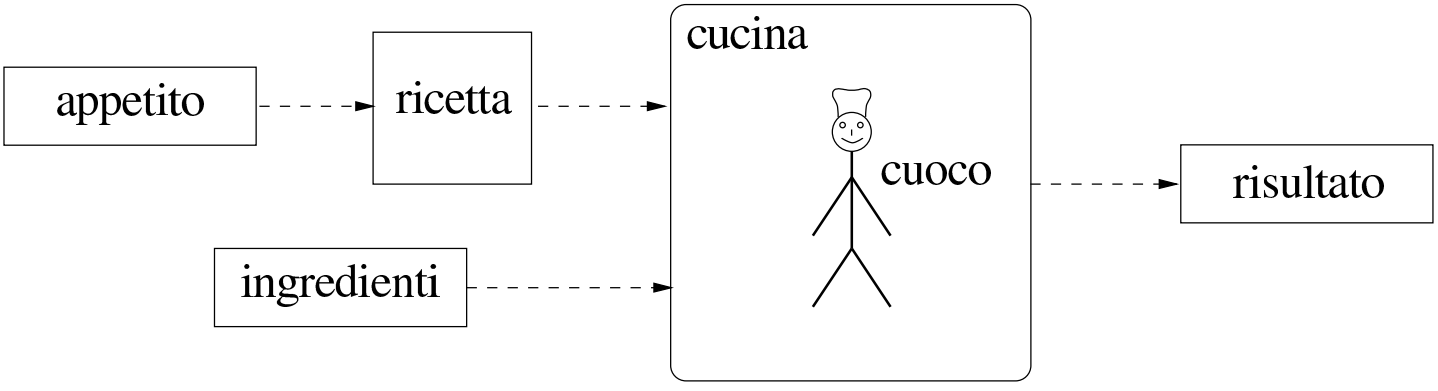
Introduzione all'uso

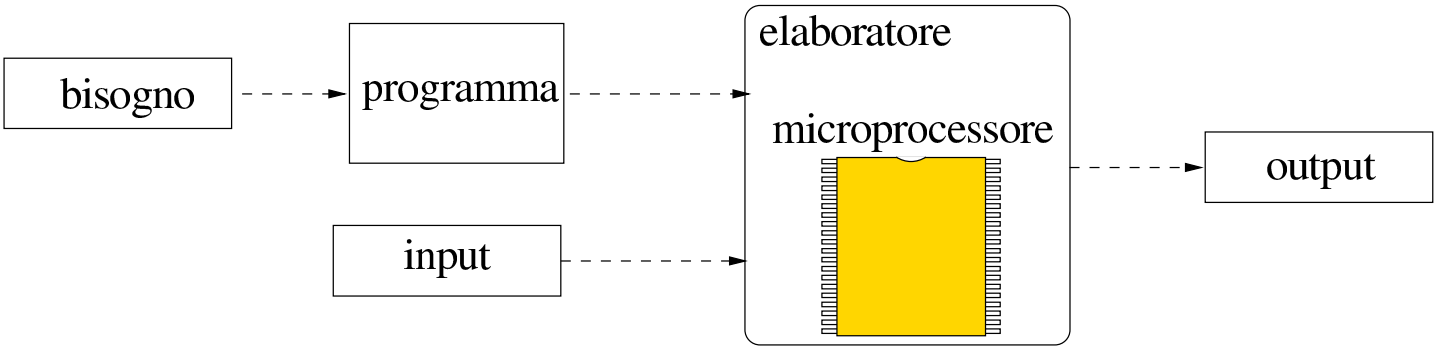
3.1 L'elaboratore

Per comprendere la struttura di un elaboratore si può immaginare il comportamento di un cuoco nella sua cucina. Il cuoco prepara delle pietanze, o piatti, che gli sono stati ordinati, basandosi sulle indicazioni delle ricette corrispondenti. Le ordinazioni vengono effettuate dai clienti che si rivolgono al cuoco perché hanno appetito.



- L'elaboratore è la cucina;
- il cuoco è il microprocessore o CPU;

- l'appetito è il bisogno da soddisfare ovvero il problema da risolvere;
- la ricetta è il programma che il microprocessore deve eseguire;
- gli ingredienti sono l'input del programma;
- le pietanze o i piatti sono l'output del programma.

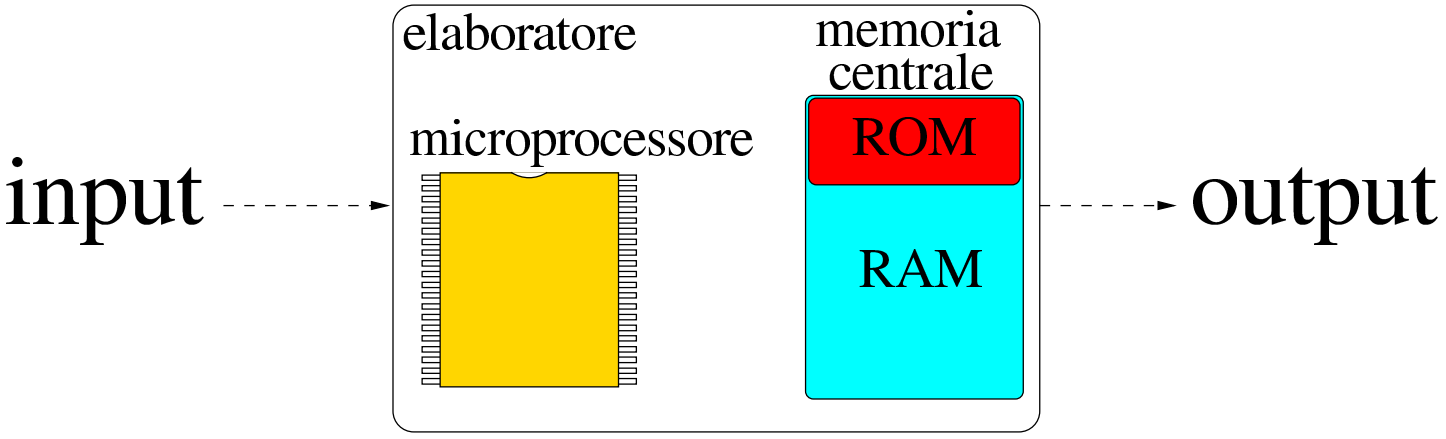


Per poter lavorare, il cuoco appoggia tutto quanto, ingredienti e ricetta, sul tavolo di lavoro. Su una parte del tavolo sono incise alcune istruzioni che al cuoco servono sempre, in particolare quelle che il cuoco deve eseguire ogni volta che la cucina viene aperta: pulire il tavolo, controllare tutti gli strumenti (pentole, tegami, coltelli, cucchiai ecc.) e ricevere le ordinazioni assieme alle ricette. Senza queste istruzioni di inizio, il cuoco non saprebbe nemmeno che deve accingersi a ricevere delle ordinazioni. Il tavolo di lavoro viene pulito alla chiusura della cucina e ciò che vi è rimasto sopra, si perde, a esclusione delle ricette incise; pertanto, oltre al tavolo, il cuoco può avvalersi di una dispensa, nella quale conserva ricette, ingredienti, pietanze pronte o in fase di preparazione.

Il tavolo di lavoro del cuoco corrisponde alla ***memoria centrale*** (o *core*) che si suddivide storicamente in una parte volatile, RAM, e in una parte non volatile, ROM. La ROM è quella parte di memoria che non può essere alterata (nell'esempio del cuoco, si tratta di istruzioni

incise sul tavolo); la RAM è il resto della memoria che può essere alterata a piacimento dalla CPU (il resto del tavolo) e che viene azzerata al termine del lavoro. La dispensa, se c'è, è la ***memoria di massa***.

L'elaboratore è pertanto una macchina composta da una o più CPU che si avvalgono di una memoria centrale per trasformare l'input (i dati in ingresso) in output (i dati in uscita).



Il cuoco deve seguire una procedura ordinata per svolgere il proprio lavoro senza intoppi, soprattutto quando gli si richiede l'esecuzione di più ricette simultaneamente. Tale procedura corrisponde nell'elaboratore al ***sistema operativo***, ovvero al programma più importante, attivato al momento dell'accensione dell'elaboratore, il quale è responsabile dell'esecuzione degli altri programmi applicativi e del controllo del loro utilizzo di risorse.

3.2 Dispositivi

L'elaboratore, per poter ricevere l'input e per poter produrre all'esterno l'output, ha bisogno di ***dispositivi***; per esempio tastiera, schermo e unità di memorizzazione. I dispositivi sono qualcosa che è separato dall'elaboratore inteso come l'insieme di CPU e memoria centrale. A seconda del tipo e della loro collocazione, questi posso-

no essere interni o periferici, ma tale distinzione è quasi scomparsa nel linguaggio normale, tanto che a volte si usa il termine «periferica» per fare riferimento genericamente a un dispositivo qualunque. Tuttavia, vale la pena di distinguere fra tre tipi di dispositivi fondamentali: dispositivi di memorizzazione; dispositivi per l'interazione tra l'utente e l'elaboratore; interfacce di rete.

I dispositivi di memorizzazione sono ciò che è in grado di conservare dati anche dopo lo spegnimento della macchina. Il supporto di memorizzazione vero e proprio potrebbe essere parte integrante del dispositivo stesso oppure essere rimovibile. I supporti di memorizzazione possono essere di qualunque tipo, anche se attualmente si è abituati ad avere a che fare prevalentemente con dischi o memorie solide. In passato si è usato di tutto e il primo tipo di supporto di memorizzazione sono state le schede di cartoncino perforate.

Anche i dispositivi per l'interazione con l'utente possono avere qualunque forma possibile e immaginabile, anche se tradizionalmente si tende a fare riferimento principalmente a tastiera, schermo e mouse.

Le interfacce di rete sono i dispositivi che consentono la connessione tra diversi elaboratori in modo da permettere la condivisione di risorse e la comunicazione in generale.

Se si lascia da parte il periodo delle schede perforate, si può dire che il primo tipo di strumento per l'interazione tra utente e macchina sia stato la telescrivente: una sorta di macchina da scrivere in grado di ricevere input dalla tastiera e di emettere output attraverso la stampante. In questo modo, l'input umano (da tastiera) era fatto di righe di testo terminate da un codice per il ritorno a capo (interruzione di riga, o *new-line*) e nello stesso modo era composto l'output

che appariva su carta.

Con l'introduzione della telescrivente si sono sviluppati i sistemi operativi ***interattivi***, i quali consentono l'uso di programmi che interagiscono con l'utente. Questo tipo di funzionamento si contrappone a quello in cui è necessario predisporre prima tutti i dati necessari e solo alla fine dell'elaborazione si ottiene un risultato (qualunque esso sia, comprese le segnalazioni di errore), tipico dei sistemi a schede perforate.

La telescrivente era (ed è) un terminale dell'elaboratore. Con il tempo, la stampante della telescrivente è stata sostituita da uno schermo che emette un flusso di testo dal basso verso l'alto, così come scorrerebbe la carta a modulo continuo attraverso una stampante. Attualmente, anche la tastiera (e il mouse) tendono a essere rimpiazzati da schermi interattivi, sensibili al tatto.

Quando si utilizza un terminale per elaboratore che emette soltanto informazioni testuali, riproducendo il funzionamento tradizionale della telescrivente, se il sistema lo prevede, si individua quello principale con il nome di ***console***.

In un sistema Unix, i vari componenti hardware di un elaboratore (escluse le interfacce di rete) sono rappresentati da file speciali, noti come ***file di dispositivo*** (contenuti normalmente nella directory '`/dev/`', ovvero *device*). Quando si vuole accedere direttamente a un dispositivo, lo si fa utilizzando il nome del file di dispositivo corrispondente. Esistono due categorie fondamentali di file di dispositivo: ***a carattere***, cioè in grado di gestire i dati in blocchetti di un

solo byte per volta, oppure ***a blocchi***, cioè in grado di gestire i dati solo in blocchi (settori) di una dimensione fissa.

Il dispositivo a caratteri tipico è la console o la vecchia porta seriale, mentre il dispositivo a blocchi tipico è l'unità di memorizzazione di massa ad accesso diretto.

Alcuni file di dispositivo non fanno riferimento a componenti hardware veri e propri. Il più noto di questi è '`/dev/null`' utilizzato come fonte per il «nulla» o come pattumiera senza fondo.

I nomi utilizzati per distinguere i file di dispositivo, sono scelti in base a qualche criterio mnemonico e all'uso più frequente. Tuttavia non è detto che un dispositivo debba chiamarsi in un modo rispetto a un altro.

3.2.1 Tastiera

«

La tastiera è una tavoletta composta da un insieme di tasti, ognuno dei quali genera un impulso particolare. È l'elaboratore che si occupa di interpretare e tradurre gli impulsi della tastiera. Questo sistema permette poi di attribuire ai tasti la funzione che si vuole.

Ciò significa anche che non esiste uno standard generale di quello che una tastiera deve avere. Di solito si hanno a disposizione tasti che permettono di scrivere le lettere dell'alfabeto inglese, i simboli di punteggiatura consueti e i numeri; tutto il resto è opzionale. Tanto più opzionali sono i tasti a cui si attribuiscono solitamente funzioni particolari. Questa considerazione è importante soprattutto per chi non vuole rimanere relegato a una particolare architettura dell'elaboratore.

3.2.2 Schermo

Il terminale più semplice è composto da una tastiera e uno schermo, ma questa non è l'unica possibilità. Infatti, ci possono essere terminali con più schermi, ognuno per un diverso tipo di output.

Nel tempo, l'uso dello schermo si è evoluto, dalla semplice emissione sequenziale di output, come avviene con una stampante, a una sorta di guida di inserimento di dati attraverso modelli-tipo o formulari. Le maschere video sono questi modelli-tipo attraverso cui l'input della tastiera viene guidato da un campo all'altro. Successivamente si è passati alla fase grafica, nella quale si è inserito l'uso di un dispositivo di puntamento, costituito solitamente da un mouse. L'ultima fase è invece quella degli schermi sensibili al tatto, con i quali è possibile fare a meno di tastiera e mouse.

Tutte le fasi di evoluzione dello schermo (e della tastiera) conservano però la possibilità di lavorare secondo le vecchie modalità. Quindi su uno schermo sensibile al tatto è possibile riprodurre una tastiera; un programma grafico può riprodurre il comportamento di una maschera di inserimento testuale; oppure, attraverso un programma di emulazione apposito, è possibile interagire come se si utilizzasse un terminale a caratteri tradizionale.

3.2.3 Stampante

Le stampanti tradizionali sono solo in grado di emettere un flusso di testo, come avveniva con le telescriventi. Più di recente, con l'introduzione delle stampanti ad aghi, si è aggiunta la possibilità di comandare direttamente gli aghi in modo da ottenere una stampa grafica.

Ma quando la stampa diventa grafica, entrano in gioco le caratteristiche particolari della stampante. Per questo, l'ultima fase evolutiva della stampa è stata l'introduzione dei linguaggi di stampa, tra cui il più importante è stato ed è PostScript, come mezzo di definizione della stampa in modo indipendente dalle caratteristiche della stampante stessa. Così, l'output generato e inviato dalle stampanti può essere costruito sempre nello stesso modo, lasciando alle stampanti l'onere di trasformarlo in base alle proprie caratteristiche e capacità.

I sistemi operativi in multiprogrammazione (*multitasking*) predisposti per l'uso di una stampante, di norma gestiscono questa funzione attraverso una coda di stampa (*spool*). Tradizionalmente, i sistemi Unix utilizzano un demone denominato '**`lpd`**' per la gestione della stampa, in grado anche di ricevere richieste di stampa remote e di inviare richieste di stampa a elaboratori remoti.

3.2.4 Dispositivi di memorizzazione

«

I dispositivi di memorizzazione sono fondamentalmente di due tipi: ad accesso sequenziale e ad accesso diretto. Nel primo caso, i dati possono essere memorizzati e riletti solo in modo sequenziale, senza la possibilità di accedere rapidamente a un punto desiderato, come con i nastri magnetici. Nel secondo caso, i dati vengono registrati e riletti accedendovi direttamente, come avviene con i dischi e le memorie solide.

I dispositivi di memorizzazione ad accesso diretto possono essere trattati in due modi diversi a seconda delle circostanze: possono essere visti come dei file enormi, oppure come contenitori di file (attraverso l'organizzazione di un file system). La visione che normalmente si ha delle unità di memorizzazione contenenti file e directory è

un'astrazione gestita automaticamente dal sistema operativo. Questa astrazione si chiama file system.

3.3 File e file system

In prima approssimazione, il ***file*** è un'unità di informazioni che si compone in pratica di una sequenza di codici. I dispositivi di memorizzazione ad accesso diretto, muniti di file system, consentono la gestione di diversi file, mentre quelli ad accesso sequenziale permettono la gestione di un solo file su tutta la loro dimensione.

Quando il file viene visto come una semplice sequenza di codici corrispondenti a testo normale, lo si può immaginare come un testo dattiloscritto: la sequenza di caratteri viene interrotta alla fine di ogni riga da un codice invisibile che fa riprendere il testo all'inizio di una riga successiva. Questo codice di interruzione di riga, spesso identificato con il termine *new-line*, cambia a seconda della piattaforma utilizzata.

Il file system è il sistema che organizza i file all'interno dei dispositivi di memorizzazione ad accesso diretto. Questo significa che tutto ciò che è contenuto in un file system è in forma di file.

Il modo più semplice per immaginare un file system è quello di un elenco di nomi di file abbinati all'indicazione della posizione in cui questi possono essere trovati. Questo sistema elementare può forse essere utile in presenza di dispositivi di memorizzazione particolarmente piccoli dal punto di vista della loro capacità.

Generalmente, si utilizzano elenchi strutturati, per cui da un elenco si viene rimandati a un altro elenco più dettagliato che può contenere l'indicazione di ciò che si cerca o il rinvio a un altro elenco ancora.

Questi elenchi sono chiamati ***directory*** (o cartelle in alcuni sistemi) e sono file con questa funzione speciale.

Per questo motivo, la struttura di un file system assume quasi sempre una forma a stella (o ad albero), nella quale c'è un'origine da cui si diramano tutti i file. Le diramazioni possono svilupparsi in modo più o meno esteso, a seconda delle esigenze.

Data l'esistenza di questo tipo di organizzazione, si utilizza una notazione particolare per indicare un file all'interno di un file system. Precisamente si rappresenta il ***percorso*** necessario a raggiungerlo:

- una barra obliqua rappresenta la directory principale, altrimenti chiamata anche radice, o *root*;
- un nome può rappresentare indifferentemente una directory o un file;
- un file o una directory che discendono da una directory genitrice, si indicano facendo precedere una barra obliqua.

Per esempio, '`/uno/due/tre`' rappresenta il file (o la directory) '`tre`' che discende da '`due`', che discende da '`uno`', che a sua volta discende dall'origine.[1]

Il tipo di file system determina le regole a cui devono sottostare i nomi dei file. Per esempio, ci possono essere situazioni in cui sono consentiti simboli speciali, come il carattere spazio, e altre in cui l'uso di tali simboli non è ammesso. Nello stesso modo, la lunghezza massima dei nomi è sottoposta a un limite.

Oltre a questo, il file system permette di annotare delle informazioni accessorie che servono a qualificare i file, per esempio per poter distinguere tra directory e file contenenti dati normali.

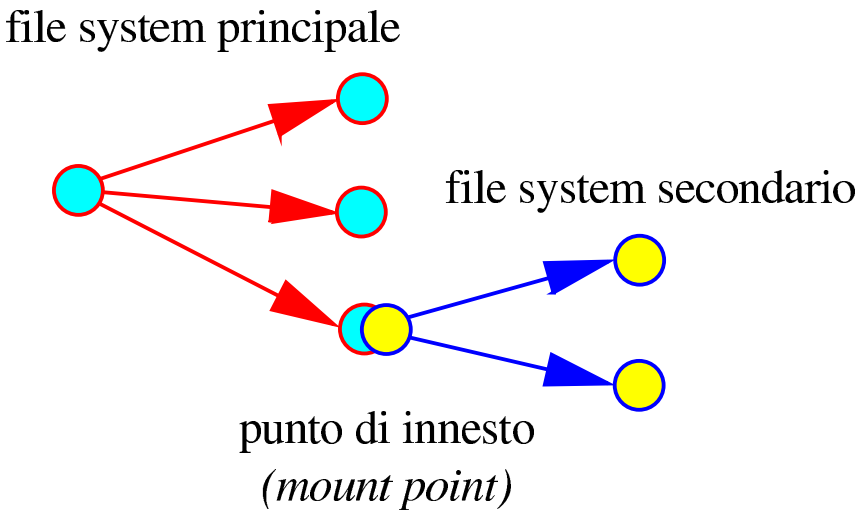
Tradizionalmente si utilizzano due nomi convenzionali per poter fare riferimento alla directory in cui ci si trova e a quella precedente (nel senso di quella che la contiene):

- '`.`' un punto singolo rappresenta la directory in cui ci si trova;
- '`..`' due punti in sequenza rappresentano la directory genitrice, ovvero quella che contiene la directory che si sta osservando.

Nei sistemi operativi Unix non esiste la possibilità di distinguere tra un'unità di memorizzazione e un'altra, come avviene nei sistemi MS-Windows, in cui i dischi o le partizioni sono contrassegnati da una lettera dell'alfabeto ('`A:`', '`B:`', '`C:`'). Nei sistemi Unix, tutti i file system cui si vuole poter accedere devono essere concatenati assieme, in modo da formare un solo file system globale.

Quando un sistema Unix viene avviato, si attiva il file system principale, o *root*, quindi possono essere collegati a questo altri file system a partire da una directory o sottodirectory di quella principale. Dal momento che per accedere ai dati di un file system diverso da quello principale occorre che questo sia collegato, nello stesso modo, per poter rimuovere l'unità di memorizzazione contenente questo file system, occorre interrompere il collegamento. Ciò significa che, nei sistemi Unix, non si può inserire un'unità di memorizzazione esterna, accedervi immediatamente e toglierla quando si vuole: occorre dire al sistema di collegare il file system di tale unità, quindi la si può usare come parte dell'unico file system globale. Al termine si deve interrompere questo collegamento e solo allora si può rimuovere l'unità di memorizzazione.

Figura 3.4. Collegamento di un file system secondario in corrispondenza di un punto di innesto (o *mount point*).



L'operazione con cui si collega un file system secondario nel file system globale viene detta *mount*, per cui si utilizza normalmente il verbo ***montare*** o innestare con questo significato; l'operazione inversa viene detta *unmount* e conseguentemente si utilizza il verbo ***smontare*** o separare. La directory a partire dalla quale si inserisce un altro file system è il *mount point*, che potrebbe essere definito come il ***punto di innesto***.

3.4 Sistema operativo

Il sistema operativo è ciò che regola il funzionamento di tutto l'insieme di queste cose, secondo quelle che sono definite ***politiche di gestione***, creando un'astrazione della macchina reale.

L'astrazione che viene messa in atto dal sistema operativo crea quella che si può definire ***macchina virtuale***, la quale, se il sistema operativo è predisposto per farlo, può disporre di una memoria virtuale maggiore rispetto alla memoria centrale reale e può gestire più processi elaborativi in modo apparentemente simultaneo, anche se nella

realtà il microprocessore è unico.

La gestione simultanea di più processi elaborativi richiede al sistema operativo la capacità di gestire la memoria (virtuale) in modo da isolare le aree concesse a ognuno di loro; in pratica, la memoria usata da un processo non deve interferire con quella di un altro (a parte i casi in cui la condivisione di un'area di memoria avviene volutamente per scambiare delle informazioni). Un sistema operativo che consente l'esecuzione di un solo programma alla volta viene detto ***monoprogrammato*** o ***uniprogrammato***, mentre un sistema in grado di mettere in funzione più programmi è ***multiprogrammato***.

L'esecuzione di più programmi simultaneamente con un microprocessore singolo avviene solo in apparenza, secondo l'astrazione creata dal sistema operativo, perché nella realtà si tratta dell'esecuzione sequenziale di piccole porzioni di ogni programma, a turno. Quando una risorsa viene impiegata secondo dei turni per porzioni di tempo ben individuate, questo fatto viene definito in inglese come *time sharing*, dove le porzioni di tempo elementari sono definite come *time slice*.

È molto importante che un sistema operativo funzioni in multiprogrammazione, perché nell'elaboratore fisico si creano facilmente dei colli di bottiglia che costringono il microprocessore a rimanere in attesa di eventi esterni alla sua responsabilità diretta. Pertanto, con la multiprogrammazione, i processi elaborativi che utilizzano in pratica porzioni fisiche non impegnate possono ancora sfruttare il microprocessore che altrimenti resterebbe bloccato inutilmente.

In un sistema operativo multiprogrammato, i processi elaborativi devono avere una priorità di esecuzione, che consenta al sistema di fare

delle preferenze nell'assegnare loro le porzioni di tempo (*time slice*) del microprocessore. Le priorità di esecuzione dei processi elaborativi servono principalmente a garantire che il sistema nel suo complesso funzioni correttamente, anche quanto le richieste da parte dei programmi raggiungono livelli insostenibili.

Un sistema operativo multiprogrammato può, eventualmente, essere predisposto per la gestione di più utenze simultaneamente. Si parla in questi casi si sistemi operativi ***multiutente***, mentre nel caso contrario i sistemi sono ***monoutente***. Un sistema operativo multiutente deve disporre di meccanismi in grado di distinguere gli utenti e di assegnare loro privilegi differenti.

Tra i componenti che costituiscono un sistema operativo comune, sono molto importanti: il kernel; la shell; i programmi di servizio. Per fare una distinzione grossolana, in un sistema GNU/Linux, il kernel è Linux, mentre i programmi di servizio e di solito anche la shell sono originati dal progetto GNU (*http://www.gnu.org*). Quando si parla di sistemi GNU si intendono qui quei sistemi operativi basati principalmente sul lavoro del progetto GNU (*http://www.gnu.org*), tra cui, in particolare, GNU/Linux e GNU/Hurd. Quando si fa riferimento a sistemi Unix, in generale si intende qualcosa che riguarda anche i sistemi GNU.

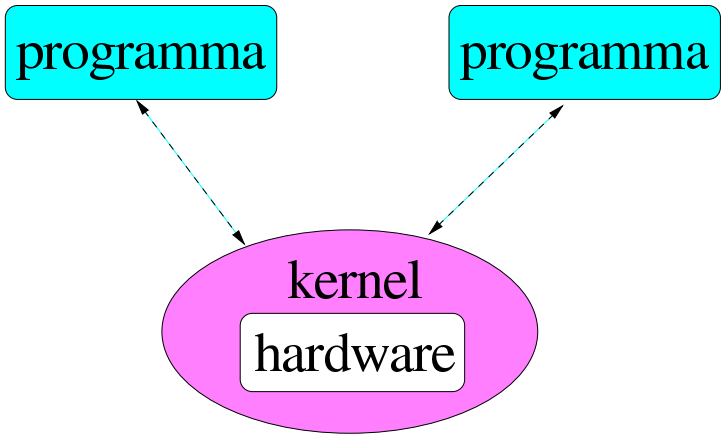
3.5 Kernel

«

Il ***kernel*** (nocciolo) è il nucleo del sistema operativo. Idealmente, è una sorta di astrazione nei confronti delle caratteristiche fisiche della macchina ed è il livello a cui i programmi si rivolgono per qualunque operazione. Ciò significa, per esempio, che i programmi non devono (non dovrebbero) accedere direttamente ai dispositivi fisici, ma pos-

sono utilizzare dispositivi logici definiti dal kernel. Questa è la base su cui si fonda la ***portabilità*** di un sistema operativo su piattaforme fisiche differenti.

Figura 3.5. Il kernel avvolge idealmente l'elaboratore e i suoi dispositivi fisici, ovvero tutto l'hardware, occupandosi di interagire con i programmi che ignorano l'elaboratore fisico.



La portabilità è quindi la possibilità di trasferire dei programmi su piattaforme differenti, ciò attuato normalmente in presenza di kernel che forniscono funzionalità compatibili.

Naturalmente esistono sistemi operativi che non forniscono kernel tanto sofisticati e lasciano ai programmi l'onere di accedere direttamente alle unità fisiche dell'elaboratore. Si tratta però di sistemi inferiori, anche se le loro caratteristiche derivano da necessità oggettive, a causa delle limitazioni architetturali dell'hardware per cui sono sviluppati.

In un sistema GNU/Linux, il kernel è costituito da un file principale, il cui nome potrebbe essere '`vmlinuz`' (oppure '`zImage`', '`bzImage`' e altri), ma può comprendere anche moduli aggiuntivi, per la gestione di componenti hardware specifici che devono poter essere attivati e disattivati durante il funzionamento del sistema.

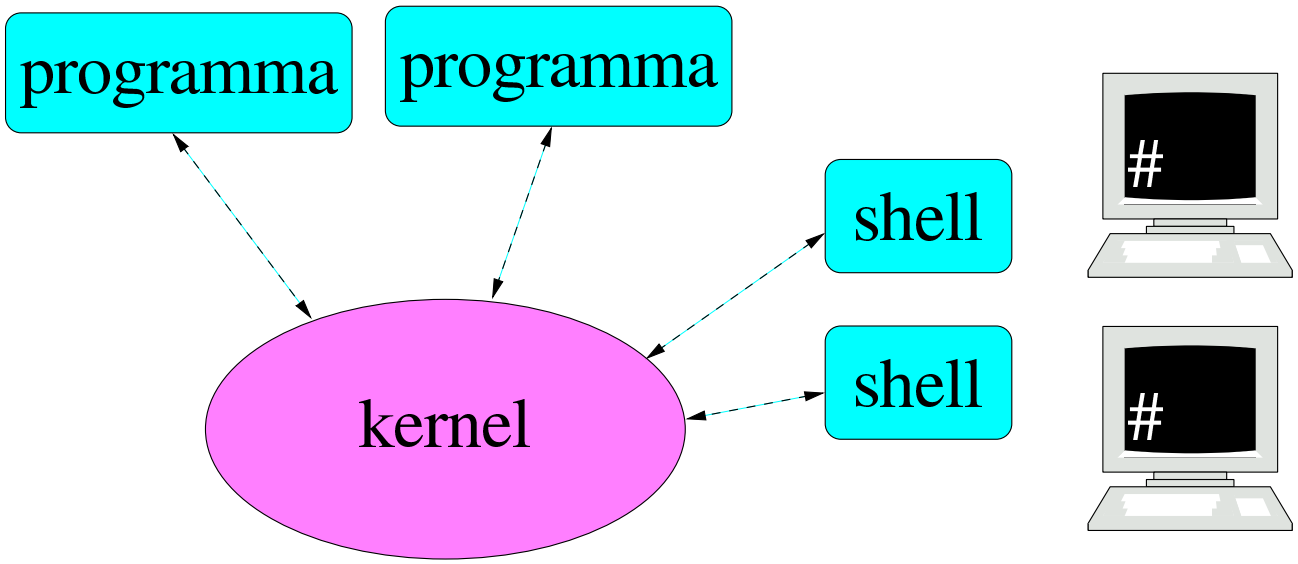
Quando il kernel Linux viene avviato (attraverso il sistema di avvio), esegue dei controlli diagnostici in base ai tipi di dispositivi (componenti hardware) per i quali è stato predisposto, quindi innesta (*mount*) il file system principale (*root*) e infine avvia la procedura di inizializzazione del sistema (Init).

3.6 Shell

«

Il kernel offre i suoi servizi e l'accesso ai dispositivi attraverso chiamate di sistema. Però, mentre i programmi accedono direttamente a questi, perché l'utente possa accedere ai servizi del sistema occorre un programma particolare che si ponga come intermediario tra l'utente (attraverso il terminale) e il kernel. Questo tipo di programma è detto ***shell***. Come suggerisce il nome (conchiglia), si tratta di qualcosa che avvolge il kernel, come se questo fosse una perla.

Figura 3.6. La shell è il programma che consente all'utente di accedere al sistema. I terminali attraverso cui si interagisce con la shell sono comunque parte dell'hardware controllato dal kernel.



Un programma shell può essere qualunque cosa, purché in grado di permettere all'utente di avviare e possibilmente di controllare i pro-

grammi. La forma più semplice e anche la più vecchia, è la riga di comando presentata da un invito, o *prompt*. Questo modo di interagire ha il vantaggio di poter essere utilizzato in qualunque tipo di terminale, compresa la telescrivente. Una shell, nella sua forma più evoluta, può arrivare a un sistema grafico di icone o di oggetti grafici simili, oppure ancora a un sistema di riconoscimento di comandi in forma vocale.

Dal punto di vista operativo, il funzionamento di un sistema Unix dipende molto dalla shell utilizzata, di conseguenza è importante la scelta della shell. La shell tipica dei sistemi Unix è una shell derivata da quella di Bourne, possibilmente aderente allo standard POSIX: la shell POSIX. Nei sistemi GNU la shell tipica è Bash (il programma eseguibile '**`bash`**'), la quale è conforme allo standard POSIX.

Una shell Unix normale svolge i compiti seguenti:

- mostra l'invito, o *prompt*, all'inserimento dei comandi;
- interpreta la riga di comando data dall'utente;
- esegue delle sostituzioni, in base a dei metacaratteri e alle variabili di ambiente;[2]
- mette a disposizione alcuni comandi interni;
- mette in esecuzione i programmi;
- gestisce la ridirezione dell'input e dell'output;
- è in grado di interpretare ed eseguire dei file script.

3.7 Programmi

«

I ***programmi di servizio*** sono un insieme di piccole applicazioni utili per la gestione del sistema operativo. Teoricamente, tutte le funzionalità amministrative per la gestione del sistema operativo potrebbero essere incorporate in una shell; in pratica, di solito questo non si fa. Dal momento che le shell tradizionali incorporano alcuni comandi di uso frequente, spesso si perde la cognizione della differenza che c'è tra le funzionalità fornite dalla shell e i programmi di servizio.

In un sistema Unix, i programmi di servizio di uso comune sono contenuti solitamente all'interno delle directory '`/bin/`' e '`/usr/bin/`'. Quelli riservati all'uso da parte dell'amministratore del sistema, l'utente '**`root`**', sono contenuti normalmente in '`/sbin/`' e '`/usr/sbin/`' dove la lettera «s» iniziale, sta per *superuser*, con un chiaro riferimento all'amministratore.

L'elaboratore non può essere una macchina fine a se stessa, ma deve servire a qualcosa. È importante ricordare che tutto nasce da un bisogno da soddisfare. I ***programmi applicativi*** sono quelli che (finalmente) servono a soddisfare i bisogni e quindi rappresentano l'unica motivazione per l'esistenza degli elaboratori.

3.8 Distinzione tra lettere maiuscole e lettere minuscole

«

Nei sistemi operativi Unix i nomi dei file si distinguono anche in base alla differenza tra le lettere maiuscole e minuscole. Questa caratteristica è molto importante; per esempio, gli ipotetici file denominati '`xy`', '`xY`', '`Xy`' e '`XY`', sono tutti diversi.

In altri sistemi operativi, come per esempio MS-Windows, i nomi di file e directory possono essere composti con lettere minuscole o minuscole, ma questi nomi non si distinguono per la combinazione differente di maiuscole e minuscole nel nome. Pertanto, in quel caso, i nomi '`xy`', '`xY`', '`Xy`' e '`XY`' corrispondono sempre allo stesso file.

Quando in un contesto si fa differenza tra maiuscole e minuscole, capita spesso di vederlo definito come *case sensitive*, mentre per converso, quando non si fa differenza, come *case insensitive*.

3.9 Root

Negli ambienti Unix si fa spesso riferimento al termine ***root*** in vari contesti e con significati differenti. *Root* è la radice, o l'origine, senza altri significati. A seconda del contesto, ne rappresenta l'origine, o il punto iniziale. Per esempio, si può avere:

- una directory *root*, che è la directory principale di un file system, ovvero la directory radice;
- un file system *root*, che è il file system principale di un gruppo che si unisce insieme;
- un utente *root*, che è l'amministratore;
- un dominio *root*, che è il dominio principale;
- una finestra *root* che è quella principale, ovvero la superficie grafica (*desktop*) su cui si appoggiano le altre finestre del sistema grafico X.

Le situazioni in cui si presenta questa definizione possono essere molte di più. L'importante, per ora, è avere chiara l'estensione del significato di questa parola.

3.10 Utenti

«

Generalmente, i sistemi Unix sono multiutente. La multiutenza implica una distinzione tra i vari utenti. Fondamentalmente si distingue tra l'amministratore del sistema, o *superuser*, e gli altri utenti. L'amministratore del sistema è quell'utente che può fare tutto ciò che vuole, soprattutto rischia di produrre gravi danni anche solo per piccole disattenzioni. L'utente comune è quello che utilizza il sistema senza pretendere di organizzarlo e non gli è possibile avviare programmi o accedere a dati che non lo riguardano.

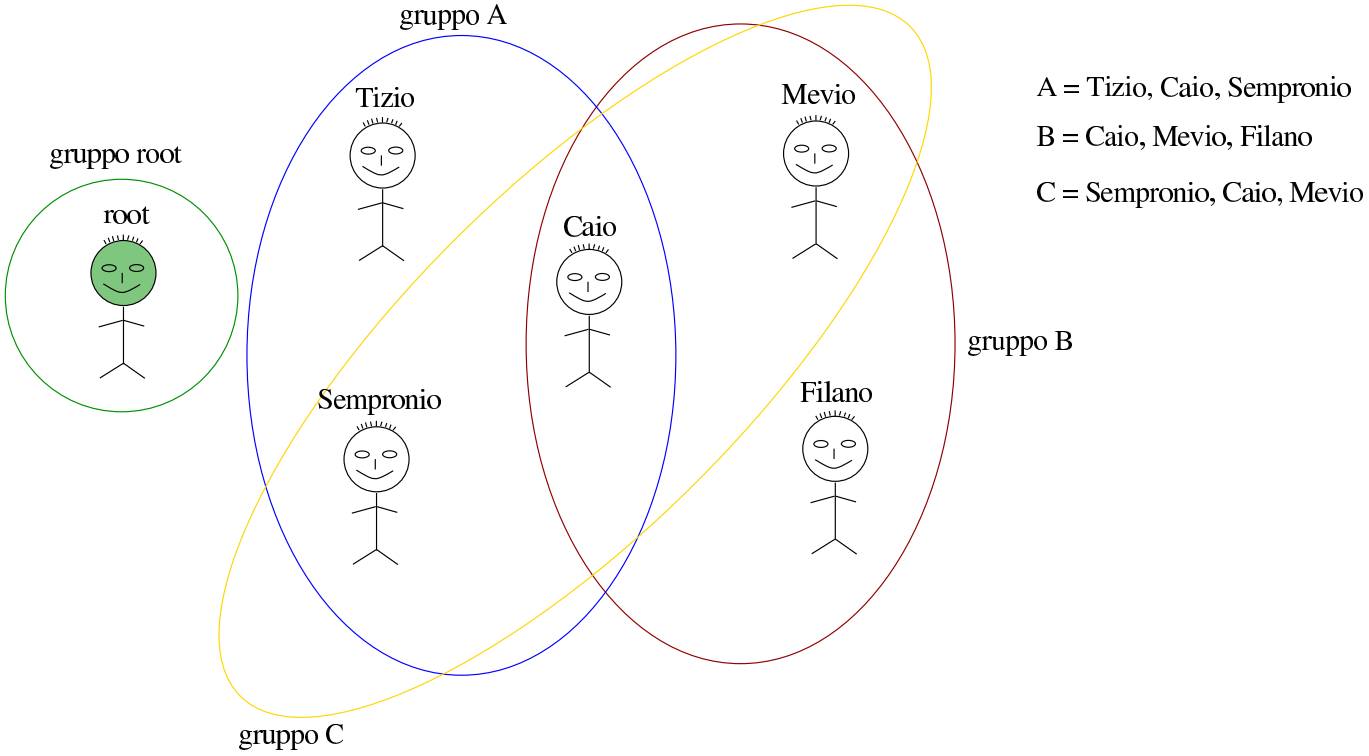
Per poter utilizzare un sistema di questo tipo, occorre essere stati registrati, ovvero, è necessario avere ottenuto un ***account***. Dal punto di vista dell'utente, l'*account* è un nome abbinato a una parola d'ordine che gli permette di essere riconosciuto e quindi di poter accedere. Oltre a questo, l'*account* stabilisce l'appartenenza a un gruppo di utenti.

Il nome dell'amministratore è sempre '**root**', quello degli altri utenti viene deciso di volta in volta.

I sistemi Unix e i programmi che su questi sistemi possono essere utilizzati, non sono predisposti per un utilizzo distratto: gli ordini non vengono discussi. Molti piccoli errori possono essere disastrosi se sono compiuti dall'utente '**root**'. È molto importante evitare il più possibile di utilizzare il sistema in qualità di utente amministratore ('**root**') anche quando si è l'unico utilizzatore del proprio elaboratore.

Tutti gli utenti di un sistema Unix sono associati a uno o più gruppi. Un utente ottiene dei privilegi in quanto tale, oppure in quanto appartenente a un certo gruppo.

Figura 3.7. Utenti e gruppi: un utente può appartenere a gruppi differenti.



3.11 Avvio

Il ***boot*** è il modo con cui un sistema operativo può essere avviato quando l'elaboratore viene acceso. Di solito, il software registrato su ROM degli elaboratori basati sull'uso di unità di memorizzazione di massa ad accesso diretto, è fatto in modo da eseguire le istruzioni contenute nel primo settore, della prima di tali unità previste. Questo settore è noto come MBR (*Master boot record*). Il codice contenuto nel settore di avvio provvede all'esecuzione del kernel (lo avvia).

La parola *boot* è il risultato dell'abbreviazione di *bootstrap*. La parola, usata normalmente al plurale, si riferisce alle linguette degli stivali che servono a calzarli (calzastivali), senza bisogno di aiuto da parte di una seconda persona. Questo fa sì che il termine venga usato in inglese anche per esprimere il concetto di «cavarsela da soli», che spiega il motivo della scelta nel contesto informatico.
In base all'analogia, anche se il suo utilizzo è desueto, è il caso di osservare che il *bootstrap* è il programma o il codice che si prende cura del caricamento del sistema operativo. Pertanto, sarebbe come dire che programmi come SYSLINUX e GRUB, installano il *bootstrap*.

Con un sistema GNU installato in un elaboratore x86, la configurazione e la gestione del sistema di avvio viene fatta principalmente attraverso SYSLINUX e GRUB, i quali permettono di avviare facilmente l'esecuzione di un kernel Linux da un'unità di memorizzazione di massa qualunque, o anche dalla rete con l'ausilio dei protocolli PXE, TFTP e DHCP.

3.12 Inizializzazione e gestione dei processi

«

I sistemi Unix funzionano generalmente in multiprogrammazione, ovvero *multitasking*, pertanto sono in grado di eseguire diversi programmi, o processi elaborativi, contemporaneamente. Per poter realizzare questo, oltre alla gestione interna al kernel, esiste esternamente un gestore dei processi elaborativi: Init, realizzato dall'eseguibile '**`init`**', che viene avviato subito dopo l'attivazione del file system principale, allo scopo di occuparsi di eseguire la procedura di inizializzazione del sistema. In pratica, esegue delle istruzioni

necessarie alla configurazione corretta del sistema particolare che si sta avviando.

3.13 Demoni

Molti servizi sono svolti da programmi che vengono avviati durante la fase di inizializzazione del sistema e quindi compiono silenziosamente la loro attività. Questi programmi sono detti ***demoni*** (*daemon*).

3.14 Gestione dei servizi di rete

Nei sistemi Unix la gestione della rete è un elemento essenziale e normalmente presente. I servizi di rete vengono svolti da dei demoni attivati in fase di inizializzazione del sistema. Nei sistemi GNU, i servizi di rete sono controllati fondamentalmente da tre programmi:

- il supervisore dei servizi di rete, costituito normalmente dal demone '**`inetd`**', che si occupa di attivare di volta in volta, quando necessario, alcuni demoni che poi gestiscono servizi specifici;
- il TCP wrapper, costituito normalmente dal programma eseguibile '**`tcpd`**', che si occupa di controllare e filtrare l'utilizzazione dei servizi offerti dal proprio sistema contro gli accessi indesiderati;
- Portmapper, costituito normalmente dal demone '**`rpc.portmap`**', oppure solo '**`portmap`**', che si occupa del protocollo RPC (*Remote procedure call*).

Un servizio molto importante nelle reti locali consente di condividere porzioni di file system da e verso altri elaboratori connessi. Questo si ottiene normalmente con il protocollo NFS che permette quindi di realizzare dei file system di rete.

3.15 Registrazione e controllo degli accessi

«

I sistemi Unix, oltre che essere in multiprogrammazione sono anche multiutente, cioè possono essere usati da più utenti contemporaneamente. La multiutenza dei sistemi Unix è da considerare nel modo più ampio possibile, nel senso che si può accedere all'utilizzo dell'elaboratore attraverso la console, terminali locali connessi attraverso porte seriali (eventualmente anche attraverso la mediazione di modem), terminali locali o remoti connessi attraverso una rete.

In queste condizioni, il controllo dell'utilizzazione del sistema è essenziale. Pertanto, ogni utente che accede deve essere stato registrato precedentemente, con un nome e una parola d'ordine, o *password*.

La fase in cui un utente viene riconosciuto e quindi gli viene consentito di agire, è detta ***login***.[3] Così, la conclusione dell'attività da parte di un utente è detta ***logout***.

3.16 Strumenti di sviluppo software

«

Tutti i sistemi operativi devono avere un mezzo per produrre del software. In particolare, un sistema operativo Unix deve essere in grado di compilare programmi scritti in linguaggio C/C++. Gli strumenti di sviluppo dei sistemi Unix, composti da un compilatore in linguaggio C/C++ e da altri programmi di contorno, sono indispensabili per poter installare del software distribuito in forma sorgente non compilata.

3.17 Arresto o riavvio del sistema

Qualunque sistema operativo in multiprogrammazione, tanto più se anche multiutente, deve prevedere una procedura di arresto del sistema che si occupi di chiudere tutte le attività in corso prima di consentire lo spegnimento fisico dell'elaboratore.

Normalmente, in un sistema Unix solo l'utente '**root**' può avviare la procedura di arresto del sistema con il comando seguente:

```
# shutdown -h now [Invio]
```

Per richiedere invece il riavvio del sistema:

```
# shutdown -r now [Invio]
```

Con un sistema GNU/Linux installato su un elaboratore con architettura x86, di solito è possibile usare la combinazione di tasti [*Ctrl Alt Canc*] per riavviare il sistema. In questo modo, anche un utente comune ha modo di spegnere il sistema senza creare danni.

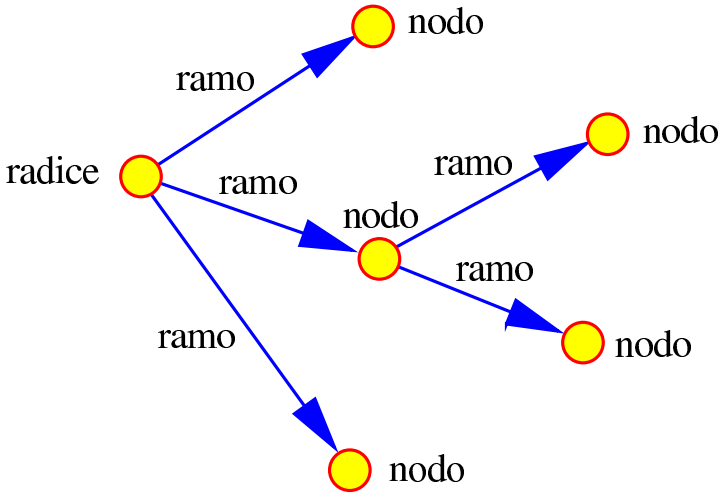
3.18 File e directory in un sistema Unix

Tutto ciò che è contenuto in un file system Unix è in forma di file: anche una directory è un file. Quando si vuole fare riferimento a un file nel senso stretto del termine, ovvero un archivio di dati,[4] se si vuole evitare qualunque ambiguità si utilizza il termine file normale, o *regular file*.

Una directory è un file speciale contenente riferimenti ad altri file. I dati contenuti in un file system sono organizzati in forma gerarchica schematizzabile attraverso un ***albero***, ovvero un tipo particolare di

grafo orientato che parte da una radice e si sviluppa in rami e nodi. La figura 3.8 mostra lo schema di un albero.

Figura 3.8. Albero.



La radice è il nodo principale di questo grafo orientato, i rami rappresentano il collegamento (la discendenza) dei nodi successivi con quello di origine (il genitore). La radice corrisponde a una directory, mentre i nodi successivi possono essere directory, file di dati o file di altro genere.

Per identificare un nodo (file o directory) all'interno di questa gerarchia, si definisce il percorso (*path*). Il percorso è espresso da una sequenza di nomi di nodi che devono essere attraversati, separati da una barra obliqua ('`/`'). Il percorso '`idrogeno/carbonio/ossigeno`' rappresenta un attraversamento dei nodi '`idrogeno`', '`carbonio`' e '`ossigeno`'.

Dal momento che il grafo di un sistema del genere ha un nodo di origine corrispondente alla radice, si distinguono due tipi di percorsi: relativo e assoluto.

- ***Percorso relativo***

  Un percorso è relativo quando parte dalla posizione corrente

(o attuale) del grafo per raggiungere la destinazione desiderata. Nel caso dell'esempio precedente, '`idrogeno/carbonio/ossigeno`' indica di attraversare il nodo '`idrogeno`' inteso come discendente della posizione corrente e quindi gli altri.

- ***Percorso assoluto***

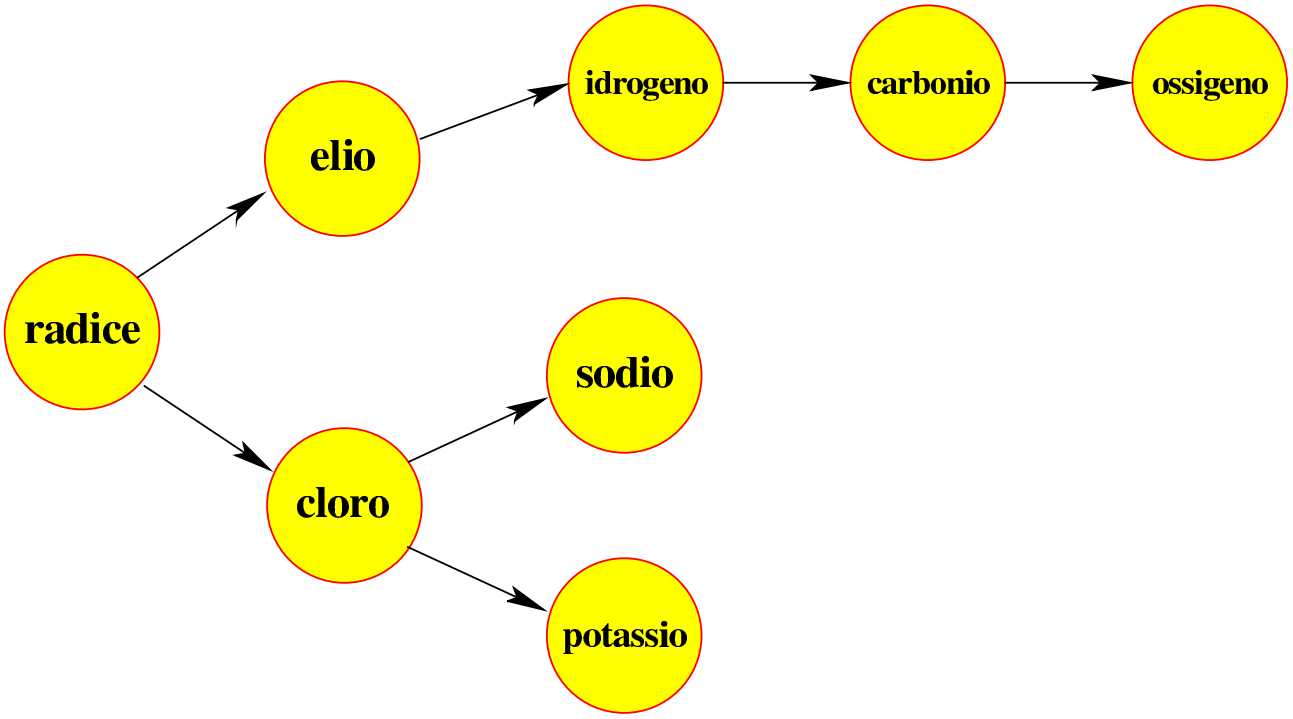
  Un percorso è assoluto quando parte dalla radice.

Il nodo della radice non ha un nome come gli altri: viene rappresentato con una sola barra obliqua ('**`/`**'), di conseguenza, un percorso che inizia con tale simbolo, è un percorso assoluto. Per esempio, '`/cloro/sodio`' indica un percorso assoluto che parte dalla radice per poi attraversare '`cloro`' e quindi raggiungere '`sodio`'.

Un albero è un grafo orientato, nel senso che i rami hanno una direzione (archi orientati), ovvero ogni nodo ha un genitore e può avere dei discendenti e il nodo radice rappresenta l'origine. Quando in un percorso si vuole tornare indietro verso il nodo genitore, non si usa il nome di questo, ma un simbolo speciale rappresentato da due punti in sequenza ('`..`'). Per esempio, '`../potassio`' rappresenta un percorso relativo in cui si raggiunge il nodo finale, '`potassio`', passando prima per il nodo genitore della posizione corrente.

In alcuni casi, per evitare equivoci, può essere utile poter identificare il nodo della posizione corrente. Il simbolo utilizzato è un punto singolo ('`.`'). Per cui, il percorso '`idrogeno/carbonio/ossigeno`' è esattamente uguale a '`./idrogeno/carbonio/ossigeno`'.

Figura 3.9. Uno schema compatibile con gli esempi descritti.



3.19 Collegamenti

«

Un albero è tale purché esista uno e un solo percorso dalla radice a un qualunque altro nodo. Nei file system Unix non è necessariamente così; pertanto sono schematizzabili attraverso grafi orientati, ma non necessariamente degli alberi. Infatti è possibile inserire dei collegamenti aggiuntivi, o *link*, che permettono l'utilizzo di percorsi alternativi. Si distinguono due tipi di questi collegamenti: simbolici e fisici (*hard*).

- ***Collegamenti fisici*, *hard link***

  Un collegamento fisico, o *hard link*, è un collegamento che una volta creato ha lo stesso livello di importanza di quelli originali e non è distinguibile da quelli.

- ***Collegamento simbolico*, *link simbolico*, *symlink***

> Il collegamento simbolico, o *link* simbolico, è un file speciale contenente un riferimento a un altro percorso e quindi a un altro nodo del grafo di directory e file.

quando si preferisce l'uso di collegamenti simbolici, lo si fa per poter distinguere la realtà (o meglio l'origine) dalla finzione: utilizzando un collegamento simbolico si dichiara apertamente che si sta indicando un'alternativa e non si perde di vista il percorso originale.

3.20 Nomi dei file

Non esiste una regola generale precisa che stabilisca quali siano i caratteri che possono essere usati per nominare un file. Esiste solo un modo per cercare di stare fuori dai guai: il simbolo '`/`' non deve essere utilizzato essendo il modo con cui si separano i nomi all'interno di un percorso in un sistema Unix; inoltre conviene limitarsi all'uso dell'alfabeto latino non accentato, dei numeri, del punto e del trattino basso.

Per convenzione, nei sistemi Unix i file che iniziano con un punto sono classificati come nascosti, perché vengono mostrati e utilizzati solo quando sono richiesti espressamente.

I file che iniziano con un punto sono nascosti per una buona ragione: si vuole evitare che utilizzando i metacaratteri si faccia riferimento alla directory stessa (‘`.`’) e alla directory genitrice (‘`..`’). Per tale ragione, si deve fare molta attenzione quando ci si vuole riferire a questi file nascosti. Potenzialmente, il comando ‘**`rm -r .*`**’ non si limita a eliminare i file e le directory che iniziano con un solo punto iniziale, ma tenta di eliminare anche ‘`.`’ e ‘`..`’, cioè tutto il contenuto della directory corrente e di quella genitrice!

3.21 Permessi

«

I file di un file system Unix appartengono simultaneamente a un utente e a un gruppo di utenti. Per questo si parla di utente e gruppo proprietari, oppure semplicemente di proprietario e di gruppo.

L’utente proprietario può modificare i permessi di accesso ai suoi file, limitando questi anche per se stesso. Si distinguono tre tipi di accesso: lettura, scrittura, esecuzione. Il significato del tipo di accesso dipende dal file cui questo si intende applicare.

Per i file normali:

- l’accesso in lettura permette di leggerne il contenuto;
- l’accesso in scrittura permette di modificarne il contenuto;
- l’accesso in esecuzione permette di eseguirlo, se si tratta di un eseguibile binario o di uno script di qualunque tipo.

Per le directory:

- l'accesso in lettura permette di leggerne il contenuto, ovvero di poter conoscere l'elenco dei file in esse contenuti (di qualunque tipo essi siano);
- l'accesso in scrittura permette di modificarne il contenuto, ovvero di creare, eliminare e rinominare dei file;
- l'accesso in «esecuzione» permette di attraversare una directory, pertanto diventa un permesso di attraversamento, ovvero un permesso di accesso.

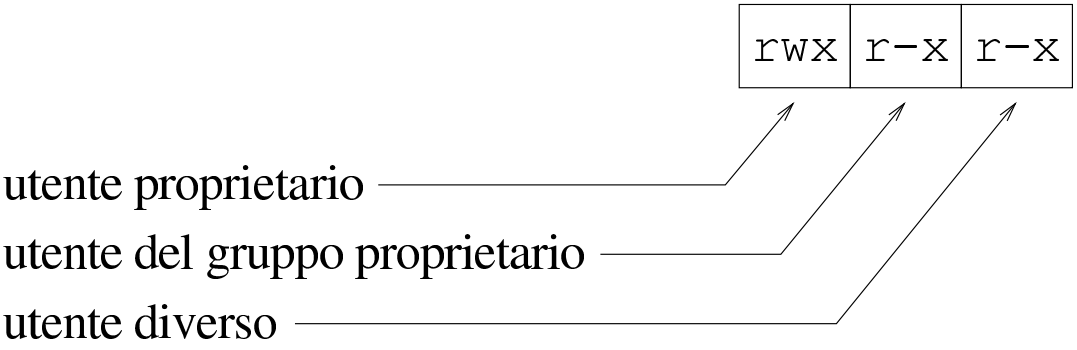
I permessi di un file permettono di attribuire privilegi differenti per gli utenti, a seconda che si tratti del proprietario del file, di utenti appartenenti al gruppo proprietario[5], oppure si tratti di utenti diversi. Così, per ogni file, un utente può ricadere in una di queste tre categorie: proprietario, gruppo o utente diverso.

I permessi si possono esprimere in due forme alternative: attraverso una stringa alfabetica oppure un numero.

I permessi possono essere rappresentati attraverso una stringa di nove caratteri in cui possono apparire le lettere '**r**', '**w**', '**x**', oppure un trattino ('**-**'). La presenza della lettera '**r**' indica un permesso di lettura, la lettera '**w**' indica un permesso di scrittura, la lettera '**x**' indica un permesso di esecuzione o di attraversamento.

I primi tre caratteri della stringa rappresentano i privilegi concessi al proprietario stesso, il gruppetto di tre caratteri successivo rappresenta i privilegi degli utenti appartenenti al gruppo, il gruppetto finale di tre caratteri rappresenta i privilegi concessi agli altri utenti.

Figura 3.10. Associazione tra le classi di utenti e i permessi secondo lo schema tradizionale dei sistemi Unix.



I permessi possono essere rappresentati attraverso tre cifre numeriche, in cui la prima rappresenta i privilegi dell'utente proprietario, la seconda quelli del gruppo e la terza quelli degli altri utenti. Il permesso di lettura corrisponde al numero quattro, il permesso di scrittura corrisponde al numero due, il permesso di esecuzione o attraversamento corrisponde al numero uno. Il numero che rappresenta il permesso attribuito a un tipo di utente, si ottiene sommando i numeri corrispondenti ai privilegi che si vogliono concedere.

Tabella 3.11. Esempi di rappresentazione di permessi.

Stringa	Numero	Descrizione
`rw-r--r--`	644_8	L'utente proprietario può accedervi in lettura e scrittura (4+2), mentre sia gli appartenenti al gruppo, sia gli altri utenti, possono solo accedervi in lettura.
`rwxr-x---`	750_8	L'utente proprietario può accedervi in lettura, scrittura ed esecuzione (4+2+1); gli utenti appartenenti al gruppo possono accedervi in lettura e in esecuzione (4+1); gli altri utenti non possono accedervi in alcun modo.

Stringa	Numero	Descrizione
`rw-------`	600_8	L'utente proprietario può accedervi in lettura e scrittura (4+2), mentre tutti gli altri non possono accedervi affatto.

Tabella 3.12. Tutti i casi di permessi di scrittura, lettura, esecuzione o attraversamento, nel contesto di interesse.

Stringa	Binario	Ottale	Descrizione
`r--`	100_2	4_8	Consente la lettura del contenuto.
`-w-`	010_2	2_8	Consente la modifica del contenuto.
`--x`	001_2	1_8	Consente l'avvio di un programma o l'attraversamento di una directory.
`rw-`	110_2	6_8	Consente la lettura e la modifica del contenuto.
`r-x`	101_2	5_8	Consente la lettura del contenuto e l'avvio di un programma o l'attraversamento di una directory.
`-wx`	011_2	3_8	Consente la modifica del contenuto e l'avvio di un programma o l'attraversamento di una directory (se il programma deve essere interpretato non può funzionare, perché manca il permesso di lettura).
`rwx`	111_2	7_8	Consente la lettura, la modifica del contenuto e l'avvio di un programma o l'attraversamento di una directory.

3.21.1 Permessi speciali: S-bit

«

I permessi dei file sono memorizzati in una sequenza di 9 bit, dove ogni gruppetto di tre rappresenta i permessi per una categoria di utenti (il proprietario, il gruppo, gli altri).

Assieme a questi 9 bit ne esistono altri tre, posti all'inizio, che permettono di indicare altrettante modalità: SUID (*Set user identifier*), SGID (*Set group identifier*) e Sticky (*Save text image*). Si tratta di attributi speciali che riguardano prevalentemente i file eseguibili. Solitamente non vengono usati e per lo più gli utenti comuni ignorano che esistano.

Quanto descritto, serve in questa fase a conoscere il motivo per il quale spesso i permessi espressi in forma numerica (ottale) sono di quattro cifre, con la prima che normalmente è azzerata (l'argomento viene ripreso nella sezione 20.7).

Per esempio, la modalità 0644_8 rappresenta il permesso per l'utente proprietario di accedervi in lettura e scrittura, mentre agli altri utenti si concede di accedervi in sola lettura.

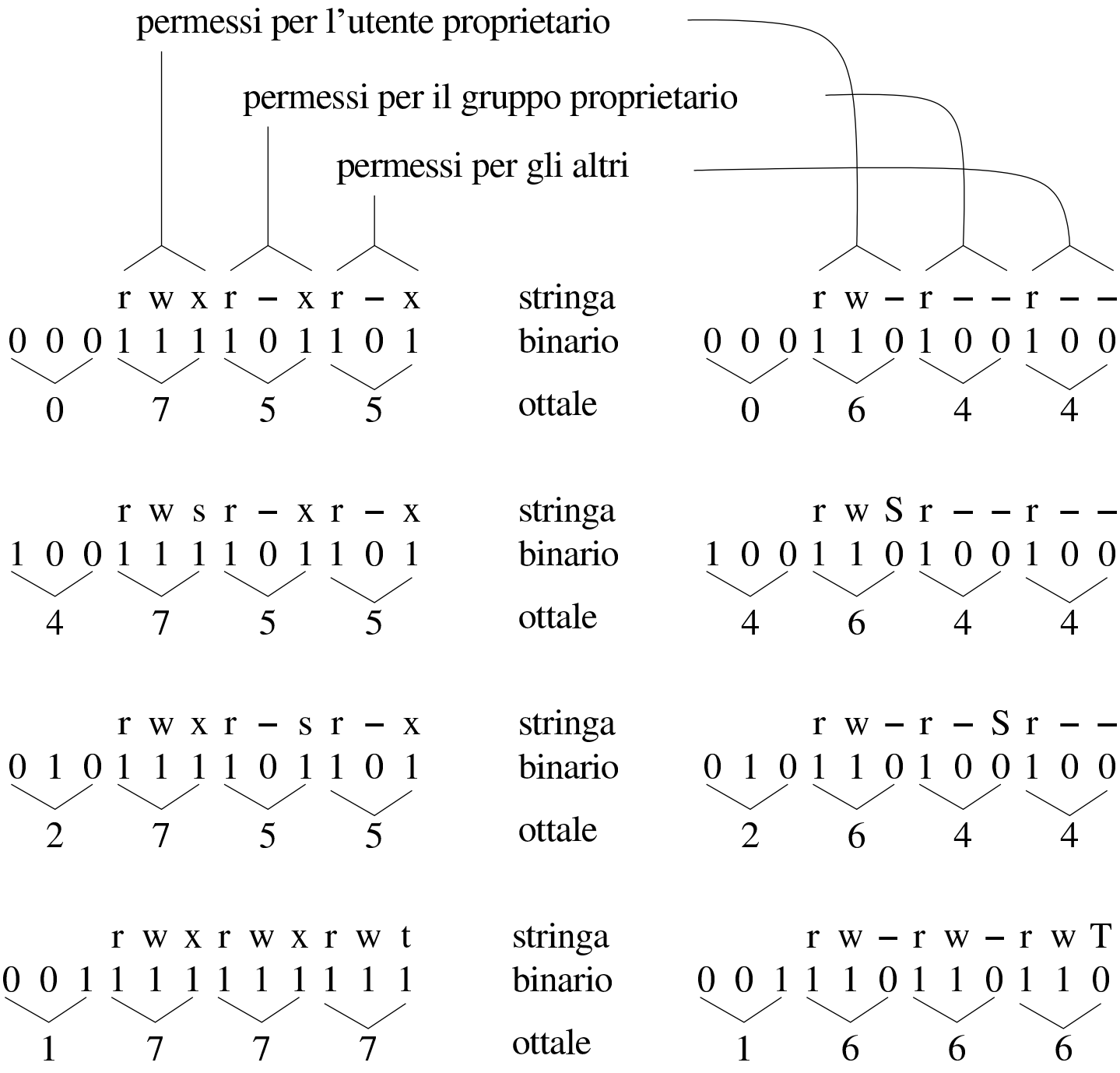
L'indicazione della presenza di questi bit attivati può essere vista anche nelle rappresentazioni in forma di stringa. L'elenco seguente mostra il numero ottale e la sigla corrispondente.

- SUID = 4 = '`--s------`'
- SGID = 2 = '`-----s---`'
- Sticky = 1 = '`--------t`'

Come si può osservare, questa indicazione prende il posto del permesso di esecuzione. Nel caso in cui il permesso di esecuzio-

ne corrispondente non sia attivato, la lettera ('**s**' o '**t**') appare maiuscola.

Figura 3.13. Esempi di rappresentazione di permessi, con i bit aggiuntivi.



3.22 Date

«

Tra gli attributi di un file in un file system Unix ci sono anche tre indicazioni data-orario:

- la data e l'ora di creazione: viene aggiornata in particolare quando si cambiano le informazioni del file che sono contenute nell'inode: permessi e proprietà (l'inode viene descritto più avanti);
- la data e l'ora di modifica: viene aggiornata quando si modifica il contenuto del file;
- la data e l'ora di accesso: viene aggiornata quando si accede al file anche solo in lettura.

3.23 Utenza e accesso

«

Una volta avviato un sistema Unix, prima che sia disponibile l'invito della shell, ovvero il *prompt*, occorre che l'utente sia riconosciuto dal sistema stesso, attraverso la procedura di accesso (*login*). Quello che viene chiesto è l'inserimento del nome dell'utente (così come è stato registrato) e subito dopo la parola d'ordine (*password*) abbinata a quell'utente. Eccezionalmente può trattarsi di un utente senza parola d'ordine, così come avviene per i mini sistemi su unità di memorizzazione esterne, fatti per consentire le operazioni di manutenzione eccezionale.

Si distingue solo tra due tipi di utenti: l'amministratore, il cui nome è '**`root`**', e gli altri utenti comuni. L'utente '**`root`**' non ha alcun limite di azione, gli altri utenti dipendono dai permessi attribuiti ai

file (e alle directory) oltre che dai vincoli posti direttamente da alcuni programmi.

Tecnicamente, nulla vieta di usare un elaboratore personale solo utilizzando i privilegi dell'utente '**root**'; tuttavia, ciò non conviene, perché si perde di vista il significato della gestione dei permessi sui file e sulle directory, ma soprattutto si rendono vani i sistemi di sicurezza predefiniti contro gli errori. Quindi, l'utente '**root**' deve stare sempre molto attento a quello che fa proprio perché può accedere a qualunque funzione o file del sistema, inoltre il sistema non pone alcuna obiezione al suo comportamento. Invece, un utente comune è vincolato dai permessi sui file e dai programmi che possono impedirgli di eseguire certe attività, di conseguenza, è possibile lavorare con meno attenzione.

3.24 Interpretazione dei comandi

Come già è stato indicato, l'interpretazione dei comandi è compito della shell e comporta anche la sostituzione di alcuni simboli che hanno un significato speciale.

Il ***glob*** (o *globbing*) è il metodo attraverso il quale, tramite un modello simbolico, è possibile indicare un gruppo di nomi di file. Di solito, si possono utilizzare i simboli seguenti:

*	l'asterisco rappresenta un gruppo qualsiasi di caratteri, compreso il punto, purché questo punto non si trovi all'inizio del nome;
?	il punto interrogativo rappresenta un carattere qualsiasi, compreso il punto, purché questo punto non si trovi all'inizio del nome;

`[…]`	le parentesi quadre permettono di rappresentare un carattere qualsiasi tra quelli contenuti al loro interno, o un intervallo di caratteri possibili.

Dal momento che è la shell a eseguire la sostituzione dei modelli simbolici, la sintassi tipica di un programma di servizio è la seguente:

programma [***opzioni***] [***file***…]

Di solito, la shell si occupa di eseguire la sostituzione del carattere tilde (‘**~**’). Nei sistemi Unix, ogni utente ha una directory personale, conosciuta comunemente come directory *home*:

`~`	la tilde si espande nel percorso della directory personale dell’utente che sta operando;
`~`***nome***	la tilde che precede il nome di un nominativo utente si espande nel percorso della directory personale di quel utente.

Le ***variabili di ambiente*** sono gestite dalla shell e costituiscono uno dei modi attraverso cui si configura un sistema. I programmi possono leggere alcune variabili di loro interesse e modificare il proprio comportamento in base al loro contenuto.

Una riga di comando può fare riferimento a una variabile di ambiente; la shell provvede a sostituirne l’indicazione con il suo contenuto:

`$`***nome***	il dollaro davanti a un nome serve a richiedere la sostituzione con il contenuto della variabile corrispondente a quel nome.

3.25 Ridirezione e condotti

I programmi, quando vengono eseguiti, hanno a disposizione alcuni canali standard per il flusso dei dati (input/output). Questi sono: standard input, standard output e standard error.

- ***Standard input***

  Lo standard input viene utilizzato come fonte standard per i dati in ingresso (input) nel programma.

- ***Standard output***

  Lo standard output viene utilizzato come destinazione standard per i dati in uscita (output) dal programma.

- ***Standard error***

  Lo standard error, viene utilizzato come destinazione standard per i dati in uscita dal programma derivati da situazioni anomale.

Lo standard input è rappresentato di norma dai dati provenienti dalla tastiera del terminale. Lo standard output e lo standard error sono emessi normalmente attraverso lo schermo del terminale.

Per mezzo della shell si possono eseguire delle ridirezioni di questi flussi di dati, per esempio facendo in modo che lo standard output di un programma sia inserito come standard input di un altro, creando così un condotto (*pipeline*).

Si ridirige lo standard input utilizzando il simbolo minore ('**<**') seguito dalla fonte alternativa di dati. Il programma a sinistra del simbolo '**<**' riceve come standard input il contenuto del file indicato a destra.

programma < ***file_di_dati***

L'esempio seguente visualizza il contenuto del file '`elenco.txt`' dopo averlo riordinato:

`$` **`sort < elenco.txt`** [*Invio*]

Si ridirige lo standard output utilizzando il simbolo maggiore ('**>**') seguito dalla destinazione alternativa dei dati. Il programma a sinistra del simbolo '**>**' emette il suo standard output all'interno del file indicato a destra che viene creato per l'occasione.

programma > ***file_di_dati***

Lo standard output può essere aggiunto a un file preesistente; in tal caso si utilizza il simbolo maggiore per due volte di seguito: '**>>**'. I due esempi seguenti mostrano la differenza nell'uso di '**>**' e di '**>>**'.

- `$` **`ls > elenco.txt`** [*Invio*]

  Genera il file '`elenco.txt`' con il risultato dell'esecuzione di '**`ls`**'.

- `$` **`ls >> elenco.txt`** [*Invio*]

  Aggiunge al file '`elenco.txt`' il risultato dell'esecuzione di '**`ls`**'.

Si ridirige lo standard error utilizzando il simbolo '**2>**' seguito dalla destinazione alternativa dei dati. Il programma a sinistra del simbolo '**2>**' emette il suo standard error all'interno del file indicato a destra che viene creato per l'occasione.

programma `2>` *file_di_dati*

Lo standard error può essere aggiunto a un file preesistente; in tal caso si utilizza il simbolo '**2>>**'. I due esempi seguenti mostrano la differenza nell'uso di '**2>**' e di '**2>>**'.

- `$` **controlla 2> errori.txt** [*Invio*]

  Genera il file '`errori.txt`' con il risultato dell'esecuzione dell'ipotetico programma '**controlla**'.

- `$` **controlla 2>> errori.txt** [*Invio*]

  Aggiunge al file '`errori.txt`' il risultato dell'esecuzione dell'ipotetico programma '**controlla**'.

Si ridirige lo standard output di un programma nello standard input di un altro, utilizzando il simbolo barra verticale ('**|**'). Il programma a sinistra del simbolo '**|**' emette il suo standard output nello standard input di quello che sta a destra.

programma1 | *programma2* [| *programma3*...]

Nella rappresentazione schematica delle sintassi dei programmi, questo simbolo ha normalmente il significato di una scelta alternativa tra opzioni diverse, parole chiave o altri argomenti. In questo caso fa proprio parte della costruzione di un condotto.

Segue la descrizione di alcuni esempi.

- `$` **ls | sort** [*Invio*]

Riordina il risultato del comando '**ls**'.

- `$` **`ls | sort | less`** [*Invio*]

  Riordina il risultato del comando '**ls**' e quindi lo fa scorrere sullo schermo con l'aiuto del programma '**less**'.

3.26 Comandi e programmi di servizio di uso comune

«

Il ***comando*** è ciò che si dà a una shell per eseguire una qualche funzione, per esempio per avviare un programma, eventualmente con certe opzioni e certi argomenti. Il comando viene interpretato dalla shell, la quale decide se deve eseguirlo personalmente, attraverso delle funzioni interne, oppure se deve avviare un programma separato.

Le funzioni interne di una shell, alle quali ci si riferisce come se si trattasse di programmi, sono note come ***comandi interni***, dal momento che non si tratta di programmi indipendenti; tuttavia, spesso, i programmi di servizio di uso comune, quando non sono troppo sofisticati e usano un sistema di input/output elementare, vengono chiamati anch'essi «comandi».

La sintassi che descrive la composizione di un comando segue delle regole molto semplici.

- Le ***metavariabili***, scritte in questo modo, descrivono l'informazione che deve essere inserita al loro posto.
- Le altre parole rappresentano dei termini chiave che, se usati, devono essere indicati così come appaiono nello schema sintattico.

- Quello che appare racchiuso tra parentesi quadre rappresenta una scelta facoltativa: può essere utilizzato o meno.
- La barra verticale ('**|**') rappresenta la possibilità di scelta tra due possibilità alternative: quello che sta alla sua sinistra e quello che sta alla sua destra. Per esempio, '**uno|due**' rappresenta la possibilità di scegliere una tra le parole '**uno**' e '**due**'.
- Quello che appare racchiuso tra parentesi graffe rappresenta una scelta obbligatoria e serve in particolare per evitare equivoci nell'interpretazione quando si hanno più scelte alternative, separate attraverso il simbolo '**|**'. Seguono alcuni esempi:

  – `{uno | due | tre}`

  rappresenta la scelta obbligatoria di una tra le parole chiave '**uno**', '**due**' e '**tre**';

  – `{-f` ***file*** `| --file=`***file***`}`

  rappresenta la scelta obbligatoria di una tra due opzioni equivalenti.

- I puntini di sospensione rappresentano la possibilità di aggiungere altri elementi dello stesso tipo di quello che li precede. Per esempio, '***file*...' rappresenta la metavariabile «file» che può essere seguita da altri valori dello stesso tipo rappresentato dalla metavariabile stessa.

Naturalmente, può capitare che i simboli utilizzati per rappresentare la sintassi, servano per comporre un comando. I casi più evidenti sono:

- i condotti (*pipeline*) che utilizzano la barra verticale per indicare il flusso di dati tra un programma e il successivo;
- le parentesi graffe usate in alcuni linguaggi di programmazione.

Quando ciò accade, occorre fare attenzione al contesto per poter interpretare correttamente il significato di una sintassi, osservando, se ci sono, gli esempi proposti.

Il programma di servizio tipico ha la sintassi seguente:

```
programma [opzioni] [file...]
```

In questo caso, il nome del programma è proprio '**programma**'.

Normalmente vengono accettate una o più opzioni facoltative, espresse attraverso una lettera dell'alfabeto preceduta da un trattino ('**-a**', '**-b**',...). Queste possono essere usate separatamente oppure, spesso si possono raggruppare con un solo trattino seguito da tutte le lettere delle opzioni che si intendono selezionare. Quindi, i due comandi seguenti dovrebbero essere equivalenti:

programma -a -b [*Invio*]

programma -ab [*Invio*]

I programmi più recenti includono opzioni descrittive formate da un nome preceduto da due trattini. In presenza di questi tipi di opzioni, non si possono fare aggregazioni nel modo appena visto.

A volte si incontrano opzioni che richiedono l'indicazione aggiuntiva di un altro argomento.

La maggior parte dei programmi di servizio esegue delle elaborazioni su file, generando un risultato che viene emesso normalmente

attraverso lo standard output. Spesso, quando non vengono indicati file negli argomenti, l'input per l'elaborazione viene ottenuto dallo standard input.

Alcuni programmi permettono l'utilizzo del trattino ('**-**') in sostituzione dell'indicazione di file in ingresso o in uscita, allo scopo di fare riferimento, rispettivamente, allo standard input e allo standard output.

3.27 Programma o eseguibile

In generale, quando si usa il termine «programma» non si chiarisce quale sia la sua estensione reale. Si può usare questo termine per identificare qualcosa che si compone di un solo file eseguibile, oppure un piccolo insieme composto da più componenti che vengono comandate da un solo sistema frontale.

Spesso, in particolare all'interno di questo documento, quando si vuole fare riferimento a un programma inteso come un insieme di componenti, oppure come qualcosa di astratto per il quale nel contesto non conta il modo in cui viene avviato, lo si indica con un nome che non ha enfatizzazioni particolari e generalmente ha l'iniziale maiuscola. Per esempio, questo è il caso della shell Bash, a cui si è accennato, il cui file eseguibile è in realtà '**`bash`**'.

Per evitare ambiguità, quando si vuole essere certi di fare riferimento a un file eseguibile, si specifica proprio che si tratta di questo, cioè di un «eseguibile», mostrandolo attraverso enfatizzazioni di tipo dattilografico, scrivendo il nome esattamente nel modo in cui ciò va fatto per avviarlo.

3.28 ABC dei comandi Unix

«

Nelle sezioni seguenti vengono descritti in modo sommario alcuni programmi di servizio fondamentali. Gli esempi mostrati fanno riferimento all'uso di una shell POSIX, come Bash che costituisce attualmente lo standard per i sistemi GNU.

È importante ricordare che negli esempi viene mostrato un invito differente a seconda che ci si riferisca a un comando impartito da parte di un utente comune o da parte dell'amministratore: il dollaro ('**$**') rappresenta un'azione di un utente comune, mentre il cancelletto ('**#**') rappresenta un'azione dell'utente '**root**'.

3.28.1 ls: «list»

«

`ls` [*opzioni*] [*file*...]

Elenca i file contenuti in una directory. Segue la descrizione di alcuni esempi.

- `$` **ls** [*Invio*]

  Elenca il contenuto della directory corrente.

- `$` **ls -l *.doc** [*Invio*]

  Elenca tutti i file che terminano con il suffisso '`.doc`' che si trovano nella directory corrente. L'elenco contiene più dettagli sui file essendoci l'opzione '**-l**'.

3.28.2 cd: «change directory»

«

`cd` [*directory*]

Cambia la directory corrente. Segue la descrizione di alcuni esempi.

- `$` **`cd /tmp`** [*Invio*]

  Cambia la directory corrente, facendola diventare '`/tmp/`'.

- `$` **`cd ciao`** [*Invio*]

  Cambia la directory corrente, spostandosi nella directory '`ciao/`' che discende da quella corrente.

- `$` **`cd ~`** [*Invio*]

  Cambia la directory corrente, spostandosi nella directory personale dell'utente.

- `$` **`cd ~daniele`** [*Invio*]

  Cambia la directory corrente, spostandosi nella directory personale dell'utente '**`daniele`**'.

3.28.3 mkdir: «make directory»

«

`mkdir` [*opzioni*] *directory*...

Crea una directory. Segue la descrizione di alcuni esempi.

- `$` **`mkdir cloro`** [*Invio*]

  Crea la directory '`cloro/`', come discendente di quella corrente.

- `$` **`mkdir /sodio/cloro`**[*Invio*]

  Crea la directory ‘`cloro/`’, come discendente di ‘`/sodio/`’.

- `$` **`mkdir ~/cloro`**[*Invio*]

  Crea la directory ‘`cloro/`’, come discendente della directory personale dell’utente attuale.

3.28.4 cp: «copy»

«

`cp` [***opzioni***] ***origine*...** ***destinazione***

Copia uno o più file (incluse le directory) in un’unica destinazione.

Se vengono specificati solo i nomi di due file normali, il primo viene copiato sul secondo, viene cioè generata una copia che ha il nome indicato come destinazione. Se il secondo nome indicato è una directory, il file viene copiato nella directory con lo stesso nome di origine. Se vengono indicati più file, l’ultimo nome **deve** essere una directory, all’interno della quale vengono generate le copie di tutti i file indicati. Di conseguenza, quando si utilizzano i metacaratteri, la destinazione deve essere una directory. In mancanza di altre indicazioni attraverso l’uso di opzioni adeguate, le directory non vengono copiate.

I file elencati nell’origine potrebbero essere in realtà dei collegamenti simbolici. Se non viene specificato diversamente attraverso l’uso delle opzioni, questi vengono copiati così come se fossero file normali; cioè la copia è ottenuta a partire dai file originali e non si ottiene quindi una copia dei collegamenti.

Opzione	Descrizione
`-a`	Equivalente a '**`-dpR`**', utile per l'archiviazione o comunque per la copia di collegamenti simbolici così come sono.
`-d`	Copia i collegamenti simbolici mantenendoli come tali, invece di copiare il file a cui i collegamenti si riferiscono.
`-f`	Sovrascrittura forzata dei file di destinazione.
`-l`	Crea un collegamento fisico invece di copiare i file.
`-p`	Mantiene le proprietà e i permessi originali.
`-r`	Copia file e directory in modo ricorsivo (includendo le sottodirectory), considerando tutto ciò che non è una directory come un file normale.
`-R`	Copia file e directory in modo ricorsivo (includendo le sottodirectory).

Segue la descrizione di alcuni esempi.

- `$` **`cp -r /test/* ~/prova`** [*Invio*]

  Copia il contenuto della directory '`/test/`' in '`~/prova/`' copiando anche eventuali sottodirectory contenute in '`/test/`'.

  Se '`~/prova`' esiste già e non si tratta di una directory, questo file viene sovrascritto, perdendo quindi il suo contenuto originale.

- `$` **`cp -r /test ~/prova`** [*Invio*]

Copia la directory ‘/test/’ in ‘~/prova/’ (attaccando ‘test/’ a ‘~/prova/’) copiando anche eventuali sottodirectory contenute in ‘/test/’.

- $ **cp -dpR /test ~/prova** [*Invio*]

  Copia la directory ‘/test/’ in ‘~/prova/’ (attaccando ‘test/’ a ‘~/prova/’) copiando anche eventuali sottodirectory contenute in ‘/test/’, mantenendo inalterati i permessi e riproducendo i collegamenti simbolici eventuali.

3.28.5 ln: «link»

«

`ln` [*opzioni*] *origine*... *destinazione*

Crea uno o più collegamenti di file (incluse le directory, ma solo se si tratta di collegamenti simbolici) in un’unica destinazione.

Se vengono specificati solo i nomi di due file normali, il secondo diventa il collegamento del primo. Se il secondo nome indicato è una directory, al suo interno vengono creati altrettanti collegamenti quanti sono i file e le directory indicati come origine. I nomi utilizzati sono gli stessi di quelli di origine. Se vengono indicati più file, l’ultimo nome **deve** corrispondere a una directory.

È ammissibile la creazione di collegamenti che fanno riferimento ad altri collegamenti.

Se ne possono creare di due tipi: collegamenti fisici e collegamenti simbolici. Questi ultimi sono da preferire (a meno che ci siano delle ragioni per utilizzare dei collegamenti fisici). Se non viene richiesto diversamente attraverso le opzioni, si generano dei collegamenti fisici invece che i consueti collegamenti simbolici.

Opzione	Descrizione
`-s`	Crea un collegamento simbolico.
`-f`	Sovrascrittura forzata dei file o dei collegamenti già esistenti nella destinazione.

Segue la descrizione di alcuni esempi.

- `$` **`ln -s /test/* ~/prova`** [*Invio*]

  Crea, nella destinazione '`~/prova/`', i collegamenti simbolici corrispondenti a tutti i file e a tutte le directory che si trovano all'interno di '`/test/`'.

- `$` **`ln -s /test ~/prova`** [*Invio*]

  Crea, nella destinazione '`~/prova`', un collegamento simbolico corrispondente al file o alla directory '`/test`'. Se '`~/prova`' è una directory, viene creato il collegamento '`~/prova/test`'; se '`~/prova`' non esiste, viene creato il collegamento '`~/prova`'.

3.28.6 rm: «remove»

«

`rm` [***opzioni***] ***nome***...

Rimuove i file indicati come argomento. In mancanza dell'indicazione delle opzioni necessarie, non vengono rimosse le directory.

Opzione	Descrizione

`-r`
 	
`-R`
	Rimuove il contenuto delle directory in modo ricorsivo.

Segue la descrizione di alcuni esempi.

- `$` **`rm prova`** [*Invio*]

  Elimina il file ‘`prova`’.

- `$` **`rm ./-r`** [*Invio*]

  Elimina il file ‘`-r`’ che inizia il suo nome con un trattino, senza confondersi con l’opzione ‘**`-r`**’ (ricorsione).

- `$` **`rm -r ~/varie`** [*Invio*]

  Elimina la directory ‘`varie/`’ che risiede nella directory personale, insieme a tutte le sue sottodirectory eventuali.

Si faccia attenzione al comando seguente:

`#` **`rm -r .*`** [*Invio*]

Teoricamente, questo comando potrebbe eliminare tutti i file e le directory a partire dalla directory genitrice! Si osservi che se la directory corrente discende immediatamente dalla directory radice, ciò significherebbe cancellare tutta la gerarchia.

Questo è comunque un errore tipico di chi vuole cancellare tutte le directory nascoste (cioè quelle che iniziano con un punto) contenute nella directory corrente. Il disastro potrebbe avvenire perché nei sistemi Unix, ‘**`.*`**’ rappresenta anche la directory corrente (‘**`.`**’) e la

directory precedente o genitrice ('. .'). In alternativa, onde evitare disguidi, conviene piuttosto un comando come quello seguente, con cui si è certi di intervenire solo su nomi che sono lunghi almeno tre caratteri complessivi (punto compreso):

rm -r .??* [*Invio*]

3.28.7 mv: «move»

«

`mv` [***opzioni***] ***origine*...** ***destinazione***

Sposta i file e le directory. Se vengono specificati solo i nomi di due elementi (file o directory), il primo viene spostato e rinominato in modo da ottenere quanto indicato come destinazione. Se si indicano più elementi (file o directory), l'ultimo attributo **deve** essere una directory, all'interno della quale si spostano tutti gli elementi elencati. Nel caso di spostamenti attraverso file system differenti, vengono spostati solo i cosiddetti file normali (quindi: niente collegamenti e niente directory).

Segue la descrizione di alcuni esempi.

- `$` **mv prova prova1** [*Invio*]

  Cambia il nome del file (o della directory) '`prova`' in '`prova1`'.

- `$` **mv * /tmp** [*Invio*]

  sposta, all'interno di '`/tmp/`', tutti i file e le directory che si trovano nella directory corrente.

3.28.8 cat: «concatenate»

«

`cat` [*opzioni*] [*file*...]

Concatena dei file e ne emette il contenuto attraverso lo standard output. Il comando emette di seguito i file indicati come argomento attraverso lo standard output (sullo schermo). Se non viene fornito il nome di alcun file, viene utilizzato lo standard input. Segue la descrizione di alcuni esempi.

- `$` **`cat prova prova1`** [*Invio*]

  Mostra di seguito il contenuto di ‘`prova`’ e ‘`prova1`’.

- `$` **`cat prova prova1 > prova2`** [*Invio*]

  Genera il file ‘`prova2`’ come risultato del concatenamento in sequenza di ‘`prova`’ e ‘`prova1`’.

3.29 Glossario introduttivo

«

3.29.1 Sistema operativo in generale

«

- ***boot*, *bootstrap***

  L’avvio, ovvero il caricamento del sistema operativo, viene individuato in inglese con il termine *boot*, mentre la porzione di codice che si occupa effettivamente di attuare l’avvio è il *bootstrap* (calzastivale). Pertanto, in senso figurato, il caricamento del sistema operativo viene paragonato all’atto di calzare uno stivale e rimane il termine *boot*, apparentemente privo di significato in questo contesto (dal momento che significa soltanto «stiva-

le»), per rappresentare l'avvio stesso (e non ciò che deve essere avviato).

- ***monoprogrammazione***, ***multiprogrammazione***

  Un sistema operativo è monoprogrammato quando consente la gestione di un solo processo elaborativo per volta; ovvero, quando la memoria centrale può contenere il codice di un solo programma per volta. Un sistema è invece multiprogrammato quando è in grado di assegnare a più processi elaborativi porzioni di memoria separate, eseguendoli in modo apparentemente simultaneo, suddividendo il tempo di utilizzo della CPU.

- ***time sharing***, ***time slice***

  Quando una risorsa viene condivisa da più processi elaborativi attraverso la suddivisione del tempo di utilizzo in «fettine», si definisce che il suo utilizzo avviene in modo *time sharing*, dove le porzioni di tempo assegnate sono note come *time slice*.

3.29.2 Utenza

«

- ***Account***

  Il termine *account* rappresenta letteralmente un conto, come quello che si può avere in banca. All'interno di un sistema operativo Unix, si ha un *account* quando si è stati registrati (e di conseguenza è stato ottenuto un UID) ed è possibile accedere attraverso la procedura di accesso.

- ***GID***

  Group identifier, *Group ID* o numero identificativo del gruppo di utenti.

- ***Login*, *logout*, *procedura di accesso***

  Con procedura di accesso si vuole fare riferimento al procedimento attraverso il quale un utente accede e può interagire con il sistema, dopo una fase di identificazione, che di solito consiste nell'indicazione di un nominativo-utente e di una parola d'ordine. In particolare *login* identifica l'ingresso dell'utente, mentre *logout* identifica la conclusione dell'attività dello stesso.

- ***UID***

  User identifier, *User ID* o numero identificativo dell'utente.

- ***Password*, *passphrase*, *parola d'ordine***

  Si riferisce a una parola o a una frase utilizzata come mezzo di verifica dell'identificazione per poter accedere a un servizio di qualunque genere.

3.29.3 File e file system

«

- ***Second-extended*, *Ext2*, *Ext3*, *Ext4***

  Il file system nativo del sistema GNU/Linux è il tipo Second-extended, il quale prevede delle varianti: Ext2, Ext3 e Ext4. Le varianti Ext3 e Ext4 contengono delle estensioni che consentono di ridurre la possibilità di perdite di dati, mantenendo la compatibilità con Ext2; ma nel caso di Ext4 si può contare solo su una compatibilità parziale.

- ***FAT***

  File allocation table. La FAT è una parte componente di un tipo di file system usato nei sistemi MS-Windows fino agli anni 1990. È così particolare che tale tipo di file system viene chiamato con questa stessa sigla: FAT.

- ***Glob*, *globbing*, *metacaratteri***

  Quando si vuole identificare un gruppo di file (e directory) attraverso una sola definizione si utilizza il meccanismo del *glob* che comporta l'uso di metacaratteri: si tratta di solito dell'asterisco, del punto interrogativo e delle parentesi quadre.

- ***Mount*, *unmount*,**

  Nei sistemi operativi Unix, quando si vuole accedere ai dati memorizzati su disco, non si può fare riferimento a un file appartenente a una certa unità come avviene nei sistemi MS-Windows. Si deve sempre fare riferimento al file system globale. Per fare questo, tutti i dischi a cui si vuole accedere devono essere uniti tramite un procedimento simile all'innesto di rami. Il termine *mount* indica un collegamento, o l'innesto, del contenuto di un disco nel file system globale; il termine *unmount* indica il distacco di un disco dalla struttura globale.

- ***Newline*, *interruzione di riga***

  Con questo termine si vorrebbe fare riferimento al codice necessario per indicare la fine di una riga di testo e l'inizio di quella successiva. Utilizzando questo nome si dovrebbe evitare di fare riferimento direttamente al codice effettivo in modo che il concetto possa essere adatto a diversi sistemi.

  I sistemi Unix più comuni utilizzano il codice <*LF*>. Nei sistemi MS-Windows si utilizza invece la coppia <*CR*><*LF*>. Per convertire un file di testo, è possibile utilizzare un filtro che trasformi il carattere <*LF*> in <*CR*><*LF*>, o viceversa. Spesso, programmi che svolgono questi compiti hanno nomi del tipo '**`unix2dos`**' e '**`dos2unix`**'.

Normalmente, negli ambienti Unix si confonde tranquillamente il termine *new-line* con il codice <*LF*>. Ciò costituisce un problema, perché ci sono situazioni in cui è importante chiarire che si tratta del codice <*LF*> in modo indipendente dalla piattaforma a cui si applica il concetto. Pertanto, quando si incontra il termine *new-line*, è indispensabile fare attenzione al senso del testo, usando un po' di buon senso.

- ***Record***

  Il record è in generale una registrazione di qualunque tipo. In informatica, il record corrisponde di solito a una riga di un file di dati. Un record è normalmente suddiviso in campi, o *field*, per cui si può fare un'analogia con un archivio a schede: l'archivio è il file, le schede sono i record e i campi sono i vari elementi indicati nelle schede.

- ***Regular file***

  Nei sistemi operativi della famiglia Unix, quando si parla di file, si intendono anche le directory oltre che altri oggetti con funzioni particolari. Per specificare che si parla di un file puro e semplice, comprendendo in questa categoria anche gli eseguibili, si parla di *regular file* o di file normale.

3.29.4 Rete

«

- ***Cliente*, *Client***

  Un «cliente» è generalmente un programma che usufruisce di un servizio offerto da un «servente». Tuttavia, spesso si usa questo termine, in modo informale, anche per identificare un nodo di rete

che, nell'ambito di un certo contesto, dipende da servizi offerti dall'esterno.

- ***Dominio***, ***nome di dominio***, ***nome a dominio***

  Normalmente, con il termine «dominio» si intende fare riferimento al nome che ha un certo nodo di rete in una rete Internet. Questo nome è composto da vari elementi che servono a rappresentare una gerarchia di domini, in modo simile a ciò che si fa nei file system con la struttura delle directory. Un «nome a dominio» può rappresentare una posizione intermedia di questa gerarchia, oppure anche il nome completo di un nodo di rete.

  Nella terminologia giuridica italiana si usa la definizione «nome a dominio».

- ***Host***

  Host è l'oste, ovvero, colui che ospita. Il termine *host* viene usato nell'ambito delle connessioni in rete per definire i nodi di rete, intesi come elaboratori, che svolgono e ospitano qualche tipo di servizio.

- ***Nodo di rete***

  Il nodo di rete è un elaboratore o un altro componente specializzato che è inserito in una rete e ha un indirizzo valido nella stessa (indirizzo riferito al livello 3, secondo il modello ISO-OSI; sezione 32.1).

- ***Protocollo***

  Il protocollo è un linguaggio convenzionale di comunicazione tra programmi (per esempio, un programma cliente comunica con un servente attraverso un protocollo determinato).

- ***Proxy***

  Il termine proxy viene usato in informatica in varie circostanze per identificare un servizio che si comporta in qualche modo come un procuratore, o un procacciatore di qualcosa. Il classico esempio di proxy è il servente che si inserisce tra una rete locale e una rete esterna, allo scopo di eseguire gli accessi verso la rete esterna per conto dei nodi della rete locale, senza che questi possano avere alcun contatto diretto con l'esterno. Di solito, questo tipo di proxy incorpora una memoria cache per ridurre gli accessi ripetuti alle stesse risorse esterne. Tuttavia è bene tenere a mente che questa definizione si usa anche per altri tipi di servizi meno appariscenti.

- ***Servente***, ***Server***

  Un servente è generalmente un programma che fornisce un servizio attraverso la rete. Tuttavia, spesso si usa questo termine, in modo informale, anche per identificare un nodo di rete che ospita servizi importanti.

- ***TCP/IP***

  La sigla TCP/IP rappresenta l'insieme dei protocolli usati per le reti conformi agli standard di Internet.

- ***URL***, ***URI***, ***IRI***

  Uniform resource locator, *Uniform resource identifier*, *Internationalized resource identifier*. È il modo con cui si definisce un indirizzo che identifica precisamente una risorsa di rete, come una pagina HTML, un file in un servizio FTP e altro ancora. Le due definizioni hanno estensioni differenti; in particolare, URI include URL e URN (sezione 54.1), mentre IRI è una nuova ver-

sione di URI che consente l'uso dei caratteri previsti da Unicode, ovvero dalla codifica universale.

3.29.5 Programmi, esecuzione e processi elaborativi

- ***Init*, *procedura di inizializzazione del sistema***

  Init è il programma che viene avviato dal kernel allo scopo di avviare il sistema. Init si avvale di script per avviare dei programmi che rimangono sullo sfondo e per sistemare tutto ciò che c'è da fare prima che il sistema sia a regime. Tutto l'insieme viene indicato come procedura di inizializzazione del sistema, includendo sia l'avvio, sia l'arresto, distinguendo anche diversi livelli di esecuzione.

- ***Job***

  Il termine *job* viene usato spesso nella documentazione Unix in riferimento a compiti di vario tipo, a seconda del contesto.

  - ***Job di shell***

    Le shell POSIX e in particolare Bash, sono in grado di gestire i *job* di shell che rappresentano un insieme di processi generati da un solo comando.

  - ***Job di stampa***

    Si tratta di stampe inserite nella coda di stampa (*spool*).

  - ***Job di scheduling***

    Si tratta di comandi la cui esecuzione è stata pianificata per un certo orario o accodata in attesa di risorse disponibili.

  Le situazioni in cui il termine *job* viene adoperato possono essere anche altre, ma gli esempi indicati bastano per intendere l'ampiezza del significato.

- ***Log*, *registrazioni***

  In informatica, il *log* equivale al ***giornale di bordo*** delle navi. Il *log* è quindi un sistema automatico di registrazione di avvenimenti significativi. I file che contengono queste annotazioni sono detti file di *log* e potrebbero essere identificati anche come i file delle registrazioni. In generale, il *log* è un registro e le annotazioni che vi si fanno sono delle registrazioni.

- ***PID***

  Process identifier, *Process ID* o numero identificativo del processo.

- ***Pipe*, *pipeline*[6], *condotto***

  Si tratta di una tubazione immaginaria attraverso la quale, generalmente, si convoglia l'output di un programma verso l'input di un altro. La connessione di più programmi in questo modo è compito della shell e di solito si utilizza il simbolo '`|`' per indicare questa operazione. A volte, quando il contesto lo consente, il simbolo '`|`' viene anche chiamato *pipe*.

- ***Run level*, *livello di esecuzione***

  Quando si utilizza una procedura di inizializzazione del sistema in stile System V, che è poi quella normale, si distinguono diversi ***livelli di esecuzione***, in modo da poter definire quali parti del sistema devono essere attivate e quali no, a seconda delle esigenze.

  Il livello di esecuzione è un numero non negativo che parte da zero, il cui significato dipende dal modo in cui il sistema è configurato. Di solito il livello zero è riservato per la fase di preparazione allo spegnimento, il livello uno è riservato al funzionamento mo-

noutente e il livello sei è riservato alla fase di preparazione al riavvio del sistema.

- ***Script***

  Uno script è un file di comandi che costituisce in pratica un programma interpretato. Normalmente, l'interprete di uno script è anche una shell.

- ***Shell***

  La shell di un sistema operativo è quel programma che si occupa di interpretare ed eseguire i comandi dati dall'utente, attraverso una riga di comando. Il termine shell, utilizzato per questo scopo, nasce proprio dai sistemi operativi Unix.

- ***Standard error***

  Il file o il dispositivo predefinito per l'emissione dei dati relativi a segnalazioni di errore è lo standard error. Di solito si tratta del video della console o del terminale da cui si opera. Lo standard error, di norma, può essere ridiretto utilizzando il simbolo '`2>`' seguito dal nome del file o del dispositivo da utilizzare.

- ***Standard input***

  Il file o il dispositivo predefinito per l'inserimento dei dati, è lo standard input. Di solito è la tastiera della console o del terminale da cui si opera. Per terminare l'inserimento occorre fornire il carattere di fine file (<*EOT*>, ovvero <^*d*>) che di solito si ottiene con la combinazione [*Ctrl d*].

  Lo standard input, di norma, può essere ridiretto con il simbolo minore ('`<`') seguito dal nome del file o del dispositivo da utilizzare, oppure con la barra verticale ('`|`') quando si vuole utilizzare l'output di un comando come input per il comando successivo.

- ***Standard output***

  Il file o il dispositivo predefinito per l'uscita dei dati, è lo standard output. Di solito è il video della console o del terminale da cui si opera. Lo standard output, di norma, può essere ridiretto utilizzando il simbolo maggiore ('**>**') seguito dal nome del file o del dispositivo da utilizzare, oppure può essere diretto a un comando seguente attraverso la barra verticale ('**|**').

- ***Unix domain socket***, ***socket di dominio Unix***

  Si tratta di un sistema di comunicazione tra le applicazioni basato su un tipo di file speciale: il socket. Alcuni demoni offrono servizi attraverso questo tipo di comunicazione stando in ascolto in attesa di una richiesta di connessione da parte delle applicazioni clienti.

- ***Utility***, ***utilità***, ***programma di utilità***, ***programma di servizio***

  Un'*utility*, ovvero un programma di utilità, o meglio un programma di servizio, è un programma utile e pratico che svolge il suo compito senza tanti fronzoli e senza essere troppo appariscente. Di solito, i programmi di questo tipo sono quelli che fanno parte integrante del sistema operativo.

3.29.6 Varie

«

- ***Case sensitive***, ***case insensitive***

  Con queste due definizioni si intende riferirsi rispettivamente alla «sensibilità» o meno verso la differenza tra le lettere maiuscole e minuscole. Generalmente, i sistemi Unix sono sensibili a questa differenza, nel senso che distinguono i nomi anche in base

alla combinazione di lettere maiuscole e minuscole, ma esistono circostanze in cui questa distinzione non c'è o si vuole ignorare.

- ***Core***

  Negli ambienti Unix, *core* è sinonimo di memoria centrale, o RAM. Questa parola deriva dal fatto che gli elaboratori usati inizialmente con il sistema UNIX erano dotati di memoria RAM realizzata attraverso un reticolo di nuclei ferromagnetici: la memoria a nuclei, ovvero *core*. Per questo motivo, spesso, quando un processo termina in modo anormale, il sistema operativo scarica in un file l'immagine che questo processo ha in memoria. Questo file ha il nome '`core`' (ovviamente) e può essere analizzato successivamente attraverso strumenti diagnostici opportuni.

  Per un approfondimento sulla memoria a nuclei si veda: *Core memory*, *http://www.science.uva.nl/museum/CoreMemory.php* , dell'università di Amsterdam; *Yet another PDP-11 Page, How core memory works*, *http://www.psych.usyd.edu.au/pdp-11/core.html* , di John Holden.

- ***Memoria cache***, ***cache***

  La memoria cache è una porzione di memoria centrale utilizzata per conservare o trattenere delle informazioni di uso frequente, in modo da non doverle richiedere ogni volta dalla loro fonte originaria. A titolo di esempio, si usa una memoria cache per alleggerire gli accessi alle unità di memorizzazione di massa, oppure per evitare il ripetersi di richieste di dati uguali attraverso la rete. Il termine «memoria cache» si usa però anche per file provvisori che hanno lo scopo di contenere informazioni che potrebbero essere riutilizzate in tempi brevi, contenuti solitamente a partire dalla directory '`/var/cache/`'.

- ***Memoria tampone*, *buffer***

  La memoria tampone è un tipo di memoria cache, con funzionalità più limitate, indispensabile per il trasferimento dei dati, usata idealmente come una spugna per assorbire i dati da una parte e per scaricarli dall'altra. Per esempio, per poter accedere a un byte singolo contenuto in un'unità di memorizzazione di massa, è necessario utilizzare un'area di memoria centrale corrispondente almeno alla dimensione del blocco di dati minimo per tale unità. Questa area di memoria è da considerare come un «tampone».

- ***Daemon*, *demone***

  Il *daemon*, o demone, è un programma che funziona sullo sfondo (*background*) e, normalmente, compie dei servizi in modo ripetitivo, come in un circolo vizioso. Questo termine è tipico degli ambienti Unix, mentre con altri sistemi operativi si utilizzano definizioni differenti, per esempio servente. Per tradizione, la maggior parte dei programmi demone ha un nome che termina con la lettera «d».

- ***Espressione regolare*, *regular expression*, *regexp***

  L'espressione regolare (si veda la sezione 23.1) è un modello per la ricerca di stringhe. Viene usata da diversi programmi di servizio.

- *Implementation*, ***realizzazione***

  Il verbo inglese *to implement* rappresenta il modo con cui una caratteristica progettuale particolare viene definita in pratica in un sistema determinato. In altre parole, si tratta della soluzione pratica adottata per assolvere a una funzione determinata, soprattutto quando le indicazioni originarie per raggiungere il risulta-

to sono incomplete. In forma ancora più stringata, si tratta della **realizzazione** di qualcosa in un contesto determinato.

Si osservi comunque che in italiano è bene evitare l'uso del verbo «implementare», preferendo piuttosto forme alternative, possibilmente più chiare, per esprimere il concetto che si intende.

- ***Internazionalizzazione*, *i18n***

  L'internazionalizzazione è l'azione con cui si realizza o si modifica un programma, in modo che sia sensibile alla «localizzazione». La sigla deriva dal fatto che tra la lettera «i» e la lettera «n» di *internationalization* ci sono 18 lettere.

- ***Localizzazione*, *l10n***

  La localizzazione è la configurazione attraverso la quale si fa in modo che un programma determinato si adatti alle particolarità linguistico-nazionali locali. La sigla deriva dal fatto che tra la lettera «l» e la lettera «n» di *localization* ci sono 10 lettere.

- ***Terminale*, *TTY***

  Alle origini, il modo normale per interagire con un elaboratore è stato l'uso della telescrivente: *teletype*. Da questo nome deriva la sigla TTY usata normalmente per identificare un terminale generico. La console è il terminale principale che fa parte dell'elaboratore stesso. Quando si parla di terminale si intende attualmente un complesso formato da una tastiera e da un video.

  Quando si parla di un flusso di dati proveniente da un terminale, come nel caso dello standard input, si fa riferimento a quanto

inserito tramite la tastiera. Quando si parla di un flusso di dati verso un terminale, come nel caso dello standard output, si fa riferimento a quanto viene emesso sullo schermo.

3.30 Unità di misura

«

Nell'ambito informatico ha preso piede un'abitudine poco scientifica di utilizzare unità di misura e prefissi moltiplicatori che non sono conformi allo standard internazionale, definito dal SI, ovvero il *Sistema internazionale di unità* (BIPM: *Bureau international des poids et mesures*, http://www.bipm.org/).

È importante che chi si avvicina all'uso dell'elaboratore non faccia confusione: i prefissi moltiplicatori sono quelli che sono riassunti nella tabella 3.20.

I moltiplicatori riferiti alle unità di misura hanno un significato e un valore ben preciso. È un errore l'uso dei termini «kilo», «mega», «giga» e «tera», per rappresentare moltiplicatori pari a 2^{10}, 2^{20}, 2^{30} e 2^{40}, come si fa abitualmente per misurare grandezze riferite a bit o a byte.

Tabella 3.20. Prefissi del *Sistema internazionale di unità* (SI).

Nome	Simbolo	Valore	Note
yotta	Y	10^{24}	
zetta	Z	10^{21}	
exa	E	10^{18}	
peta	P	10^{15}	
tera	T	10^{12}	
giga	G	10^{9}	

Nome	Simbolo	Valore	Note
mega	M	10^6	
kilo	k	10^3	Lettera «k» minuscola.
hecto, etto	h	10^2	
deca	da	10	
		1	Nessun moltiplicatore.
deci	d	10^{-1}	
centi	c	10^{-2}	
milli	m	10^{-3}	
micro	μ	10^{-6}	
nano	n	10^{-9}	
pico	p	10^{-12}	
femto	f	10^{-15}	
atto	a	10^{-18}	
zepto	z	10^{-21}	
yocto	y	10^{-24}	

Lo standard IEC 60027-2 introduce un gruppo nuovo di prefissi da utilizzare in alternativa a quelli del SI, per risolvere il problema dell'ambiguità causata dall'uso improprio dei prefissi del SI in ambito informatico. A questo proposito, una discussione particolareggiata su questo argomento si può trovare nel documento *Standardized units for use in information technology*, di Markus Kuhn, *http://www.cl.cam.ac.uk/~mgk25/information-units.txt* . La tabella 3.21 riporta l'elenco di questi prefissi speciali.

Tabella 3.21. Prefissi IEC 60027-2.

Origine	Nome	Simbolo	Valore	Note
kilobinary	kibi	Ki	2^{10}	Si usa la «K» maiuscola.
megabinary	mebi	Mi	2^{20}	

Origine	Nome	Simbolo	Valore	Note
gigabinary	gibi	Gi	2^{30}	
terabinary	tebi	Ti	2^{40}	
petabinary	pebi	Pi	2^{50}	
exabinary	exbi	Ei	2^{60}	
zettabinary	zebi	Zi	2^{70}	
yottabinary	yobi	Yi	2^{80}	

La sezione 47.3.3 contiene una descrizione più dettagliata a proposito del modo corretto di rappresentare le grandezze e le unità di misura.

3.31 Riferimenti

«

- Eric S. Raymond, *The Unix and Internet Fundamentals HOWTO*, *http://tldp.org/HOWTO/Unix-and-Internet-Fundamentals-HOWTO/*
- Eric S. Raymond, *Jargon File Resources*, *http://www.catb.org/~esr/jargon/*

[1] Il tipo di barra obliqua che si utilizza dipende dal sistema operativo. La barra obliqua normale corrisponde al sistema tradizionale.

[2] La sostituzione dei metacaratteri, è il procedimento attraverso il quale alcuni caratteri speciali vengono tradotti in un elenco di nomi di file e directory corrispondenti. Negli ambienti Unix si utilizza il termine *globbing* per fare riferimento a questo concetto.

[3] La parola *login* va pronunciata separando le due sillabe: «log-in». Lo stesso valga per la parola *logout* che va pronunciata: «log-out».

[4] Il termine «file» non viene tradotto in italiano a causa di una carenza storica dell'informatica meccanizzata, rispetto agli altri paesi occidentali. Il termine che più si può avvicinare al concetto di file, sulla scorta di quanto si fa in francese o in tedesco, è «schedario elettronico». Un file di oggi ha poco a che vedere con uno schedario; tuttavia, la prima forma di elaborazione dati meccanica è avvenuta attraverso schede perforate, rendendo appropriata la quella definizione. Infatti, in francese si chiama «fichier», mentre in tedesco si usa «datei», come variante di «kartei», ovvero «schedario».

[5] Per gruppo proprietario si intende quello che è stato attribuito ai file in questione.

[6] Queste parole si pronunciano: «p-a-i-p» e «p-a-i-p-l-a-i-n».