suffissi `H`, `D`, `W` per indicare rispettivamente ore, giorni e settimane.

Alle direttive seguono poi i vari record, in genere uno per riga, formati da campi separati da spazi o tabulazioni. Il primo campo deve iniziare dal primo carattere della riga, altrimenti viene considerato **blank**. Se necessario si possono specificare i campi su più righe, racchiudendoli fra parentesi tonde. Infine si possono inserire commenti ovunque precedendoli con il carattere `;`. Il formato dei vari record è simile, la sintassi generale è la seguente:

```
{<domain>|@|<blank>} [<ttl>] [<class>] <type> <rdata> [<comment>]
```

Il primo campo indica il nome di dominio che viene ricercato; esso può essere immesso esplicitamente, ma se il campo è vuoto (cioè **blank**) viene usato l'ultimo nome immesso; infine si può usare il carattere speciale `"@"`, che viene automaticamente espanso nel nome della zona indicato dal comando **zone** che fa riferimento al file, che viene detta *origine*. Il campo `ttl` viene di norma omesso, in quanto di solito si usa la direttiva `$TTL` per specificarlo una volta per tutte. Il campo `class` indica la classe dei dati del record; nel nostro caso usando indirizzi internet sarà sempre `IN` per tutti i vari tipi di record, esistono altre classi ma non sono usate. Segue il campo `type` che deve essere sempre specificato e definisce il tipo di record, infine segue il campo (o i campi) `rdata` che contiene i dati da inviare in risposta alle richieste relative al record.

L'ordine in cui si dichiarano i record non è essenziale, ma è convenzione specificare sempre per primo un **SOA** che dichiara che il nostro server è *autoritativo* per quel dominio (di nuovo si ricordi quanto detto in sez. 9.1.1), dato che non ce ne può essere più di uno per file. Questo significa anche la necessità di un file separato per ogni dominio che si vuole definire. Si prosegue poi con la dichiarazione dei record di tipo **NS** che definiscono i nameserver per il dominio in questione, a cui fanno seguito gli altri tipi di record.

La prima colonna di un record **SOA** è il nome del dominio per il quale si dichiara di avere autorità, nel caso in esempio si è usato il carattere speciale `@`, altrimenti si sarebbe dovuto specificare il nome del dominio completo nella forma `localhost.`, con il `"."` finale in quanto i nomi vanno dichiarati in forma assoluta (facendo riferimento alla radice che è appunto `"."`) segue la classe `IN` ed il tipo **SOA**. Nel caso di un record **SOA** i dati prevedono il server che farà da nameserver principale per il dominio, la e-mail del responsabile, nella forma solita ma con un `"."` al posto della `"@"` (nel caso la posta andrà a `root@localhost`).

Le restanti informazioni, inserite fra parentesi per poterle dividere su più righe, sono usate nella replicazione dei dati fra nameserver primari e secondari: una funzionalità avanzata del

protocollo che consente di mantenere delle repliche automatizzate dei dati di un nameserver, aggiornate secondo quanto stabilito da questi parametri, su un server di riserva (detto appunto secondario) che entra in azione solo quando il primario fallisce. Di questa l'unica da modificare è quella identificata come **serial** che indica un numero seriale (crescente) da aggiornare tutte le volte che si effettua una modifica alla zona.

Tornando all'esempio, i due record successivi dichiarano rispettivamente (il record **NS**) la macchina nel dominio che fa da nameserver (nel caso quella identificata dal nome del dominio stesso) e l'indirizzo del dominio (il record **A**).

Vista la particolarità del dominio **localhost** vediamo un esempio più significativo, su come impostare un nostro dominio locale per risolvere i nomi della macchine di una rete locale. Nel caso l'esempio è tratto dalla mia configurazione della rete di casa. Si è scelto il dominio **earthsea.ea**, i cui dati sono mantenuti nel file **db.earthsea** il cui contenuto è:

```
$TTL 100000
@ IN SOA gont.earthsea.ea. piccardi.gont.earthsea.ea. (
    2      ; serial
    10800  ; refresh after 3 hour
    3600   ; retry after 1 hour
    604800 ; expire after 1 week
    86400  ) ; minimum TTL 1 day

;
; Name server
;
@           IN      NS      gont.earthsea.ea.
;
; Mail server
;
;
;
;
;
;
; Indirizzi
;
gont        IN      A       192.168.1.1
ns          IN      CNAME   gont
www         IN      CNAME   gont
hogen       IN      A       192.168.1.2
lorbaner    IN      A       192.168.1.17
karegoat    IN      A       192.168.1.19
oppish      IN      A       192.168.1.168
localhost   IN      A       127.0.0.1
```

Di nuovo si è cominciato con un record di tipo **SOA** che dichiara l'autorità per **earthsea.ea**, in questo caso il nameserver è la macchina **gont.earthsea.ea.**, anch'essa identificata dal suo nome di dominio assoluto. Si tenga conto che se non si usa il nome assoluto al nome viene sempre aggiunta l'*origine* della zona, per cui o si usa questa notazione o si specifica il nameserver semplicemente con **gont**. L'amministratore della macchina sono io per cui la posta sarà inviata a me con l'indirizzo **piccardi@gont.earthsea.ea**.

Il secondo record definisce il nameserver, cioè **gont.earthsea.ea.**: si noti che di nuovo, come per il precedente, non lo si è definito con l'indirizzo IP ad esso corrispondente; tutto quello che occorre infatti è usare un nome che corrisponda ad un record di tipo **A**. Nel caso si è specificato un solo nameserver, ma se ne possono definire più di uno; l'informazione sarà passata nelle richieste di risoluzione, che potranno essere rivolte ad uno qualunque dei server della lista.

Il record successivo, di tipo **MX**, serve a definire a quale macchina viene inviata la posta per il dominio **earthsea.ea**, di nuovo si è specificato **gont**, ma stavolta, per dare un esempio di come si possono scrivere le cose in maniera più compatta, non si è dichiarato il dominio (che essendo **blank** corrisponde a quello precedente, e cioè sempre **earthsea.ea**, si è tralasciata la classe **IN** che essendo quella di default può non essere specificata esplicitamente, e si è indicato **gont** senza

usare il nome assoluto. Il valore inserito 10 inserito prima del nome indica la priorità del server di posta, è possibile infatti specificare più di un server di posta, con valori diversi, in modo che verrà sempre contattato per primo quello con il valore più basso, passando ai successivi solo in caso di fallimento dei precedenti. Si tenga presente che anche questi record è opportuno facciano riferimento a nomi associati a record di tipo A.

Infine si sono inseriti i vari record di tipo A contenenti gli indirizzi, ed i record di tipo CNAME che definiscono dei nomi alternativi per le stesse macchine. Per quanto visto finora il formato di questi record è evidente.

Una volta definito il nostro dominio occorre anche definire la risoluzione inversa, questo è fatto tramite il file `db.192.168.1`, il cui contenuto è il seguente:

```
$TTL 100000
@ IN SOA gont.earthsea.ea. piccardi.gont.earthsea.ea. (
    2      ; serial
    10800  ; refresh after 3 hour
    3600   ; retry after 1 hour
    604800 ; expire after 1 week
    86400  ) ; minimum TTL 1 day
;
; Name server
;
@      IN      NS      gont.earthsea.ea.
;
; Indirizzi
;
1      IN      PTR      gont.earthsea.ea.
2      IN      PTR      hogen.earthsea.ea.
17     IN      PTR      karegoat.earthsea.ea.
19     IN      PTR      lorbaner.earthsea.ea.
168    IN      PTR      oppish.earthsea.ea.
```

Di nuovo si comincia con un record di tipo SOA, sostanzialmente identico al precedente dato che nameserver e amministratore sono gli stessi. Si deve definire anche il nameserver per questo dominio inverso, che ovviamente è sempre `gont.earthsea.ea.` (qui specificare `gont` non avrebbe funzionato, l'indirizzo assoluto invece funziona in quanto il dominio è stato già definito prima).

Seguono infine i record di tipo PTR che contengono il nome delle singole macchine, in questo caso però gli indirizzi devono corrispondere ad un solo nome, quello canonico e contrariamente a prima non si possono associare più nomi allo stesso indirizzo. Anche se è possibile associare due nomi allo stesso IP molti sistemi non sono preparati a questa evenienza, che è bene pertanto escludere onde evitare comportamenti inattesi.

Completati i nostri file possiamo riavviare il servizio, e controllare i risultati. Al solito usiamo `dig` per interrogare il nostro server:

```
[piccardi@gont piccardi]$ dig any earthsea.ea

; <<>> DiG 9.2.2 <<>> any earthsea.ea
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56071
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;earthsea.ea.                IN      ANY

;; ANSWER SECTION:
earthsea.ea.                100000 IN      NS      gont.earthsea.ea.
earthsea.ea.                100000 IN      SOA     gont.earthsea.ea. piccardi.gont
.earthsea.ea. 2 10800 3600 604800 86400

;; AUTHORITY SECTION:
earthsea.ea.                100000 IN      NS      gont.earthsea.ea.

;; ADDITIONAL SECTION:
gont.earthsea.ea.          100000 IN      A       192.168.1.1

;; Query time: 16 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Wed Mar 26 21:41:06 2003
;; MSG SIZE rcvd: 123
```

ed otteniamo il risultato voluto, adesso possiamo provare a risolvere un'altra macchina:

```
[piccardi@gont piccardi]$ dig lorbaner.earthsea.ea

; <<>> DiG 9.2.2 <<>> lorbaner.earthsea.ea
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24592
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;lorbaner.earthsea.ea.      IN      A

;; ANSWER SECTION:
lorbaner.earthsea.ea.      100000 IN      A       192.168.1.17

;; AUTHORITY SECTION:
earthsea.ea.                100000 IN      NS      gont.earthsea.ea.

;; ADDITIONAL SECTION:
gont.earthsea.ea.          100000 IN      A       192.168.1.1

;; Query time: 2 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Wed Mar 26 21:42:36 2003
;; MSG SIZE rcvd: 89
```

ed infine proviamo con la risoluzione inversa, ricordiamoci che in questo caso **dig** vuole l'opzione **-x**, ed otterremo:

```
[piccardi@gont anime]$ dig -x 192.168.1.17

; <<>> DiG 9.2.2 <<>> -x 192.168.1.17
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30928
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;17.1.168.192.in-addr.arpa.      IN      PTR

;; ANSWER SECTION:
17.1.168.192.in-addr.arpa. 100000 IN      PTR      karegoat.earthsea.ea.

;; AUTHORITY SECTION:
1.168.192.in-addr.arpa. 100000 IN      NS      gont.earthsea.ea.

;; ADDITIONAL SECTION:
gont.earthsea.ea.      100000 IN      A      192.168.1.1

;; Query time: 2 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Wed Mar 26 21:52:09 2003
;; MSG SIZE rcvd: 112
```

e di nuovo è tutto a posto.

9.3 Configurazioni avanzate

Tratteremo in questa sezione le configurazioni più avanzate di BIND, come l'impostazione di eventuali secondari, la gestione delle delegazioni, ed le varie tipologie di controllo degli accessi disponibili nel programma. Chiuderemo la sezione con un breve accenno alla configurazione della versione precedente (bind 4) del demone.

9.3.1 La delegazione di una zona

Come accennato una delle caratteristiche fondamentali del funzionamento del DNS è la capacità di *delegare* la risoluzione di parti dell'albero dei nomi ad altri server, in modo da distribuire sia il carico di rete che quello amministrativo.

9.3.2 La gestione di un secondario

Una delle caratteristiche del DNS è quella di consentire in maniera naturale la distribuzione di carico e la ridondanza attraverso la possibilità di gestire dei DNS secondari cui trasferire in maniera automatica tutte le informazioni di una certa zona.

Appendice A

Sinossi dei comandi principali

A.1 Comandi per la gestione dei file

Comando	Significato e sintassi
<code>ls</code>	Mostra il contenuto di una directory. <code>ls [OPTION] ... [FILE] ...</code>
<code>mv</code>	Sposta e rinomina i file. <code>mv [OPTION] ... SOURCE DEST</code> <code>mv [OPTION] ... SOURCE... DIRECTORY</code>
<code>cp</code>	Copia i file. <code>cp [OPTION] ... SOURCE DEST</code> <code>cp [OPTION] ... SOURCE... DIRECTORY</code>
<code>rm</code>	Cancella file e directory. <code>rm [OPTION] ... [FILE] ...</code>
<code>ln</code>	Crea un link. <code>ln [OPTION] ... TARGET [LINK_NAME]</code> <code>ln [OPTION] ... TARGET... DIRECTORY</code>

Tabella A.1: Principali comandi di base per i file.

Comando	Significato e sintassi
<code>cat</code>	Concatena i file stampandoli sullo standard output. <code>cat [OPTION] [FILE] ...</code>
<code>less</code>	Visualizza i file stampandoli sullo standard output. <code>less [FILE] ...</code>
<code>tail</code>	Visualizza l'ultima parte di un file. <code>tail [OPTION] [FILE] ...</code>
<code>head</code>	Visualizza la prima parte di un file. <code>head [OPTION] [FILE] ...</code>

Tabella A.2: Principali comandi per il contenuto dei file.

A.2 Comandi per la gestione dei processi

Comando	Significato e sintassi
<code>ps</code>	Riporta una fotografia dei processi in corso. <code>ps [OPTIONS]</code>
<code>top</code>	Mostra i processi in corso sul sistema. <code>top [OPTIONS]...</code>
<code>kill</code>	Manda un segnale ad un process. <code>kill [-signal -s signal] pid ...</code>
<code>killall</code>	Uccide i processi per nome. <code>killall [OPTIONS] process</code>
<code>nice</code>	Lancia un programma con priorità modificata. <code>nice [OPTION] [COMMAND [ARG]]...</code>

Tabella A.3: Principali comandi per la gestione dei processi.

A.3 I permessi dei file

Nome	Significato e sintassi
<code>u</code>	Privilegi dell'utente. <code>u</code> sta per <i>user</i>
<code>g</code>	Privilegi del gruppo. <code>g</code> sta per <i>group</i>
<code>o</code>	Privilegi di tutti gli altri. <code>o</code> sta per <i>other</i>
<code>r</code>	Permessi di lettura. <code>r</code> sta per <i>read</i>
<code>w</code>	Permessi di scrittura. <code>w</code> sta per <i>write</i>
<code>x</code>	Permessi di esecuzione. <code>x</code> sta per <i>execute</i>

Tabella A.4: I permessi dei file.

Comando	Significato e sintassi
<code>chmod</code>	Cambia i permessi di accesso ai file. <code>chmod [OPTIONS]</code>
<code>chown</code>	Cambia proprietari e gruppi ai file. <code>chown [OPTIONS]</code>
<code>chgrp</code>	Cambia il gruppo proprietario ai file. <code>chgrp [OPTIONS]</code>

Tabella A.5: Principali comandi per la gestione dei permessi dei file.

A.4 Comandi per la localizzazione dei file

Comando	Significato e sintassi
<code>locate</code>	Elenca i file del database che si adattano col nome cercato. <code>locate [OPTIONS] file</code>
<code>whereis</code>	Cerca i file di tipo binario, sorgente e pagina di manuale per un comando. <code>whereis [OPTIONS] file</code>
<code>which</code>	Individua la posizione di un comando. <code>which [OPTION] file</code>
<code>find</code>	Cerca file in una gerarchia di directory. <code>find [PATH] espressione</code>
<code>updatedb</code>	Aggiorna il database con l'elenco dei file presenti sul sistema. <code>updatedb [OPTIONS]</code>

Tabella A.6: Principali comandi per la localizzazione dei file.

A.5 Comandi per la documentazione

Comando	Significato e sintassi
<code>man</code>	È un'interfaccia ai manuali di riferimento in linea. <code>man [OPTIONS] file</code>
<code>whatis</code>	Mostra le descrizioni delle pagine di manuale. <code>whatis [OPTIONS] nome</code>
<code>apropos</code>	Ricerca nei nomi e nelle descrizioni delle pagine di manuale. <code>apropos [OPTION] parola chiave</code>
<code>find</code>	Cerca file in una gerarchia di directory. <code>find [PATH] espressione</code>
<code>comando -help</code>	Mostra le opzioni di un comando. <code>comando -help</code>

Tabella A.7: Principali comandi per la documentazione.

A.6 Comandi per la gestione dei tempi

Comando	Significato e sintassi
<code>date</code>	Mostra o configura data e orario. <code>date [OPTION]... [FORMAT]</code>
<code>hwclock</code>	Mostra e configura l'orologio hardware. <code>hwclock [OPTIONS]</code>

Tabella A.8: Principali comandi per la gestione dei tempi.

A.7 Comandi di archiviazione e compressione

Comando	Significato e sintassi
<code>tar</code>	Programma di archiviazione. <code>tar [OPTION] file1 file2 ..</code>
<code>gzip</code>	Comprime e scompime file. <code>gzip [OPTIONS] file</code>
<code>bzip2</code>	Programma di archiviazione. <code>bzip2 [OPTION] file1 file2 ..</code>

Tabella A.9: Principali comandi per di archiviazione.

A.8 Gestione dei pacchetti

Comando	Significato e sintassi
apt-get	Gestore di pacchetti di Debian GNU/Linux. <code>apt-get [OPTIONS]</code>
rpm	Gestore di pacchetti di RedHat. <code>rpm [OPTIONS]</code>

Tabella A.10: Principali comandi per di archiviazione.

A.9 I comandi diagnostici

Comando	Significato e sintassi
ping	Invia ICMP ECHO REQUEST agli host sulla rete . <code>ping [OPTIONS] host</code>
traceroute	Mostra i passaggi da un host ad un altro. <code>traceroute [OPTIONS] host</code>
netstat	Mostra le connessioni sulla ret. <code>netstat [OPTIONS]</code>

Tabella A.11: Principali comandi diagnostici.

A.10 I client dei servizi base

Comando	Significato e sintassi
telnet	Interfaccia utente al protocollo Telnet. <code>telnet [OPTION] host porta</code>
ftp	Client del protocollo File Transfer Protocol. <code>ftp [OPTIONS] host porta</code>
finger	Client per il controllo delle informazioni degli utenti. <code>finger [OPTION] user</code>
whois	Client per il servizio di whois. <code>whois [OPTIONS] keyword</code>

Tabella A.12: Principali client dei servizi base.

Appendice B

Indice degli argomenti per LPI

B.1 Argomenti LPI 101

<i>Topic</i>	Titolo	Riferimento
1.101.1	Configure Fundamental BIOS Settings	sez. 5.2.1 e 5.4.1
1.101.3	Configure Modem and Sound cards	sez. 5.4.1
1.101.4	Setup SCSI Devices	sez. 5.4.2
1.101.5	Configure different PC expansion cards	sez. 5.4.1
1.101.6	Configure Communication Devices	sez. 5.4.3
1.101.7	Configure USB devices	sez. 5.4.4
1.102.1	Design hard disk layout	sez. 1.2.3
1.102.2	Install a boot manager	sez. 5.3.2 e 5.3.3
1.102.3	Make and install programs from source	sez. 4.2.1
1.102.4	Manage shared libraries	sez. 3.1.2
1.102.5	Use Debian package management	sez. 4.2 e 4.2.3
1.102.6	Use Red Hat Package Manager (RPM)	sez. 4.2.2
1.103.1	Work on the command line	sez. 2
1.103.2	Process text streams using filters	sez. 2.2.5 e 2.3
1.103.3	Perform basic file management	sez. 2.2
1.103.4	Use streams, pipes, and redirects	sez. 2.1.5
1.103.5	Create, monitor, and kill processes	sez. 1.3
1.103.6	Modify process execution priorities	sez. 1.3.1
1.103.7	Search text files using regular expressions	sez. 2.2.2
1.103.8	Perform basic file editing operations using vi	sez. 2.4.3
1.104.1	Create partitions and filesystems	sez. 5.2.2, 5.2.3 e 5.2.3
1.104.2	Maintain the integrity of filesystems	sez. 5.2
1.104.3	Control mounting and unmounting filesystems	sez. 5.2.2
1.104.4	Managing disk quota	sez. 6.3
1.104.5	Use file permissions to control access to files	sez. 1.4.2, 1.4.3 e 1.4.4
1.104.6	Manage file ownership	sez. 1.4.2, 1.4.3 e 1.4.4
1.104.7	Create and change hard and symbolic links	sez. 1.2.2
1.104.8	Find system files and place files in the correct location	sez. 2.2.2
1.110.1	Install e Configure XFree86	sez. 3.4.2
1.110.2	Setup a display manager	sez. 3.4.3
1.110.4	Install e Customize a Window Manager Environment	sez. 3.4.4

B.2 Argomenti LPI 102

<i>Topic</i>	Titolo	Riferimento
1.105.1	Manage/Query kernel and kernel modules at runtime	sez. 5.1 e 5.1.4
1.105.2	Reconfigure, build, and install a custom kernel and kernel modules	sez. 5.1.1, 5.1.2 e 5.1.3
1.106.1	Boot the system	sez. 5.3 e 5.3.1
1.106.2	Change runlevels and shutdown or reboot system	sez. 5.3.4
1.107.2	Manage printers and print queues	sez. 3.5.2
1.107.3	Print files	sez. 3.5.4
1.107.4	Install and configure local and remote printers	sez. 3.5.3
1.108.1	Use and manage local system documentation	sez. 2.3.1 e 3.1.5
1.108.2	Find Linux documentation on the Internet	sez. 2.3.1
1.108.5	Notify users on system-related issues	sez. 3.1.4
1.109.1	Customize and use the shell environment	pag. 72 e ss.
1.109.2	Customize or write simple scripts	sez. 2.1.6
1.111.1	Manage users and group accounts and related system	sez. 1.4.1
1.111.2	Tune the user environment and system environment variables	sez. 3.2.2
1.111.3	Configure and use system log files to meet administrative and security needs	sez. 3.3.3 e 3.3.4
1.111.4	Automate system administration tasks by scheduling jobs to run in the future	sez. 3.3.1
1.111.5	Maintain an effective data backup strategy	sez. 4.1
1.111.6	Maintain system time	sez. 2.3.2
1.112.1	Fundamentals of TCP/IP	sez. 7.2.1
1.112.3	TCP/IP configuration and troubleshooting	sez. 7.3 e 7.4
1.112.4	Configure Linux as a PPP client	sez. 7.7.3
1.113.1	Configure and manage inetd, xinetd, and related services	sez. 8.1
1.113.2	Operate and perform basic configuration of sendmail	sez. ??
1.113.3	Operate and perform basic configuration of Apache	sez. ??
1.113.4	Properly manage the NFS, smb, and nmb daemons	sez. 8.4 e 8.5
1.113.5	Setup and configure basic DNS services	sez. 7.4.3, 7.4.2, 3.1.3, 9.2.2 e 9.3
1.113.7	Set up secure shell (OpenSSH)	sez. 8.3
1.114.1	Perform security administration tasks	sez. ??
1.114.2	Setup host security	sez. 4.3.2 e 4.3.3
1.114.3	Setup user level security	sez. 6.3

Appendix C

GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other written document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

C.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be

a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, L^AT_EX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

C.2 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

C.3 Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

C.4 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled “Acknowledgements” or “Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.
- Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

C.5 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

C.6 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

C.7 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

C.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

C.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

C.10 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License or any later version applies to it, you have the option of following the terms and conditions either of that specified version or of any

later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Indice analitico

- arithmetic expansion*, 62, 84
- backup, 147–149
- bootloader*, 3, 18, 184, 185, 192, 193, 207–215
- broadcast*, 266, 269, 276, 295, 298, 321, 322, 326
- canale DMA, 220
- comando
 - addgroup, 162
 - adduser, 162
 - apropos, 90
 - apt-get, 159
 - arp, 296
 - atq, 117
 - atrm, 117
 - at, 117
 - badblocks, 200
 - batch, 117
 - cancel, 144
 - cat, 64
 - cfdisk, 195
 - chatr, 49
 - chat, 305
 - chfn, 163
 - chgrp, 47
 - chmod, 46
 - chown, 47
 - chroot, 33
 - chsh, 163
 - cksum, 89
 - cpio, 152
 - cp, 12
 - crontab, 116
 - cut, 84
 - date, 93
 - delgroup, 162
 - deluser, 162
 - depmod, 187
 - dhclient, 322
 - diff, 173
 - dig, 342
 - dpkg, 158
 - dump, 153
 - e2fsck, 204
 - e2image, 202
 - echo, 57
 - edquota, 255
 - env, 58
 - expand, 86
 - exportfs, 332
 - false, 71
 - fdformat, 197
 - fdisk, 193
 - find, 76
 - finger, 302
 - fmt, 86
 - fold, 86
 - free, 35
 - fsck, 203
 - ftp, 301
 - getent, 167
 - getty, 40
 - gpsswd, 163
 - grep, 79
 - groupadd, 162
 - groupmod, 162
 - groups, 42
 - grpconv, 167
 - grpunconv, 167
 - grub-install, 214
 - head, 83
 - history, 63
 - hostname, 288
 - host, 340
 - hwclock, 94
 - id, 42
 - ifconfig, 273
 - info, 91
 - init, 215
 - insmod, 186
 - isapnp, 222
 - killall, 37
 - kill, 35
 - ldconfig, 107

ldd, 106
less, 83
lilo, 210
ln, 10
locate, 75
login, 40
logrotate, 120
lpadmin, 142
lpc, 135
lpd, 135
lpinfo, 144
lpq, 134
lprm, 135
lpr, 134
lpstat, 144
lp, 143
lsaddr, 48
lsmod, 191
lspci, 224
lsusb, 236
ls, 7
lvcreate, 249
lvdisplay, 250
lvextend, 250
lvreduce, 251
lvremove, 251
lvresize, 251
lvscan, 250
make, 156
mandb, 111
manpath, 112
man, 90
md5sum, 89
mkdir, 14
mke2fs, 198
mkfifo, 48
mkfs.msdos, 197
mkfs, 197
mknod, 48
mkreiserfs, 197
modinfo, 190
modprobe, 187
more, 82
mount, 21
mtr, 292
mv, 12
nc, 299
netcat, 299
netstat, 293
newgrp, 163
nice, 38
nl, 86
parted, 252
passwd, 162
paste, 85
patch, 174
ping, 290
pnpdump, 222
pr, 85
pstree, 26
ps, 27
pump, 321
pvcreate, 246
pvdisplay, 246
pvmove, 247
pvscan, 246
pwconv, 167
pwd, 14
pwunconv, 167
quotacheck, 253
quotaoff, 254
quotaon, 254
quota, 256
raidhotadd, 243
raidhotremove, 243
raidstart, 243
raidstop, 243
rdev, 209
renice, 38
repquota, 256
resize2fs, 251
resize_reiserfs, 252
restore, 154
rmdir, 14
rmmod, 188
rm, 11
route, 276
rpcinfo, 298
rpm, 157
run-parts, 116
scp, 328
scsiinfo, 230
sed, 86
seq, 71
setserial, 231
sg, 163
smbclient, 337
smbmount, 338
sort, 88
source, 72

- split, 84
- ssh-add, 331
- ssh-copy-id, 330
- ssh-keygen, 329
- ssh, 328
- startx, 129
- su, 163
- swapoff, 207
- swapon, 206
- syslinux, 209
- tac, 88
- tail, 83
- tar, 149
- tcpdchk, 318
- tcpdmatch, 318
- tcpd, 316
- tee, 95
- telinit, 217
- telnet, 298
- test, 71
- time, 92
- top, 33
- touch, 88
- traceroute, 291
- tree, 15
- true, 71
- tr, 85
- tune2fs, 200
- umount, 25
- unexpand, 86
- uniq, 88
- update-grub, 215
- updatedb, 114
- useradd, 161
- usermod, 162
- vgchange, 248
- vgcreate, 247
- vgdisplay, 248
- vgextend, 247
- vgmerge, 249
- vgreduce, 247
- vgscan, 248
- vgsplit, 249
- vidmode, 209
- wc, 89
- whatis, 90
- which, 60
- whoami, 42
- whois, 303
- xargs, 94
- xf86cfg, 124
- xf86config, 124
- xinit, 129
- xrdb, 131
- xterm, 129
- yes, 95
- command expansion*, 62
- configurazione
 - .bash_login, 73
 - .bash_profile, 73
 - .bashrc, 73
 - .profile, 73
 - .xinitrc, 129
 - /etc/X11/XF86Config-4, 124
 - /etc/X11/xdm/xdm-config, 130
 - /etc/X11/xdm/xdm.options, 130
 - /etc/apt/sources.list, 159
 - /etc/bootptab, 320
 - /etc/cron.allow, 116
 - /etc/cron.deny, 116
 - /etc/crontab, 115
 - /etc/cups/cupsd.conf, 140
 - /etc/dhclient.conf, 322
 - /etc/dhcpd.conf, 324
 - /etc/ethers, 296
 - /etc/exports, 332
 - /etc/fstab, 23
 - /etc/gdm/gdm.conf, 130
 - /etc/group, 165
 - /etc/gshadow, 167
 - /etc/host.conf, 286
 - /etc/hostname, 287
 - /etc/hosts.allow, 317
 - /etc/hosts.deny, 317
 - /etc/hosts, 287
 - /etc/inetd.conf, 310
 - /etc/inittab, 216
 - /etc/isapnp.conf, 222
 - /etc/issue.net, 110
 - /etc/issue, 110
 - /etc/kde3/kdm/kdmrc, 130
 - /etc/ld.so.cache, 107
 - /etc/ld.so.conf, 107
 - /etc/lilo.conf, 210
 - /etc/login.defs, 110
 - /etc/logrotate.conf, 120
 - /etc/lprng/lpd.conf, 137
 - /etc/lprng/lpd.perms, 137
 - /etc/manpath.config, 111
 - /etc/modules.conf, 188

- /etc/modules, 190
- /etc/motd, 110
- /etc/mtab, 25
- /etc/networks, 289
- /etc/pam.conf, 169
- /etc/pam.d, 168
- /etc/passwd, 164
- /etc/ppp/options, 305
- /etc/ppp/peers, 305
- /etc/printcap, 136
- /etc/profile, 73
- /etc/protocols, 289
- /etc/raidtab, 240
- /etc/rc.local, 113
- /etc/resolv.conf, 285
- /etc/rpc, 297
- /etc/samba/smb.conf, 334
- /etc/securetty, 110
- /etc/services, 288
- /etc/shadow, 166
- /etc/shells, 113
- /etc/skel, 113
- /etc/ssh/ssh_config, 328
- /etc/ssh/sshd_config, 326
- /etc/syslog.conf, 117
- /etc/updatedb.conf, 114
- /etc/xinetd.conf, 312
- menu.lst, 214
- named.conf, 343
- default gateway*, 271, 279, 280, 283, 321, 326
- demone
 - anacron, 116
 - atd, 117
 - bootpd, 319
 - crond, 115
 - dhcpcd, 323
 - gdm, 130
 - inetd, 310
 - kdm, 130
 - lpd, 137
 - named, 342
 - nmbd, 334
 - portmap, 298
 - pppd, 304
 - smbd, 334
 - ssh-agent, 330
 - sshd, 326
 - syslogd, 117
 - xdm, 130
 - xinetd, 312
- device mapper*, 244, 249
- directory radice, 3, 13–25, 33, 176, 203, 208, 209, 215, 216
- disciplina di linea, 299
- editor
 - emacs, 96
 - jed, 102
 - joe, 101
 - nano, 103
 - pico, 103
 - vi, 99
- espressioni regolari, 79–81, 87
- fifo*, 6, 7, 48
- filename globbing*, 60–62, 76, 80, 81, 135, 152
- filesystem /proc
 - /proc/bus/pci, 224
 - /proc/bus/usb, 235
 - /proc/dma, 220
 - /proc/filesystem, 21
 - /proc/interrupts, 220
 - /proc/ioports, 221
 - /proc/kernel/modprobe, 185
 - /proc/mdstat, 243
 - /proc/mounts, 25
 - /proc/scsi, 229
 - /proc/swaps, 206
 - /proc/sys/net/ipv4/ip_forward, 278
- Fully Qualified Domain Name*, 287
- hash*, 89, 164, 197, 306, 336, 337
- history*, 62
- hostname*, 110, 119, 286, 287, 301
- inode, 7–12, 199
- interrupt, 220
- kmod*, 185
- login manager*, 129
- MAC address*, 109, 246, 262, 267, 295–297, 319–321, 325
- memoria virtuale, 2, 34, 36, 205
- modo promiscuo, 276
- mount point*, 18, 21, 24, 25, 154, 203, 253, 333, 338
- multicast*, 266, 269, 276, 294
- Name Service Switch*, 108–109, 285, 288, 297
- nice*, 32, 34, 37–38, 314
- nome a dominio, 284, 285, 288

- path search*, 59
- pipe*, 66, 95
- Point to Point Protocol*, 267, 304
- porte, 271–273
 - destinazione, 273
 - effimere, 273
 - riservate, 273
 - sorgente, 273
- prompt*, 55
- Remote Procedure Call*, 297–298
- resolver*, 284–288
- router*, 266, 270–271, 277–279
- routing*, 267, 270–271
- shell
 - ash, 53
 - bash, 52
 - csh, 53
 - ksh, 53
 - sh, 52
 - tcsh, 53
 - zsh, 53
 - built-in
 - alias, 60
 - bg, 41
 - cd, 14
 - declare, 57
 - export, 57
 - fg, 41
 - jobs, 41
 - set, 56
 - type, 60
 - umask, 46
 - unalias, 60
 - unset, 56
 - di login, 72, 73
 - interattiva, 64, 72
 - istruzione
 - case, 70
 - elif, 69
 - else, 69
 - fi, 69
 - for, 70
 - function, 71
 - if, 69
 - then, 69
 - until, 70
 - while, 70
- socket, 7, 265–266, 270, 272, 294, 295, 309–312, 315, 331
- standard error*, 63, 65, 300, 312
- standard input*, 63–65, 75, 299, 309, 312
- standard output*, 63, 65, 75, 300, 309, 312
- swap*, 2, 24, 44, 195, 205–207
- system call*, 2, 3, 7, 29
- tabella di instradamento, 276–280, 294
- umask, 46, 73, 316
- variabili
 - di ambiente, 57
 - EDITOR, 58, 82
 - HOME, 58
 - LD_LIBRARY_PATH, 108
 - MANPATH, 112
 - PAGER, 82
 - PATH, 58–60
 - SSH_AUTH_SOCK, 331
 - TERM, 58
 - USER, 57
 - di shell, 56–57
 - HISTFILESIZE, 63
 - HISTFILE, 63
 - HISTSIZE, 63
 - IFS, 54
 - PS1, 55
 - speciali, 69
- window manager*, 131
- zombie, 29–30, 32, 36

Bibliografia

- [1] S. Piccardi, *Guida alla Programmazione in Linux*, vol. 1. <http://gatil.truelite.it>, 2001.
- [2] M. Cooper, *Advanced Bash-Scripting Guide*. <http://www.tldp.org/LDP/abs/>, 2000.
- [3] J. E. F. Friedl, *Mastering regular expression*. O'Reilly, 1997.
- [4] W. R. Stevens, *TCP/IP Illustrated, Volume 1, the protocols*. Addison Wesley, 1994.