Amministrare GNU/Linux

Simone Piccardi

23 marzo 2004

Indice

Capitolo 1

L'architettura di un sistema GNU/Linux

1.1 L'architettura del sistema.

Prima di addentrarci nei dettagli di funzionamento 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 si che spesso i detrattori di GNU/Linux ne denuncino la presunta mancanza di *innovatività*, ma anche le case hanno da secoli le stesse basi architetturali (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 provvedere 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.

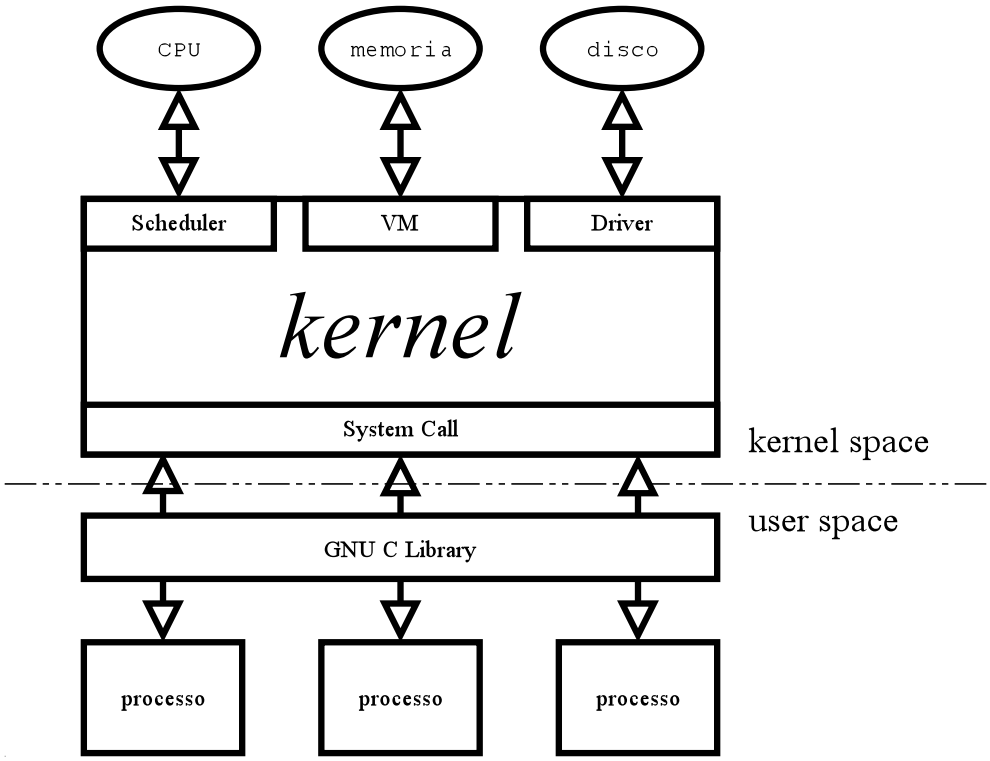

Figura 1.1: Struttura del sistema

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* appunto) in cui essi vedono se stessi come se avessero piena disponibilità della CPU e della memoria.

Sarà sempre il kernel ad eseguire al suo interno le operazioni di accesso alle risorse richieste dai vari processi, e a decidere volta per volta qual'è il processo che deve essere eseguito (realizzando così il multitasking) il tutto in modo sostanzialmente trasparente ai processi stessi.

Una conseguenza di questa separazione è che non è possibile ad un singolo programma 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.

Per illustrare meglio la distinzione fra kernel space e user space prendiamo in esame la procedura di avvio del sistema. 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*,[1] 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.

[1] questo 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.

Fra questi processi ci sarà anche quello che si occupa di dialogare con la tastiera e lo schermo della console, quello che chiede nome e password dell'utente che si vuole collegare, e quello che una volta completato il collegamento (procedura che viene chiamata *login*) mette a disposizione dell'utente l'interfaccia da cui inviare i comandi, sia questa una shell a riga di comando (su cui torneremo in cap. 2) o una interfaccia grafica.

È da rimarcare come tutti i comandi di base di un sistema GNU/Linux, come quelli per vedere la lista dei file, o quelli per entrare nel sistema, siano dei normali programmi come tutti gli altri, che vengono eseguiti dal kernel e fanno operazioni attraverso le funzioni (dette *system call*) che esso mette a disposizione, esattamente come accadrebbe se si fosse eseguito un programma di scrittura o di disegno.

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)[2] che altri sistemi operativi (come Windows) hanno al loro interno: tutte le operazioni di normale amministrazione di un sistema sono sempre realizzate tramite dei normali programmi.

1.1.2 Il funzionamento del sistema

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 infatti occorre avere, oltre al kernel, anche tutti i programmi che permettano all'utente di eseguire le varie operazioni con i dischi, i file, le periferiche.

Per questo al kernel vengono sempre uniti degli opportuni programmi di gestione 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, Slackware[3]), quelle che si trovano sui CD con i quali si installa "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.

Anche se il kernel tratta tutti i programmi allo modo, non tutti hanno la stessa importanza. Nella sezione precedente ad esempio abbiamo già incontrato uno programma particolare, `init`, che è quello che si cura dell'inizializzazione del sistema quando questo viene fatto partire. Una caratteristica fondamentale dell'architettura Unix infatti (ci torneremo in sez. 1.3) è quella per cui qualunque processo può a sua volta avviarne di nuovi,[4] per cui sarà cura di `init` mettere in esecuzione tutti i programmi necessari al funzionamento del sistema.

Benché in teoria sia possibile far partire qualunque altro programma al posto di `init`[5] tutti i sistemi Unix usano questo specifico programma come primo processo lanciato all'avvio del sistema. Così a seconda dei programmi che `init` mette in esecuzione (che a loro volta potranno lanciarne di altri) ci si può trovare davanti ad un terminale a caratteri o ad una interfaccia

[2] questa 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.

[3] in rigoroso ordine alfabetico!

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

[5] ed in casi di emergenza si può lanciare una shell al suo posto, o per usi particolare, ad esempio in sistemi embedded che devono svolgere un solo compito, un altro programma specifico.

grafica, e a seconda di quanto deciso dall'amministratore[6] si avrà un server di posta, o un server web, 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* ultima versione ufficiale) e quelli derivati dal codice sviluppato all'università di Berkley, detto BSD (da *Berkley Software 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 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 (quasi sempre in maniera assolutamente non avvertibile) le prestazioni 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*. Lo spazio grezzo, che è normalmente diviso in settori contigui di dimensione fissa, viene cioè organizzato in maniera tale da permettere il rapido reperimento delle informazioni memorizzate su questi settori che possono essere sparsi sul disco, per presentarli in quello che l'utente vede come un 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

[6] di norma lo si fa in fase di installazione, ma lo si può cambiare anche in seguito.

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.

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 design 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. 1.2.5). 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 layer della 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 periferiche[7] 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 dell'harddisk 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 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 comunicare 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.

[7] con la sola eccezione delle interfacce ai dispositivi di rete, che non rientrano bene nell'astrazione.

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

Tabella 1.1: I vari tipi di file riconosciuti da Linux

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 avviene anche grazie al fatto che in Linux l'accesso ai file viene effettuato attraverso una interfaccia, detta *Virtual File System*, che prevede una serie di operazioni generiche applicabili ad un qualunque oggetto del sistema il quale, secondo la filosofia del *tutto è un file*, è accessibile tramite essa.

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.[8] 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 definire l'operazione di spostamento `llseek`).

Funzione	Operazione
`open`	apre il file.
`read`	legge dal file.
`write`	scrive sul file.
`llseek`	si sposta all'interno del file.
`ioctl`	accede alle operazioni di controllo.
`readdir`	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*),

[8] ed altre ancora più complesse non sono state riportate.

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:

```
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 file[9] 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 *invisibili*.
`-i`	scrive il numero di `inode` (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 specificarlo 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, 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.

[9] con 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.

Il sesto campo è il tempo di *ultima modifica* del file e l'ultimo campo il nome del file. I tempi dei file (mantenuti automaticamente dal kernel quando opera si essi) in un sistema unix-like 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 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 di sistema unix-like.

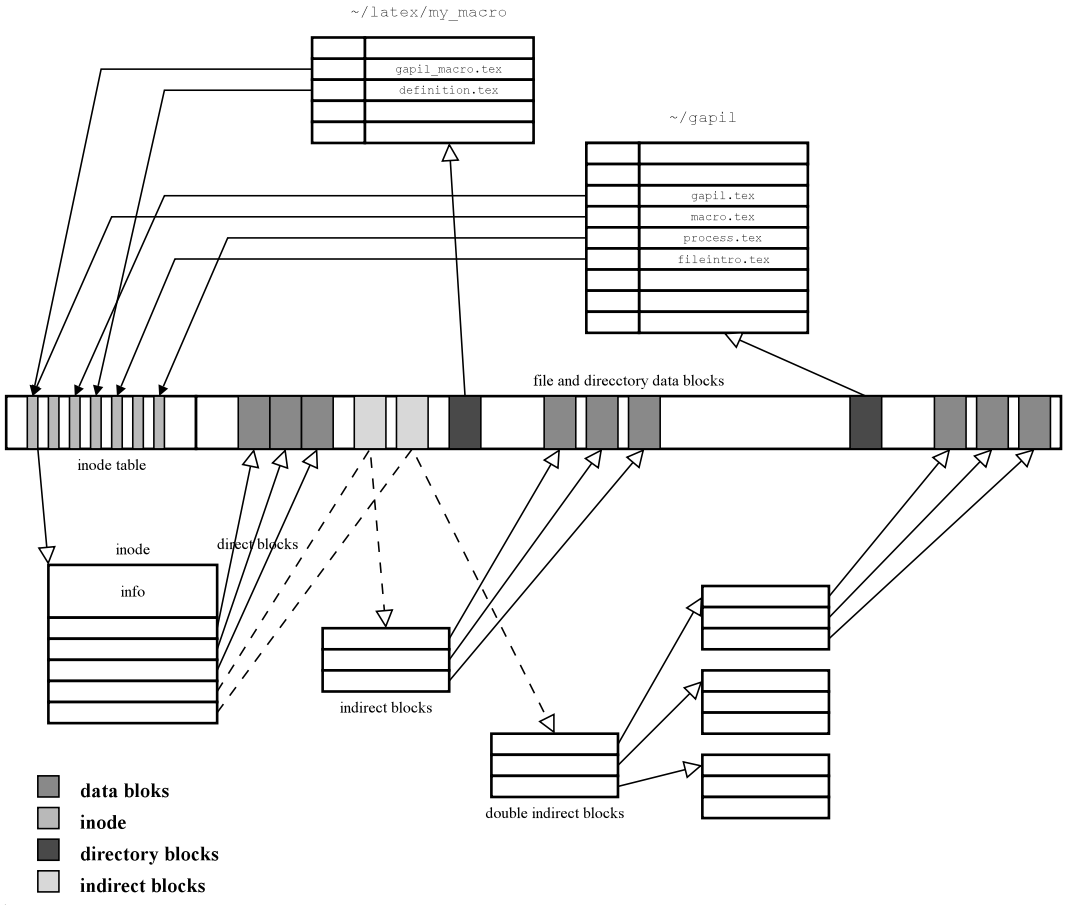

Figura 1.2: Strutturazione dei dati all'interno di un filesystem.

Se si va ad esaminare con maggiore dettaglio la strutturazione dell'informazione all'interno del 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 un file all'interno di un filesystem è il cosiddetto *inode*, a ciascun file infatti corrisponde un *inode*, che lo identifica univocamente. L'*inode* contiene tutte le informazioni riguardanti il file: il tipo di file, i permessi di accesso, le dimensioni, i puntatori ai blocchi fisici che contengono i dati e così via; 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 di output di `ls` visto in precedenza, dal valore numerico riportato nel secondo campo, e ci permette di dire se un file ha degli *hard link* (anche se non possiamo sapere dove sono).

Il comando generico che permette di creare dei link è `ln` che prende come parametri il file originale ed il nome del link. La differenza rispetto a Windows è che in un sistema unix-like i link sono di due tipi, oltre agli hard link appena illustrati infatti esistono anche i cosiddetti *link 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 `-i` 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` segnalerà 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 `ln -s file link`. In questo caso viene creato un nuovo file, di tipo *symbolic link*, e con un suo diverso *inode*, come mostrato nell'esempio precedente, il cui contenuto sarà il nome del 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 a cui questo punta.

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 *hard link* ad una directory (in Linux questa non è usabile).

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 ad una directory attraverso un path 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 basta 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 una ulteriore lista di riferimenti relativi a tutti i file che sono aperti, per cui anche se si cancellano tutti i nomi dalle varie directory in cui possono comparire, ma resta qualche processo attivo che ha aperto quel file, lo spazio disco non sarà rilasciato. Questo ci permette di capire anche un comportamento che può sembrare anomalo, quello per cui, in certe situazioni, pur cancellando un file non si recupera spazio disco.

Il comando `rm` e prende come parametri 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`. 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 (sovrascrivendo eventuali file presenti con lo stesso nome). 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
`-f`	forza la sovrascrittura del nuovo file se esso esiste già.
`-i`	richiede conferma in caso di sovrascrittura.
`-u`	esegue lo spostamento solo se la destinazione è più vecchia della sorgente.

Tabella 1.5: Principali opzioni del comando `mv`.

Dato che il comando si limita a cambiare di una voce associata ad un numero di *inode* all'interno di una directory, i tempi dei file non vengono mai modificati con l'uso di `mv`, fintanto che lo spostamento avviene all'interno dello stesso filesystem. 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 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.[10] 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` può prendere come argomenti o due nomi di file o una lista di file seguita da una directory; nel primo caso effettua una copia del primo file sul secondo, nel secondo copia tutti file della lista nella directory specificata.

[10] il kernel fornisce delle system call che permettono di cambiare i tempi di ultimo accesso e modifica di un file, così che il comando `mv` può 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.

Dato che il comando funziona copiando il contenuto di un file su un secondo 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`.

Opzione	Significato
`-f`	forza la sovrascrittura della destinazione se essa esiste già.
`-i`	richiede conferma in caso di sovrascrittura.
`-p`	preserva tempi, permessi e proprietari del file.
`-l`	crea degli hard link al posto delle copie.
`-s`	crea dei link simbolici al posto delle copie.
`-d`	copia il link simbolico invece del file da esso indicato.
`-r`	copia ricorsivamente tutto il contenuto di una directory.
`-R`	identico a `-R`.
`-a`	combina le opzioni `-dpR`.
`-L`	segue sempre i link simbolici.

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 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`.

1.2.3 La struttura dell'organizzazione delle directory

Un'altra delle caratteristiche specifiche 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*[11] 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 di memorizzazione dei dati, verranno poi *montati* (vedi sez. 1.2.5) successivamente in opportune sotto-directory della radice.

I nomi dei file sono indicati con un *pathname* o *percorso*, che descrive il cammino che occorre fare nell'albero 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, o in maniera relativa, partendo dalla cosiddetta *directory di lavoro corrente*.

Esempio	Formato
`/home/piccardi/gapil/gapil.tex`	assoluto
`gapil/gapil.tex`	relativo

Tabella 1.7: Formato dei pathname assoluti e relativi.

Quest'ultima è una caratteristica specifica di ogni processo, che viene ereditata dal padre alla sua creazione (vedi sez. 1.3.1). Quando si entra nel sistema la directory di lavoro corrisponde alla *home*[12] 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 ogni directory contiene sempre almeno due voci: la directory `.` che fa riferimento a se stessa, e la directory `..`

[11] l'operazione di rendere visibili i file dentro un filesystem è chiamata così.

[12] ogni utente ha una sua directory personale nella quale può tenere i suoi file, che viene chiamata così.

che fa riferimento alla directory sovrastante, in questo modo anche con dei pathname relativi si possono fare riferimenti a directory poste in sezioni diverse dell'albero. Si noti come entrambe queste voci (dato che entrambe iniziano per `.`) siano *invisibili*.

La shell inoltre, 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`. Infine `cd` riconosce il carattere `-` che indica il ritorna alla precedente directory di lavoro, torneremo su questo con qualche dettaglio in più in sez. 2.1.3.

1.2.4 Il *Filesystem Hierarchy Standard*

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 directory e, come accennato in sez. 1.1.1, viene montata direttamente dal kernel in fase di avvio. Per questo 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 (`init` compreso). Per questo `/` è l'unica directory che non può venire smontata, e può essere cambiata solo con un riavvio.[13]

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
...
|   |-- zmore
|   `-- znew
|-- boot
|   |-- System.map-2.4.20
...
|   |-- os2_d.b
|   `-- vmlinuz-2.4.20
|-- cdrom
|-- dev
|   |-- MAKEDEV -> /sbin/MAKEDEV
...
|   |-- xdb8
|   `-- zero
|-- etc
|   |-- GNUstep
...
|   |-- xpdf
|   `-- xpdfrc
|-- floppy
```

[13] in realtà essa può essere cambiata per i processi lanciati usando il comando `chroot`, in modo da restringere il loro accesso ai file contenuti in una sezione particolare dell'albero, il resto del sistema però continuerà a vedere la `/` originale.

```
|-- home
|   `-- piccardi
|-- initrd
|-- lib
|   |-- cpp -> /usr/bin/cpp-2.95
...
|   |-- modules
|   `-- security
|-- lost+found
|-- mnt
|   `-- usb
|-- opt
|-- proc
|   |-- 1
...
|   |-- uptime
|   `-- version
|-- root
|-- sbin
|   |-- MAKEDEV
...
|   |-- update-grub
|   `-- update-modules
|-- tmp
|   `-- ssh-XXBiWARl
|-- 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.

La organizzazione dell'albero delle directory è standardizzata in maniera molto accurata da un documento che si chiama *Filesystem Hierarchy Standard*, a cui tutte le distribuzioni si stanno adeguando. 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 bootloader
`/dev`	file di dispositivo
`/etc`	file di configurazione della macchina
`/lib`	librerie essenziali e moduli del kernel
`/mnt`	*mount point* per filesystem temporanei
`/opt`	pacchetti software addizionali
`/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 che possono essere condivisi (come i programmi di sistema) fra più stazioni di lavoro (ad esempio utilizzando un filesystem di rete) 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).[14]

Directory	**Contenuto**
`/lib<qual>`	librerie in formati alternativi
`/home`	home directory degli utenti
`/root`	home directory di *root*
`/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.

[14] le 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*, ma oggi sono in completo disuso.

`/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 quando si è un *single user mode*). 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.

`/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 script[15] 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) 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 i moduli devono essere installati in `/lib/modules`.

`/mnt` Contiene i *mount point* 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 `/`). 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 delle 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` e 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 utilizzati da sistema si potranno leggere i file `/proc/interrupts` e `/proc/dma`, mentre se si potranno impostare varie caratteristiche del sistema scrivendo nei file si `/proc/sys`.

`/root` È la home directory dell'amministratore. Di norma la si installa sullo stesso filesystem della radice.

[15] gli *script*, su cui torneremo in sez. 2.1.4, 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 è un vero di un 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 il 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 `/` in modo da poterla montare 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.
- `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 e amministrativi, 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 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/.

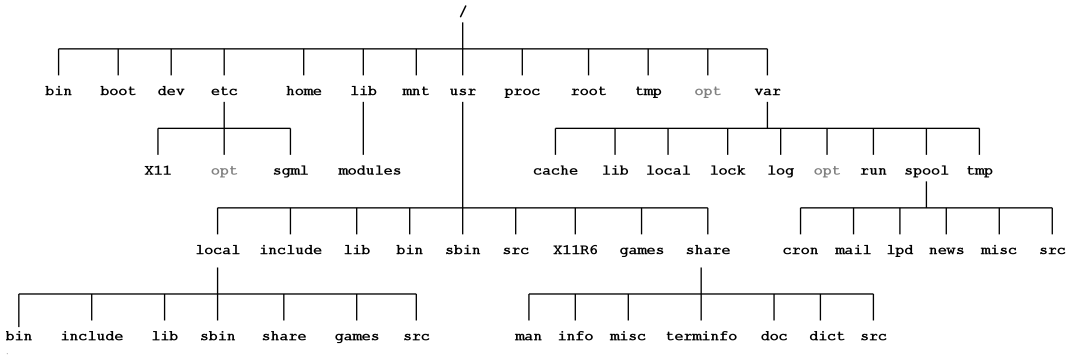

Figura 1.3: Struttura tipica delle directory, secondo il Filesystem Hierarchy Standard.

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 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.

1.2.5 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. 1.2.4) 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,[16] tutti gli altri *volumi* (che siano filesystem contenuti in partizioni diverse dello stesso disco o in altri dischi,

[16] e 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;[17] 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,[18] 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. 1.2.4) 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 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`,

[17] Si 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`.

[18] questo 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`	*Acorn Disc Filing System*, il filesystem del sistema operativo RiscOS.
`cramfs`	*Compressed ROM File System*, un filesystem su ROM per sistemi embedded.
`ext2`	*Second Extended File System*, il filesystem standard di Linux.
`ext3`	*Second Extended File System* (filesystem standard di Linux) in versione *journalled*.
`hfs`	*Hyerarchy File System*, il filesystem del MacOS (non MacOS X).
`hpfs`	*? File System*, il filesystem di OS/2.
`iso9660`	Il filesystem dei CD-ROM, secondo lo standard ISO 9660.
`jfs`	*Journalling File System*, il filesystem *journalled* della IBM portato su Linux.
`minix`	*Minix File System*, il filesystem del sistema operativo Minix.
`ntfs`	*NT File System*, il filesystem di Windows NT.
`qnx4`	*QNX4 File System*, il filesystem usato da QNX4 e QNX6.
`reiserfs`	*Reiser File System* un filesystem *journalled* per Linux.
`romfs`	Il filesystem
`ufs`	*Unix File System*, il filesystem usato da vari Unix derivati da BSD (SunOS, FreeBSD, NetBSD, OpenBSD e MacOS X).
`vxfs`	*Veritas VxFS File System*, filesystem standard di UnixWare disponibile anche su HP-UX e Solaris.
`xfs`	*? File System*, il filesystem di IRIX, portato dalla SGI su Linux.
`befs`	*BeOS File System*, 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 contente le informazioni relative al bus USB.
`nfs`	*Network File System*, filesystem per la condivisione di file attraverso la rete attraverso il protocollo NFS creato da Sun.
`coda`	*Coda? File System*, filesystem distribuito su rete che supporta funzionalità evolute come autenticazione, replicazione e operazioni disconesse.
`smbfs`	*SMB File System*, 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`.

`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).

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 `Debian.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 Debian.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 cifra-

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 soprassedute 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.

tura usato e l'opzione `keybits` per specificare la lunghezza (in bit) della chiave; la password di accesso verrà chiesta sul terminale.

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), `-v` che aumenta la verbosità 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 comporta 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 campi 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, 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.

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.

Gli ultimi due campi sono relativi alla manutenzione del filesystem, il quinto campo indica se effettuare il *dump*[19] 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 elencati 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,[20] 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

[19] è un valore utilizzabile solo per i filesystem (attualmente ext2 e ext3) che supportano il comando di backup `dump`; 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.

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

occorre identificare qual'è il file di dispositivo da usare e scegliere nel filesystem una directory su cui montarlo. Deve poi essere specificato il filesystem da usare (o `auto` se si vuole tentare il riconoscimento automatico).

Nell'esempio si noti come per ZIP e floppy si sia consentito agli utenti di montare il filesystem, ma si sia disabilitato il montaggio all'avvio, e pure il controllo dello stato del filesystem, dato che non è detto che il floppy o lo ZIP siamo sempre nel driver. Lo stesso vale per il CDROM e il masterizzatore, per i quali si è pure aggiunto l'opzione di montaggio in read-only. Si noti inoltre l'opzione speciale per il filesystem di root, per il quale si è indicato di rimontare il filesystem in read only nel caso di errori. Nel caso di disco fisso andrà poi scelto se montarlo all'avvio o meno, e in questo caso usare il sesto campo per indicare in quale ordine rispetto agli altri dovrà essere effettuato il controllo del filesystem (il primo deve essere il filesystem usato come radice).

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.

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 riutilizzo[21].

Il comando in questo caso è `umount`[22] che prende come parametro sia il mount point che il dispositivo e distacca il relativo filesystem dall'albero dei file. Si tenga presente che fintanto che il filesystem è utilizzato questa operazioni 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

[21] con 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.

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

come i processi sono organizzati in forma gerarchica, quali sono caratteristiche e proprietà che contraddistinguono ciascuno di essi porta con sé, e come vengono gestiti all'interno del sistema.

1.3.1 Le proprietà dei processi

Come spiegato in sez. 1.1.2 la caratteristica principale dell'architettura dei processi in un sistema unix-like è che qualunque processo può a sua volta lanciarne degli altri, che vengono detti *processi figli* mentre il processo originale è detto *processo padre*. Inoltre tutti i processi nel sistema possono essere creati solo in questo modo e pertanto tutti i processi avranno un padre, con l'unica eccezione di `init` 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 del tutto speciale, come radice dell'albero. La classificazione può essere mostrata direttamente con il comando `pstree`:

```
init-+-atd
     |-bdflush
     |-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
```

che evidenza in maniera grafica l'*albero genealogico* di ciascun processo attivo nel sistema, alla cui radice, come dicevamo, c'è sempre `init`.

Si tenga presenta che questa architettura, in cui i processi possono creare 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 chiunque. 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 chiamate al sistema diverse, la prima delle quali crea un nuovo processo, identico al padre, che poi usa la seconda per eseguire un altro programma. Il meccanismo però permette anche (una modalità di funzionamento comune con i server di rete) di scrivere un programma unico in cui il processo padre esegue la parte che si occupa di ricevere le richieste, e per ciascuna di esse fa eseguire ad un figlio creato apposta le operazioni necessarie alla fornire le relative risposte.

Proprietà	**Descrizione**
`PID`	PID del processo.
`PPID`	PID del padre del processo.
`UID`	*effective user id* del processo.
`GID`	*effective group id* del processo.
`CMD`	linea di comando con cui è stato lanciato il processo.
`STAT`	stato del processo.
`NI`	valore di *nice* (vedi sez. 1.3.3) del processo.
`TTY`	terminale di riferimento del processo.
`SID`	SID o *session id* (vedi sez. 1.3.4) del processo.
`PGID`	*process group id* (vedi sez. 1.3.4) del processo.
`%CPU`	percentuale del tempo di CPU usato rispetto al tempo reale di esecuzione.
`%MEM`	percentuale della memoria utilizzata.
`START`	tempo di avvio.
`TIME`	tempo di CPU utilizzato.
`USER`	*effective user id* del processo
`RUSER`	*real user id* del processo
`GROUP`	*effective group id* del processo
`RGROUP`	*real group id* del processo

Tabella 1.12: Le principali proprietà dei processi e ed il nome della relativa colonna nella visualizzazione effettuata da `ps`.

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, derivate dalle varie versioni che del comando sono state realizzate nel tempo. In Linux il comando supporta la maggior parte di queste; quelle derivata da SysV devono essere precedute da un `-`, quelle derivate da BSD non devono essere precedute da un `-`, mentre le estensioni GNU usano un `-`.

Per ogni processo attivo il kernel mantiene tutta una serie di proprietà ed informazioni necessarie alla sua gestione ed al controllo delle varie operazioni che esso può compiere, la principali delle quali sono riportate in tab. 1.12 insieme al nome che `ps` utilizza come intestazione della colonna che ne visualizza il valore. Torneremo sul loro significato fra breve.

Veniamo allora al funzionamento di `ps`, se eseguito su un terminale da un utente senza dare nessuna opzione esso mostra l'elenco dei processi appartenenti a detti utente attivi sul quel terminale; specificando l'opzione `a` verranno visualizzati anche i processi lanciati da altri utenti (sempre facenti riferimento allo stesso terminale), mentre con l'opzione x si visualizzano tutti i processi non legati ad un terminale; infine l'opzione `f` permette di mostrare la gerarchia dei processi, così si può ottenere un elenco completo dei processi attivi con una cosa del tipo:

```
[piccardi@hogen piccardi]$ ps axf
  PID TTY      STAT   TIME COMMAND
    6 ?        SW     0:00 [kupdated]
    5 ?        SW     0:00 [bdflush]
    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
```

ed in questo caso, avendo usato le opzioni `a` e `x`, si sono selezionati 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`, come spiegheremo fra poco).

Con questo formato il comando ci mostra solo alcune informazioni di base; già queste però sono sufficienti ad introdurre alcune delle caratteristiche essenziali dei processi. La prima colonna, `PID`, contiene il cosiddetto *process id* del processo, un numero che il kernel utilizza per poter identificare univocamente ciascun processo. Questo numero viene assegnato alla creazione del processo, ed è unico fintanto che il processo resta attivo. È il numero che si deve usare quando si vuole fare riferimento ad un processo specifico.

La seconda colonna, `TTY`, mostra il nome del terminale di controllo del processo. Ogni processo interattivo è sempre associato ad un terminale di controllo, che corrisponde appunto 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 con l'interfaccia a riga di comando, ad esempio è un programma ad interfaccia grafica) il non esiste un terminale di controllo e la colonna riporta un valore di `?`.

La terza colonna, `STAT`, riporta lo stato del processo; il sistema infatti prevede cinque possibili stati diversi per i processi, riportati in tab. 1.13, torneremo su questo a breve, in quanto ci permette di spiegare varie caratteristiche della gestione dei processi. Seguono infine la colonna `TIME` che indica il tempo di 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
runnable	R	Il processo è in esecuzione o è pronto ad essere eseguito (cioè è in attesa che gli venga assegnata la CPU).
sleep	S	Il processo è in attesa di un risposta dal sistema, ma può essere interrotto da un segnale.
uninterrutible sleep	D	Il processo è in attesa di un risposta dal sistema (in genere per I/O), e non può essere interrotto in nessuna circostanza.
stopped	T	Il processo è stato fermato con un `SIGSTOP`, o è tracciato.
zombie	Z	Il processo è terminato ma il suo stato di terminazione non è ancora stato letto dal padre.

Tabella 1.13: 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 chiamata al sistema 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. 1.13 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) questo non è

possibile ed allora si ha uno stato di *ininterrutible sleep*, allora se il processo resta bloccato[23] nello stato `D` non può più essere terminato se non con il riavvio della macchina; tutto questo 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 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 che 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 chiamata al sistema 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 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 si 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 riceve 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.[24] 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.

[23] in 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.

[24] si vedrà cioè, ad una successiva esecuzione di `ps` con opzioni che permettano di visualizzare il PID del padre, che questo è diventato 1.

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.[25] 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 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.[26]

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 [bdflush]
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
```

[25] per 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.

[26] quest'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.

```
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 sono mantenute nella forma dei relativi UID (*user id*) e GID (*group id*).[27] Ogni processo in realtà mantiene diverse versioni di questi identificatori,[28] 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,[29] 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 l'utilizzo della CPU con una opportuna metrica, 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. 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`.

Infine ci sono alcune proprietà dei processi, non direttamente visualizzabili con `ps`, che comunque sono importanti e vanno menzionate. Come accennato in sez. 1.2.3, ogni processo ha una directory di lavoro rispetto alla quale risolve i pathname relativi; anche questa è una caratteristica 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 cui ogni processo oltre alla directory di lavoro corrente, porta con se pure 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 redirezione dell'output; in tal caso di

[27] come 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*.

[28] Linux 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.

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

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 bdflush
    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 khubd
  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*, 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), l'utente cui esso appartiene (colonna `USER`); la priorità ed il valore di *nice* (per i dettagli vedi sez. 1.3.3) rispettivamente nelle colonne `PR` e `NI`; i dati dell'uso della memoria (colonne `VIRT`, `RES` e `SHR`). Segue 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`).

In modalità interattiva è possibile dare una serie di comandi, con `k` si può inviare un segnale (di default `SIGTERM`, vedi sez. 1.3.2) mentre con `r` si può cambiare il valore di *nice* (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 dei risultati. L'elenco completo, oltre che nella pagina di manuale, può essere stampato a video con `h`.

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
```

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.[30] In generale ciascuno di essi ha un compito o un significato specifico, ad esempio il segnale `SIGSEGV` viene inviato dal kernel per segnalare ad un processo una *Segment Violation*, cioè un accesso illegale alla memoria (un errore di programmazione che di default comporta la terminazione immediata del processo).

Gran parte dei segnali (tutti eccetto `SIGKILL` e `SIGSTOP`) possono essere *intercettati* dal processo, che può eseguire una opportuna funzione al loro arrivo. Se ciò non viene fatto viene eseguita una azione di default che nella maggior parte dei casi è la terminazione immediata del processo. Ad esempio il segnale `SIGTERM` (che è quello che il comando `kill` invia di default) serve per richiedere la terminazione di un processo; esso infatti è associato alla pressione di ctrl-C sulla tastiera, che lo invia al processo che sta girando al momento sul terminale. In questo caso l'azione di default è quella di terminare immediatamente il processo, ma il segnale può essere intercettato per eseguire delle operazioni di pulizia come ad esempio cancellare dei file temporanei prima dell'uscita.

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

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 processo venga bloccato anche nell'esecuzione della funzione di gestione; dato che non è possibile intercettare `SIGKILL` si ha a disposizione un mezzo infallibile[31] 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. 1.14. Al solito la documentazione completa è nella pagina di manuale, accessibile con `man killall`.

Opzione	Significato
`-g`	invia il segnale al *process group* 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 1.14: 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*[32] 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.

[31] quasi 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.

[32] sono 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.

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*,[33] 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 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 una *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[34] mette sempre in esecuzione, fra tutti quelli in stato *runnable*, il processo che ha una priorità dinamica più bassa.

Questo fa si 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 "*carini*" 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 su vuole cambiare la priorità. L'opzione principale è `-n nice` 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 ...]
```

[33] questo 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.

[34] lo *scheduler* è la parte di kernel che decide quale processo deve essere posto in esecuzione.

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.

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. 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*[35] 0, 1 e 2.

Convenzionalmente[36] 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. 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. 4.1.4). 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.[37]

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
```

[35] come 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*.

[36] si 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.

[37] di 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*.

```
 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 vengano mandati in esecuzione sospendendo momentaneamente l'accesso al terminale. Se, come accade per `xmms`, `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`, questo ferma il processo, che può essere fatto ripartire in background con il comando `bg`.[38]

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

[38] attenzione, questo, come i seguenti `fg` e `jobs`, è un comando interno della shell, e non corrisponde a nessun eseguibile su disco.

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 come Linux deve fornire dei meccanismi di identificazione dei vari utenti sulla base dei quali poi viene imposto il controllo degli accessi e delle operazioni che questi possono fare.

La struttura di sicurezza del sistema è estremamente semplice, tanto da essere in certi casi considerata troppo primitiva[39] e prevede una distinzione fondamentale fra 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 e nel caso si nega 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.[40]

Per questo motivo tutti gli Unix 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.

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 si 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 il sistema può provvedere al controllo degli accessi, e porre una serie di limitazioni a quello che l'utente può fare, impedendo al processo 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 l'utente che lo sta eseguendo è `whoami`, che ne stampa il nome di login (comunemente detto *username*. 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 trattate 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. Il comando prende molte

[39] 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*.

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

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.

1.4.2 I permessi dei file

Anche la gestione dei permessi dei file è tutto sommato piuttosto semplice; esistono estensioni[41] 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 ElemDaytimeTCPCuncServ.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.

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

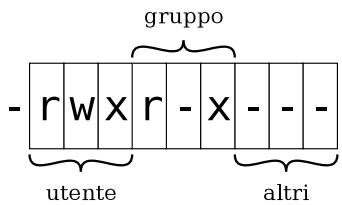

Figura 1.4: Legenda dei permessi dell'output di `ls`.

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`, 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 stessi[42].

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 significato[43] solo al caso di programmi eseguibili.[44] 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 si 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.

[42] 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

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

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

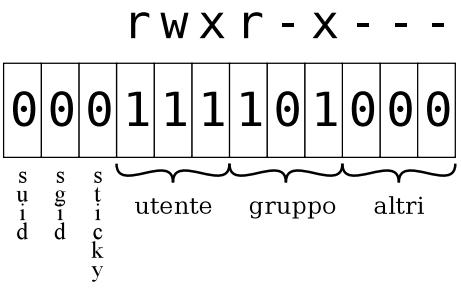

Figura 1.5: Schema dei bit dei permessi mantenuti nell'inode.

La situazione completa dei bit associati ai permessi è illustrata in fig. 1.5. Data corrispondenza diretta con il contenuto dell'inode, ed 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 i gruppi, e per il proprietario. Per cui i permessi di fig. 1.5, corrispondenti ai valori già mostrati in fig. 1.4, si 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 *suid bit*, *sgid bit* sono attivi verrà rispettivamente mostrata una `s`[45] nel permesso di esecuzione per utente e gruppo. Lo *sticky bit* invece modifica il permesso di esecuzione per tutti in una `t`.[46]

Finora abbiamo parlato di permessi applicati ai file normali, ma gli stessi permessi possono essere anche applicati ad una directory[47] 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.

[45] o una `S` qualora il corrispondente permesso di esecuzione non sia attivo.

[46] che come per i precedenti diventa una `T` qualora il corrispondente permesso di esecuzione non sia attivo.

[47] ed 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.

1.4.4 I comandi per la gestione dei permessi dei file

Prima di entrare nei dettagli dei comandi che permettono di cambiare permessi e proprietari di file e directory, occorre dare qualche dettaglio in più sulle modalità con cui nuovi file e directory vengono creati. Il comportamento di default del sistema infatti è di creare i nuovi file con i permessi di lettura e scrittura attivati per *tutti* dove con tutti qui si vuole intende proprietario, gruppo e tutti gli altri, cioè un permesso numerico 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 non è cosa molto saggia dal punto di vista della sicurezza. Per questo motivo il kernel mantiene anche, per ogni processo, una proprietà specifica, la *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 di cancellare dai permessi tutti i bit in essa specificati.

In genere la *umask* viene impostata negli script di avvio grazie al comando `umask`. Dato che poi 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 controllare il valore di questo parametro invocando direttamente lo stesso comando,[48] che prende come parametro il nuovo valore. Questo può essere specificato 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, l'opzione `-S` stampa il valore in forma simbolica. Di norma un utente personalizza questo valore negli script di avvio della shell.

Allora se un permesso ha una *umask* di `022` 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 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 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 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 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). Quando si usa il programma con un link simbolico verranno cambiati i permessi del programma referenziato dal link.

[48] anche questo è un comando interno della shell, dato che l'invocazione di un comando esterno non servirebbe a nulla, in quanto la chiamata al sistema che esegue l'operazione opera solo sul processo corrente.

Il comando supporta anche l'uso dell'opzione `-R` che esegue la modifica dei permessi ricorsivamente quando lo si utilizza per una directory. 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 directory). Per questo esistono i due comandi `chown` e `chgrp`. Il primo cambia il proprietario ed il secondo il gruppo. Entrambi prendono come parametri utente (gruppo per `chgrp`) ed i file su cui operare. Le opzioni sono simili a quelle di `chmod`, ed in particolare `-R` esegue dei cambiamenti ricorsivamente e `-reference` usa i valori di un altro file. Inoltre `chown` supporta per il nome utente la sintassi `username.group` che permette di cambiare anche il gruppo.

Si tenga presente che un utente normale può cambiare solo il gruppo dei file che gli appartengono, ed assegnarli soltanto a gruppi cui egli appartiene. Solo l'amministratore ha la capacità piena di cambiare proprietario e gruppo di un file.

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 root che può inviare segnali a qualunque processo.

Altre operazioni privilegiate sono quelle che riguardano la rete. 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 è il montaggio dei filesystem, come già visto in sez. 1.2.5, 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, 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* [49] che lo identificano univocamente.[50]

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.

[49] questi due numeri sono il meccanismo con cui storicamente vengono identificati i dispositivi all'interno del kernel (cosi 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.

[50] per 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`.

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 la progettazione del sistema, anche l'interfaccia a riga di comando deriva da alcune scelte progettuali precise. Arnold Robbins spiega molto chiaramente questa filosofia in un articolo riportato anche nella pagina *info* 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 cosiddetta *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 comando[1] 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 e comunque specifiche allo scopo del comando, e sono descritte dettagliatamente nella relativa pagina di manuale.

[1] ne abbiamo incontrati già alcuni in cap. 1, e ne vedremo molti altri fra breve.

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

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

- **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 Kb 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. 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 completo 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 (sul mio sistema conta la bellezza di 5266 righe, ed in effetti più che una pagina è un manuale!). Per questo in seguito faremo riferimento, quando necessario, alle varie sezioni in cui essa è divisa.

2.1.3 Funzionalità generali

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 semplicemente quello di leggere una riga di comando dalla tastiera, riconoscere qual è il programma che volete mettere in esecuzione e ricostruire i vari parametri da passare poi al programma stesso per poi eseguirlo. Così quando scrivete qualcosa del tipo:

```
piccardi@anarres:~/Truelite/documentazione/corso$ rm pippo pluto paperino
```

la shell capirà che volete invocare il comando `rm`, individuerà il file che contiene il relativo programma su disco, e lo lancerà passandogli come parametri le tre stringhe `pippo`, `pluto` e `paperino`.[3]

[3] il meccanismo è un po' più complesso di quando non sembri, la shell infatti nel costruire la lista dei parametri esegue sempre una scansione della riga di comando e considera uno spazio vuoto (composto da uno o più spazi, tabulatori, ecc.) come separatore fra due parametri diversi. Questo comportamento può essere modificato indicando un carattere diverso nella variabile (spiegheremo le variabili a breve) `IFS`.

Benché la filosofia di Unix sia quella di utilizzare un apposito comando per effettuare ciascun compito specifico, la shell fornisce direttamente alcune funzionalità tramite alcuni *comandi interni* (detti anche *built-in*), che non eseguono a nessun programma a parte, e che di norma servono a svolgere compiti che relativi all'uso stesso dell'interfaccia.

Un esempio classico di questi comandi interni è `cd`, che andando a modificare la directory di lavoro corrente della shell non può essere eseguito come comando esterno (in quanto questo cambierebbe directory di lavoro corrente per se 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 in seguito.

Ma oltre a lanciare i comandi, già l'esempio precedente con `rm` ci mostra come la shell esegua 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 una linea di comando 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 (parleremo delle variabili fra poco) `PS1`,[4] nel cui contenuto, oltre ai normali caratteri, si possono inserire una serie di caratteri di controllo, i principali dei quali sono riportati in tab. 2.1, che vengono automaticamente *espansi* in un determinato valore (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 *username* 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 opzioni di visualizzazione per il prompt di shell.

Due esempi possibili sono il prompt usato da Debian, che usa per `PS1` il valore `\u@\h:\w\$`, 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 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, utilizzando una semplice assegnazione del tipo:

[4] in 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`.

```
VARIABILE=valore
```

Per convenzione le variabili si scrivono con lettere maiuscole, (ma è appunto solo una convenzione, è possibile usare qualunque tipo di lettera ed il carattere _), una volta definita una variabile è poi possibile recuperarne il valore (ad esempio all'interno di una riga di comando) precedendone il nome con il carattere $, così se nella variabile MAIL mettete la directory in cui si trova la vostra posta elettronica, potrete guardarne il contenuto con un comando del tipo:

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

L'elenco delle variabili di shell già definite (vedremo in sez. 2.1.4 come effettuarne l'impostazione in maniera standard) si può ottenere con il comando set,[5] che ne stampa a video la lista coi relativi valori, mentre per cancellarne una si può usare il comando[6] unset seguito dal nome della variabile.

Se la shell provvede degli opportuni comandi interni per assegnare e cancellare le variabili, altrettanto non accade per la visualizzazione, in quanto questo compito può essere eseguito attraverso l'uso di uno dei tanti comandi specialistici della *Unix toolbox*, ed in particolare tramite il comando echo, il cui solo compito è stampare in uscita la stringa (o le stringhe) passata come parametro: siccome la shell esegue automaticamente l'espansione delle variabili prima di passare i parametri ad un comando, si potrà usare echo per leggere il contenuto di una variabile con un comando del tipo:

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

la variabile USER infatti verrà espansa nella stringa piccardi dopo di che la shell passerà quest'ultima come parametro ad echo, che lo scriverà in uscita.

Il comando echo 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, dato che sul terminale si avrebbe l'interferenza da parte del prompt. La seconda 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 pagina di manuale accessibile con man echo.

La bash definisce di suo (o utilizza qualora siano definite) tutta una serie di veribili, 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.

La shell fornisce inoltre il supporto per una funzionalità prevista dallo standard ANSI C, che permette di passare il valore di alcune di queste variabili anche ai programmi che vengono messi in esecuzione, attraverso l'uso del cosiddetto *ambiente* (in inglese *environment*).[7] 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; per visualizzare le variabili presenti nell'ambiente occorre allora usare l'apposito comando interno env, che al solito ne stampa l'elenco a video; se invece

[5] in 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 utlizzato con le opportune opzioni, per impostare una lunga serie di altre proprietà della shell, per l'elenco completo si può fare riferimento alla sua descrizione nella sezione SHELL BUILTIN COMMANDS della pagina di manuale.

[6] anche questo, come il precedente è un comando interno interno alla shell.

[7] in sostanza un programma quando viene lanciato con la opportuna chiamata al sistema (i curiosi possono fare riferimento al capitolo due di GaPiL), deve ricevere una serie di informazioni da chi lo lancia; una parte di queste informazioni sono i parametri, che vengono presi direttamente da quanto scritto nella riga di comando con il meccanismo cui abbiamo accennato in una nota precedente, l'altra parte sono appunto le variabili di ambiente.

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 da ignorare nel *filename globbing*.
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.
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.

si vuole inserire una variabile nell'ambiente si deve usare il comando `export` seguito dal nome della stessa (che deve essere già definita),[8] mentre la sua rimozione si effettua sempre con `unset`.

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` ne indica la *home directory*, `TERM` che specifica il tipo di terminale su cui si sta operando, `PATH` che specifica la lista delle directory dove cercare i comandi, ecc.

La variabile `PATH` ci introduce ad un'altra caratteristica della shell: quella del cosiddetto *path search*. Come illustrato nell'esempio iniziale la shell vi consente di non specificare il pathname completo quando invocate un programma, così se volete leggere una pagina di manuale basta usare il comando `man` e non `/usr/bin/man`. La shell, quando gli viene richiesto un nome di programma senza specificarne il pathname assoluto, lo cerca nel cosiddetto *PATH*, cioè nella lista di directory contenute appunto nella variabile d'ambiente `PATH`. La shell fornisce poi anche il comando `which` che, dato un comando presente nel *PATH*, stampa a video il pathname completo, con qualcosa del tipo di:

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

in questo modo è possibile capire quale sia effettivamente il comando che viene usato qualora ne esistano più versioni in diverse directory del *PATH*.[9]

Si tenga presente però che `which` funziona solo nel caso di programmi esterni, se indica un comando interno questo non viene considerato, per capire alloa se un comando è interno o esterno, e di che tipo, si può usare `type`, ottenendo qualcosa del tipo:

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

La variabile `PATH` ha la forma di una lista di pathname assoluti di directory separati da un carattere di due punti,[10]. Un esempio del valore di `PATH` potrebbe essere:

[8] la `bash` supporta anche la definizione della variabile nella stessa linea in cui viene esportata nell'ambiente, 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 manrenere la compatibilità.

[9] la regola comunque è che viene usato il primo che viene trovato nella scansione, che viene eseguita nell'ordine in cui le directory sono elencate, se non ve ne sono non viene stampato nulla.

[10] è un formato comune usato nelle variabili di ambiente tutte le volte che si deve specificare una lista di directory in cui effettuare ricerche, lo reincontreremo spesso, ad esempio in sez.

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

se si vuole aggiungere un'altra directory al `PATH`, ad esempio la directory corrente, che di norma non vi è mai inclusa,[11] questo può essere fatto semplicemente ridefinendo la variabile con qualcosa del come:

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

e di nuovo si è usato la capacità della shell di riutilizzare le variabili. Si noti anche come una riga del genere non esegua nessun comando, ma si limiti a lavorare con le funzionalità interne della shell.

Un'altra funzionalità che la shell mette a disposizione è quella degli *alias*: prima ancora di cercare se una certa parola corrisponde ad un comando presente in una delle directory indicate dal *PATH*, la shell controlla se essa corrisponde ad un *alias*; ci permette così di ideare nuovi comandi, definire abbreviazioni per quelli più usati, o ridefinire il nome stesso di un comando per farlo comportare in maniera diversa. Un *alias* si crea appunto con il comando `alias` associando un comando alla nuova parola chiave con un qualcosa 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.

Un'altra funzionalità estremamente comoda della shell è il cosiddetto *filename globbing*, si può cioè operare su gruppi di file con nomi simili utilizzando dei *caratteri jolly* (detti anche *wildcard*); se cioè si indica un nome inserendo uno di questi caratteri, la shell verificherà quali file ci sono nella directory corrente, e se ne troverà di corrispondenti *espanderà* il nostro nome nella corrispondente lista.[12]

Come nel caso del DOS, il carattere `*` viene espanso in un numero arbitrario di caratteri mentre il carattere `?` viene sempre sostituito da un solo carattere. Ma oltre a queste due *wildcard* 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 fa i due estremi) usando due caratteri separati da un `-`. Infine quando si può invertire la selezione precedendo la lista o l'intervallo con un carattere di `"^"`.

Vediamo allora alcuni esempi di tutte queste opzioni relative al *filename globbing*: come primo esempio si potranno vedere i PDF presenti in una directory con qualcosa come:

[11] è una domanda frequente nei newsgroup di sviluppo quella dei principianti che chiedono perché non riescono ad eseguire il programma che hanno appena creato: è appunto per questo, e la scelta ha una ragione precisa; se infatti si inserisse la directory corrente nel PATH un utente malizioso potrebbe mettere la sua versione "*taroccata*" di un qualche comando fondamentale, che voi rischiereste così di eseguire, e se lo fate da amministratore potreste trovarvi in guai seri; è vero che se la si mette in fondo i comandi usuali hanno la precedenza, ma qualcuno potrebbe sempre mettere `la` e aspettarvi al varco per un comune errore di battitura di `ls` ...

[12] questo significa che al comando sarà passato, invece di un singolo argomento, la lista separata da spazi dei nomi corrispondenti all'espansione dei caratteri jolly.

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

mentre si potranno selezionare tutti i file con nomi lunghi esattamente sette caratteri con qualcosa del tipo:

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

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

```
piccardi@anarres:~/Truelite/documentazione/corso$ ls *[c,g]
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 che iniziano con un maiuscola potremo usare:

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

infine se ci interessano i PDF che non iniziano 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
```

ma si tenga presente che il meccanismo dell'espansione è tale che è il risultato che viene passato come parametro al comando, per cui ad esempio se si fosse eseguito:

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

si sarebbe ottenuto un risultato forse inaspettato, che non riporta file di tre lettere. Questo avviene perché in questo caso l'unico file di tre lettere presente è la directory `CVS`, per cui otteniamo il risultato di `ls CVS`, che è appunto il modo in cui viene espanso il precedente comando.

Infine una ultima modalità espansione dei nomi dei file si ha utilizzando il carattere tilde `"~"` per indicare la home directory dell'utente corrente, mentre si può usare la notazione `~username` per indicare la home di un'altro utente usando il relativo pathname.

Il *filename globbing* e l'uso delle variabili di ambiente è molto utile, però talvolta presenta degli inconvenienti; ad esempio se si vuole passare come parametro il nome di un file contenente uno dei caratteri jolly si avrebbe un problema, dato che la shell cercherebbe di espanderlo, stessa cosa che avverrebbe per il carattere `$`, usato per eseguire l'espansione delle variabili. Un inconveniente analogo ci sarebbe per per i nomi dei file che contengono degli spazi, dato che lo spazio viene usato per separare i parametri fra di loro verrebbero passati due parametri e non una stringa unica.

Per risolvere questi problemi esistono dei metodi per disabilitare del tutto o in parte le capacità di interpretare in maniera speciale i vari caratteri. Questo può essere fatto, a livello di intere stringhe (risolvendo così il problema degli spazi), indicando i parametri per i quali non si vuole effettuare l'espansione del filename globbing fra virgolette (`"`), in questo il testo fra virgolette viene interpretato come una stringa e si potranno usare spazi, asterischi e punti interrogativi. In questo modo però resta attiva l'espansione delle variabili, si può eliminare anche questa usando al posto delle virgolette gli apici semplici (`'`). Infine si possono *proteggere* i singoli caratteri (impedendone l'interpretazione) precedendoli con una barra rovesciata `"\"`.

Un'altra caratteristica estremamente utile della shell è quella che viene chiamata *command expansion*, si può cioè usare come parametro nella nostra riga di comando il risultato di un altro comando, racchiudendo questo fra due apici inversi (`‘`), o nella costruzione `$()`; ad esempio se nel file `elenco` si è scritto una lista di file si potrà effettuarne 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.

Simile alla *command expansion* per la sua sintassi è la cosiddetta *arithmetic expansion* che permette di valutare delle semplici espressioni aritmetiche con una costruzione `$(())` all'interno della quale si inserisce l'espressione da calcolare, che poi sarà sostituita nella linea di comando dal risultato del calcolo. Considerato che si possono usare delle variabili all'interno dell'espressione se ne possono trarre ulteriori potenzialità di utilizzo.

Un'altra delle funzionalità più usate della shell 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 salva ogni linea di comando eseguita in un apposito file, che di default è `.bash_history` (nella nella home dell'utente), ma che può essere cambiato in qualunque altro file specificato dalla variabile `HISTFILE`. Nel file vengono salvate fino ad numero di righe massimo specificato attraverso la variabile `HISTSIZE`.[13]

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 un 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
```

[13] e si può anche indicare una dimensione massima del file con `HISTFILESIZE`.

```
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`.

2.1.4 Modalità di invocazione e "*configurazione*" della shell

Finora abbiamo parlato della shell come il programma che implementa l'interfaccia a riga di comando; in realtà la shell è molto di più, ed è uno dei componenti essenziali per far funzionare il sistema:[14] tutti i sistemi di avvio, come descritto in sez. 5.3.4, usano infatti degli *script di shell* per eseguire le loro operazioni.

Infatti una delle caratteristiche della shell è che, come 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 che è avete a disposizione un vero e proprio linguaggio di programmazione (anche se non molto elegante) con tanto di variabili e direttive di iterazione, condizionali, ecc. che vi permette di effettuare anche compiti di notevole complessità.

La descrizione di questo linguaggio non è affrontabile adesso, le sue caratteristiche si trovano comunque nella sezione `SHELL GRAMMAR` della pagina di manuale. Quello che è importante sottolineare qui è che la shell non solo può essere invocata in maniera interattiva, quando è associata ad un terminale, ma può anche essere lanciata non interattivamente facendole eseguire direttamente quello che appunto si chiama uno `script`, invocandola come un qualunque altro comando cui si passa come parametro il file dello script, con qualcosa del tipo `bash script`.

In realtà gli script possono sfruttare una funzionalità peculiare del *link-loader*[15] di Linux (che si trova comunque in tutti i sistemi unix-like), che oltre ad eseguire i comandi binari nel formato standard[16] è in grado di eseguire come se fossero binari pure gli script, che identifica 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 un comando. In tal caso il *link-loader* lancia automaticamente il comando (con le opzioni che possono seguire sempre sulla prima riga del file) dandogli come parametro il nome file stesso.

In questo modo, dato che per la shell (e per tutti i linguaggi di scripting che usano questa funzionalità, come il python o il perl) il carattere `#` indica l'inizio di una riga di commento, basta far iniziare il nostro script con questa riga, cui far seguire una serie di istruzioni, e poi renderlo eseguibile,[17] per renderlo del tutto equivalente ad un comando binario.

L'esecuzione degli script però 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*[18] a cui fa eseguire

[14] si noti infatti che essa deve trovarsi, come richiesto dal FHS trattato in sez. 1.2.4, sotto `/bin`, dato che è essenziale per l'avvio del sistema.

[15] questo 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.

[16] in 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.

[17] cioè attivarne il permesso di esecuzione con `chmod +x`.

[18] cioè un'altra istanza della shell, che viene eseguito da un altro processo.

lo script. Questo vuol dire ad esempio che se nello script si effettuano operazioni che modificano le proprietà della shell (come le variabili di ambiente o gli alias trattati in sez. 2.1.3) 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`,[19] cui dare come parametro 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 necessari quando la si invoca per eseguire una script.

Un'altra modalità è quella della *shell di login*. Questa è la modalità che viene usata dal programma `login` quando da all'utente un accesso al sistema e che in genere è alla radice di ogni altro programma lanciato dall'utente.[20] Queste modalità in realtà cambiano pochissimo, quello che le contraddistingue è che nel caso di shell di login viene sempre letto ed eseguito (all'interno della shell corrente) il contenuto del file `/etc/profile`, 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, ripreso 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 il `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).

[19] una sintassi alternativa, più sintetica ma anche più criptica, è quella di usare `.` al posto di `source`.
[20] in generale un utente può sempre lanciare, dopo il login, varie altre shell interattive.

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* 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 disponibile in tutti i casi. Un esempio è:

```
# ~/.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.

2.1.5 La redirezione dell'I/O

Abbiamo tenuto per ultima la funzionalità fondamentale della shell, quella che ci fornisce la vera potenza dell'interfaccia a riga di comando, permettendoci di combinare fra loro i vari comandi, il meccanismo che ci permette di costruire la nostra catena di montaggio: la *redirezione 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 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 parametro), e scrivono i loro dati in uscita sullo standard output.[21]

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 parametri) e a scriverne il contenuto sullo standard output. Se non si specifica nessun file come parametro il comando legge, come da standard, sullo 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
```

[21] lo 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 redirige lo standard output, non è così, essendo i due file distinti.

dove il comando reagisce all'invio ristampando sul terminale quanto appena scritto e attendendo nuovo input.[22]

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 c\texttt{more}\texttt{more}ompatible 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`.

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 error su un file specificato dall'utente dopo l'operatore.[23]

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:

[22] questo 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.

[23] in 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`.

```
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 parametri, riscrivendolo sullo standard output, che nel caso è stato rediretto sul nuovo file, per cui quest'ultimo risulterà essere la *concatenazione* del contenuto di tutti i file passati come parametri.

Così se si è spezzato un file di grosse dimensioni in una serie 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
```

si andrebbe incontro ad una sgradita sorpresa. Quello che avviene con la redirezione 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 redirezione si comporta esattamente come `cat`) si sarebbe perso il contenuto originale di `file`.

Per ovviare a questo problema è disponibile un altro operatore di redirezione, `>>`, 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 redirezione di ingresso ed uscita, per quanto utile, è però marginale rispetto ad una forma di redirezione 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*,[24] 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

[24] che 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.

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
`cmd < file`	redirige lo standard input: legge l'input del comando `cmd` dal file `file`.
`cmd > file`	redirige lo standard output: scrive l'output del comando `cmd` sul file `file`.
`cmd >> file`	redirige lo standard output: scrive l'output del comando `cmd` accodandolo al contenuto del file `file`.
`cmd 2> file`	redirige lo standard error: scrive gli errori del comando `cmd` sul file `file`.
`cmd1 | cmd2`	*pipelining*: redirige lo standard output del comando `cmd1` sullo standard input del comando `cmd2`.
`cmd > file 2>&1`	redirige lo standard output e lo standard error: scrive errori e risultati del comando `cmd` sul file `file`.
`cmd1 2>&1 | cmd2`	redirige lo standard output e lo standard error del comando `cmd1` sullo standard input del comando `cmd2`.

Tabella 2.3: Principali modalità di redirezione.

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. 2.3 un riassunto delle principali modalità di redirezione utilizzate nella shell. Comprese alcune delle più esoteriche che permettono di redirigere più file.

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 redirezione 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; queste sono introdotte per convenzione da parametri passati nella linea di comando che iniziano con un carattere `-` e di norma costituite di una sola lettera, ad esempio molto spesso l'opzione `-h` stampa 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 anche

da molti altri, per la presenza di una libreria che automatizza[25] in maniera efficiente la gestione delle opzioni, 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 *verbosità* 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.

I comandi del progetto GNU supportano (attraverso la stessa libreria) 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,[26] in modo tutti i parametri che seguono vengano usati direttamente senza essere considerati una opzione anche se comincia 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 parametri. Per questo scrivendo `-` prima del nome del file l'interpretazione delle opzioni sarà terminata ed il nome (anche se inizia per `-`) sarà preso come parametro. Invece per l'uso del `-` vale la convenzione, per quei comandi che prendono come parametro un file, di considerarlo (a seconda del contesto) come sinonimo dello standard input o dello standard auto.

Un'altra caratteristica comune dei comandi è che tutti riportano 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. Questo è molto importante, specie per gli script che non possono vedere i messaggi di errore e hanno solo questo come informazione; è compito della shell ricevere lo stato di uscita di un comando (essendo lei il processo che lo ha lanciato) ed esso viene sempre memorizzato nella variabile speciale `?`, così che si può visualizzare lo stato di uscita di un comando 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 licidi
ls: licidi: No such file or directory
piccardi@anarres:~/Truelite/documentazione$ echo $?
1
```

e nel primo caso il comando è stato eseguito correttamente, mentre nel secondo no.

[25] essa infatti permette di mettere le opzioni in qualsiasi posizione lungo la linea di comando, per cui non ci si deve preoccupare di doverle specificare prima dei normali parametri come richiesto da alcuni programmi che non la usano.

[26] è un'altra delle funzionalità introdotte dalla libreria di cui parlavamo.

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. 2.1.3, 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 parametro. Il vantaggio di questo programma è la sua velocità, esso infatti non 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 parametro di espressioni analoghe a quelle usate dalla shell per il *file 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` che viene eseguito in genere una volta al giorno (fra i lavori periodici di `cron` che vedremo in sez. 4.3.1), 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 parametro 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. 2.4, 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 2.4: 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. 2.5.

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 2.5: Principali opzioni di `find` per la ricerca.

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 numero di link), che è stato indicato in tab. 2.5 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
`-exec`	esegue un comando usando come argomento il nome del file.
`-print`	stampa il nome del file terminato con un a capo.
`-print0`	stampa il nome del file terminato con un carattere NUL (il valore 0).
`-fprint file`	scrive il nome del file sul file `file`.
`-ok`	come `-exec` ma chiede conferma del comando.

Tabella 2.6: Principali opzioni di `find` per specificare azioni.

L'elenco delle opzioni principali è riportato in tab. 2.6, 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 un 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 ininfluente 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*.[27]

L'uso elementare di `grep` è banale, il comando prende come primo parametro 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
```

Le opzioni principali 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 innumerevoli, ed altrettanto complesse sono le sue capacità di ricerca basate sulle espressioni regolari, al solito si rimanda alla pagina di manuale per i dettagli.

Come gli altri comandi Unix anche `grep` legge, qualora non gli sia passato nessun argomento, dallo standard input e scrive sullo standard output; diventa allora evidente la sua utilità come filtro per selezionare a piacere, sulla base delle opportune corrispondenze le righe di un file. Si noti inoltre come si possano effettuare ricerche sempre più mirate semplicemente concatenando in successione diverse chiamate al comando.

2.2.3 I comandi 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

[27] le *espressioni regolari*, o *regex*, dall'inglese *regular expressions*, sono una specie estensione del sistema del globbing (che abbiamo illustrato in sez. 2.1.3) 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.

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.[28] 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 supportare la riscrittura dello schermo, e che tutte le tastiere ad avere i tasti di freccia, venne creato `less` come *evoluzione*[29] 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.

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 provvede 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 comandi precedenti; basterà tagliare prima la coda del file con `head` e poi la testa con `tail`, costruendo una linea di comando del tipo:[30]

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

[28] in realtà si può usare una *regular expression*, e compiere quindi anche ricerche molto complesse.
[29] si, volevano davvero fare gli spiritosi!
[30] dove si è usata la *arithmetic expansion* brevemente descritta in sez. 2.1.3.

Vale la pena poi menzionare esplicitamente l'opzione `-f` di `tail` che quando usata fa si 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 scrivano in append 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 parametro il file da *affettare* (ma se non lo si specifica al solito legge dallo standard input, consentendo così a esempio di creare un file e suddividerlo al volo), e lo divide in tanti file di dimensione uguale che chiama `xaa`, `xab`, ecc. Aggiungendo un secondo parametro si può specificare un prefisso diverso da `x` per i nuovi file. La dimensione dei file viene specificata con l'opzione `-b` se la si vuole in byte (l'opzione supporta anche i suffissi `m` e `k` per indicare Mb e Kb) o `-C` se le si vuole in linee. Infine se due lettere non bastano per indicizzare i file che si generano si può usare l'opzione `-a` per specificarne un numero diverso.

Se si vuole tagliare un file per colonne invece che per righe si può usare invece il comando `cut`. Il comando opera sul file passato come parametro (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--
```

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
```

Come contraltare a `cut`, il comando `paste` permette di concatenare file diversi in colonna. Il comando prende come parametri il nome 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 indicandola 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 Comandi vari

Altri comandi di uso abbastanza comune sono `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 parametro 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. 1.2.2). 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 parametro 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 comando che permette di filtrare il contenuto di un file è `uniq`, che elimina le linee adiacenti uguali; il comando prende come parametro un nome di file (ma se non viene specificato legge lo standard input) e stampa il risultato sullo standard output. Al solito le varie 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`, che come il nome (*Word Count*) viene usato per contare le parole contenute in un file. Il comando prende come parametro 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*,[31] che confrontate permettono di verificare l'integrità di un file). Entrambi prendono come parametri 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 redirezione 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.[32]

Come accennato in sez. 2.1.3 ciascun comando di norma supporta da suo una opzione `-help` che permette di visualizzarne brevemente la sintassi. Dato che questa informazione è in genere 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 dopo il nome del comando in maiuscolo nella prima riga di ciascuna pagina. Ciascuna sezione sezione contiene la documentazione relativa ad un certo argomento, secondo quanto riportato in tab. 2.7.

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 sempre si accede con `man man`.

Si tenga presente che il comando `man` richiama la pagina relativa al nome che si è passata come parametro, cercando in sequenza 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

[31] si chiamano così delle opportune funzioni matematiche, dette anche *hash* 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.

[32] sigla che sta, a seconda dell'umore del momento, per `Read The Fine Manual` o `Read The Fucking Maunal`.

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 /dev.
(5)	formati dei file di configurazione.
(6)	giochi.
(7)	varie (convenzioni, informazioni generiche su argomenti).
(8)	comandi di amministrazione.

Tabella 2.7: Sezioni delle pagine di manuale.

comando passwd e per il file /etc/passwd) verrà mostrata 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 permette però di verificare se esistono più pagine associate allo stesso nome con il comando whatis, che ne stampa l'elenco delle pagine 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 parametro nelle descrizioni brevi del comando che compaiono nella intestazione della pagina di manuale (come quelle appena mostrate nell'output di whatis) per cui potremo eseguire:

```
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, dei file che aggiunge, rispetto alle pagine di manuale, delle caratteristiche più avanzate e molto comode come la possibilità di navigare usando link anche 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 Info installati, e sarà possibile iniziare la navigazione spostandosi con le freccie, 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/`.

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.[33] Di solito il kernel lo legge all'avvio per impostare il valore

[33] è l'orologio che si trova sulla scheda madre, ed è alimentato della batteria che si trova su di essa.

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.[34]

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[[CC]YY][.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. 2.8, 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. 2.8, ma anche descrizioni verbali (solo in inglese, però) come `8 day ago`, o `1 mounth`. La descrizione di queste stringhe può essere 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:

[34] il 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 2.8: Direttive di formattazione per il comando `date`.

```
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 Comandi di ausilio per la redirezione

Nonostante la grande flessibilità degli operatori di redirezione 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 redirezione.

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 i parametri di quest'ultimo, che provengono dalla linea di comando.

È però possibile provvedere 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 i parametri, effettuando invece la lettura dei parametri dallo standard input, così da poterli ricevere da un comando precedente tramite una pipe.

Il comando `xargs` prende come parametri 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 parametri, separati da spazi, così come letti dallo standard input. È possibile proteggere la presenza di spazi all'interno dei parametri così come si fa con la shell, usando le virgolette o la barra trasversa. Se non si passa nessun parametro 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 su 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 redirige 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`.

2.3.4 Comandi vari

Concludiamo riportando alcuni comandi di varia utilità che non rientrano direttamente in nessuna categorie a se stante. Il primo di questi è `yes`, che ha il semplice compito (se invocato senza parametri) 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`). Il comando non ha nessuna opzione specifica e prende come parametro una stringa, che verrà usata al posto di `y`.

Un comando simile è `seq`, che invece di una stringa costante permette di stampare una sequenza di numeri, il comando può essere invocato con un numero di parametri variabile da uno a tre numeri interi: un solo parametro indica il numero finale a cui fermare la sequenza, partendo da 1 ed incrementando di uno, specificando due parametri si indicano il valore iniziale ed il valore finale, che viene raggiunto sempre a passi di uno, infine con tre parametri il secondo viene usato come valore di incremento nell'intervallo che inizia con il primo e finisce col terzo.

Questo comando è molto utile soprattutto negli script, quando si vogliono fare dei cicli, lo si può infatti usare per generare una lista di numeri con `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 ritornare alla shell rispettivamente una 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, o `false` che invece restituisce un valore non nullo, equivalente alla condizione falsa e al fallimento del comando. Questi vengono usati spesso come shell di default (vedi sez. 3.2.2) per gli account degli utenti di sistema che non devono avere una shell.

2.4 Gli editor

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

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

2.4.1 Introduzione

Come abbondantemente evidenziato in sez. 2.1 Linux nasce come tutti gli Unix con l'interfaccia a linea di comando, e tutte le funzionalità del sistema sono accessibili fin da quel livello.

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

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

Dato che, come vedremo meglio in cap. 4, una delle caratteristica fondamentali di Linux è quella di tenere tutte le configurazioni in file di testo, uno dei programmi più usati da ogni amministratore di sistema è *l'editor di testi*. Nonostante la nascita di tante interfacce di configurazione grafica infatti, il completo controllo sul comportamento di un programma si può avere soltanto attraverso i relativi file di configurazione, per cui l'editor resta lo strumento principale per una amministrazione professionale di un sistema unix-like.

Per questo in questa sezione daremo una breve panoramica sull'uso dei più comuni. Inoltre restando nell'ottica dell'amministrazione base parleremo esclusivamente di editor accessibili da console, e quindi utilizzabili anche attraverso connessioni remote con dei semplici modem.

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

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

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

Per molti `emacs` è l'editor. Sicuramente è il più potente: dentro `emacs` si può davvero fare di tutto. Esempi sono: navigare fra i file e in internet, leggere la posta e le news, programmare (con evidenziazione della sintassi e scorciatoie ai costrutti principali di qualunque linguaggio), fare debug, scrivere dispense come queste, giocare (a tetrix o a qualche avventura testuale), farsi psicanalizzare dal doctor.

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

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

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

2.4.3 La sintassi di emacs

Come per tutti gli editor una sessione di emacs si inizia con il comando `emacs nomefile`, dato il comando si otterrà una finestra come quella mostrata in fig. 2.1: nella prima riga si trova il menu dei comandi (cui si accede con F10), ad esso segue la sezione principale, (vuota nell'esempio) dove compare il testo del file ed in cui ci si muove con le frecce, una barra di stato (quella che comincia per `--`) in cui compaiono varie informazioni (il nome del file, se sono state fatte modifiche, la modalità[35]; e nella la riga finale il cosiddetto *minibuffer*, dove compaiono brevi messaggi (come nell'esempio, che riporta la scritta `(New file)`) ed in cui si scrivono i comandi. In certi casi, in corrispondenza dei comandi inviati, la sezione principale può venire automaticamente divisa in 2 parti per permettere di inserire o mostrare ulteriori informazioni.

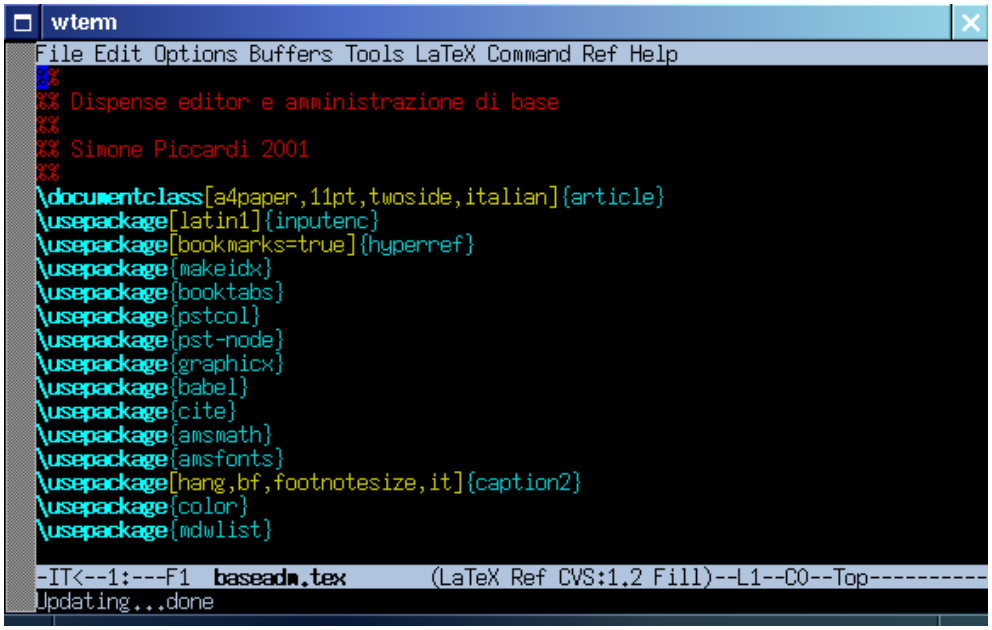

Figura 2.1: Schermata di avvio dell'editor `emacs`.

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

Quella che segue è la lista dei comandi principali:

- aprire un file: `C-x C-f`
- salvare un file: `C-x C-s`
- salvare con nome: `C-x C-w nomefile`
- uscire: `C-x C-c` chiede se salvare le eventuali modifiche
- annullare: `C-_`, `C-`
  o `C-x u`, ripetendo si torna ulteriormente indietro nell'annullamento
- seleziona: `C-x spazio` all'inizio e poi spostarsi con le frecce
- taglia: `C-w` sulla regione selezionata

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

- incolla: `C-y`
- cancella: `C-d` in avanti e `backspace` indietro
- ricerca: `C-s testo` ricerca incrementale sul testo specificato, `C-s` cerca il successivo `C-r` cerca il precedente.
- help: `C-h ?` poi scegliere nella finestra quello che si vuole
- annulla (comando): `C-g` o `ESC ESC ESC`

2.4.4 Editor: `vi`

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

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

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

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

2.4.5 La sintassi di `vi`

Come accennato `vi` è un editor modale, il comando `vi nomefile`, apre il file in modalità comando in una finestra principale (mostrata in fig. 2.2), dove compare il testo (essendo il file nuovo nella figura le righe vuote sono sostituite da `~`) ed in cui ci si muove con le frecce, lasciando libera l'ultima linea, usata per dare i comandi o ricevere le informazioni (nel caso il fatto che il file è nuovo e che si è sulla prima riga).

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

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

Quella che segue è la lista dei comandi principali:

- aprire un file: `:ex file`
- salvare un file: `:w`
- salvare con nome: `:w nomefile`

```
wterm
%%
%% Dispense editor e amministrazione di base
%%
%% Simone Piccardi 2001
%%
\documentclass[a4paper,11pt,twoside,italian]{article}
\usepackage[latin1]{inputenc}
\usepackage[bookmarks=true]{hyperref}
\usepackage{makeidx}
\usepackage{booktabs}
\usepackage{pstcol}
\usepackage{pst-node}
\usepackage{graphicx}
\usepackage{babel}
\usepackage{cite}
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage[hang,bf,footnotesize,it]{caption2}
\usepackage{color}
\usepackage{mdwlist}

%\usepackage{footnote}
%\usepackage{mdwtab}
baseadm.tex: unmodified: line 1
```

Figura 2.2: Schermata di avvio dell'editor `vi`.

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

Data la sua *età* `vi` supporta, oltre alle classiche frecce, lo spostamento nel file con i tasti `h`, `j`, `k`, `l` (e nelle versioni più *ridotte*, solo con questi) che effettuano rispettivamente lo spostamento a sinistra, basso, alto e destra. Inoltre una caratteristica peculiare è che qualunque comando di questo tipo può essere moltiplicato se lo si fa precedere da un numero, per cui per abbassarsi di 10 righe si può scrivere qualcosa tipo `10j`.

Infine 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. È possibile però cancellare caratteri (con `x` per quello corrente, con `X` il precedente), parole (con `dw`) e righe (con `dd`), ed usare i moltiplicatori appena illustrati per selezioni multiple, così come si può incollare (di nuovo più volte se si usano i moltiplicatori) quanto appena cancellato con `p`.

2.4.6 Editor: `joe`

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

di solito il comando si può usare in modalità emulazione (ad esempio invocandolo con `jmacs`) usando le sintassi di altri editor.

Il comando `joe nomefile`, apre il file in una finestra principale, mostrata in fig. 2.3, dove una riga di stato in testa, seguita dalla sezione principale in cui compare il testo (nell'esempio viene mostrata la scritta `New File`) ed in cui ci si muove con le frecce, l'ultima linea viene usata per mostrare i messaggi e per dare i comandi o ricevere le informazioni, e scompare quando non è più necessaria.

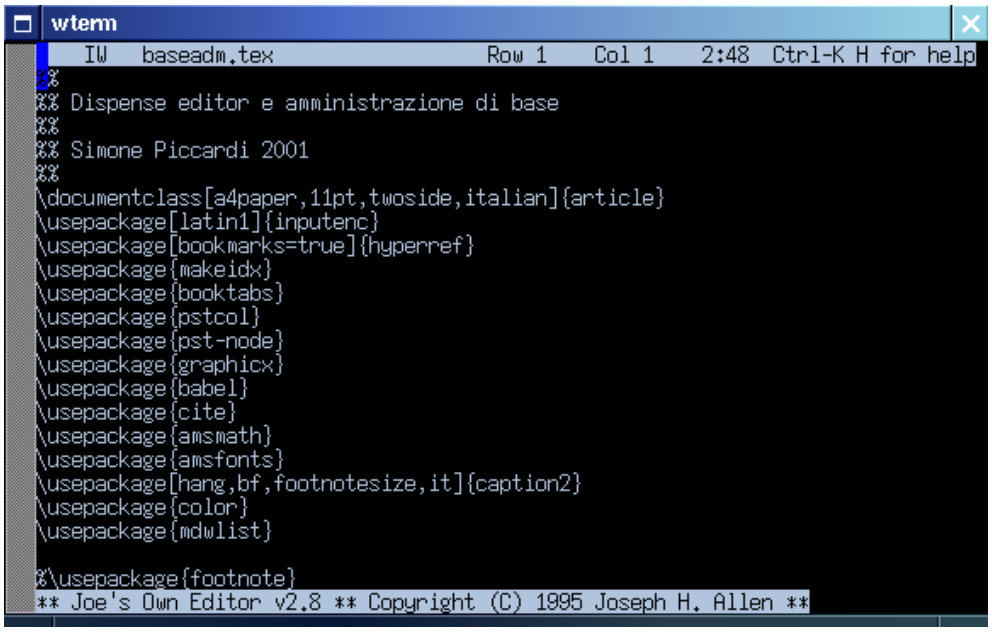

Figura 2.3: Schermata di avvio dell'editor `joe`.

Quella che segue è la lista dei comandi principali:

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

2.4.7 Editor: jed

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

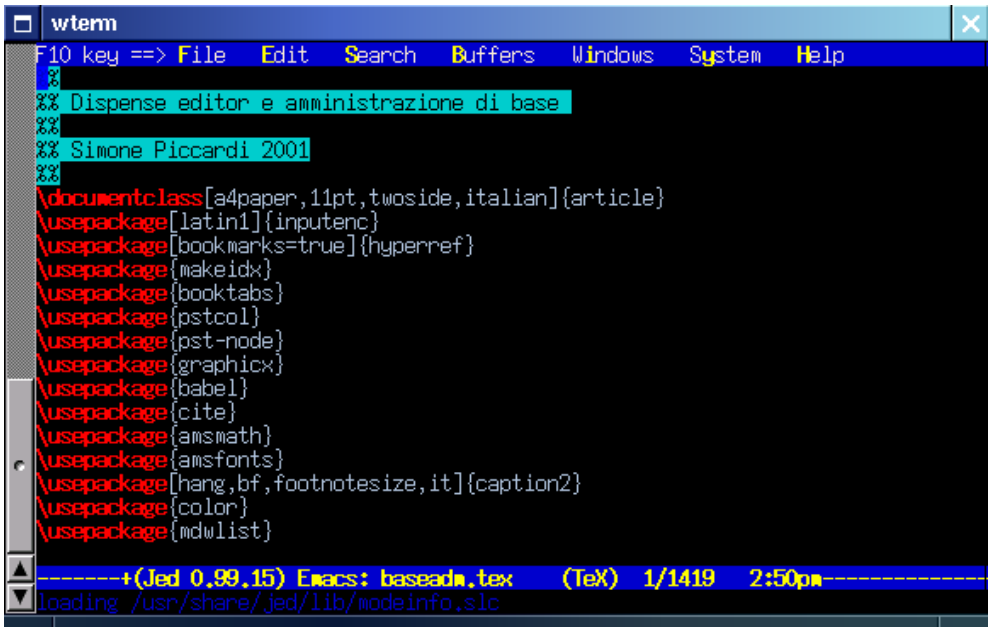

Figura 2.4: Schermata di avvio dell'editor `jed`.

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

2.4.8 Editor: `pico` e `nano`

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

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

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

2.4.9 Gli altri editor

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

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

Figura 2.5: Schermata di avvio dell'editor `nano`.

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

Capitolo 3

Amministrazione ordinaria del sistema

3.1 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.

3.1.1 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.

In genere il software viene distribuito in forma sorgente in degli archivi compressi detti *tarball* in quanto sono creati con il programma `tar`, che permette di archiviare il contenuto di una directory all'interno di un unico file di archivio, conservando tutte le informazioni relative ai vari file (permessi, tempi, ecc.). Il comando permette anche di comprimere al volo l'archivio, che di norma viene poi creato con un nome che usa l'estensione `.tar.gz`[1] dato che il programma di compressione 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 sorgente la prima cosa da fare è scaricarsi l'archivio degli stessi dal sito di sviluppo dello stesso: è consigliato usare i vari *mirror* disponibili, dato che questi di norma sono meno *occupati* del sito originale, e probabilmente anche più *vicini* 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`.

Il comando prende varie opzioni (che possono essere raggruppate insieme e specificate anche senza l'uso del `-`), le principali sono `c`, che permette di creare un archivio, `t` che ne verifica il contenuto, e `x` che lo decomprime. Con `f` si indica il file che contiene l'archivio (che deve essere specificato immediatamente dopo, per cui se si raggruppano le opzioni questa deve andare per ultima). Infine con `z` si indica di utilizzare il programma di compressione `gzip` mentre con `j` si indica `bzip2`. Infine si consiglia l'uso di `v` che durante l'esecuzione delle varie operazioni stampa a video il nome dei file su cui si sta operando.

Pertanto se il nostro pacchetto software è `pacchetto.tar.gz` se ne potrà verificare il contenuto con il comando `tar -tvzf pacchetto.tar.gz` e lo si potrà decomprimere con `tar -xvzf`

[1] talvolta abbreviata in `tgz`, per l'uso di sistemi obsoleti che hanno problemi coi nomi di file che hanno troppi caratteri "`.`" al loro interno.

`pacchetto.tar.gz`; questo creerà 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* GNU[2] 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. In genere questi sono disponibili nei dischi di installazione come pacchetti con lo stesso nome delle librerie relative, estesi con un `-dev`. In questo caso quello che c'è da fare è installare questi pacchetti con il relativo meccanismo di gestione, 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 la 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 è la procedura standard prevede di installare i pacchetti compilati dai sorgenti in `/usr/local/`, per cui per poterli utilizzare dovete avere le directory di questa gerarchia secondaria nel PATH (vedi sez. 2.1.3) ed essere in grado di usare le eventuali librerie installate dal pacchetto (secondo quanto vedremo in sez. 4.1.2).

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

[2] un 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.

purtroppo) provvedono uno speciale *bersaglio*[3] 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.

3.1.2 La gestione dei pacchetti con `rpm`

Uno dei primi sistemi di gestione dei pacchetti, il *RedHat Package Manager* è stato introdotto dalla RedHat nella sua distribuzione; data la sua efficacia esso è stato poi adottato da molte altre distribuzioni, divenendo probabilmente il più diffuso sistema di gestione automatizzata dei pacchetti.

Il *RedHat Package Manager* si basa su un programma, `rpm`, che serve a gestire l'installazione e la rimozione dei pacchetti, i quali sono distribuiti in file in un apposito formato, anch'esso caratterizzato dalla estensione `.rpm`. In realtà RPM fa molto di più che installare e disinstallare. Il comando tiene traccia di ogni pacchetto installato e di dove è stato installato, 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 permettono di norma anche una configurazione automatica del pacchetto stesso.

Il comando è complesso e prende molteplici opzioni, come molteplici sono le funzionalità che provvede. Le due opzioni più semplici sono `-i` che installa un pacchetto e `-e` che lo cancella. 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. 3.1, la descrizione dettagliata del comando con l'elenco completo delle opzioni sono al solito disponibili nella relativa pagina di manuale.

Opzione	Significato
`-i`	Installa il pacchetto.
`-U`	Esegue l'upgrade del pacchetto.
`-e`	Rimuove il pacchetto.
`-q`	Interroga il database per la presenza di un pacchetto.
`-qa`	Stampa tutti i pacchetti installati.
`-ql`	Stampa i file contenuti in un pacchetto.
`-qf`	Stampa il pacchetto che contiene un file.
`-V`	Verifica lo stato di un pacchetto.
`-K`	Verifica la firma digitale di un pacchetto.
`-v`	Stampa più informazioni nell'esecuzione delle operazioni.
`-h`	Stampa una barra di progressione.

Tabella 3.1: Principali opzioni del comando `rpm`.

Una delle funzionalità più importanti di un gestore di pacchetti è quella di essere in grado

[3] si chiamano così, dall'inglese *target*, i parametri che di norma si passano a `make` e che indicano in genere 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, in genere 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.

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).

Nel caso in questione `rpm` è in grado, sia pure in forma non troppo sofisticata (il che porta a quella situazione che viene chiamata *dependency hell*, quando un pacchetto dipende da un'altro che a sua volta dipende da una altro che a sua volta ... e così via) di avvisare l'utente quali sono i file che devono essere già presenti nel sistema perché un certo pacchetto possa essere installato. In questo modo il programma permette di evitare di installare pacchetti che non funzioneranno a causa della mancanza di un qualche componente a loro essenziale. Il meccanismo però si limita a questo, sta all'amministratore trovare, scaricare ed installare i pacchetti mancanti.

3.1.3 La gestione dei pacchetti di Debian

Uno dei più grandi vantaggi di Debian è proprio il suo sistema di gestione dei pacchetti. Esso è basato su un programma di gestione dei singoli pacchetti con funzionalità analoghe a quelle di `rpm`, ma sopra di esso è stata costruita una infrastruttura molto più complessa che rende estremamente semplice e funzionale la gestione dei pacchetti.

I pacchetti Debian sono distribuiti in file con l'estensione `.deb` ed usano un formato diverso rispetto agli RPM; il programma che permette di installare questi pacchetti è `dpkg`, ed è sostanzialmente un analogo (in realtà c'era da prima) di `rpm`, in quanto provvede le stesse funzionalità: installa e rimuove pacchetti mantenendo un database dei pacchetti e dei relativi file installati, provvede l'esecuzione di script in fase di installazione e rimozione, è in grado di accorgersi di eventuali conflitti con pacchetti già installati, e delle dipendenze da altri pacchetti, può interrogare il database dei pacchetti e ottenere informazione sul loro contenuto.

Le opzioni principali del pacchetto sono `-i` che esegue l'installazione di un file `.deb` e `-r` che rimuove un pacchetto dal sistema. Inoltre con `-l` viene stampata la lista dei pacchetti installati (se non si specifica nulla, se si specifica un nome viene ricercata la presenza di un pacchetto con quel nome), con `-L` si stampa la lista dei file contenuti nel pacchetto e con `-S` quella dei pacchetti che contengono un file corrispondente alla stringa passata come parametro. Le altre opzioni principali sono riportate in tab. 3.2, al solito le istruzioni complete e tutte le altre opzioni sono descritte nella pagina di manuale.

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 *stato* di un pacchetto.
`-p`	Stampa informazioni su un pacchetto installato.
`-I`	Stampa informazioni su un `.deb`.
`-c`	Stampa il contenuto di un `.deb`.

Tabella 3.2: Principali opzioni del comando `dpkg`.

Benché ormai anche i `.deb` risultino piuttosto usati (dato che sono diventate parecchie le distribuzioni basate su Debian) la loro diffusione su internet come file a se stante è piuttosto ridotta. Questo avviene perché in realtà `dpkg` è solo la parte di basso livello del sistema di gestione dei pacchetti di Debian, ed è piuttosto raro dover usare questo comando per installare un pacchetto.

Uno dei grandi vantaggi di questa distribuzione (e di quelle che usano il suo sistema di gestione) è costituito dalla interfaccia di alto livello chiamata *Advanced Package Tool*, che permette di cancellare completamente il problema delle dipendenze. I pacchetti Debian infatti sono organizzati per indicare in maniera coerente da quali altri pacchetti essi dipendono, diventa 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 a detta directory (cosa che può essere fatta in una molteplicità di modi, i principali dei quali sono via HTTP o via FTP).

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, ed 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 nel 90% dei casi è comunque 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.

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.

Opzione	**Significato**
`install`	Installa un pacchetto.
`remove`	Rimuove un pacchetto.
`clean`	Cancella l'archivio dei pacchetti scaricati.
`update`	Scarica la lista aggiornata dei pacchetti.
`upgrade`	Esegue l'upgrade dei pacchetti aggiornati.
`dist-upgrade`	Esegue l'upgrade della distribuzione.
`autoclean`	Cancella dall'archivio i pacchetti con vecchie versioni.

Tabella 3.3: Principali opzioni del comando `apt-get`.

Le opzioni principali di `apt-get` sono riportate in tab. 3.3, l'elenco completo, insieme alla descrizione dettagliata di tutte le caratteristiche del comando, è al solito disponibile nella relativa pagina di manuale.

3.2 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.

3.2.1 I comandi per la gestione degli utenti e gruppi

Nelle prime versioni di Unix tutte le informazioni relative ad utenti e gruppi presenti nel sistema erano memorizzate in due file, `/etc/passwd` e `/etc/group`, su cui torneremo in sez. 3.2.2. Dato che come tutti i file di configurazione del sistema anche questi sono file di testo, in teoria non ci sarebbe nessuna necessità di programmi specifici per che vadano ad operare su detti file, dato che possono essere modificati a mano con un qualunque editor.[4]

Nei sistemi moderni però il meccanismo di gestione di utenti e gruppi è stato completamente modularizzato attraverso l'uso di PAM (un acronimo che sta per *Pluggable Authentication Method*) che permette di mantenere i dati ed effettuare i relativi controlli usando i supporti più disparati (server NIS, vari database, server LDAP, ecc.). In questo caso non è più possibile andare ad effettuare le modifiche a mano con un editor. L'uso di un supporto modulare però fa sì che si possano utilizzare in maniera trasparente gli stessi comandi che in origine operavano solo sui file di testo.

Il primo comando che prendiamo in esame è `useradd`, che permette di aggiungere un nuovo utente al sistema. Il comando prende come parametro 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.

É possibile comunque impostare ciascuna di queste proprietà (e molte altre) attraverso le opportune opzioni: ad esempio `-p` permette di specificare[5] la password dell'utente, `-s` la shell, `-G` eventuali altri gruppi di appartenenza, `-m` la creazione della home directory, con tanto di creazione di tutti i file contenuti nella directory `/etc/skel` (vedi sez. 4.2.4). 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 99[6] e più grande di tutti gli altri valori utilizzati per gli altri utenti.

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 `/etc/skel` nella home.
`-g`	imposta il gruppo di iniziale.
`-G`	imposta eventuali gruppi aggiuntivi.
`-o`	permette di specificare un user ID già esistente.

Tabella 3.4: Principali opzioni del comando `useradd`.

Le principali opzioni sono riportate in tab. 3.4, 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 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

[4] cosa 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.

[5] che 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.

[6] i valori fra 0 e 99 sono usati normalmente per gli utenti corrispondenti ad alcuni servizi di sistema.

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. 3.4 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. 3.2.2).

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.

Analogo a `usermod` è invece il comando `groupmod` che permette di modificare un gruppo, in particolare il comando permette di cambiare il group ID usando `-g` (e supporta sempre l'opzione `-o` per poter specificare un gruppo già assegnato), e 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 directory[7] 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 gruppo,[8] 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 attribuiti, 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 parametro; 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 parametro questo indica l'utente di cui si vuole cambiare la password, ma si 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. 3.2.2), dette opzioni con il relativo significato sono riportate in tab. 3.5, e possono essere utilizzate solo dall'amministratore.

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

[7] si 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.

[8] gestendo anche, rispetto ai corrispondenti programmi base, l'eliminazione dei file non contenuti nella home.

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 3.5: Opzioni del comando `passwd` per la gestione delle informazioni relative alle *shadow password*.

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 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 il nome del (gruppo o dell'utente) del quale si vogliono avere i diritti e richiedono la relativa password. Se usati on 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. 4.2.5) e `chfn` che permette di cambiare le informazioni mantenute nel campo chiamato *Gecos* (vedi sez. 3.2.2), 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.

3.2.2 Il database degli utenti

Come accennato in sez. 3.2.1 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 delle *shadow password*, 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 tutti i programmi ottengono il nome che corrisponde ad un certo user ID). Nel file sono mantenute poi anche altre informazioni come la shell di default, 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 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. 3.2.1) in casi di emergenza può essere utile modificare a mano il file. Il formato del file è molto semplice, per ogni utente deve essere presente una riga composta da 7 campi separati dal carattere `:`, non sono ammessi né commenti né righe vuote. Il significato dei sette campi è il seguente:

- nome di login (user name)
- password cifrata (opzionale)
- identificatore numerico dell'utente (user ID)
- identificatore numerico del gruppo di default
- nome e cognome dell'utente, con eventuali altri campi di commento, separati da virgole. Il campo è detto anche *Gecos*.
- home directory dell'utente
- shell di default

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`.[9]

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 editare questo file, ad esempio se si è persa la password di root, nel qual caso occorrerà far partire il computer con un disco di recupero e togliere la `x` lasciando il campo vuoto, così che la richiesta della password venga disabilitata al successivo riavvio, in modo da poter entrare per ripristinarla.

[9] si ricordi quanto detto in sez. 2.3.1 riguardo le sezioni delle pagine di manuale.

Analogo a `/etc/passwd` per mantenere l'elenco dei gruppi c'è `/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
- la password del gruppo
- il valore numerico del 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
...
piccardi:x:1000:
...
```

Benché l'algoritmo crittografico con cui si calcolano le password sia piuttosto robusto, mantenere leggibili le password cifrare espone comunque ad un attacco a forza bruta (cioè alla possibilità che qualcuno tenti di provarle tutte) e 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 se pure 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 il 3 al posto della e o l'1 al posto della i. Tutti trucchetti ampiamente noti e banali da reimplementare in un programma di password cracking, che li riapplicherà alle parole del suo dizionario.

Per questo motivo non è comunque molto sicuro lasciare leggibili a tutti le password cifrate, che potrebbero essere soggette ad una analisi di questo tipo. Per questo 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 ulteriori.

Il file che contiene le password cifrate è `/etc/shadow`, ed in questo caso è protetto in lettura così che solo l'amministratore può accedervi. In esso oltre alle password sono memorizzate una serie di informazioni ulteriori che permettono un controllo molto più dettagliato su di esse, con date di scadenza, date in cui sono state cambiate ecc. Il formato del file è sempre lo stesso, i campi sono 9 e sono sempre separati con dei `:`, il loro contenuto è:

- nome di login (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:n34MlzgKs8uTM: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:$1$KSRp2lZ3$s9/C2ms0Ke9UTaPpQ98cv1:11189:0:99999:7:::
...
```

di nuovo i dettagli 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.

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 del file è:

```
root:*::
daemon:*::
bin:*::
sys:*::
adm:*::
tty:*::piccardi
...
piccardi:x::
...
```

Capitolo 4

I file di configurazione ed i servizi di base

4.1 I file di configurazione

In questa sezione tratteremo in maniera generica la gestione 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 alcuni dei file di configurazione del sistema, come quelli che controllano il comportamento delle librerie dinamiche e quelli usati per il login, rimandando la trattazione di quelli relativi ad altri servizi di sistema nel prosieguo del capitolo. Si tenga comunque presente che alcuni file di configurazione (in particolare `fstab` e `mtab`) sono già stati trattati in precedenza (in sez. 1.2.5).

4.1.1 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. 1.2.4, 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 da opportuni programmi (che non è affatto detto siano sempre gli stessi). Ciò comporta che i formati dei file di configurazione possano essere anche i più vari, dipendendo ciascuno dalla convenzione 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 applicate.

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

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

ha invece il grande vantaggio che le modifiche ad un singolo file 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.

Una seconda cosa di cui bisogna tenere conto è che Unix 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 impostate per tutto sistema.

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

4.1.2 La gestione e configurazione 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 sono chiamate *shared object*. Questo è il meccanismo che permette di inserire tutto il codice comune usato dai programmi all'interno di opportune librerie,[1] in modo che non sia necessario reinserirlo tutte le volte all'interno di ciascun programma.

Perché però poi detto codice possa essere utilizzato dai singoli programmi occorre una apposita infrastruttura; come brevemente accennato in sez. 2.1.4 uno dei compiti fondamentali del sistema, quello di eseguire i programmi, viene eseguito attraverso un programma speciale, il *link-loader*, che non è prettamente un programma a se stante, ma una parte di codice che viene sempre eseguita preventivamente tutte le volte che si deve lanciare ogni programma[2] e che permette di identificare le librerie condivise che contengono le funzioni necessarie al programma stesso, e di caricarle automaticamente in memoria, in maniera trasparente all'utente.

Per verificare quali librerie 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 detto comando ha bisogno. Il comando prende come argomento il pathname assoluto del comando 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)
```

[1] e abbiamo visto in sez. 1.2.4 come ci siano delle directory specifiche previste dal *Filesystem Hyerarchy Standard* che devono contenere i relativi file.

[2] a meno che questo non sia stato compilato staticamente, cioè in maniera da includere al suo interno tutto il codice che altrimenti sarebbe disponibile attraverso delle librerie condivise.

e si può usare l'opzione `-v` per avere una descrizione più dettagliata, o altre opzioni che al solito sono descritte nella relativa pagina di manuale.

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*.[3] 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 usani ne abbiano a risentire,[4] 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.[5]

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 effetturare la ricerca in maniera efficiente viene utilizzato un apposito file di *cache* in cui tutte le informazioni relative alla funzioni presenti ed alle librerie in cui esse sono mantenute sono state indicizzate, in modo da rendere la ricerca veloce. 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; come dicevamo infatti queste vengono sempre cercate per major version, ma la libreria installata avrà comunque una 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        26104 Sep 21 14:56 librt-2.3.2.so
lrwxrwxrwx    1 root     root           14 Sep 22 14:44 librt.so.1 -> librt-2.3.2.so
```

se invocato con l'opzione `-v` il comando stampa tutte le librerie usate nella ricostruzione, mentre con `-N` e `-X` si blocca 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 queste possono essere specificate con un opportuno file di configurazione, `/etc/ld.so.conf`, che contiene la lista delle directory che contengono le librerie condivise usate dai programmi 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 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
```

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 una lista di directory dove effettuare la ricerca direttamente sulla linea di comando. Per le altre opzioni

[3] al solito è una convenzione, 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.

[4] 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.

[5] questo è il motivo per cui il problema noto come *DLL hell* presente in windows, dove questa distinzione non esiste, non è presente in un sistema unix-like.

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 lista di ulteriori directory[6] in verrà effettuata la ricerca (questa volta senza usare l'indicizzazione, per cui il sistema è più lento) per altre librerie.

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 programmi[7] del sistema che continueranno ad usare la versione abituale.

4.1.3 Il *Name Service Switch* e `/etc/nsswitch.conf`

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, queste informazioni sono memorizzate in specifici file di configurazione mantenuti sotto `/etc` (ne tratteremo alcuni in seguito, ad esempio quelli relativi alla gestione di utenti e gruppi in sez. 3.2). I sistemi moderni però permettono di mantenere queste informazioni anche in maniera diversa, ad esempio su un database centralizzato, o su un server LDAP.[8].

Il *Name Service Switch* è una estensione alle funzionalità delle librerie standard del C che permette di mantenere, in maniera modulare, queste informazioni su una serie di supporti diversi, e di specificare al sistema deve cercarle ed in quale ordine utilizzare le varie fonti.

Il file che permette al sistema di specificare su quale supporto si trovano le varie classi di informazioni e il relativo ordine di utilizzo è appunto `/etc/nsswitch.conf`; il suo formato usuale è qualcosa del tipo:

```
# /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.

passwd:         compat
group:          compat
shadow:         compat

hosts:          files dns
networks:       files

protocols:      db files
services:       db files
ethers:         db files
rpc:            db files
```

[6] nella stessa forma usata per `PATH`, cioè separate da dei "`:`".

[7] o 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.

[8] un acronimo che sta per *Lightweight Directory Access Protocol*, un protocollo dedicato alla gestione generica di elenchi di informazioni, che viene implementato tramite un server di rete.

```
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. In genere la prima colonna, terminata da un `:` indica il tipo di servizio, di seguito vengono elencate, separate da spazi, le modalità con cui questo viene fornito.

Per ciascuna di queste 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_NOME.so.X` (dove `X` fa riferimento alla versione corrente delle `glibc`).

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.

4.1.4 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 il login.

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. 4.1, l'elenco completo è riportato nella pagina di manuale di `getty`.

Opzione	Significato
`\d`	data.
`\l`	nome del terminale.
`\m`	architettura (i486, ppc, ecc.).
`\n`	hostname.
`\r`	versione del kernel.
`\s`	nome del sistema (Debian, RedHat, ecc).
`\t`	ora.

Tabella 4.1: 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 login effettuati via rete; se invece si usa `ssh` detti file vengono completamente 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 controlla usa serie di funzionalità della 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. 3.2). 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
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.[9]

4.2 Altri file

Raccogliamo in questa sezione le informazioni relative ad una serie di altri file di configurazione relativi a caratteristiche del sistema che non sono facilmente raggruppabili sotto un fattore comune.

4.2.1 Il file `rc.local`

Abbiamo esaminato la procedura di avvio del sistema in sez. 5.3.4, da quanto detto allora è evidente che per far eseguire un proprio script basterà creare un opportuno link simbolico e se lo si vuole eseguire per ultimo basterà usare un numero alto; in alcune distribuzioni però si preferisce usare un file specifico per eseguire dei comandi dopo che tutta la procedura di avvio è stata completata.

In genere questo file è appunto `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 startup, 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.

[9] non è comunque una buona idea neanche quella di usare `telnet`, quindi è meglio se proprio lasciate perdere la cosa.

4.2.2 Il file /etc/hostname

Una caratteristica specifica di ciascuna macchina è il suo nome, comunemente detto *hostname*, che è quello che viene stampato da vari comandi (come `uname`, il prompt della shell o le schermate di avvio). Benché questo 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 impostato attraverso l'uso del comando `hostname`.

Di norma buona parte delle distribuzioni chiedono il nome in fase di installazione e lo memorizzano nell'omonimo file `/etc/hostname` (in alcune viene invece usato `/etc/HOSTNAME`) che viene usato negli script di avvio per leggere il nome della macchina quando questo viene impostato con il comando `hostname`. Perciò si deve essere consapevoli che fintanto 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.

4.2.3 Il file /etc/hosts

Questo file serve per associare nomi e numeri IP. Benché il file faccia riferimento ad una caratteristica della rete, esso deve comunque essere presente anche quando la macchina non è in rete, in quanto esiste comunque l'interfaccia locale, e molti programmi fanno comunque riferimento a questo file per la risoluzione del nome.

Quando si è in rete conviene specificare in questo file la risoluzione dei nomi delle macchine a cui si accede più di frequente in modo da evitare di effettuare la richiesta di risoluzione del nome via DNS. Se la macchina è isolata conviene assegnare al *localhost* il nome della macchina.

Se si è attivata un'interfaccia di rete è sempre utile associare ad essa il nome della propria macchina, altrimenti di nuovo sarebbe invocato il DNS, che in caso di mancanza di connessione può portare anche a lunghi ritardi (dovuti all'attesa di una risposta dal DNS) nella partenza dei programmi più comuni.

Il formato del file è molto semplice, ogni linea definisce una associazione fra indirizzo numerico e indirizzo simbolico, le linee vuote o che iniziano per `#` vengono ignorate. La pagina di manuale fornisce una descrizione completa. Le singole righe hanno il formato:

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

Un esempio di questo file, così come installato sulla mia macchina di casa, è il seguente:

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

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

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

al solito le informazioni complete sul formato del file sono riportate nella relativa pagina di manuale.

4.2.4 La directory `/etc/skel`

Come accennato in sez. 3.2.1 è possibile creare uno *scheletro* del contenuto della home di un nuovo utente in modo che questo sia automaticamente disponibile tutte le volte che se ne crea uno.

La directory `/etc/skel`, come il nome suggerisce, è quella che contiene questo scheletro. Tutti i file e le directory che si vuole siano creati nella home 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.

4.2.5 Il file `/etc/shells`

Questo è il file che contiene la lista delle shell valide che l'utente può selezionare con il comando `chsh`. 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 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`.

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; ma non potrà però cambiare quella con cui entra nel sistema al login.

4.3 I servizi di base

In questa sezione prenderemo in esame alcuni servizi di base del sistema, come quelli per l'esecuzione periodica dei comandi, il sistema di gestione dei log, ecc. In genere, come per i servizi più complessi, come quelli legati ai server di rete, questi sono realizzati da degli appositi programmi detti *demoni* (si ricordi quanto detto in sez. 1.3.4) che lavorano in background ed il cui comportamento è controllato dai relativi file di configurazione.

4.3.1 Il servizio *cron*

Il servizio di schedulazione dei lavori periodici, cioè tutte quelle operazioni che devono essere eseguite a periodi fissi, come ad esempio la creazione del database di `locate` visto in sez. 2.2.2, vendono gestiti dal servizio chiamato *cron*, che viene implementato dal demone `crond`. Il demone ha il compito di svegliarsi ogni minuto ed eseguire ogni programma è stati programmato per quel momento.

Il principale file di configurazione di `crond` è `/etc/crontab` contiene l'elenco dei comandi periodici del sistema, esso viene usato dal programma `crond` per eseguire una serie di azioni periodiche di manutenzione del sistema. In genere si deve intervenire su questo file solo quando si vuole o cambiare uno degli orari a cui le operazioni di default vengono eseguite o per inserire un nuovo comando periodico.

Il formato del file segue la solita regola di ignorare righe vuote ed inizianti per `#`, ogni riga deve contenere una assegnazione di una variabile di ambiente o la specificazione di una azione periodica. L'azione viene specificata da una serie di 7 campi separati da spazi o tabulatori, i

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

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

Il demone `crond` di Linux supporta poi alcune estensioni non presenti in altri Unix: si può usare una lista (separata da virgole) per indicare più valori, un intervallo, specificando gli estremi separati con un `-` o un periodo indicato con il carattere `/` seguito dal divisore del valore (ad esempio `*/2` nel primo campo implica ogni due minuti). Dettagli ulteriori, come sempre, nella pagina di manuale (accessibile con `man 5 crontab`).

Un esempio di `/etc/crontab`, preso da una Debian Sid, è il seguente:

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

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

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

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

A meno di non avere esigenze molto particolari, il contenuto standard di questo file già prevede una serie di azioni giornaliere, settimanali e mensili che vengono eseguite in maniera automatica. Queste azioni sono eseguite (attraverso il comando `run-parts`) dagli script presenti nelle directory elencate nell'esempio precedente, per cui se non si hanno esigenze specifiche non è il caso di intervenire su questo file ma di aggiungere 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 contenenti caratteri estranei (come punto, caratteri di sottolineatura, tilde, ecc.)

Il servizio *cron* però è in grado di eseguire anche i comandi richiesti da un singolo utente; in questo caso questi deve creare un suo *crontab* personale, questo si fa con il comando `crontab`. 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 parametro 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 dei lavori programmati usando l'opzione `-l` che scrive sullo standard output il contenuto del crontab dell'utente.[10] Con `-r` invece si cancella la rimozione completa della tabella corrente. Infine con `-u user` si potrà modificare la tabella dell'utente specificato.

[10] la 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 direttamente come input per `crontab -`.

Si tenga presente che è possibile per l'amministratore riservare l'uso di questo servizio solo ad alcuni utenti, creando il file `/etc/cron.allow`, nel quale si devono elencare gli username di chi lo può 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 dovrà contenere la lista di coloro che non possono usarlo.

4.3.2 Il servizio *at*

Abbiamo visto nella sezione precedente come si possono programmare job periodici 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 un tempo successivo. Il servizio comporta diverse modalità di gestione della esecuzione differita, e vari programmi per la gestione della stessa.

Il comando di base, che permette di programmare l'esecuzione di un programma ad un dato momento, è `at`. Il comando vuole come parametro 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 comando questo viene messo, per ciascun utente, in una opportuna coda. Gli utenti possono vedere la lista dei propri comandi in coda con `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 del job.

Un job può essere rimosso dalla coda con il comando `atrm`, che prende come parametro l'identificativo dello stesso ottenibile con `atq`. Di nuovo un utente normale può operare solo sui propri job, 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 file `/etc/at.allow` e `/etc/at.deny`, il cui formato e significato è identico a quello degli analoghi `cron.allow` e `cron.deny`.

4.3.3 Il servizio *syslog*

Il servizio di *syslog* è il servizio usato dai programmi 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, `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

Servizio	Significato
`auth`	servizio identico a `authpriv`, deprecato.
`authpriv`	messaggi relativi ad autenticazione e sicurezza.
`cron`	messaggi dei demoni di schedulazione (`at` e `cron`) .
`daemon`	demoni di sistema che non hanno una categoria di servizio a se stante.
`ftp`	servizio FTP (*File Transfere Protocol.*
`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 `auth`.
`syslog`	messaggi interni generati da `syslogd`.
`user`	messaggi generici a livello utente.
`uucp`	messaggi del sistema UUCP (*Unix to Unix CoPy*, un meccanismo di comunicazione predente internet).

Tabella 4.2: I servizi standard in cui sono classificati i messaggi del *syslog*.

`-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. La pagina di manuale fornisce una descrizione completa.

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

Il *servizio* identifica una categoria di servizi di sistema per conto dei quali si vuole registrare il messaggio, e viene specificato tramite una delle parole chiave riportate in tab. 4.2, 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. 4.3 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 verranno registrati.

Priorità	Significato
`debug`	messaggio di debug.
`info`	messaggio informativo.
`notice`	situazione normale, ma significativa.
`warning`	avvertimento.
`warn`	sinonimo di `warning`, deprecato.
`err`	condizione di errore.
`error`	sinonimo di `err`, deprecato.
`crit`	condizione critica.
`alert`	si deve intervenire immediatamente.
`emerg`	il sistema è inusabile.
`panic`	sinonimo di `emerg`, deprecato.

Tabella 4.3: 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 si 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 un 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 del file `/etc/syslog.conf` è il seguente, così come presente su una Debian Sid:

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

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

auth,authpriv.*                 /var/log/auth.log
*.*;auth,authpriv.none          -/var/log/syslog
#cron.*                         /var/log/cron.log
daemon.*                        -/var/log/daemon.log
kern.*                          -/var/log/kern.log
lpr.*                           -/var/log/lpr.log
mail.*                          /var/log/mail.log
user.*                          -/var/log/user.log
uucp.*                          -/var/log/uucp.log

#
# Logging for the mail system. Split it up so that
# it is easy to write scripts to parse these files.
#
mail.info                       -/var/log/mail.info
mail.warn                       -/var/log/mail.warn
mail.err                        /var/log/mail.err
#
# Some 'catch-all' logfiles.
```

```
#
*.=debug;\
   auth,authpriv.none;\
   news.none;mail.none     -/var/log/debug
*.=info;*.=notice;*.=warn;\
   auth,authpriv.none;\
   cron,daemon.none;\
   mail,news.none          -/var/log/messages

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

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

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

Commentare

4.3.4 Il sistema di rotazione dei file di log

I file di log (la cui produzione è governata dal demone `syslogd` la cui configurazione è stata trattata in dettaglio in sez. 4.3.3) sono una delle caratteristiche più utili di un sistema unix-like in quanto vi sono registrati messaggi (errori, avvertimenti, notifiche, etc.) dai vari servizi e sono di importanza fondamentale per capire le ragioni di eventuali malfunzionamenti.

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

Il meccanismo di rotazione viene governato attraverso il file `/etc/logrotate.conf` e gli ulteriori file di configurazione presenti nella directory `/etc/logrotate.d/`, in cui sono mantenute le impostazioni per la rotazione dei vari file di log del sistema. Il formato del file

`/etc/logrotate.conf` è sempre lo stesso, le linee vuote o che iniziano per `#` vengono ignorate. Il comando `man logrotate` fornisce una descrizione completa delle varie opzioni del file.

In genere si mettono in questo file solo alcune opzioni generali, le opzioni specifiche per ciascun servizio vengono messe nella directory `/etc/logrotate.d/`, in questo modo si fa si 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 `include /etc/logrotate.d` presente in `logrotate.conf` così che non ci sia necessità di modificare quest'ultimo ogni volta che si installa un altro pacchetto.

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 preso dalla versione installata su una Debian:

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

# keep 4 weeks worth of backlogs
rotate 4

# send errors to root
errors root

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

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

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

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

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

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

Commentare e aggiungere tabella con le opzioni principali.

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.

[1] si chiama così il kernel pubblicato su `http://www.kernel.org`, curato dal *mantainer* ufficiale (lo stesso Linus o chi lui ha delegato al compito).

[2] si 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 quali[3] 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 *mantainer* (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 particolare che segue il rilascio di una nuova serie stabile in cui non si è ancora dato vita ad una nuova versione di sviluppo. 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) 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 degli errori.

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 che in HTTP) e sono disponibili in tre forme,[4] 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 Mb abbondanti) oppure attraverso dei *patch* che permettono di passare da una

[3] accade 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.

[4] in realtà esiste anche una forma di distribuzione tramite il protocollo `rsync`, che permette di ridurre la quantità di dati da scaricare.

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 2.4.23 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 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 di è il seguente:

```
--- linux-2.4.20-ben8/arch/ppc/kernel/ppc_ksyms.c       2002-11-23 10:52:30.000000000 +01
+++ linux-2.4.20-ben8-xfs-lolat/arch/ppc/kernel/ppc_ksyms.c     2003-05-15 17:12:38.00000
@@ -163,6 +163,7 @@ EXPORT_SYMBOL(_outsw_ns);
 EXPORT_SYMBOL(_insl_ns);
 EXPORT_SYMBOL(_outsl_ns);
 EXPORT_SYMBOL(ioremap);
+EXPORT_SYMBOL(ioremap_bot);
 EXPORT_SYMBOL(__ioremap);
 EXPORT_SYMBOL(iounmap);
 EXPORT_SYMBOL(iopa);
@@ -196,6 +197,7 @@ EXPORT_SYMBOL(flush_dcache_range);
 EXPORT_SYMBOL(flush_icache_user_range);
 EXPORT_SYMBOL(flush_icache_page);
 EXPORT_SYMBOL(flush_dcache_page);
+EXPORT_SYMBOL(local_flush_tlb_all);
 EXPORT_SYMBOL(xchg_u32);
 #ifdef CONFIG_ALTIVEC
 EXPORT_SYMBOL(last_task_used_altivec);
```

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 6 righe del primo file a partire dalla 163, e 7 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 redirezione.

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. 3.1.1, 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*[5] del comando `make`.

Una delle caratteristiche peculiari di Linux (torneremo sull'agomento 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. É 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`.[6] 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 sono nella forma:

```
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 23
EXTRAVERSION = -ben1

KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)
```

[5] si ricordi quanto accennato in sez. 3.1.1 relativamente al funzionamento di questo comando.

[6] in 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‘`.

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.

In genere questa è l'unica modifica 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,[7] dato che questi non sono altro che codice, come quello dei programmi ordinari, anche se un po' particolare.[8] 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 del boot con un *kernel panic*.[9]

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 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`,

[7] un file oggetto, in genere identificato dall'estenzione `.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.

[8] ed 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).

[9] si chiama così un crash fatale del kernel; questo può avvenire solo per errori fatali nell'esecuzione dello stesso (in caso di bug particolarmente gravi), o in fase di boot quando mancano le componenti essenziali per l'avvio del sistema.

quelle attivate non sono commentate ed assegnate al relativo valore, alcune indicano dei valori generici (come il tipo di processore o la codifica NLS[10] 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.

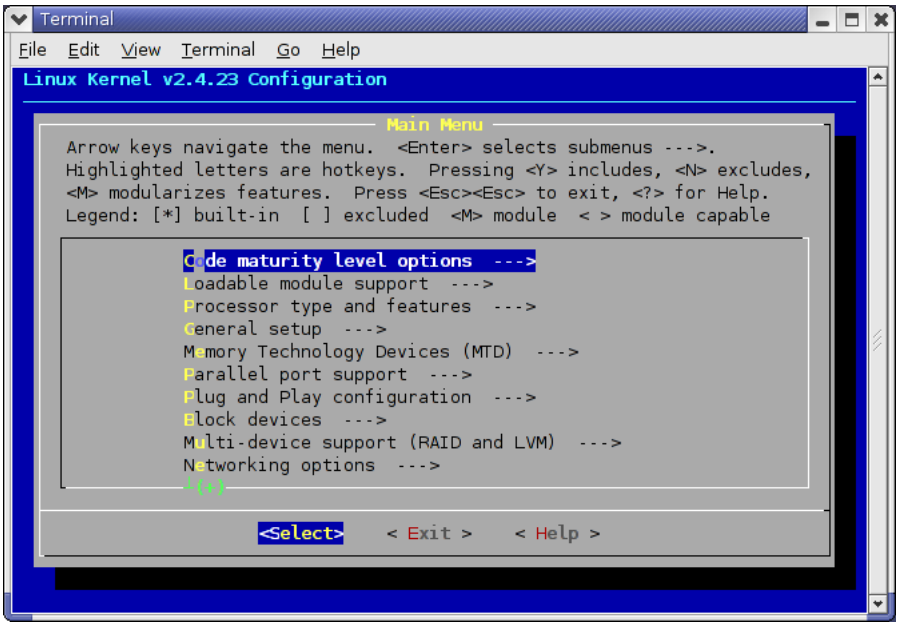

Figura 5.1: Schermata di avvio della configurazione del kernel con `make menuconfig`.

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

[10] il *Native Language Support*, indica la codifica dei vari codici ascii per le stringhe, come `iso8859-1`.

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 pacchetto[11] (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 (nel caso è il menù principale), seguito da un breve riassunto dei principali comandi disponibili. Le freccie 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 freccie 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.

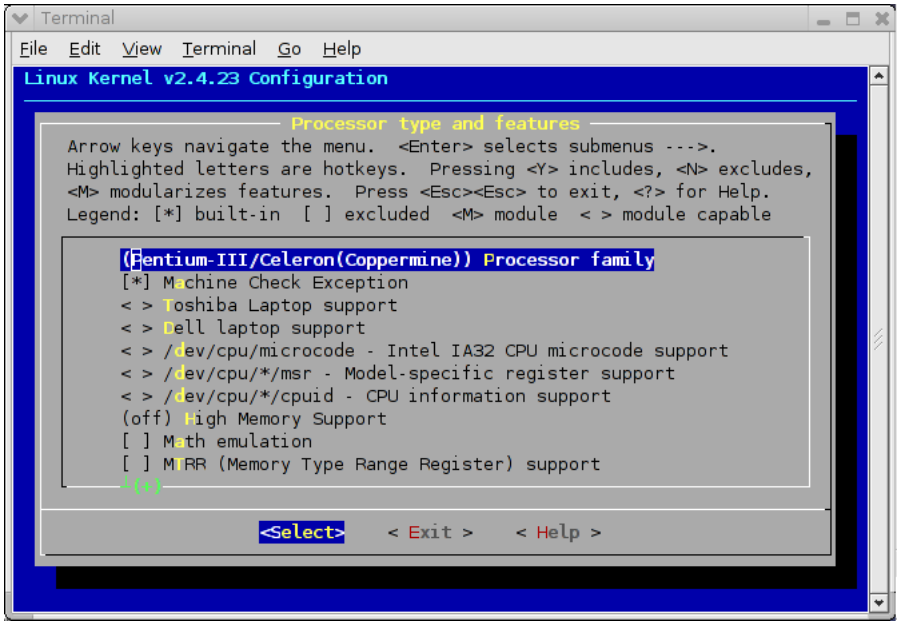

Figura 5.2: Schermata di configurazione del kernel con `make menuconfig`.

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`" (atti-

[11] le 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`.

vabile 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 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.x[12] 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;[13] 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 **Simmetric multi-processing** che consente l'uso di macchine multiprocessore. Sempre 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**.

[12] alcune di queste sono presenti solo nelle versioni più recenti.

[13] il 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.

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.). 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`. 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 dipendenze[14] per la compilazione, il secondo, se si è abilitato il controllo della versione dei moduli, è calcolare le informazioni per il versionamento nei simboli.[15] Pertanto quando se non si è abilitato il versionamento è necessario eseguire questo comando soltanto la prima volte

[14] cioè determinare quali file di dichiarazione (i `.h`) sono necessari per produrre i relativi file binari (i `.o`) contenenti il codice di kernel e moduli.

[15] si chiamano così i nomi delle funzioni che vengono dichiarate all'interno di un modulo, ma possono essere 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.

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,[16] quanto perché nel primo caso il kernel viene caricato nella cosiddetta *low memory* (cioè sotto i primi 640kb) e viene utilizzato un meccanismo d'avvio diverso, mentre nel secondo caso viene caricato sopra 1Mb.

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).

Target	**Significato**
`config`	interfaccia di configurazione a linea di comando.
`oldconfig`	interfaccia di configurazione a linea di comando, che riutilizza i valori precedentemente immessi.
`menuconfig`	interfaccia di configurazione a grafica testuale, basata sulle librerie `ncurses`.
`xconfig`	interfaccia di configurazione grafica.
`dep`	crea le dipendenze per la compilazione e le informazioni per il versionamento dei moduli.
`depend`	identico a `dep`.
`zImage`	crea una immagine compressa del kernel (valida su tutte le architetture).
`bzImage`	crea una immagine compressa del kernel (su architettura PC) con una diversa procedura di avvio.
`vmlinux`	crea una immagine non compressa del kernel.
`modules`	compila i moduli.
`modules_install`	installa i moduli nella relativa directory.
`clean`	cancella tutti i file oggetto (i `.o`) presenti prodotti da una predente compilazione.
`mrproper`	oltre a quanto esegue `clean`, cancella anche le informazioni sulle dipendenze.
`distclean`	oltre a quanto esegue `mrproper` 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.

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

[16] la compressione è identica nei due casi e viene sempre effettuata con `gzip`, nonostante il nome `bzImage` possa trarre in inganno, `bzip` non viene mai usato.

usato come test di efficienza.[17] 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`[18] che consente di specificare come parametro dell'opzione un numero di processi da eseguire in parallelo,[19] ciascuno dei quali compilerà parti indipendenti,[20] 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. 1.2.4 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*.[21]

Il contenuto di questo file è utilizzato dal processo interno al kernel che invia gli eventuali errori (tramite il servizio del *syslog*) riscontrati nell'esecuzione del kernel (detti *oops*) per scrivere i nomi delle funzioni coinvolte invece dei loro indirizzi.[22] La sua assenza perciò non comporta problemi di funzionamento del sistema, ma solo una

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 si 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-2.4.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
```

[17] ad esempio può facilmente causare il surriscaldamento della CPU, per cui viene utilizzato spesso per verificare se un *overclocking* è andato a buon fine.

[18] in realtà la funzionalità è della versione GNU di `make`, non è detto la si ritrovi su altre versioni.

[19] la cosa è particolarmente efficiente su macchine multiprocessore.

[20] questo è possibile sfruttando appunto le informazioni sulle dipendenze usate da `make` per affidare a processi diversi la compilazioni di sorgenti indipendenti.

[21] è 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.

[22] la funzionalità di per se 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.

```
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, `kerneld`, 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 una apposita funzione interna al kernel[23] 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.[24] 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`.

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‘`,[25] 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`.

[23] la 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.

[24] data 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.

[25] si 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.

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 vecchi `kerneld`) 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 *auto-clean* per il modulo, che viene controllato da `kerneld` 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 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`. 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`.

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'*autoclean*.
`-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`.

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 una 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`,[26] dato che in tal caso esegue il calcolo delle dipendenze con i moduli presenti 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.

[26] nelle ultime versioni questa è opzionale.

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 in uso, né contenere simboli usati da un altro modulo (cioè on 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`.[27] 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 un 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,[28] ed già associati all'unico modulo che può essere utilizzato. Esistono però tutta una serie di funzionalità che non sono necessariamente associate ad un unico modulo: il caso classico è quello del controller SCSI, 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 `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
```

[27] in alcuni sistemi più vecchi può essere usato invece il file `/etc/conf.modules`, che è deprecato e non deve essere più utilizzato.

[28] invocando `modprobe -c` specificando un file di configurazione vuoto (con `-C`) si può stampare la configurazione di default.

```
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 si 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 si 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`.

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`.

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
```

Opzione	Significato
`path`	definisce una directory dove cercare i moduli.
`keep`	mantiene le directory predefinite per la ricerca dei moduli .
`option`	definisce un parametro opzionale da passare al modulo quando viene caricato.
`alias`	definisce un nome da associare a un modulo.
`pre-install`	definisce un comando da eseguire prima di caricare il modulo specificato.
`install`	definisce un comando da usare al posto di `insmod` per caricare il modulo specificato.
`post-install`	definisce un comando da eseguire dopo aver caricato il modulo specificato.
`pre-remove`	definisce un comando da eseguire prima di rimuovere il modulo specificato.
`remove`	definisce un comando da usare al posto di `rmmod` per caricare il modulo specificato.
`post-remove`	definisce un comando da eseguire dopo aver rimosso il modulo specificato.

Tabella 5.4: Principali direttive del file di configurazione `modules.conf`.

```
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 `prm_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 di dischi e filesystem per quanto riguarda quelle tutte le operazioni non connesse all'uso corrente degli stessi, che è già stato trattato in sez. 1.2.5. Tratteremo inoltre l'utilizzo dei sistemi di RAID software ed il *Logical Volume Manager*.

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 dimensione 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 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,[29] 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 Addessing*, in cui anche per i dischi IDE l'accesso è eseguito, come per i dischi SCSI,[30] 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 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 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`.

[29] per i dettagli sui registri e su tutte le varie limitazioni storicamente susseguitesi si può leggere il *Large Disk HOWTO*.

[30] il 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.

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. I

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. 1.2.5 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 ripartizionare un disco[31] è `fdisk`, sulla sua base sono stati creati anche altri programmi (ad esempio il `diskdrake` usato dall'installatore di Mandrake che ha una interfaccia utente grafica ed è pertanto più facile da usare), ma noi faremo riferimento solo a questo e alla variante `cfdisk`.

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)
```

[31] su un PC, non tratteremo qui le altre architettura, che hanno programmi diversi.

```
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
```

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 (anche se inutile) con una 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. Altri valori usati da Linux sono

per indicare le partizioni usate per il RAID software e per quelle usate per il *Logical Volume Manager*.

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 `cfdisk`, 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 `cfdisk`, dalla quale è poi possibile dare tutti i comandi.

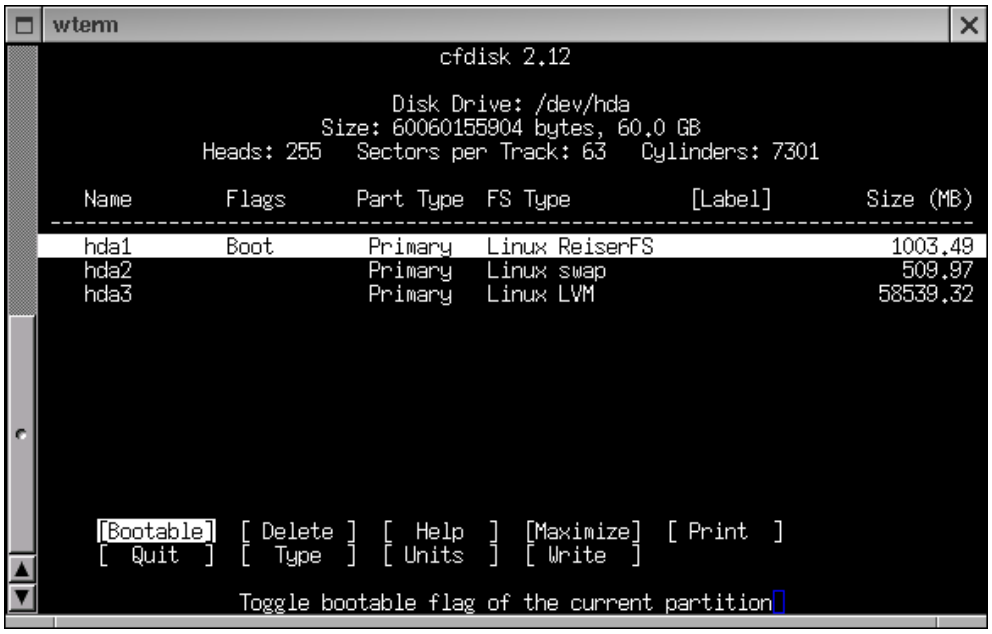

Figura 5.3: Schermata del comando `cfdisk`.

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.[32] 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, se se ne modificano le dimensioni o la posizione le informazioni del filesystem non saranno più utilizzabili.

Per questo in genere occorre pianificare bene le dimensioni delle partizioni fin dall'inizio, perché modificarle in un secondo tempo comporta quasi sempre la necessità di dover fare un backup dei contenuti. Esiste comunque un comando molto utile, `parted` (per il quale rimandiamo alla relativa documentazione, disponibile sia nella pagina di manuale, che nelle FAQ distribuite con il pacchetto), che permette di ridimensionare un filesystem e restringere le relative partizioni, anche se l'operazione resta complessa e delicata.

Per questo nell'installazione occorre normalmente decidere una *strategia di partizionamento*, cioè decidere quante partizioni fare e come assegnarle; per questo occorre tenere ben presenti le indicazioni su contenuto delle varie directory di sistema illustrate in sez. 1.2.4. 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

[32] al 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.

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 motivi si danneggiano file essenziali per il sistema, si avrà sempre a disposizione l'altra partizione con un sistema pienamente funzionale.

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 path 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 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*[33] (in breve LVM) che permette di unire partizioni e dischi fisici diversi in dei *volumi logici* all'interno dei quali creare i relativi filesystem.

Detti volumi possono essere ridimensionati a piacere senza necessità di toccare le partizioni o i dischi che stanno al di sotto, per cui usando i programma per il ridimensionamento dei filesystem si possono allargare o restringere questi ultimi in maniera molto comoda (con certi filesystem le operazioni possono essere eseguite a "*caldo*", con il filesystem montato). In questo modo se lo spazio si esaurisce basterà allargare il volume logico, e se lo spazio disco sottostante non è sufficiente basterà aggiungere 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.

5.2.3 La creazione di un filesystem

Abbiamo visto in sez. 1.2.5 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*.

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.

[33] si 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.

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, che sono comunque delle operazioni operazioni distinte nonostante alcuni sistemi operativi le eseguano insieme, vengono sempre tenute separate.

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 filesytem `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.

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 *hashing* per la ricerca veloce dei file all'interno delle directory.
`msdos`		`mkfs.msdos`
	`-F N`	Specifica la dimensione della FAT (*File Allocation Table*).
	`-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.

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 dispositivo[34] 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*[35] 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 genere[36] 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 sulla base della dimensione del disco e del tipo di uso che si vuole fare del filesystem.[37] 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

[34] anche se volendo si può anche usare un file normale che potrà poi essere montato in loopback.

[35] la 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.

[36] per 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.

[37] 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.

del filesystem.[38] 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
`dir_index`	usa una struttura ad alberi ed hash per la gestione del contenuto delle directory.
`filetype`	memorizza il tipo di file nelle voci delle directory.
`has_journal`	crea anche il file per il giornale (equivalente all'uso di `-j`).
`journal_dev`	richiede che sul dispositivo indicato sia creato un giornale invece di un filesystem.
`sparse_super`	crea un filesystem con meno copie del *superblock* 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.

Parametro	Significato
`size`	Specifica la dimensione del giornale, da specificare in megabytes. 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 `mke2fs -O journal_dev`.

Tabella 5.7: Parametri per la gestione del giornale di un filesystem `ext3` specificabili tramite l'opzione `-J` di `mke2fs`.

Un problema che si può avere nella gestione dei dischi è quello dei settori difettosi. In genere

[38] alcuni filesystem più evoluti permettono di cambiare questo parametro anche in un secondo tempo.

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 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 legge una lista di questi blocchi 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 modificarne 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`.

Le altre opzioni principali sono illustrate in tab. 5.8, al solito si può ottenere l'elenco completo ed una descrizione dettagliata nella pagina di manuale del comando.

5.2.4 Controllo e riparazione di un 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

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 *kernel panic*).
-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 *superblock* (come `dumpe2fs`).
-L	Imposta il nome del volume, per un massimo di 16 caratteri. È questo nome (detto anche *label*) 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.
-U	Imposta l'*universally unique identifier* (UUID) del filesystem, un numero (espresso in cifre esadecimali) che identifica il filesystem (in maniera analoga alla *label*).

Tabella 5.8: Principali opzioni per il comando `tune2fs`.

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:/usr/src/linux/Documentation# 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.[39]

In generale il funzionamento dei filesystem in Linux è estremamente stabile, ed una volta creati è praticamente impossibile[40] 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 di norma `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 questo viene automaticamente avviato.

[39] 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.

[40] a meno di non usare kernel sperimentali, nel qual caso si stanno cercando rogne, ed è anche possibile trovarle.

In genere la procedura di controllo e riparazione può essere molto lunga e complessa, specie per filesystem di grandi dimensioni, in quanto prevede una serie di controlli accurati per identificare informazioni incoerenti e parziali, che comportano varie scansioni del contenuto del filesystem. Questo può significare dei tempi che possono diventare ore o addirittura giorni per i dischi più grandi; per questo molti dei filesystem più avanzati supportano il cosiddetto *journalling*, che permette di evitare questo procedimento e 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: nel primo, in cui l'interruzione è avvenuta durante l'aggiornamento del filesystem, questo sarà in uno stato incoerente, ma il giornale sarà coerente e al riavvio basterà utilizzarlo per riprendere la registrazione da dove era stata interrotta. 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 sarà comunque 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 comunque, specie se si usa `ext3`, è comunque opportuno mantenere un controllo periodico sul filesystem, in quanto errori sul disco, cavi difettosi o bug del kernel possono comunque corrompere 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 font-end per i singoli programmi specifici di ciascun tipo di filesystem, che 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 il comando viene invocato con l'opzione `-A` (di solito questo viene fatto nella procedura di avvio) questo 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. 1.2.5). Nel qual caso prima prova ad eseguire il controllo per il filesystem radice e poi per 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 interattiva. 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 *journalling*; ed anche il tradizionale `ext2` ha ottenuto questa funzionalità con la nuova versione `ext3`. Comunque l'uso di un giornale ha un impatto sulle prestazioni del filesystem, non è detto che questo venga sempre utilizzato, per cui la diffusione di `ext2` è ancora molto ampia. Inoltre per la facilità della conversione (tutto quello che occorre fare è aggiungere il giornale al filesystem con `tune2fs -j` e rimontarlo come `ext3`) si può 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,[41] 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 rilettura. 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 la dimensione dei blocchi.
`-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 *no* a tutte le domande.
`-t`	Stampa delle statistiche di esecuzione.
`-y`	Esegue il controllo rispondendo automaticamente *yes* a tutte le domande.

Tabella 5.9: Principali opzioni per il comando `e2fsck`.

In genere non c'è necessità di eseguire direttamente `mke2fs`, in quanto di norma questo viene eseguito dagli script di avvio. Nel caso lo di 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 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,[42] 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 filesystem[43], 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

[41] eccetto 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.

[42] chiaramente se il filesystem è danneggiato così gravemente da non poter neanche essere montato si avrà un *kernel panic*.

[43] per 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*.

dei blocchi, l'opzione `-s` permette di specificare il *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.[44] 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 L'uso del RAID

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 prevedeva cinque diversi tipi di architetture RAID (da RAID-1 a RAID-5) ciascuna delle quali provvedeva 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*. 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.

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 predefinita[45] 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

[44] in effetti ci sono alcuni programmi per questa operazione, ma non vengono usati, proprio perché inutili.

[45] questa viene di norma definita in fase di creazione dell'array, ed è uno dei parametri fondamentali per le prestazioni dello stesso.

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 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.[46]

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.

[46] è caratteristica comune degli hard disk avere tempi di risposta nettamente diversi per la lettura sequenziale di dati, rispetto al posizionamento (il cosiddetto *seek time*.

Questo livello cerca di bilanciare i vantaggi di affidabilità del RAID-1 con le prestazioni del RAID-0, ma non viene usato molto spesso però 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 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 distrugga il vostro array.

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 l'insieme dei dischi come un dispositivo unico. É senz'altro 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.

Linux supporta vari controller RAID, sia SCSI che IDE. Nel primo caso 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*" (se si hanno più controller) 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`.

A parte il supporto nel kernel, per l'utilizzo del RAID software è necessario utilizzare 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.

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 di root 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.

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.

5.3 La gestione dell'avvio del sistema

In questa sezione prenderemo in esame la procedura di avvio del sistema, dettagliando i vari passaggi (ed i relativi programmi e meccanismi di configurazione) che portano dall'accensione della macchina al pieno funzionamento del sistema.

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 usato 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 disco[47] scelto per l'avvio[48] 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.

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 dei *bootloader* 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; non ci occuperemo di questi casi.

Una volta avviato il kernel effettuerà una serie di operazioni preliminari, come le inizializzazioni delle infrastrutture generiche, (in particolare CPU, memoria e bus PCI), 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.

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 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 parametri all'avvio. In particolare questo significa che il sistema userà come radice quella in uso quando viene compilata l'immagine. In realtà alcuni parametri di avvio in questo caso possono essere modificati con il comando `rdev`. Questo prende come parametro il file contenente l'immagine di

[47] si 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.

[48] per molti BIOS questo può essere solo il primo disco IDE, anche se BIOS più recenti permettono di usare anche altri dischi

un kernel e permette di modificare la directory radice, la modalità video VGA e la dimensione di un RAM disk di avvio.

Nel caso di un CDROM il meccanismo è sostanzialmente lo stesso dei floppy, ma il CDROM deve essere 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.

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 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 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 memorizzato il settore su cui essa si trova) caricarla in memoria, passare tutti gli eventuali parametri di avvio specificati in fase di configurazione, e poi eseguirla.[49] 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. ('/')
```

[49] qualora lo si sia richiesto, si può caricare in memoria anche l'immagine di un ram-disk, che serve come filesystem iniziale per il kernel.

```
#
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-HOWO' and 'kernel-parameters.txt' in
# the Linux kernel 'Documentation' directory.
#
# 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. 1.2.4, 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[50] 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) 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`.[51] 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

[50] in realtà è comunque possibile ottenere la riga di comando se ti tiene premuto il tasto di shift.

[51] si noti che non è assolutamente detto che questa debba stare sullo stesso disco da cui si lancia il kernel.

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 ramdisk 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 `resctricted` si impedisce all'utente di passare dei parametri al kernel sul prompt, a 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 di si avvia il sistema da un disco di recupero (ad esempio perché 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 HURD[52] ma è in grado di avviare qualunque altro tipo di sistema (compreso Linux e BSD).

[52] HURD 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 all'inizio.

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 qualunque dei

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 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 `hdX` (con 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 la 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 specie di shell da cui eseguire i vari comandi interni. Questa è disponibile anche da Linux, se lo si lancia con il comando `grub`, la differenza è che essa è disponibile anche quando viene lanciato all'avvio. I comandi di GRUB sono molteplici ed oltre a quelli per impostare l'avvio dei vari kernel, prevedono anche capacità di ricerca dei file, di visualizzazione e confronto del loro contenuto, con tanto auto-completamento di comandi e nomi, ed è presente anche un *help on line*. 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 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`.[53] 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 che comportano o l'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 quello 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 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 avvio all'interno di essi. Nel caso di Linux solo una distribuzione, la Slackware, ha adottato questa modalità, tutte le altre 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 Linux[54] ha adottato questo sistema.

L'avvio alla SysV avviene sulla base dei cosiddetti *runlevel*, una serie di *modalità operative* del sistema, in cui vengono lanciati un particolare insieme di programmi, che garantiscono 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 all'avvio il sistema in un certo *runlevel*, a seconda di quanto specificato nel suo file di configurazione, `/etc/inittab`.

I *runlevel* validi sono 7, numerati da 0 a 6, normalmente il *runlevel* 0 è usato per fermare il sistema, mentre il *runlevel* 6 per riavviarlo; il *runlevel* 1 serve per andare in *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 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.

Il formato di `/etc/inittab` è molto semplice. Le linee vuote o che iniziano per `#` vengono ignorate, le altre devono essere nel formato:

[53] in realtà passando l'opzione `init=...` si può lanciare un programma qualunque indicandone il path.

[54] Linux è stato sviluppato da zero, per cui non deriva da nessuna delle due famiglie di Unix, per questo in genere ha cercato di prendere il meglio da entrambe.

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

dove `id` deve essere una sequenza unica di 1-4 caratteri che identifica la linea (ed in certi casi delle azioni speciali), `runlevel` la lista dei runlevel (espressa coi numeri di cui sopra) cui la azione specificata dalla parola chiave `action` si applica e `process` il programma che deve essere lanciato. I dettagli al solito si trovano con `man inittab`. Un esempio di 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.
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
# What to do when the power fails/returns.
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop
# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

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 `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 `respwan` si chiede che il comando sia messo in esecuzione immediatamente, senza attendere la sua conclusione per proseguire coi successivi, e 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,[55] 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 ogni caso l'elenco completo di tutte le azioni e del relativo comportamento, insieme a tutti i dettagli del formato di `inittab`, è riportato nella pagina di manuale, accessibile con `man inittab`.

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

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

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 parametro pari al numero dello stesso.

Il meccanismo di avvio di SysV si basa 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 ferme 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`,[56] dove `N` corrisponde al numero del *runlevel* passato come parametro 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 `S` o `K` e 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 il parametro `stop` a quelli che iniziano per `K` (che sta per *kill*) ed il parametro `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).

[55] si 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.

[56] direttamente sotto `/etc` o dentro `/etc/init.d` a seconda delle distribuzioni.

Capitolo 6

Un'introduzione ai concetti fondamentali delle reti

6.1 Le 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 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.

6.1.1 L'estensione

Una delle più immediate forme di classificazione di una rete, 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 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). 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 si fa riferimento per indicare la struttura che connette le 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 la 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 di area intermedia fra LAN e WAN, in genere una rete cittadina, in cui le varie reti locali vengono integrate preliminarmente fra di loro grazie a delle infrastrutture dedicate. Vengono di norma gestite 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 reti di comunicazione usate per connettere computer e dispositivi di uso personale, su distanze di pochi metri.

DAN *Desktop Area Network*, che fa riferimento alle reti di comunicazione usate per connettere vari dispositivi periferici ad un computer a brevissima distanza (la scrivania appunto).

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.

6.1.2 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 qui 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 di base. Le tipologie 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, e ricevere 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 sostenere tutto il traffico possibile fra ogni stazione che vi è collegata e può essere saturato e divenire un collo di bottiglia. Inoltre in caso di rottura del canale tutto il traffico di rete è 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 (un *hub* o uno *switch*) centrale 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.

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 limitando 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.

6.1.3 I protocolli

Come abbiamo visto 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 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. 6.1.

Livello	**Nome**	
Livello 7	*Application*	*Applicazione*
Livello 6	*Presentation*	*Presentazione*
Livello 5	*Session*	*Sessione*
Livello 4	*Transport*	*Trasporto*
Livello 3	*Network*	*Rete*
Livello 2	*DataLink*	*Collegamento Dati*
Livello 1	*Physical*	*Connessione Fisica*

Tabella 6.1: 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 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.

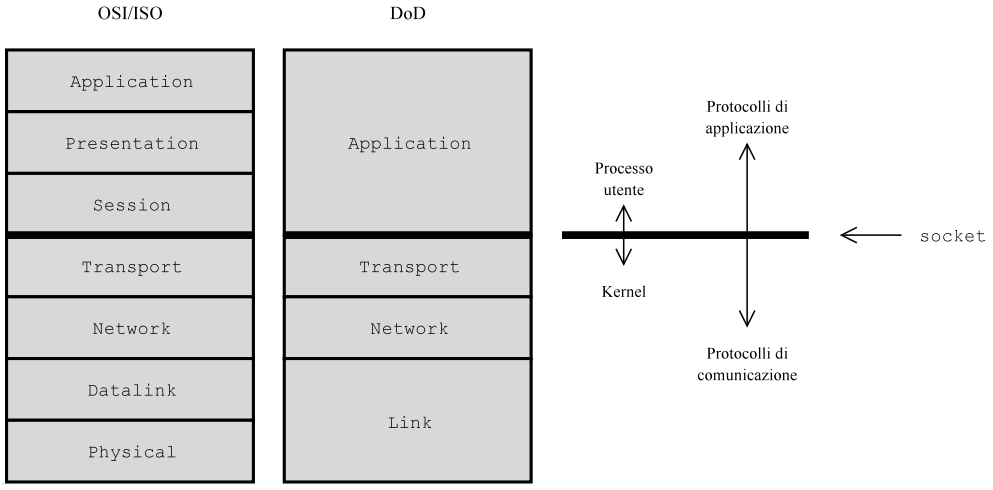

Figura 6.1: Confronto fra il modello OSI ed il modello TCP/IP nella loro relazione con il sistema.

Così come ISO/OSI anche il modello del TCP/IP è stato strutturato in livelli (riassunti in tab. 6.2); un confronto fra i due modelli è riportato in fig. 6.1, dove viene evidenziata anche la corrispondenza fra i rispettivi livelli (che comunque è approssimativa). Si è indicato in figura

anche dove, nel sistema operativo, viene inserita l'interfaccia di programmazione (i cosiddetti *socket*) per l'accesso alla rete.

Livello	Nome		Esempi
Livello 4	*Application*	*Applicazione*	Telnet, FTP, etc.
Livello 3	*Transport*	*Trasporto*	TCP, UDP
Livello 2	*Network*	*Rete*	IP, (ICMP, IGMP)
Livello 1	*Link*	*Collegamento*	ethernet, PPP, SLIP

Tabella 6.2: I quattro livelli del protocollo TCP/IP.

Il nome del modello deriva dai due principali protocolli su cui è basata internet, il TCP (*Trasmission 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:

Applicazione É 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 le due 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 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 spacchettate e reimpacchettate per lo smistamento verso la destinazione finale. Qui il postino prenderà la vostra lettera dal pacchetto arrivato dall'aereoporto e la metterà nella cassetta della posta del vostro destinatario, il quale la aprirà e leggerà quello che gli avete scritto.

Questo è quello che succede anche quando volete spedire dei dati via rete, secondo il procedimento che è illustrato in fig. 6.2. Ad ogni passaggio attraverso un livello del protocollo al 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

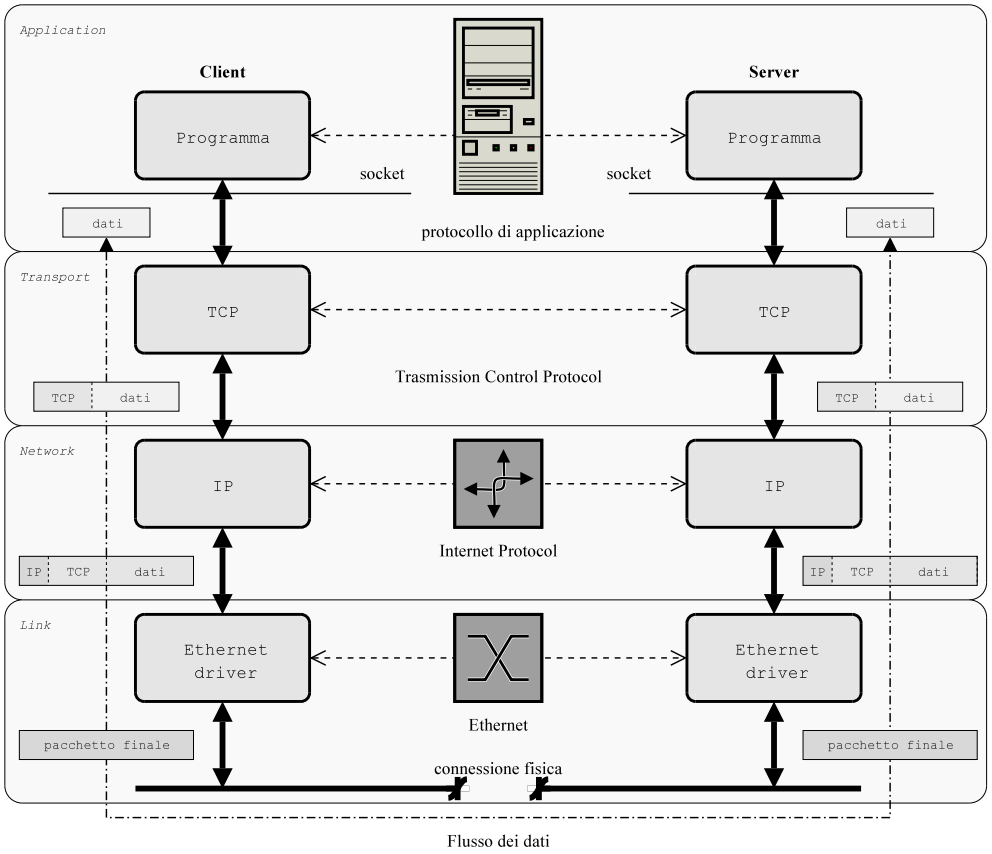

Figura 6.2: Strutturazione del flusso dei dati nella comunicazione fra due applicazioni attraverso i protocolli della suite TCP/IP.

per passare in un pacchetto diverso, ma vengono reimbustati 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 si 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.

6.2 Il TCP/IP.

Come accennato in sez. 6.1.3 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 protocollo[1] 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.

[1]in 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.

6.2.1 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.

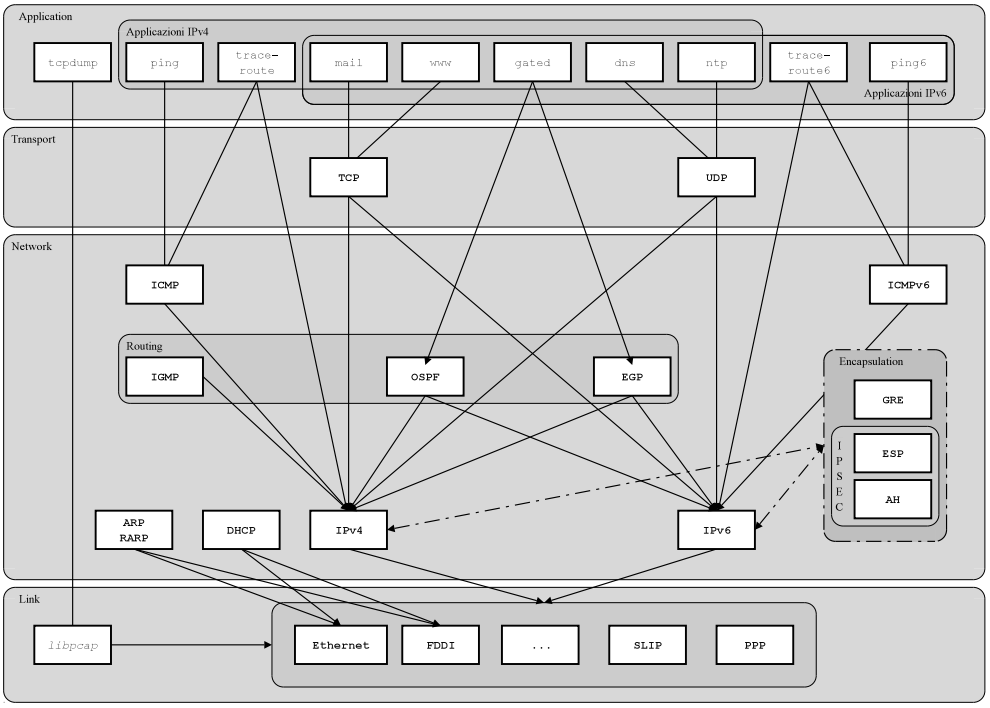

Figura 6.3: Panoramica sui vari protocolli che compongono la suite TCP/IP.

Come illustrato in sez. 6.1.3 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. 6.3 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. 6.2 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.

In fig. 6.3 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 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 *Trasmission 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'acknoweledgment, i timeout, la ritrasmissione, etc. 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 2 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 ICPMv4 per distinguerlo da ICMPv6.

IGMP *Internet Group Management Protocol.* É un protocollo di livello 2 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 Ring o FDDI che hanno associato un indirizzo fisico (il *MAC address*) alla interfaccia, ma non serve in connessioni punto-punto.

[2] in 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.

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. Viene usato a volte per 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 routing 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 routing 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.* Provvede 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.

PPP *Point-to-Point Protocol.* È un protocollo a livello 1 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 1 che permette di trasmettere un pacchetto IP attraverso una linea seriale.

DHCP *Dinamic Host Protocol.* È un protocollo di livello 1 che permette di inviare informazioni di inizializzazione alle stazioni presenti in una LAN, come numero di IP, indirizzi di gateway e nameserver, file per il boot via rete, ecc.

6.2.2 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. 6.1.3, 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 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 ad un telefono possono

[3] vengono chiamati *autonomous systems* i raggruppamenti al livello più alto della rete.

essere collegati più numeri, vedremo più avanti 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 IP usando una apposita notazione che viene chiamata *dotted decimal*, usando i normali numeri decimali; un esempio 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 numero 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, il numero 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, partendo dal 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. 6.3), 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 6.3: 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`.

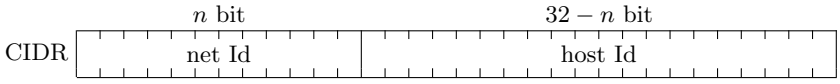

Tabella 6.4: Uno esempio di indirizzamento CIDR.

La suddivisione riportata in tab. 6.3 è largamente inefficiente in quanto se un utente necessita di anche solo un indirizzo in più dei 256 disponibili[6] 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 il numero di rete e quello destinati a indicare l'host finale può essere piazzato in qualunque punto dei 32 bit totali (vedi tab. 6.4), permettendo così di accorpare più classi A 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`.

Per concludere questa panoramica sugli indirizzi occorre accennare all'indirizzo di *broadcast*, mostrato insieme agli altri visti finora nello specchietto in tab. 6.5, in cui si è riassunta la struttura di un indirizzo IP di esempio.

In ogni rete Internet infatti esiste un indirizzo riservato, che per convenzione è sempre ottenuto mettendo ad 1 tutti i bit della porzione dell'indirizzo riservata all'host, che serve ad inviare un messaggio contemporaneamente a tutti i computer presenti su quella rete.[7]

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 6.5: Specchietto riassuntivo della struttura degli indirizzi IP.

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.

[4] il *multicast* è progettato per la comunicazione simultanea verso più stazioni che si possono mettere in ascolto su uno di questi indirizzi.

[5] e questo alle volte può creare problemi, dato che non è detto che la rete in questione sia ancora classificabile in questo modo.

[6] in 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.

[7] questo permette di usare direttamente le capacità di *broadcasting* di alcune interfacce di rete che supportano questa modalità di comunicazione.

Infine alcuni indirizzi sono trattati a parte e considerati riservati, ad esempio dalla classe A è stata rimossa l'intera rete `127.0.0.0` che viene associata ad interfaccia di rete virtuale, indicata dalla sigla `lo`, interna al singolo nodo. Si tratta di una interfaccia assolutamente virtuale che effettua la comunicazione in locale facendo passare i dati attraverso il kernel; di solito le si associa l'indirizzo `127.0.0.1`, il *localhost*, che viene utilizzato per comunicare, senza influenzare la rete, quando un programma che usa i socket TCP/IP deve fare riferimento a se stesso.

Un altro indirizzo speciale è l'indirizzo nullo `0.0.0.0` che viene usato per indicare un indirizzo generico, lo si usa in genere per indicare l'insieme di tutti gli indirizzi possibili, 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.

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. 6.6.

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 6.6: Le classi di indirizzi IP riservate per le reti private.

6.2.3 Il *routing*

L'identificazione di un nodo nella 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 infatti 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*.

In questo caso il problema si può sostanzialmente dividere in due, cioè nella parte relativa alla propria rete locale, ed in quello che succede una volta usciti da essa. L'argomento del *routing* è uno dei più complessi nella gestione delle reti, ma dal punto di vista di una rete locale tutto si riduce al problema dell'instradamento dei pacchetti che escono dai computer che ne fanno parte. 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 all'esterno occorre in qualche modo passare dal centralino.

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 *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 l'indirizzo completo della macchina che fa da ponte, che anch'essa avrà un suo numero IP.

Una volta usciti dalla rete locale la situazione si complica molto, ed in questo caso l'analogia telefonica non ci aiuta, perché di solito il collegamento telefonico nasce a *commutazione di linea* (cioè con una serie di interruttori si metteva effettivamente in collegamento elettrico i due telefoni), mentre internet funziona 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, ma i dati inviati vengono passati da un nodo della rete all'altro fino ad arrivare alla loro destinazione.

Per questo, come dice poi lo stesso nome di *instradamento*, un'analogia che può spiegare un po' meglio le cose, ed 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 li 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 li 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.

6.2.4 I servizi e le porte.

Finora abbiamo parlato quasi esclusivamente di IP; come accennato nell'introduzione (si ricordi quanto detto in sez. 6.1.3) questo è solo uno dei protocolli di internet, e copre soltanto il *livello di rete*. Abbiamo già visto che per poter effettuare delle comunicazioni in generale i programmi necessitano di creare delle connessioni, e per far questo ci sono i socket, che usano i 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*, e 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 dei numeri IP come 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 numero 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,[8] 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

[8] per 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.

alla volta, mentre con internet potete sia ascoltare che trasmettere, da e verso qualunque altro telefono e su quanti canali volete[9] in contemporanea.

Questo avviene perché, come spiegato, TCP/IP è un insieme di protocolli, ed IP (quello dei numeri) serve solo a gestire la trasmissione dei pacchetti attraverso una rete. Per poter effettuare uno scambio di dati occorre una modalità per stabilire una *connessione*, e per far questo occorre andare più in là di quanto si fa con IP, che serve solo ad inviare pacchetti da un computer all'altro, per introdurre ad esempio, dei meccanismi che garantiscano che i pacchetti arrivino davvero a destinazione, che non siano persi, modificati, ecc.

Per questo si usano i protocolli del livello di *trasporto*, come UDP e TCP, i quali a loro volta introducono il concetto di *porta* per poter gestire la possibilità di avere più connessioni[10] in contemporanea, dedicate a servizi diversi, cioè allo scambio di dati specifici come la posta o il web, che vanno a costituire l'ultimo livello (quello di *applicazione*) della struttura mostrata in fig. 6.1.

Così quando si vuole inviare della posta elettronica si comunicherà attraverso una di queste porte, mentre quando si vuole leggere una pagina web se ne userà un'altra. 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 effettuarlo[11] (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 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, ed i programmi di posta la invieranno sulla 25.

Le corrispondenze fra servizi, identificati da nomi simbolici, e numeri delle porte loro assegnate viene mantenuta nel file `/etc/services`. Di norma non è un file che sia necessario modificare, ma è utile per avere un riferimento.

Al di sopra della porta 1024 qualunque utente può mettere un suo servizio, alcuni però sono stati usati tradizionalmente da alcuni servizi, ed il file `/etc/services` tiene conto anche di questi. Il formato del file è molto semplice, un elenco di numeri a ciascuno dei quali è associato un nome simbolico che individua il servizio ad esso associato dalle convenzioni internazionali. Un estratto del file 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
```

[9] in realtà lo si può fare fino ad un numero massimo di 65535 porte, pari a $2^{16} - 1$.

[10] per UDP è più corretto parlare di canali di comunicazione in quanto non c'è una connessione.

[11] per questo non sarà mai possibile sfondare una "*porta*" sul vostro computer, se su di essa non c'è un server, così come non possono mandarvi offese sulla radio, se non siete in ascolto.

```
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
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 ignorato; ogni riga ha il formato:

```
nome            numero/protocollo    alias
```

dove `nome` é l'identificativo simbolico del servizio, `numero` è il numero di porta ad esso assegnato, `protocollo` indica se si tratta di UDP o TCP, e `alias` è la lista di eventuali altri nomi associati allo stesso servizio.

È guardando in questo file che vari programmi attinenti alla rete (ad esempio `netstat`) che riportano o richiedono un numero di porta per un servizio possono utilizzare il nome simbolico di quest'ultimo (www, FTP, telnet), così come viene in esso riportato, invece che un numero.

Capitolo 7

L'amministrazione di base

7.1 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 che operano sulla rete, sono altrettanto numerose e complesse le configurazioni che si possono affrontare. Per questo 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 tutto il resto è costruito, ed esamineremo sia i comandi di base con i quali si effettua la configurazione manuale della rete, che i principali programmi diagnostici e di controllo.

7.1.1 Il supporto nel kernel

Come mostrato fig. 6.1 è il kernel che fornisce il supporto dei protocolli necessari (per i livelli di trasporto, di rete ed il collegamento fisico) per il funzionamento della rete e delle interfacce di comunicazione.

In genere tutte le distribuzioni provvedono dei kernel standard che sono già predisposti a supportare tutto quello che serve nei casi più comuni. Nel caso il vostro hardware non sia supportato, o vi necessiti un protocollo o una funzionalità non previsti nel kernel corrente vi occorrerà ricompilare il kernel[1].

Si da per noto il procedimento per la ricompilazione del kernel, per cui non ci dilungheremo su questa procedura, ma solo sulle opzioni di compilazione che ci interessano. Le sezioni relative alla rete sono due, `Networking options` e `Network device support`, come si può vedere dalla seguente schermata del menù di configurazione riportata in fig. 7.1.

Nelle prima delle due opzioni segnalate con un asterisco in fig, 7.1 si possono attivare i protocolli di rete di alto livello (sopra il collegamento) necessari. Di solito per il TCP/IP tutto quello che serve è attivato di default, è necessario intervenire qui solo se si vuole supportare un altro protocollo (come IPX o Appletalk), o attivare alcune opzioni specialistiche (su cui eventualmente torneremo più avanti).

Nella seconda parte si attivano invece i driver per le varie schede ed i protocolli di basso livello. In genere anche in questo caso la configurazione di default per la maggior parte delle distribuzioni fornisce il supporto per le condizioni di uso più comune; se però si ha una scheda di rete non supportata dal kernel di default può essere necessario accedere alla sottosezione `Ethernet (10 or 100Mbit)`[2] dove troverete le opzioni per una vasta scelta di schede.

[1] nel caso di kernel modulare può bastare la compilazione dei moduli necessari

[2] a meno che non abbiate una gigabit o qualche scheda WAN, nel qual caso dovrete cercare nelle relative sezioni.

```
Linux Kernel v2.4.16 Configuration
--------------------------------------------------------------------------
+------------------------------- Main Menu -------------------------------+
|  Arrow keys navigate the menu.  <Enter> selects submenus --->.          |
|  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, |
|  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.     |
|  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable     |
| +---------^(-)-------------------------------------------------------+ |
| |         Multi-device support (RAID and LVM)  --->                   | |
| |       * Networking options  --->                                     | |
| |         Telephony Support  --->                                      | |
| |         ATA/IDE/MFM/RLL support  --->                                | |
| |         SCSI support  --->                                           | |
| |         Fusion MPT device support  --->                              | |
| |         IEEE 1394 (FireWire) support (EXPERIMENTAL)  --->            | |
| |         I2O device support  --->                                     | |
| |       * Network device support  --->                                 | |
| +         Amateur Radio support  --->                                  | |
| +---------v(+)-------------------------------------------------------+ |
+-------------------------------------------------------------------------+
|                    <Select>    < Exit >    < Help >                     |
+-------------------------------------------------------------------------+
```

Figura 7.1: Schermata di configurazione per la compilazione del kernel; le opzioni di configurazione per il supporto di rete sono indicate con un asterisco

In tal caso la difficoltà maggiore sarà quella di selezionare il modulo che corrisponde alla vostra scheda, cosa che potrete fare ricorrendo ad una lettura dell'Ethernet HOWTO che di solito si trova nella documentazione allegata con la vostra distribuzione o all'indirizzo del link precedente, nel quale è riportata una lunga lista di schede supportate.

Se non avete idea di quale sia la scheda che avete sulla macchina, e vi secca aprire il case, potete sempre ricorrere al comando `lspci` (si suppone che abbiate una scheda PCI, per le schede ISA si può usare `pnpdump`, ma è senz'altro meglio andare a comprarsi una scheda più recente), per ottenere un un risultato 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 40)
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 40)
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/8139C+ (rev 10)
01:00.0 VGA compatible controller: nVidia Corporation NV11 [GeForce2 MX/MX 400] (rev a1)
```

che vi mostra come la vostra scheda di rete sia una Realtek 8139. In questo caso dovrete semplicemente attivare il relativo supporto nella sezione delle schede ethernet `RealTek RTL-8139 PCI Fast Ethernet Adapter support` nella sezione `Ethernet (10 or 100Mbit)`.

Nel caso abbiate optato per un supporto modulare dovrete inoltre configurare opportunamen-

te il file `/etc/modules.conf`, che permette di associare un modulo ad una specifica interfaccia, così da specificare in maniera univoca a quale interfaccia viene associata una certa scheda; in particolare una sezione del file tipo:

```
alias eth0 3c59x
alias eth1 eepro100
```

dice di usare la 3Com Vortex per la prima interfaccia e la Intel EtherExpress per seconda.

Inoltre se lo ritenete opportuno potete far caricare i moduli relativi alle interfacce di rete all'avvio della macchina, questo viene fatto indicando i rispettivi nomi nel file `/etc/modules`.

7.1.2 Il comando `ifconfig`

Il primo passo per poter utilizzare la rete è quello di assegnare ad una interfaccia di rete il suo numero IP. Il comando che vi consente di fare questo è `ifconfig` che permette anche di impostare le varie caratteristiche delle interfacce di rete. Tutto quello che serve nella maggior parte dei casi sono soltanto le opzioni che permettono di attivare e disattivare una interfaccia.

La sintassi completa del comando, come riportata dalla pagina di manuale, è la seguente:

```
ifconfig [interface]
ifconfig interface [aftype] options | address ...
```

se si specifica solo un nome di interfaccia il comando mostra lo stato dell'interfaccia specificata, se si vuole lo stato di tutte le interfacce, comprese quelle non attive, occorre dare solo un'opzione `-a`.

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 mostrato in fig. 7.2.

Come si vede 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.

Si può notare come il comando ci riporti le varie caratteristiche delle interfacce: nella prima riga viene scritto il tipo di collegamento usato, e se presente l'indirizzo fisico; nella seconda riga il tipo ed il valore degli indirizzi associati all'interfaccia, quindi segue una riga con stato corrente dell'interfaccia e tutta una serie di altre informazioni statistiche sul traffico da essa sostenuto.

Valore	Tipo indirizzo
`inet`	Indirizzo IPv4
`inet6`	Indirizzo IPv6
`ax25`	Indirizzo AX25
`ddp`	Indirizzo AppleTalk
`ipx`	Indirizzo Novell IPX

Tabella 7.1: Valori del parametro `aftype` del comando `ifconfig`.

L'argomento `aftype` specifica il tipo di indirizzo che si vuole associare all'interfaccia. I valori possibili sono riportati in tab. 7.1, il valore di default, sottinteso quando non si specifica nulla, è `inet`, che indica il protocollo TCP/IP, dato che il caso più comune è relativo all'impostazione di indirizzi IP.

Il risultato riportato in fig. 7.2 ci mostra anche che l'interfaccia `eth0` è come suol dirsi *multihomed*; ha cioè assegnati due indirizzi diversi, (identificati dalle stringhe `eth0` ed `eth0:0`). È sempre possibile assegnare più indirizzi ad una sola interfaccia, purché si sia abilitato il relativo

```
[root@havnor 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)
```

Figura 7.2: Risultato del comando `ifconfig`.

supporto nel kernel; l'opzione è indicata come `IP aliasing`, nella sezione `Networking options`, e l'assegnazione si può far specificando delle interfacce "*virtuali*" nella forma `eth0:X` dove `X` è un numero crescente a partire da `0`.

Prima di poter configurare una interfaccia occorre verificare che essa non sia già attiva. Supponiamo che si tratti di `eth0`. Il comando `ifconfig eth0` ci mostrerà lo stato dell'interfaccia (dando un errore nel caso il supporto nel kernel non sia attivato), qualora essa sia già attiva e debba essere riconfigurata può essere disattivata con il comando:

```
[root@havnor root]# ifconfig eth0 down
```

utilizzando l'opzione `down`; dopo di che si può assegnarle un indirizzo ed attivarla in un colpo solo con il comando:

```
[root@havnor root]# ifconfig eth0 192.168.1.100
```

in cui è sottintesa l'opzione `up`.

Si ricordi che ad ogni indirizzo è sempre associata una rete, nel caso specifico però essa sembra non comparire; questo è dovuto al fatto che, se non viene specificato esplicitamente, il

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 multicast
BROADCAST	L'interfaccia è in modalità broadcast
POINTOPOINT	L'interfaccia è in modalità punto-punto
LOOPBACK	L'interfaccia è in modalità loopback

Tabella 7.2: Stati riportati nella terza riga del comando `ifconfig` e relativo significato.

comando appena visto assegna automaticamente all'interfaccia una netmask corrispondente alla classe cui l'indirizzo appartiene (nel caso sarebbe pari a `255.255.255.0`, dato che l'indirizzo è di classe C). La rete su cui si allaccia l'interfaccia può anche essere specificata esplicitamente indicando la netmask con una opzione del tipo:

```
[root@havnor root]# ifconfig eth0 192.168.1.100 netmask 255.255.0.0
```

Oltre alle due opzioni `up` e `down` appena illustrate, il comando `ifconfig` supporta molte altre opzioni. Le principali sono elencate di seguito, per le altre, che normalmente vengono lasciate al valore di default, si rimanda alla lettura della pagina di manuale:

`[-]arp` attiva e disattiva (con `-`) l'uso del protocollo ARP per l'interfaccia. Se non specificato il default è `arp`.

`media type` seleziona il tipo di mezzo (in genere il cavo), tramite il parametro `type` (che può assumere ad esempio valori come `10base2`, `10baseT`, ecc.); di solito viene impostato automaticamente usando il default che è `auto`.

`multicast` abilita il multicast sull'interfaccia, normalmente è selezionato automaticamente quando si attiva l'interfaccia.

`[-]promisc` attiva e disattiva (con `-`) il modo promiscuo (in cui tutti i pacchetti vengono ricevuti) per l'interfaccia.

`netmask addr`
imposta la netmask per l'indirizzo.

7.1.3 Il comando `route`

Avere attivato l'interfaccia ed averle assegnato un numero di IP è solo il primo passo, perché sia possibile utilizzare la rete occorre anche impostare *l'instradamento*; per questo si usa il comando `route`, che si chiama così proprio perché serve a specificare la strada che i pacchetti possono prendere per arrivare a destinazione. La sintassi completa del comando, come riportata dalla pagina di manuale, è la seguente:

```
route [-CFvnee]

route  [-v]  [-A  family]  add [-net|-host] target [netmask Nm] [gw Gw]
       [metric N] [mss M] [window W]  [irtt  I]  [reject]  [mod]  [dyn]
       [reinstate] [[dev] If]

route  [-v]  [-A  family]  del [-net|-host] target [gw Gw] [netmask Nm]
       [metric N] [[dev] If]
```

Il comando permette di manipolare la *tabella di instradamento* 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.

Le opzioni fondamentali sono del comando sono due: `add`, che permette di aggiungere una voce alla tabella, e `del` che permette la cancellazione. Se non si specifica nessuna delle due opzioni il comando viene usato per mostrare lo stato corrente della tabella, e le opzioni valide sono quelle mostrate nella prima riga (per il loro significato si rimanda alla pagina di manuale).

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.

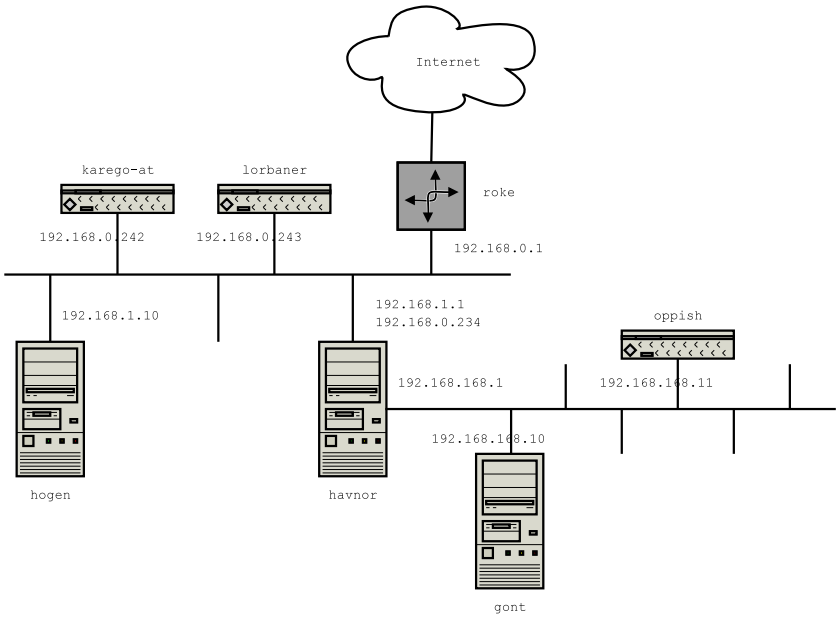

Figura 7.3: Schema di una rete di prova.

Supponiamo di avere la rete schematizzata in fig. 7.3, 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 `havnor`, che è posta a cavallo fra due reti diverse; nel nostro esempio questa dovrà fare da ponte[3] 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 disabilito; questo è controllato attraverso il filesystem `/proc`; per abilitare l'*IP forwarding* occorre eseguire il comando:

```
echo 1 > /proc/sys/net/ip_forwarding
```

Vediamo allora come configurare le rotte della nostra rete usando `route`. Il primo passo è sempre quello di controllare la situazione corrente. Quando è invocato senza parametri il

[3]tecnicamente 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 altro a livello di rete.

comando vi mostra il contenuto corrente della *tabella di instradamento*. Ad esempio, nel caso illustrato in fig. 7.3, 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
```

e si è usata l'opzione `-n` per avere un output 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*,[4] che modificano continuamente le rotte presenti nella tabella di instradamento per tenere conto delle condizioni della rete.

Se non specificato altrimenti il comando tenta anche di risolvere (vedi sez. 7.3) 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 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. 7.3, maggiori dettagli sono nella pagina di manuale.

Nome	**Descrizione**
`Destination`	rete o nodo di destinazione
`Gateway`	indirizzo del gateway (`0.0.0.0` se non impostato)
`Genmask`	*netmask* della rete di destinazione
`Flags`	flag associati alla rotta (vedi tab. 7.4)
`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 (*Maximum Segment Size*) delle connessioni TCP su questa rotta
`Window`	default per la *window size* delle connessioni TCP su questa rotta
`irtt`	valore iniziale dell'RTT (*Round Trip Time*) 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 7.3: Nomi della colonna e relativo significati per le varie informazioni riportate nell'uscita del comando `route`.

Nell'esempio mostrato si può notare come ci siano tre diverse destinazioni associate a tre diverse sottoreti, di cui due fanno capo alla stessa interfaccia (quella che in sez. 7.1.2 abbiamo

[4] abbiamo accennato in sez. 6.1.3 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.

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 quello che è il default gateway, cioè l'indirizzo cui devono essere inviati i pacchetti che non hanno una strada specificata altrimenti; in tal caso la destinazione è indicata dall'indirizzo nullo (che fa le veci della wildcard).

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 tirate su una interfaccia la rotta per la rete a cui l'indirizzo è associato viene inserita automaticamente. Il problema si pone quando la struttura della rete è più complicata, e la rete è divisa in diverse parti.

Simbolo	Significato
`U`	la rotta è attiva
`H`	la destinazione è un nodo
`G`	usa un gateway
`R`	rotta reintegrata da un instradamento dinamico
`D`	installata dinamicamente da un demone o una redirezione
`M`	modificata da un demone o una redirezione
`A`	installata da `addrconf`
`C`	voce nella cache
`!`	rotta bloccata (impedisce l'instradamento per la destinazione specificata)

Tabella 7.4: 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 singola, una volta assegnato l'indirizzo con `ifconfig` tutto quello che resta da fare è specificare qual'è il default gateway, cioè l'indirizzo del nodo (di solito un router) che si usa per uscire in internet; questo si fa ad esempio con il comando:

```
[root@havnor root]# route add default gw 192.168.0.1
```

che ovviamente deve essere dato da root, 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. 7.3.

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 su una LAN diversa; lo stesso vale per le macchine di quella LAN, che non possono accedere direttamente al tratto di rete delle precedenti. 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 ponte fra i due tratti separati.

In tutti i casi in l'accesso ad una rete è condizionato al passaggio da una macchina specifica che fa da ponte, si deve specificare esplicitamente una rotta statica che indichi anche quest'ultima come gateway 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 `192.168.160.0` 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
```

Si tenga presente che, come nel caso precedente in cui il default gateway è specificato per qualunque indirizzo, la tabella di instradamento può contenere più rotte per la stessa destinazione. In genere il kernel le riordina, eseguendo il controllo su quale direzione far prendere ad un

pacchetto, a partire dalla rotta più specifica, ed esegue l'instradamento non appena trova una rotta valida. Così nel caso dell'esempio precedente i pacchetti uscenti da `lorbaner` destinati all'indirizzo `192.168.168.11` non arriveranno mai al gateway, ma saranno mandati verso `havnor` sulla scheda di rete corrispondente a `eth0` per poi riemergere dall'altra interfaccia di rete sul tratto su cui c'è anche `gont`.

Oltre alla rete e alla destinazione il comando `route` permette, quando si aggiunge una nuova rotta alla tabella di instradamento, di impostarne una serie di altre caratteristiche. In generale il comando `add` prende sempre due opzioni: `-net` per indicare una voce riferita ad una rete e `-host` per una stazione singola, e deve essere seguito dall'indirizzo (di rete o di nodo) della destinazione (sia in forma numerica che simbolica). Si deve inoltre, nel caso si sia indicata una rete, specificare anche la relativa netmask, e l'interfaccia da utilizzare; gli altri parametri, la cui lista completa è riportata in tab. 7.5, sono opzionali o inutilizzati per le rotte statiche; parte di essi vengono automaticamente impostati agli opportuni default per l'interfaccia cui la voce fa riferimento.

Le opzioni base (`-net` e `-host`) valgono anche per il comando `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.

Opzione	Parametro	Descrizione
`-A`	family	imposta la famiglia di indirizzi
`netmask`	Nm	imposta la maschera per rete
`gw`	Gw	imposta l'indirizzo del gateway
`reject`	–	installa una rotta bloccata
`metric`	M	imposta il valore del campo `Metric`
`mss`	M	imposta la *Maximum Segment Size* del TCP per la rotta
`window`	W	imposta la dimensione della *advertizing window* del TCP per la rotta
`irtt`	I	imposta il *Round Trip Time* iniziale per la rotta
`mod,dyn,reinstate`		flag diagnostici usati dai demoni di routing
`dev`	If	imposta l'interfaccia usata per raggiungere la destinazione

Tabella 7.5: Opzioni del comando `route`.

7.1.4 La configurazione automatica.

La configurazione della rete all'avvio viene fatta da degli opportuni script, la cui locazione dipende dalla distribuzione; in tab. 7.6 si sono riportati quelli delle principali distribuzioni. 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`.[5] Gli script usano al loro interno i comandi visti in sez. 7.1, per impostare i valori predefiniti.

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/...`

Tabella 7.6: Gli script di inizializzazione della rete per varie distribuzioni.

In generale tutte le distribuzioni provvedono dei programmi per automatizzare il processo di configurazione delle interfacce e degli indirizzi IP, memorizzando i valori usati dagli script di avvio (che normalmente non cambiano) in opportuni file di configurazione (torneremo su questo in sez. 7.1.5) che vengono letti da questi ultimi.

[5] questo non vale per Slackware che non usa il sistema dei runlevel di System V.

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. Di solito i casi 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 preoccuperà 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 frontend per `pppd`) 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, il DNS (torneremo su questo in sez. 7.3.5).

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 il numero IP da assegnare alla vostra macchina, la netmask, e l'indirizzo del gateway (oltre alla informazione sul DNS da usare).

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. 6.6, che la IANA ha destinato appositamente a questo uso, e che non vengono mai usati per macchine pubbliche su Internet.[6]

Quasi tutte le distribuzioni hanno dei programmi per configurare la rete locale,[7] 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. 7.7, ed

Distribuzione	Comando
Debian	`dpkg-reconfigure etherconf`
RedHat	`netcfg`, `netconfig`
Slackware	`netconfig`

Tabella 7.7: Comandi di configurazione della rete.

in genere si tratta di soltanto di specificare i valori richiesti. Un caso comune è quello in cui l'indirizzo non deve essere specificato a mano, ma viene impostato automaticamente grazie alla presenza di un server DHCP. Tutti i programmi di configurazione prevedono questa possibilità, nel qual caso non dovrete fornire nessuna informazione specifica, se non quella relativa all'uso del DHCP.

7.1.5 I file di configurazione delle interfacce statiche.

Come accennato tutte le distribuzioni avviano la rete all'interno di opportuni script di avvio; uno specchietto degli script usati è riportato in tab. 7.8. In genere questi script non fanno altro che

[6] se 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, automaticamente i siti che hanno i numeri che voi avete assegnato alle vostre macchine sarebbero irraggiungibili.

[7] nel 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.

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.

Distribuzione	Comando
Debian	`/etc/init.d/networking`
RedHat	`/etc/rc.d/init.d/network`
Slackware	`/etc/rc.d/rc.inet1`
Suse	`/etc/rc.d/network`

Tabella 7.8: Script di avvio della rete nelle varie distribuzioni.

L'uso manuale degli stessi comandi per avviare la rete lo abbiamo già discusso nelle sezioni precedenti, se però 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.

Debian Il file di configurazione delle interfacce[8] 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 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,[9] e le modalità della stessa; esse sono sostanzialmente tre:

`loopback` si usa per l'interfaccia di loopback.

`dhcp` usa un server DHCP

`static` assegna i valori secondo i parametri specificati nelle righe seguenti.

e se si specifica `static` si possono 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` le righe successive, indentate per chiarezza, specificano le ulteriori opzioni, nel caso in esempio si sono indicati l'indirizzo, la netmask e l'indirizzo del gateway. Quest'ultimo deve essere specificato una volta sola,

[8] in 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.

[9] `inet` indica l'usuale TCP/IP, ma sono possibili anche `ipx` per IPX, e `inet6` per IPv6.

nel caso compaiano più interfacce. 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.

Infine è possibile usare la parola chiave `mapping` per effettuare una mappatura fra interfacce *logiche* e *fisiche*,

Si tenga presente che in generale le interfacce temporanee relative all'uso di PPP non vengono mai menzionate in questo file, ma vengono configurate a parte dagli script di avvio della connessione.

RedHat i file di configurazione sono dentro `/etc/sysconfig/networking/devices`, e ce n'è uno per interfaccia, con lo stesso nome dell'interfaccia che contiene 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
```

nella forma di assegnazione di un valore ad una variabile[10] il cui significato è evidente per gran parte di esse. Si noti che nel caso si è scelto, usando `BOOTPROTO=dhcp` di fa assegnare l'IP attraverso un server DHCP, nel qual caso `IPADRR` non viene assegnato, se invece si fosse voluto assegnare un IP fisso si sarebbe dovuto utilizzare il valore `BOOTPROTO=static` ed impostare il relativo numero tramite `IPADRR`.

7.1.6 Il comando `ping`

Una volta configurate le interfacce il primo controllo da effettuare per vedere se la rete funziona è "*pingare*" un'altra macchina. Nella nostra analogia telefonica questo equivale a telefonare per sentire se dà il libero.

Il comando `ping` permette di inviare un pacchetto ICMP (abbiamo accennato a questo protocollo in sez. 6.2.1) di tipo *echo request*, che di norma causa nella macchina che lo riceve l'emissione in risposta di un alto pacchetto ICMP, un *echo reply*.[11] Il comando si invoca specificando l'IP della macchina bersaglio,[12] altrimenti deve anche funzionare il DNS, e se la rete non va per qualche altro motivo che non vi consente di raggiungere quest'ultimo non otterreste nulla anche se il resto è a posto.

Un esempio di uso 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
```

[10] in effetti si tratta proprio di questo, il file viene letto dallo script di avvio che usa queste variabili di shell per effettuare la configurazione.

[11] in sostanza si tratta di una specie di *sonar*.

[12] anche se in realtà va bene pure il nome, ma se lo usate esso deve essere scritto in `/etc/hosts` (vedi sez. 7.3.2).

```
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
```

Il comando invia un pacchetto al secondo, e nel caso riceve sempre risposta riportando il tempo che ci è voluto. Quando lo fermate (con `C-c`) vi stampa anche una statistica. Questo ci dice che la macchina con IP `192.168.168.10` è attiva ed è raggiungibile. Il comando prende una serie di opzioni, le principali, insieme al loro significato, sono riportate in tab. 7.9, per le altre 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 root 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 risoluzioni di nomi e indirizzi.
`-p pattern`	si possono specificare fino a 16 byte con cui riempire il contenuto 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 di default.

Tabella 7.9: Opzioni del comando `ping`.

Si tenga presente che quello che si può effettuare con `ping` è solo un controllo preliminare; se non ricevete risposta il motivo può dipendere da molti fattori, da un errore di configurazione ad aver usato un indirizzo sbagliato, incluso il fatto che la macchina che volete controllare può essere spenta, o che qualche router nel mezzo non funziona. 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.

7.1.7 Il comando `traceroute`

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 richiede[13] 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 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 avere tracciata tutta la strada fatta da un pacchetto.

La sintassi generica del comando, come riportata nella pagina di manuale è la seguente:

[13] questo viene fatto per evitare che i pacchetti persi nei *routing loop* continuino a circolare sulla rete, con spreco di banda.

```
traceroute [ -dFIlnrvx ] [ -f first_ttl ] [ -g gateway ]
        [ -i iface ] [ -m max_ttl ] [ -p port ]
        [ -q nqueries ] [ -s src_addr ] [ -t tos ]
        [ -w waittime ] [ -z pausemsecs ]
        host [ packetlen ]
```

un esempio del funzionamento del comando è il seguente:

```
piccardi@oppish:~/temp/video/riserva$ traceroute www.linux.it
traceroute to picard.linux.it (62.177.1.107), 30 hops max, 38 byte packets
 1  10.16.34.54 (10.16.34.54)  404.842 ms  228.168 ms  217.859 ms
 2  213.234.139.162 (213.234.139.162)  295.735 ms  271.781 ms  120.006 ms
 3  interbusiness-mix.mix-it.net (217.29.66.35)  417.619 ms  62.880 ms  296.789 ms
 4  151.99.98.225 (151.99.98.225)  209.817 ms  275.854 ms  125.990 ms
 5  151.99.75.219 (151.99.75.219)  157.818 ms  363.377 ms  64.956 ms
 6  80.17.211.190 (80.17.211.190)  477.793 ms  190.838 ms  191.940 ms
 7  r-ge10-vl2.opb.interbusiness.it (195.31.96.227)  365.857 ms  222.774 ms  310.838 ms
 8  62.86.83.194 (62.86.83.194)  70.828 ms  156.906 ms  67.908 ms
 9  sircore1-ext.publinet.it (62.177.0.1)  67.880 ms  561.587 ms  694.767 ms
10  picard.linux.it (62.177.1.107)  293.828 ms  464.118 ms  217.977 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,[14] 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 la risoluzioni di nomi e indirizzi.
`-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 7.10: Opzioni del comando `traceroute`.

Il comando `traceroute` prende numerose opzioni, che permettono di impostare varie caratteristiche dei pacchetti inviati, al solito si sono riportate le principali in tab. 7.10; per le restanti si faccia riferimento alla pagina di manuale.

Del comando esiste una versione più moderna, che presenta anche una interfaccia grafica, `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`.

7.1.8 Il comando `netstat`

Un altro comando diagnostico molto utile, anche se complesso, che permette di visualizzare una grande quantità di informazioni relative alla rete, è `netstat`. La trattazione di tutte le sue capacità comporta un approfondimento dei concetti relativi alle reti che va al di là delle possibilità di quanto possiamo affrontare in questo corso.

[14] sempre che qualche amministratore di rete troppo zelante ed un po' ignorante non si sia messo a filtrare anche i pacchetti di controllo usati dal protocollo.

Qui ne tratteremo solo un uso specifico, quello che permette di visualizzare tutte le connessioni attive sulla macchina (un po' come potrebbe fare il quadro di un centralino che mostra tutti i collegamenti in corso). Per questo è anche uno dei primi comandi che un eventuale intruso che si sia introdotto nella vostra macchina cercherà di sostituire con una sua versione "*personale*", perché può mostrare eventuali collegamenti "*indesiderati*".

Il comando usato con le opzioni di default mostra le informazioni riguardo a tutti i *socket* aperti; anche i collegamenti interni al sistema (usati da vari programmi per scambiarsi i dati con l'interfaccia dei *socket*) che di norma non han nulla a che fare con la rete. Per questo motivo è d'uopo 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 il seguente:

```
[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
```

Il comando riporta una tabella con le indicazioni relative a ciascuna connessione. Il campo *Proto* riporta il protocollo della connessione. I campi *Local Address* e *Foreign Address* indicano gli indirizzi locale e remoto della stessa (che può essere stampato in forma numerica anziché usando lo switch `-n`), 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 vari campi va di nuovo al di là delle possibilità di questo corso (specie per il campo `State`), e richiede una trattazione approfondita del protocollo TCP/IP.

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, nel caso ce ne sono vari corrispondenti a servizi come la posta, le news, il DNS, la stampa via rete, gli indirizzi sono di norma non specificati in quanto la connessione può essere effettuata su uno qualunque degli indirizzi locali, da un qualunque indirizzo esterno. Lo stato `ESTABLISHED` indica le connessioni stabilite ed attive, e riporta nei campi degli indirizzi i due capi della connessione. Altri stati che possono essere riportati sono `FIN_WAIT`, `TIME_WAIT`, e si riferiscono a connessioni che si stanno chiudendo.

7.2 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

dei server (la maggior parte dei server 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.

7.2.1 Il comando `telnet`

Uno dei servizi di base un tempo più utilizzato sulla rete è quello del *telnet*, nato per eseguire delle connessioni con 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. 8.5.2.

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.

La forma più generale del comando, come riportata dalla relativa pagina di manuale, è la seguente:

```
telnet [-468ELadr] [-S tos] [-e escapechar] [-l user] [-n tracefile]
       [host [port]]
```

i dettagli delle varie opzioni si trovano al solito sulla pagina di manuale accessibile con `man telnet`. Di norma lo si usa indicando semplicemente un indirizzo ed una porta, con un comando 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 allora lo si è usato per connettersi nel caso alla porta 25 della propria macchina, e verificare che il relativo servizio (SMTP) sia attivo (nel caso si è ottenuta la risposta di `postfix`).

Allo stesso modo lo si può usare per verificare la presenza e l'attività dei servizi sulle relative porte. Non specificando nessuna porta il comando si collega sulla porta 23, che corrisponde al servizio standard *telent*, mostrando (qualora si attivo il relativo server) una schermata di login.

7.2.2 Il comando `ftp`

Il protocollo FTP è il più vecchio dei 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 consente solo il prelievo dei file).

Dato che il protocollo, come *telnet*, non prevede alcuna cifratura dei dati, è meglio non usarlo, con l'eccezione della modalità anonima, per lo scambio di file dei vari utenti, in quanto l'autenticazione verrebbe eseguita in chiaro; per questo sono disponibili alternative come l'uso di `scp` o `sftp`.[15]

Esistono molti client per FTP, sia grafici che testuali; ma il comando di base usato per lanciare il client testuale è `ftp`. Un esempio di uso generico comando è:

[15] `sftp` è un programma che ha esattamente la stessa sintassi di `ftp`, ma consente l'uso di una connessione cifrata attraverso il protocollo SSH.

```
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,[16] e vanno verificate facendo riferimento alla relativa pagina di manuale.

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 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. 7.11. La lista completa è al solito disponibile nella 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 7.11: Comandi del protocollo FTP.

Si tenga inoltre presente che non tutti i comandi possono essere supportati dal server. Ad esempio il comando `pasv`, che abilita il modo passivo,[17] 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.

7.2.3 Il comando `finger`

Il comando `finger` veniva usato nelle origini di Unix per riportare informazioni relative agli utenti di un sistema, in modo da consentire agli altri utenti di sapere chi era collegato. Il

[16] 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 history dei comandi, il completamento dei nomi, e capacità aggiuntive.

[17] il 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.

programma funziona sia in locale che in remoto (nel qual caso si eseguirà una richiesta con un parametro del tipo `nome@host`.

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: 055-6800052
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.

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 più a non attivare questo servizio sulla rete.

7.2.4 Il comando `whois`

Il comando `whois` viene usato per contattare il servizio *whois*. Questo servizio viene utilizzato per mantenere un database dei titolari dei domini su internet. In genere lo si utilizza quando si cercano informazioni sulla registrazione dei domini, e per il contatto dei relativi proprietari.

Ad esempio avremo che:

```
[piccardi@gont piccardi]$ whois truelite.it
domain:      truelite.it
x400-domain: c=it; admd=0; prmd=truelite;
org:         Simone Piccardi
descr:       Via Kyoto, 8
descr:       50126 Firenze (FI)
admin-c:     SP1700-ITNIC
tech-c:      SP1700-ITNIC
tech-c:      TS7016-ITNIC
postmaster:  SP1700-ITNIC
zone-c:      TS7016-ITNIC
nserver:     195.110.99.23 ns1.register.it
nserver:     213.246.4.72 ns2.register.it
mnt-by:      REGISTER-MNT
created:     20020819
expire:      20030819
changed:     support@register.it 20020815
source:      IT-NIC

person:      Simone Piccardi
address:     Simone Piccardi
address:     Via Kyoto, 8
address:     50126 Firenze (FI)
address:     IT
phone:       +39 055 6800052
```

```
e-mail:      piccardi@firenze.linux.it
nic-hdl:     SP1700-ITNIC
changed:     support@register.it 20020819
source:      IT-NIC

person:      Technical Services
address:     Register.it
address:     Italy
phone:       +39 035 3230400
fax-no:      +39 035 3230312
e-mail:      support@register.it
nic-hdl:     TS7016-ITNIC
notify:      support@register.it
mnt-by:      REGISTER-MNT
changed:     support@register.it 20011016
changed:     support@register.it 20030423
source:      IT-NIC
```

7.3 La risoluzione dei nomi

Finora abbiamo parlato della configurazione di base della rete, che riguarda principalmente l'assegnazione degli indirizzi e delle rotte per lo smistamento dei pacchetti, cioè il livello più basso del protocollo TCP/IP, che in genere viene gestito direttamente dal kernel.

Uno dei servizi di base per l'accesso a internet però è quello della risoluzione dei nomi, che permette alle persone di operare con degli identificativi più significativi che non dei numeri IP. Proseguendo con la nostra analogia telefonica consideriamo l'agenda che sta sempre accanto al telefono. Ricordarsi a mente a quale sito, macchina o persona corrisponde un certo numero IP è assolutamente impraticabile; per questo occorre l'equivalente dell'agenda e dell'elenco telefonico che associa ogni numero ad un nome.

7.3.1 Introduzione

Il servizio di risoluzione dei nomi è realizzato principalmente attraverso il cosiddetto *Domain Name Service*, o DNS. Il DNS è un enorme database distribuito che associa ad un nome letterale (quello dei *nomi di dominio*, che identifica i siti internet) un indirizzo IP. Trattare i dettagli del funzionamento del protocollo del DNS va al di là di quanto può essere affrontato in questo corso, ne vedremo più avanti (in sez. 8.3.1) gli aspetti essenziali.

All'interno di una rete locale comunque il DNS è più simile al servizio "12" che all'agenda telefonica (questa in realtà è realizzata dal file `/etc/hosts`, trattato in sez. 7.3.2), in quanto esso viene realizzato da macchine che forniscono questo servizio. Esso ha anche il vantaggio (come il servizio 12 rispetto all'agenda) che viene aggiornato automaticamente qualora una associazione fra nome e numero IP dovesse cambiare.

Per questo motivo quando configurate la rete con i programmi di interfaccia creati dalle varie distribuzioni, una delle informazioni che di norma vi vengono richieste è l'indirizzo IP (numerico ovviamente) di una di queste macchine (ogni provider ne mette a disposizione almeno una) che fa le veci del vostro "12" personale.

In realtà occorre tenere presente che il servizio di risoluzione dei nomi è effettuato a più livelli da una serie di funzioni fornite dalle librerie standard del C, che vanno a costituire quell'insieme di funzionalità che viene chiamato "*resolver*" e che permette di ottenere delle associazioni fra nodi della rete e identificativi associati agli stessi. Per questo prima di trattare del DNS in quanto

servizio fornito sulla rete, prenderemo in considerazione i file di configurazione usati dalle routine del *resolver*, che sono quelle che svolgono in maniera generica il servizio di risoluzione dei nomi.

7.3.2 Il file `/etc/hosts`

Normalmente, a meno di una diversa impostazione nei vari file di configurazione che esamineremo più avanti, il primo file utilizzato dal *resolver* per effettuare una corrispondenza fra numeri IP e nomi simbolici è `/etc/hosts`. Questo file contiene un elenco di nomi di macchine, associati al relativo numero 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, il numero di IP, il nome completo (in termini di dominio, quello che viene detto FQDN, *Fully Qualified Domain Name*) ed un nome breve. Un possibile esempio di questo file è il seguente:

```
# private nets
192.168.168.10  gont.gnulinux.it       gont
192.168.168.11  oppish.gnulinux.it     oppish
```

che associa agli IP delle macchine che in fig. 7.3 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 in una rete questo file dovrebbe 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. 8.3.6 come fare lo stesso lavoro attraverso il DNS.

7.3.3 Gli altri file per i nomi di rete

Oltre alle associazioni fra numeri IP e nome simbolico di una macchina, esistono una serie di altre informazioni relative ad una rete, di norma identificate nei protocolli da valori numerici, ai quali però viene assegnato un corrispondente nome simbolico come facilitazione mnemonica.

Il primo di questi file 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 è analogo a quello di `/etc/hosts`.

Il formato del file, 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 il suo indirizzo, nella notazione *dotted decimal* (tralasciando opzionalmente gli eventuali `.0` finali), il terzo 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
```

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 l'indirizzamento CIDR non funziona.

Oltre ai nomi delle reti, i vari comandi di sistema possono avere altre necessità di usare nomi simbolici al posto dei valori numerici; un esempio di questo è nella corrispondenza fra numeri di porta e servizi illustrata in sez. 6.2.4, che viene mantenuta in `/etc/services`. Un altro file usato per questo tipo di corrispondenze è `/etc/protocols`, che associa al numero usato a livello

di IP per identificare il protocollo di trasporto usato nello strato successivo (si ricordi quanto detto in sez. 6.1.3) ad un identificativo.

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; la descrizione completa del file è riportata nella pagina di manuale accessibile con `man protocols`.

Un'altra informazione usata da parecchi programmi che hanno a che fare con la rete e non solo è il nome della macchina su cui ci si trova. Questa è in genere una informazione indipendente dal fatto di essere in rete, e viene gestita attraverso il comando `hostname` che serve sia per impostare che per mostrare il nome della stazione su cui ci si trova.

In generale la gran parte delle distribuzioni usano un file che è `/etc/hostname` o `/etc/HOSTNAME` nel quale scrivere il nome della macchina; questo poi viene usato per impostare l'hostname (in genere negli script di avvio). Di solito si usa lo stesso comando `hostname` che con l'opzione `-F` permette di impostare il nome della macchina leggendolo da un file. Anche in questo caso le righe vuote e tutto quello che segue un `#` viene ignorato.

Un secondo comando relativo all'hostname è `dnsdomainname` che invece permette di ottenere il nome di dominio in cui si trova la macchina. In questo caso però non è possibile usare questo comando per modificarne il valore, in quanto questo viene determinato dal *resolver* sulla base della configurazione di quest'ultimo.

7.3.4 Il file /etc/nsswitch.conf

In generale oltre alla risoluzione dei nomi associati alle singole stazioni esiste la necessità di poter effettuare altra associazioni, 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, con dei file come /etc/hosts o /etc/protocols, ma anche su database, con LDAP, o su un server DNS, le librerie del C GNU prevedono una modularizzazione di questi servizi attraverso il cosiddetto *Name Service Switch*.

Questo è un insieme di librerie dinamiche che permettono di mantenere le varie corrispondenze fra nomi e numeri in maniera generica su vari tipi di supporti. Il loro comportamento è regolato dal file /etc/nsswitch.conf che serve, per ciascuno dei servizi gestiti dal *Name Service Switch*, a indicare su quale supporto si trovano, ed in quale ordine cercarle, le informazioni di corrispondenza.

Per il file valgono le solite convenzioni per cui quello che segue un # e le righe vuote vengono ignorate. Il formato richiede un nome di servizio seguito da due punti, ed una lista di possibili supporti nell'ordine in cui verranno esaminati. Un estratto del file è il seguente:

```
...

hosts:          files dns
networks:       files
protocols:      db files
services:       db files
ethers:         db files
rpc:            db files

netgroup:       nis
```

ed ad esempio nel caso di risoluzione dei nomi la riga che interessa è quella relativa a hosts che 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 si effettuerebbe una ricerca su LDAP.

7.3.5 Il file /etc/resolv.conf

È nel file /etc/resolv.conf che vengono memorizzate le informazioni principali relative alla risoluzione dei nomi, come il dominio locale e l'IP del nameserver che vengono di solito indicati in fase di installazione o 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; di solito ogni provider ha il suo, e voi potete creare anche il vostro DNS locale.

Anche questo file controlla il comportamento delle funzioni del cosiddetto *resolver*, di norma contiene il nome del dominio e la lista degli IP dei nameserver; il formato 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 è il seguente:

```
nameserver 127.0.0.1
search firenze.linux.it
nameserver 195.110.99.218
```

La direttiva `nameserver` serve ad indicare la specificazione del numero IP di un nameserver, la risoluzione di un nome verrà eseguita nell'ordine in cui esse sono state specificate. Normalmente si possono specificare fino ad un massimo di 3 diversi nameserver, l'indirizzo deve sempre essere fornito in forma numerica.

La direttiva `search` specifica una lista di domini in cui cercare i nomi; di norma contiene solo il dominio locale, così quando si specifica un nome gli viene aggiunto automaticamente quel dominio. In questo modo si può specificare solo l'hostname del computer, e la ricerca assumerà che esso si trovi nella lista dei domini qui elencati.

Un elenco delle opzioni principali è riportato in tab. 7.12; la descrizione più dettagliata di tutte le opzioni 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`	lista dei domini su cui effettuare una ricerca preventiva
`domain`	nome del dominio locale, se assente viene determinato sulla base di quanto specificato con `/etc/hostname`

Tabella 7.12: Direttive del file di con `resolv.conf`.

Se siete su una rete locale dovete impostare questi valori manualmente (in genere questo viene fatto dal programma di configurazione della rete, che, quando vi chiede dominio e DNS, va a creare automaticamente questo file). 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) 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 qualche altro DNS (ce ne sono di pubblici) qualora quello del vostro provider avesse problemi.

7.3.6 Il file `/etc/host.conf`

Questo file controlla le modalità di funzionamento delle routine che eseguono la risoluzione dei nomi (il *resolver*). Al solito le righe vuote od inizianti per `#` sono ignorate, le altre devono contenere una parola chiave che esprime una direttiva seguita o meno da un valore.

Un esempio comune del contenuto di questo file è il seguente:

```
order hosts,bind
multi on
```

la prima direttiva controlla la sequenza in cui viene effettuata la risoluzione dei nomi, e dice che deve prima essere usato il file `hosts` di sez. 7.3.2, e poi le interrogazioni ai DNS esterni. La seconda permette di avere indietro tutti gli indirizzi validi di una stazione che compare più volte in `hosts`, invece di avere solo il primo.

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 7.13: Direttive del file di con `host.conf`.

Oltre a queste due le principali direttive ed il relativo significato è riportato in tab. 7.13, tutte le direttive ivi riportate prendono come argomenti `on` ed `off` che attivano e disattivano

il comportamento richiesto, se non esplicitamente richiesto inoltre il comportamento di default è equivalente ad `off`. Per i dettagli completi si può consultare la pagina di manuale con `man host.conf`.

7.4 Il protocollo PPP

Come accennato in sez. 6.2.1 il PPP è uno dei protocolli del livello di rete su cui si innestano tutti i protocolli dei livelli successivi. Nel caso specifico PPP è un protocollo generico che permette di inviare dati su diversi collegamenti seriali punto-punto.

Il protocollo si compone di tre parti, un metodo generico per incapsulare dati su collegamenti seriali, un protocollo estensibile per il controllo del collegamento (*Link Control Protocol* o LCP) ed una famiglia di protocolli (*Network Control Protocols* o NCP) per stabilire e configurare le connessioni coi protocolli di livello superiore.

In generale esso 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 gli altri protocolli. L'uso più tipico è comunque quello con i modem analogici.

7.4.1 Il demone `pppd`

Delle tre parti cui è composto il protocollo PPP, la incapsulazione è gestita dal kernel, il resto è affidato ad un demone apposito, il `pppd`, che si cura di fornire il controllo del collegamento, l'autenticazione, e l'NCP che riguarda la configurazione e la gestione del protocollo IP su PPP.

La sintassi generica con cui si può chiamare il programma, 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 service 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. 7.14, 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
`call name`	Usa le opzioni contenute nel file `name` in `/etc/ppp/peers`.
`connect script`	Usa lo script `script` (di norma `chat`) per preimpostare la linea.
`crtscts`	Attiva il controllo di flusso hardware sulle porte seriali.
`defaultroute`	Aggiunge una rotta di default alla tabella di routing usando l'IP ricevuto sulla connessione.
`lock`	Crea un file di lock per la seriale.
`demand`	Crea la connessione a richiesta quando vede del traffico.
`usepeerdns`	Chiede alla connessione gli indirizzi di due server DNS, che vengono passati allo script `/etc/ppp/ip-up` nelle variabili di ambiente `DNS1` e `DNS2` ed usati per la creazione di un file `resolv.conf` che li usi come nameserver.
`debug`	Abilita il debugging, è utile per avere più informazioni quando la connessione non funziona.

Tabella 7.14: Opzioni del 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:

```
ogin: 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 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.

7.4.2 I meccanismi di autenticazione

Al di la 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 CIFS 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 la password inizia per `@` si indica che essa è contenuta nel file specificato di seguito.

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 La gestione dei servizi generici

In genere su ogni macchina in rete viene installata una serie di piccoli servizi generici, spesso attivi solo per uso di test, come `echo` o `discard`, che non vengono effettuati da altrettanti demoni perennemente attivi, ma attraverso un apposito programma che viene chiamato *superdemone*, il cui compito è un po' quello della segretaria, che riceve le telefonate sul centralino e poi passa ciascuna di esse all'ufficio competente.

Di norma i servizi più importanti ed usati, 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. Inoltre esistono vari programmi in grado di svolgere questo compito, anche se i due principali sono il tradizionale `inetd`, ed il più moderno `xinetd`.

8.1.1 Il superdemone `inetd`

Tradizionalmente la gestione dei servizi generici è sempre stata effettuata tramite il programma `inetd`, che ancor oggi viene usato dalla gran parte delle distribuzioni. È attraverso questo programma che di norma vengono lanciati invece alcuni fra i servizi più semplici e meno usati, come `telnet` o `ftp`, che non vale la pena di mantenere in memoria. In questo caso è `inetd` che si mette in ascolto sulle rispettive porte, e lancia il relativo programma solo quando qualcuno effettua una connessione.

Il programma viene di norma lanciato dagli script di avvio, ed al solito può essere controllato manualmente con il relativo script che di norma è `/etc/init.d/inetd`. In caso di necessità lo si può eseguire anche direttamente a riga di comando, nel qual caso si può usare l'opzione `-d` per farlo partire in modalità di debug, mentre con `-q` si può importare la profondità della coda delle connessioni in ingresso per cui si sta in ascolto.

Il programma inoltre fornisce direttamente alcuni servizi elementari come `echo`, `discard` o `daytime`, per i quali non è necessario lanciare nessun programma. Ma l'aspetto interessante è che `inetd` è in generale in grado di associare ad una porta un programma qualunque. Quello che succederà è che in occasione di una connessione di rete esso eseguirà il programma collegando sia lo standard input che lo standard output dello stesso al socket, così ad esempio si può attivare il servizio `netstat` semplicemente facendogli lanciare l'omonimo programma, il cui risultato sarà inviato via rete.

8.1.2 Il file `/etc/inetd.conf`

Il file di configurazione 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 campi sono separati da spazi o tabulatori; al solito righe vuote e iniziate da `#` sono ignorate.

Il servizio da lanciare è identificato dal primo campo, tramite il suo valore simbolico così come viene indicato in `/etc/services`. Il successivo campo indica il tipo di socket: i due tipi più comuni sono `stream` o `dgram`, ma si possono indicare tutti i tipi di socket supportati con Linux, (come `raw`, `rdm`, or `seqpacket`). Il terzo campo indica il protocollo usato: di norma si tratta di `tcp` o `udp`, (ma si può specificare un qualunque nome valido riportato in `/etc/protocols`) che in pratica corrispondono rispettivamente ai tipi `stream` e `dgram`.

Il quarto campo è applicabile solo quando si è usato un socket `dgram`, e indica se il server è in grado di trattare altri pacchetti mentre sta rispondendo (nel qual caso si imposta `nowait`) o meno, nel qual caso si deve aspettare la fine del processo corrente prima di rilanciare il programma; per gli altri socket deve essere `nowait`. Il quinto campo indica l'utente per conto del quale viene eseguito il programma; se si vuole specificare anche un gruppo diverso da quello di default, lo si può fare scrivendo il campo nella forma `user.group`.

Il quinto 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`. 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 su una riga di comando, ma dovranno essere preceduti dall'argomento iniziale (che la riga di comando costruisce automaticamente, per cui non viene mai specificato), 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 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. 8.2.

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 specificato, prima dei parametri `-ant`, il nome del programma `netstat`; senza questo il risultato sarebbe stato che all'esecuzione di `/bin/netstat` nome del processo (come ottenibile attraverso il comando `ps`) sarebbe risultato `-ant`, mentre le opzioni sarebbero state perse.

A questo punto 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 0.0.0.0:709            0.0.0.0:*               LISTEN
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 0.0.0.0:884            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 0.0.0.0:119            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 0.0.0.0:444            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
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`, come se lo si fosse eseguito sulla nostra macchina.

Questa è una caratteristica generale del comportamento di `inetd`. Il superdemone infatti è in grado di eseguire un programma qualsiasi; in generale sarà compito del programma eseguito gestire la connessione di rete, ma `inetd` quando lancia il programma fa sì che standard input, standard output e standard error siano comunque associati al socket su cui è stata aperta la connessione. Questo significa che si può lanciare attraverso `inetd` un programma qualsiasi, e questo accetterà l'input dal socket e su di esso scriverà il suo output.

Nell'esempio appena mostrato quello che è successo è che è stato eseguito il programma `netstat` sul server remoto che come al solito ha prodotto la lista dei socket attivi scrivendola sullo standard output, per poi terminare. Questo ha fatto sì che noi la ricevessimo sul socket creato dalla connessione effettuata con `telnet`, che è stato automaticamente chiuso alla terminazione del processo. La cosa 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 fornisce l'elenco dei processi, o quello che vogliamo.

8.1.3 Il superdemone `xinetd`

Il programma `xinetd` nasce come estensione 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 come per i *TCP wrapper*, 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 redirezione delle connessione, delle capacità di logging estese, dei meccanismi di protezione nei confronti dei portscan, delle 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 tipico del suo contenuto 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.
        # 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 `+=` per aggiungere e `-=` per togliere dei valori agli attributi che 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 il logging; 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'RFC1413) 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 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 degli 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`.

Come accennato il programma è in grado di implementare direttamente dei controllo di accesso, che di utilizzare quelli eventualmente specificati tramite i file di controllo dei *TCP wrappers*[1] (vedi sez. 8.2.2), i due attributi `only_from` e `no_access` però 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
```

[1] Si tenga presente che l'uso dei *TCP wrapper* ha di norma la precedenza sul controllo interno, per cui se si abilita l'accesso con queste direttive, ma esso è negato dai TCP wrapper, questi ultimi avranno la meglio.

Attributo	Descrizione
instances	Numero massimo di processi lanciati per ogni servizio. Un argomento intero.
log_type	Metodologia di registrazione dei log, su file o tramite *syslog*. Opzioni FILE o SYSLOG.
log_on_success	Cosa registrare per le connessioni riuscite. Uno o più fra PID, HOST, USERID, EXIT, DURATION.
log_on_failure	Cosa registrare per le connessioni fallite. Uno o più fra HOST, USERID, ATTEMPT.
per_source	Numero massimo di connessioni per IP sorgente. Un argomento intero.
only_from	Lista degli host da cui è possibile accedere. Indirizzi in forma dotted decimal, CIDR o simbolica.
no_access	Lista degli host da cui è impossibile accedere. Indirizzi in forma dotted decimal, CIDR o simbolica.
access_times	Intervallo temporale nel quale è possibile accedere ai servizi. Intervallo temporale nella forma HH:MM-HH:MM.
cps	Rate massimo di accesso. Un argomento decimale per indicare il limite sulle connessioni al secondo, ed un argomento intero per specificare il numero di secondi da aspettare prima di accettare nuove connessioni una volta superato il limite.
nice	Valore di nice da applicare ai demoni lanciati. Un argomento intero.
max_load	Carico massimo oltre il quale la macchina smetta di accettare connessioni. Un argomento decimale.

Tabella 8.1: Attributi specificabili in generale per tutti i servizi gestiti attraverso il superdemone xinetd.

```
        type            = INTERNAL
        id              = time-stream
        socket_type     = stream
        protocol        = tcp
        user            = root
        wait            = no
}

service time
{
        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. Li si sono poi suddivisi in due sezioni, la prima contenente quelli che, a certe condizioni, devono comunque essere specificati, la seconda quelli che sono sempre opzionali. Si ricordi che anche i precedenti argomenti visti in tab. 8.1 possono essere utilizzati e saranno applicati solo al servizio in questione. Al solito nelle pagine di manuale è riportato un elenco completo di tutti gli argomenti presenti e la descrizione dettagliata di ciascuno di essi.

Attributo	**Descrizione**
`socket_type`	Tipo di socket. Prende gli stessi valori (`stream`, `dgram`, ecc.) dell'analogo parametro in `inetd.conf`.
`user`	Utente per conto del quale è lanciato il servizio. Deve essere presente in `/etc/passwd`. Si può specificare un eventuale gruppo con l'attributo `group`.
`server`	Pathname del programma server da lanciare. Va sempre specificato se il servizio non è gestito internamente.
`wait`	Indica se si deve attendere o meno la conclusione del server per lanciare un'altra istanza. Analogo dello stesso parametro in `inetd.conf`. Prende i valori `yes` e `no`.
`protocol`	Protocollo usato (analogo di `tcp` ed `udp` per `inetd.conf`). Deve essere un nome valido in `/etc/protocols`.
`port`	Porta su cui ascoltare le connessioni. Deve essere specificata se si è specificato un servizio non riportato in `/etc/services`.
`type`	Tipo di servizio. Può assumere i valori `INTERNAL` (per servizi gestiti internamente) e `UNLISTED` per servizi non presenti in `/etc/services`.
`server_args`	Eventuali argomenti da passare al programma server quando viene lanciato. Non necessita di specificare l'argomento iniziale come per `inetd`.
`disable`	Indica se attivare il servizio, il default è disattivo. Può assumere i valori `yes` e `no`.
`id`	Identificatore aggiuntivo qualora si tratti di servizi diversi con lo stesso nome.
`bind`	Consente di specificare l'interfaccia su cui fornire il servizio. Prende l'IP ad essa associato.
`redirect`	Consente di redirigere il servizio ad un'altra macchina. Prende l'indirizzo IP e la porta verso quale redirigere tutto il traffico.

Tabella 8.2: Attributi specificabili per un servizio gestito attraverso il superdemone `xinetd`.

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.

8.2 I TCP wrappers

In generale la gran parte dei demoni di rete, nati in un periodo in cui internet era una rete costituita principalmente da istituzioni ed enti e l'accesso era molto limitato, non provvedono nessun tipo di controllo degli accessi, e permettono a chiunque di collegarsi, da qualunque macchina si trovi.

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 appunto *TCP wrappers* che permettono di specificare un controllo degli accessi che consente il collegamento a certi servizi solo da parte di certe stazioni o reti, (insomma, una specie di filtro telefonico alla rovescia che impedisce che ci possano chiamare da certi numeri).

8.2.1 Il comando `tcpd` e le librerie `libwrap`

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`, da cui deriva appunto il nome, che fa da *involucro* 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 delle librerie di controllo all'interno del demone, che deve essere collegato alle stesse, in modo da poterne usare direttamente le funzionalità. Sono esempi di questa modalità servizi come SSH, LDAP o NFS.

In entrambi i casi le connessioni ai servizi controllati vengono archiviate tramite *syslog* in modo da lasciare traccia di eventuali tentativi non autorizzati di accesso, dopo di che vengono eseguiti i vari controlli che prevedono.

8.2.2 I file `hosts.allow` e `hosts.deny`

Il controllo di accesso implementato dai *TCP wrappers* è gestito attraverso questi due file, 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 ed il controllo finisce qui; 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 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:

```
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
```

dove la terza parte (`: comando shell`) è opzionale e può essere omessa.

Un altro esempio, più significativo del precedente, è quello del contenuto di `hosts.allow` riportato di seguito:

```
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*, 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. 8.6.1). 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 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 una `*` come wildcard per raggruppare indirizzi; si può anche specificare una netmask 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.

8.2.3 I comandi `tcpdchk` e `tcpdmatch`

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 gli `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.

8.3 La gestione di un server DNS

Il DNS è uno dei protocolli fondamentali per il funzionamento di internet, come già accennato 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 in maniera veloce ed efficiente.

8.3.1 Il funzionamento del DNS

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, 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 8.3: 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, corrisponden-

ze 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. 8.3, un elenco completo si può trovare nella pagina di manuale del comando `host`.

8.3.2 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. 8.3. 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. 8.3). 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.

8.3.3 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. 6.2.4) 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. 8.4, 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 8.4: 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 server[2] per le normali operazioni del DNS. L'utente dipende dalla distribuzione e dall'installazione che si è effettuata.

8.3.4 Il file `named.conf`

Il file di configurazione di `named` è `named.conf`, di solito si trova questo file o in `/etc` o in `/etc/bind/` 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 terminati 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 di `named.conf`, ripreso dal file installato di default su una Debian, è riportato in fig. 8.1; 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.

8.3.5 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

[2] dovendo 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.

```
// 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; };
        category cname { null; };
};
...
```

Figura 8.1: Sezione iniziale del file `named.conf`.

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. 8.3.2) 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. 8.3.6, 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. 8.3.1, 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`.

8.3.6 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. 8.3.1); 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 di 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. 8.3.1), 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 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
;
                        MX      10 gont
                        MX      20 oppish
;
; 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 `earhsea.ea`, in questo caso il nameserver è la macchina `gont.earhsea.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.

;; 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.

8.3.7 La configurazione con bind4

Benché ormai la gran parte dei server sia passata alle versioni più recenti di `bind` (la versione 8 o la 9) esistono ancora un certo numero di server che mantengono la vecchia versione 4. Il passaggio da questa alle successive ha visto un grosso cambiamento del server, ed in particolare della sua configurazione.

Il cambiamento principale (a parte la struttura interna del server stesso) è stato quello nei file di configurazione; la versione 4 infatti non usa `named.conf` ma un file `named.boot`, che ha una sintassi completamente diversa; le differenze per fortuna si fermano qui, ed i file che definiscono le zone sono sostanzialmente immutati.

Un possibile esempio di file `named.boot`, con una configurazione che è sostanzialmente equivalente a quella vista nei paragrafi precedenti, è il seguente:

```
;
; Bind v4 configuration file
;
directory  /etc/bind
forwarders  213.234.128.211  213.234.132.130
cache           .                  db.root
; zone definitions
; type     domain                  file
primary    0.0.127.in-addr.arpa    db.127
primary    localhost               db.local
primary    earthsea.ea             db.earthsea
primary    1.168.192.in-addr.arpa  db.192.168.1
```

La direttiva `directory` specifica la directory di lavoro corrente del server, così che tutti i pathname relativi impiegati in seguito faranno riferimento ad essa. La direttiva `primary` permette di definire un dominio per cui si è DNS primario; deve essere seguita dal nome del dominio e dal file in cui sono contenute le informazioni sui relativi resource record. La direttiva `secondary` invece permette di definire un dominio per cui si è DNS secondario, deve essere seguita dal nome del dominio, dalla lista dei server primari e dal file su cui salvare i dati. La direttiva `forwarders` permette, come l'analoga opzione di bind8, di reinviare le richiesta ad altri server DNS, la cui lista è a seguire. Infine la direttiva `cache` permette di ottenere le informazioni su un dominio da un file (è analoga ad una zona di tipo `hint` di bind8) e nel caso permette di specificare la lista dei root DNS.

Direttiva	Significato
`domain <nome>`	Imposta il nome di dominio di default.
`directory <path>`	Specifica la directory di lavoro del server.
`primary <dominio> <file>`	Definisce un dominio per cui si è DNS primario.
`secondary <dominio> <lista> <file>`	Definisce un dominio per cui si è DNS secondario.
`cache <dominio> <file>`	Definisce un file di cache per un dominio.
`forwarders <list>`	Imposta i DNS a cui reinviare le richieste.
`options <opzione>`	Imposta una opzione.

Tabella 8.5: Opzioni impostabili nel file `named.boot`.

L'elenco delle direttive principali è riportato in tab. 8.5; l'elenco completo delle opzioni si può trovare nella pagina di manuale di `named.boot`, insieme alla documentazione di bind4.

8.4 I protocolli ARP e DHCP

In questa sezione tratteremo due protocolli che sono accumunati dall'essere entrambi relativi alla gestione dei numeri IP. Il primo, l'*Address Resolution Protocol* serve per identificare che ha un certo IP su una rete locale, il secondo per effettuare l'assegnazione dinamica dei numeri IP, sempre all'interno di una rete locale.

8.4.1 Il protocollo ARP ed il comando `arp`

Come accennato in sez. 6.2.1 il protocollo ARP viene usato (dal kernel) per associare a ciascun indirizzo fisico presente sulla rete locale il relativo indirizzo IP, cosicché il kernel possa sapere a quale scheda ethernet mandare i pacchetti indirizzati verso un IP che sta sulla sua stessa sottorete.

Il protocollo viene usato per mandare delle richieste in *broadcast* (cioè che vengono ricevuta da tutte le schede ethernet su una stessa LAN), queste hanno la tipica forma "*dì 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 è riportato nella 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 *XX:XX:XX:XX:XX:XX*" 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*, e rinnovate solo dopo che sono scadute.

In certi casi può essere utile esaminare e modificare la *ARP cache*. Ad esempio una tecnica comunemente usata quando si divide in due una rete introducendo un router è quella del *proxy ARP*. Dato che il router non trasmette i pacchetti a livello 1 si fa sì che alle richieste ARP per gli IP posti al di là del router venga risposto l'indirizzo della scheda del router, che così li riceverà anche senza aver impostato su tutte le macchine una opportuna rotta statica. Ovviamente questa è una soluzione provvisoria, che come tutte le soluzioni provvisorie ha la brutta abitudine di diventare spesso definitiva.

Opzione	Significato
-a host	mostra le voci relative al nodo host, se non specificato mostra tutte le voci.
-f file	legge i valori da immettere nella cache dal file file.
-n	non effettua la risoluzioni di nomi e indirizzi.
-s host val	inserisce una voce nella cache, associando al nodo host l'indirizzo val.
-d host	cancella la voce relativa al nodo host.

Tabella 8.6: Opzioni del comando arp.

Il comando che permette di esaminare e modificare la *ARP cache* è arp, la sua sintassi, come riportata dalla pagina di manuale, è la seguente:

```
arp [-vn] [-H type] [-i if] -a [hostname]
arp [-v] [-i if] -d hostname [pub]
arp [-v] [-H type] [-i if] -s hostname hw_addr [temp]
arp [-v] [-H type] [-i if] -s hostname hw_addr [netmask nm] pub
arp [-v] [-H type] [-i if] -Ds hostname ifa [netmask nm] pub
arp [-vnD] [-H type] [-i if] -f [filename]
```

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 sono la cancellazione e l'inserimento 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. 8.6, per una spiegazione completa si può fare ricorso alla pagina di manuale con `man arp`.

Come per la risoluzione dei nomi (vedi ad esempio sez. 7.3.6) è possibile anche specificare un certo numero di corrispondenze manualmente usando un opportuno file di configurazione, che nel caso è `/etc/ethers`. Questo contiene delle corrispondenze nella forma:

```
MAC-address numero-IP
```

anche se al posto del numero IP si può anche specificare un nome, posto che questo sia risolvibile. Il formato dei MAC address è quello solito di sei byte esadecimali separati da `":"`, qualcosa del tipo:

```
08:00:20:00:61:CA  129.168.1.245
```

8.4.2 Configurazione del server DHCP

Il protocollo DHCP, sigla che sta per *Dynamic Host Configuration Protocol*, è un protocollo per la configurazione automatica delle reti che opera direttamente sopra il livello di collegamento fisico.

Il protocollo prevede la possibilità da parte di una singola stazione di effettuare delle richieste in *broadcast* sulla rete locale, alla ricerca di un server al quale chiedere quale indirizzo IP utilizzare. In questo modo l'indirizzo IP può essere assegnato dinamicamente, evitando di doverlo specificare con la procedura vista in sez. 7.1 per ciascuna delle macchine.

Si può così centralizzare l'assegnazione dei numeri IP, ed inserire anche dei limitati meccanismi di controllo sulle macchine presenti in rete, sulla base dei MAC address delle varie schede. Si tenga comunque presente che è sempre possibile assegnare un IP anche senza passare dal server DHCP, e che è possibile falsificare i MAC address, quindi il controllo effettuabile è relativo.

Dal lato server il servizio DHCP viene realizzato da un apposito demone, il programma `dhcpd`, che si incarica di ascoltare le richieste e fornire le risposte. Il server è controllato da un file di configurazione che di norma è `/etc/dhcp.conf`; un esempio è riportato in fig. 8.2.

Il formato del file è relativamente semplice, e assomiglia a quello di `named`. Vale la solita regola che linee vuote e tutto ciò che segue un `#` viene ignorato. Il contenuto è divisibile sommariamente in due categorie di direttive, la specificazione di parametri e le dichiarazioni. Un esempio di specificazione di parametri sono le righe iniziali che nel caso sono utilizzate per impostare alcuni parametri globali, come il dominio di riferimento o il nameserver, mentre un esempio di dichiarazione è quello relativo alla definizione delle sottoreti e degli host.

Un elenco delle principali opzioni impostabili come parametri (nella forma `option name valore;`) è riportato in tab. 8.7, a queste vanno aggiunte le dichiarazioni dirette dei parametri come quelle relative ai tempi massimi per cui vengono mantenute le corrispondenze[3] come `max-lease-time` e `default-lease-time` che sono specificati in secondi.

Le opzioni possono essere inserite sia direttamente nel corpo principale di `dhcpd.conf` che all'interno di dichiarazioni. Queste ultime sono sempre nella forma `keyword { parameter value; ... }` come mostrato per le due dichiarazioni relative una rete ed una stazione mostrate in fig. 8.2.

Si noti come il protocollo consenta di dividere gli IP restituiti alle stazioni in sottoreti, nel nostro caso basterà usarne una. La parola chiave `subnet` specifica una sottorete, ed è seguita dall'indirizzo della rete e dalla specificazione della relativa netmask. All'interno della dichiarazione poi si può indicare l'intervallo di indirizzi assegnati con il parametro `range`, ed il

[3] di norma una volta assegnato un numero IP il server mantiene l'associazione con la macchina cui l'ha assegnato (identificata dal MAC address) per un certo tempo, onde evitare di riassegnare immediatamente l'IP ad un'altra macchina in caso di assenza temporanea della prima, cosa che provocherebbe delle interferenze con le cache ARP (vedi sez. 8.4.1) delle macchine sulla stessa rete locale.

```
#
# Sample configuration file for ISC dhcpd for Debian
#
# $Id: netinter.tex,v 1.2 2004/03/06 18:59:52 piccardi Exp $
#

# 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;
}
```

Figura 8.2: Esempio del file di configurazione `dhcp.conf`.

relativo router. Si possono anche rispecificare (cambiandole rispetto a quelle generali impostate all'inizio del file) le varie opzioni di tab. 8.7.

Infine un'altra dichiarazione utile è quella relativa ad una stazione singola, anch'essa mostrata in fig. 8.2, introdotta dalla parola chiave `host` e dal nome della stazione, che permette di assegnare ad una macchina, identificata dal MAC address (specificato dal parametro `hardware ethernet`), un indirizzo fisso (specificato dal parametro `fixed-address`).

Per una lista completa dei parametri si può fare riferimento alla pagina di manuale del file di configurazione accessibile con `man dhcpd.conf`.

8.4.3 Uso del DHCP come client

Per poter usufruire dei servizi di un server DHCP esistono vari programmi; i più comuni e diffusi sono `pump` e `dhclient`. L'uso di questi programmi è estremamente semplice la sintassi è rispettivamente:

```
pump -i eth0
```

oppure

```
dhclient eth1 eth2
```

Di norma in fase di configurazione della rete si può sempre specificare che l'indirizzo di una interfaccia deve essere ottenuto dinamicamente, nel qual caso è compito degli script di avvio della rete chiamare autonomamente uno di questi programmi.

Distribuzione	Comando
`domain-name`	Specifica il nome di dominio in cui ci si trova (ad uso dell'impostazione automatica di `resolv.conf`).
`domain-name-servers`	Specifica una lista di nameserver (anche questo per l'impostazione automatica dei relativi campi in `resolv.conf`).
`subnet-mask`	Imposta la netmask per la rete associata all'indirizzo assegnato.
`broadcast-address`	Imposta l'indirizzo di broadcast per la rete associata all'indirizzo assegnato.
`routers`	Imposta il default gateway per la rete associata all'indirizzo assegnato.

Tabella 8.7: Opzioni impostabili nel file `dhcpd.conf`.

8.5 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.

8.5.1 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`	La versione del protocollo. **Deve** essere impostata a 2, in quanto le precedenti versioni sono insicure. È possibile abilitarle 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 `su`.
`X11Forwarding`	Abilita il *forwarding* 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 `sftp`.

Tabella 8.8: Principali opzioni di configurazione per il demone `sshd` usate nel file `sshd_config`.

Si tenga presente inoltre che `sshd` onora la presenza del file `/etc/nologin` non consentendo, quando esso esiste, la connessione ad utenti che non siamo l'amministratore, che l'utilizzo dei TCP wrappers,

8.5.2 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 in fig. 8.3; di norma, come nell'esempio, non è necessario impostare niente di diverso dai default. L'amministratore però può volere imporre delle restrizioni particolare all'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 ~
```

Figura 8.3: Estratto del file di configurazione `ssh_config`.

Esempi delle possibili opzioni impostabili sono riportati direttamente in fig. 8.3; 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, come riportata dalla pagina di manuale è:

```
scp [-pqrvBC1246] [-F ssh_config] [-S program] [-P port] [-c cipher]
    [-i identity_file] [-o ssh_option] [[user@]host1:]file1 [...]
    [[user@]host2:]file2
```

ed è simile 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 gapil.pdf 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`.

8.5.3 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.

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`; 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,[4] 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 da 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 attiene 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.

[4] le chiavi sono mantenute una per riga, in formato codificato ASCII, per cui le singole righe sono di norma molto lunghe.

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 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.

8.6 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.

8.6.1 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 prestazione 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 (*Remote Procedure Call*), un meccanismo di intercomunicazione generico basato sui protocolli di trasporto (TCP e UDP), che permette ai processi di accedere a dei servizi (nella forma di chiamate a procedure esterne) in maniera trasparente rispetto alla rete. Ciascun servizio è fornito da un apposito demone, in ascolto su una porta generica: un programma che lo voglia utilizzare deve prima rivolgersi al cosiddetto *portmapper* che registra i servizi attivi ed è in grado di indicare la porta su cui sono forniti.

Il servizio del portmapper è realizzato dal demone `portmap`, è l'unico cui viene assegnata una porta definita, la 111, che corrisponde al servizio chiamato `sunrpc`. 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:

```
/pat/to/directory machine1(option11,option12) machine2(option21,option22)
```

dove il separatore è costituito dallo spazio.[5]

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.[6]

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`.

[5] attenzione 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.

[6] questa 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.

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`.

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
`nohide`	mostra
`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`.

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:

```
/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. 8.2.2. 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`.

8.6.2 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.

8.7 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.

8.7.1 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 di 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.

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. 8.7.2.

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
```

Parametro	Significato
`security`	Il parametro 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 *workgroup name*).
`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 *ospite* (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 *NetBIOS* di NT.
`wins server`	Indica un server per la risoluzione dei nomi *NetBIOS* 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`.

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.

8.7.2 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. 8.7.1 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 `passdb 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 dato 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`.

8.7.3 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

GNU Free Documentation License

Version 1.1, March 2000

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. Secondarily, 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.

9.1 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, LaTeX 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.

9.2 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.

9.3 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.

9.4 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.

9.5 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."

9.6 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.

9.7 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.

9.8 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.

9.9 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.

9.10 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