UNIX: introduzione elementare

Guida introduttiva al sistema operativo Unix per principianti

Marco Liverani

Roma, Maggio 1995

Questo fascicolo è stato prodotto utilizzando il software LaTeX 2.09 in ambiente Linux.

Prima versione Maggio 1995

Indice

Introduzione

Unix non è certo un'invenzione recente. Le prime versioni di questo sistema operativo furono sviluppate negli Stati Uniti intorno ai primi anni '70 presso gli ormai mitici AT&T Bell Laboratories. Sono passati quindi più di venti anni da allora, un secolo se rapportato ai tempi rapidissimi di evoluzione dell'informatica. Sono cambiati l'hardware dei computer, gli scopi per cui questi vengono utilizzati, in questi venti anni è cambiato quasi tutto ciò che riguarda la microinformatica, eppure lui, Unix, è ancora lì, perfettamente integrato con i nuovi sistemi ed anzi, in questi ultimi anni sta vivendo una seconda giovinezza grazie al peso che ormai ha assunto il settore delle *workstation* basate su architettura RISC. Ma forse i motivi di questa longevità (più unica che rara, lo ripeto, nel mondo dell'informatica dove tutto invecchia rapidamente) li dobbiamo ricercare anche nel portentoso sviluppo che le reti (locali e geografiche) hanno avuto in questi ultimi anni; e Unix, in un ambiente di rete, la fa da padrone. La maggior parte degli host su Internet sono macchine Unix, ed in effetti è ben diverso collegarsi alla Rete con un Mac o un PC piuttosto che con una macchina Unix: solo con quest'ultima tutti i servizi di rete sono perfettamente integrati col sistema (davvero la rete diventa un'estenzione naturale della propria workstation). Non per niente lo slogan di Sun, leader mondiale nella produzione di workstation Unix è "The network is the computer".

Proprio perché è stato concepito in modo esemplare, Unix è anche un ottimo "oggetto didattico", un sistema da usare per imparare veramente cosa è un computer, cosa è un sistema operativo; studiare tutto questo su un pc può essere forse più comodo, ma ci darebbe un punto di vista limitato, già molto indirizzato verso un certo tipo di applicazioni, di sicuro non completo. Dopo aver studiato ed operato su una macchina Unix, potremo capire cosa è un pc con maggiore facilità, potremo anche apprezzarne meglio le differenze, la maggiore facilità d'uso ottenuta al costo di una potenza operativa assai inferiore (anche a parità di risorse hardware).

Per capire a fondo il perché di certi aspetti di Unix è forse utile conoscerne gli utenti. Non conosco un altro sistema operativo che sia stato così pesantemente modellato dai suoi utenti, per certi aspetti si potrebbe dire che Unix è parzialmente "fatto in casa", ogni installazione di Unix è ritagliata, modificata ed integrata dai suoi utenti, mediante un insieme di applicazioni piccole o grandi, programmate direttamente, o reperite per i canali del software di pubblico dominio, magari attraverso la rete Internet. Ed infatti gli utenti Unix sono un po' particolari, formano quasi una allegra "setta", si sentono (sono) la crema degli utenti di computer, amano essere autodidatti, adorano integrare il sistema con programmi sviluppati in proprio, scrivono correntemente in C e in Perl, il loro editor preferito è il vi (e presto capirete quali implicazioni ha un'affermazione di questo tipo!). Parlano con un codice linguistico molto particolare, spesso si capiscono solo tra di loro. Soprattutto, un vero utente Unix troverebbe aberrante questa guida: è troppo breve ed incompleta, e per questo non rende giustizia alla grande potenza del sistema.

Questa brevissima guida non è un manuale esaustivo su Unix, ed in effetti non è proprio un manuale su Unix. È una guida introduttiva. Su ogni argomento toccato dai capitoli di questo fascicolo sono stati scritti libri interi ed è a questi testi (qualcuno lo riporto tra le note bibliografiche) che vi consiglio di riferirvi per approfondire la conoscenza di Unix. Utilizzate questo fascicolo solo per avere una prima informazione sintetica su cosa vi si apre davanti accedendo ad una macchina Unix. Le informazioni riportate nella guida non sono errate (almeno questo è ciò che mi auguro!), ma sono profondamente incomplete. I comandi Unix sono caratterizzati da una serie infinita di opzioni che ne modificano l'esecuzione e che li rendono veramente flessibili e potenti; ho riportato in queste pagine solo pochi comandi e di ognuno di questi ho elencato solo le opzioni principali. Sta a voi, alla vostra curiosità, alle vostre necessità, di scoprire di volta in volta nuove varianti di uno stesso comando. In altre parole si può dire che in questa guida non viene spiegato in dettaglio *come* compiere determinate operazioni, ma piuttosto si è preferito puntare l'attenzione su *quali* operazioni è possibile compiere, rimandando alla documentazione del sistema per tutto ciò che riguarda i dettagli tecnici e sintattici. Consiglio di leggere questo fascicolo per intero e comunque nell'ordine in cui sono riportati i capitoli: non essendo un manuale di consultazione, ma una guida introduttiva, non avrebbe nessun senso cominciare direttamente da un capitolo diverso dal primo. Spesso, nella descrizione dei comandi o delle funzionalità di Unix, ho fatto dei paralleli con i comandi o le funzionalità del DOS, ben sapendo che per molti quest'ultimo è uno strumento più familiare. Questi paralleli, che ad un purista potrebbero sembrare (forse non a torto) del tutto inopportuni, sono stati effettuati solo per esigenze di sintesi e chiarezza.

Prima di concludere questa breve introduzione voglio raccomandarvi di sperimentare ogni cosa sulla vostra macchina, senza alcun timore di danneggiare i dati degli altri utenti del sistema. Unix dispone di numerosi sistemi di sicurezza e quindi non vi sarà facile compromettere il buon funzionamento del sistema o danneggiare il lavoro dei vostri colleghi.

* * *

Nel comporre il testo di questo fascicolo sono state impiegate alcune convenzioni tipografiche. I caratteri `typewriter` sono stati utilizzati per indicare i comandi del sistema (ciò che si deve digitare). Negli esempi invece lo stesso carattere è stato utilizzato per rappresentare i messaggi visualizzati dal sistema; ciò che negli esempi si intende digitato dall'utente è stato scritto con il carattere *slanted*. I tasti che devono essere battuti per impostare determinati comandi sono rappresentati con una cornice: $\boxed{\text{Return}}$.

Le numerose figure che compaiono alla fine dei capitoli, rappresentano alcune workstation che operano in ambiente Unix. Le immagini sono state prelevate sulla rete Internet nei siti WWW delle rispettive case costruttrici.

Ho cercato di porre la massima cura ed attenzione nella realizzazione di questo fascicolo. Nondimeno saranno sicuramente presenti numerosi refusi tipografici ed inesattezze sostanziali nel testo. Invito fin d'ora, chiunque riscontrasse errori di qualsiasi tipo, a segnalarmeli con un messaggio di posta elettronica agli indirizzi `mc2991@mclink.it` e `liverani@mat.uniroma1.it`.

M.L.

Roma, Maggio 1995

Capitolo 1

Organizzazione del sistema

In questo primo capitolo voglio cercare di dare una brevissima infarinatura generale sulla struttura teorica di un sistema Unix, volendo esagerare potrei anche dire sulla “filosofia di un sistema Unix”. Una macchina che opera con questo sistema operativo è infatti concepita in modo del tutto diverso da un personal computer; non voglio essere frainteso: oggi esistono numerose versioni del sistema operativo Unix anche per PC, ma è difficile parlare di *personal computing* in presenza di Unix, visto che ben poco spazio lascia a ciò che siamo soliti fare con il nostro pc. Soprattutto è diverso il modo di usare il sistema da parte dell’utente.

Di sicuro Unix è molto più potente del DOS, più versatile e forse anche più divertente, ma di conseguenza è più difficile da apprendere; tuttavia lavorare con questo potente strumento può dare enormi soddisfazioni: una volta imparato ad operare su un sistema Unix sarà ben difficile accettare le anguste ristrettezze a cui ci costringe il DOS.

1.1 Introduzione al sistema operativo Unix

In termini assai riduttivi possiamo dire che un sistema operativo è il software che, caricato in memoria ed eseguito al momento dell’accenzione del calcolatore (*bootstrap*), permette ai programmi applicativi di girare e di sfruttare le risorse hardware della macchina. Il sistema operativo in un certo senso impone anche all’utente il modo di operare sulla macchina, stabilendo ciò che l’utente può fare e ciò che invece non può fare, stabilendo l’ordine in cui devono essere eseguite certe operazioni e così via.

Di questi aspetti ci si accorge poco utilizzando su un personal computer un sistema operativo limitato come il DOS: in questo caso infatti poche cose è consentito fare, ma l’utente le può eseguire tutte, senza alcuna restrizione.

Il DOS d’altra parte è stato progettato per girare su un personal computer, su una macchina che quindi sarebbe stata utilizzata da un unico utente per volta e spesso da un solo utente in assoluto. Il DOS è anche concepito per poter eseguire un unico programma (*task*) per volta[1]: mentre usiamo un editor, non possiamo contemporaneamente utilizzare un programma di comunicazione o eseguire un programma di utilità.

Il sistema operativo Unix nasce intorno ai primi anni ’70 contemporaneamente all’introduzione dei *mini computer*, nuove macchine più compatte ed agili dei grandi *mainframe* prodotti fino ad allora. Le prerogative di Unix che ne determinarono subito

[1] Tralasciamo di entrare nei dettagli dell’ambiente grafico Microsoft Windows, che non è un sistema operativo, ma una semplice interfaccia grafica, e quindi non è un’ambiente multitasking.

un grande successo sono sostanzialmente quelle di essere un sistema operativo compatto e modulare, facilmente adattabile alle risorse hardware ed alle esigenze operative dell'ambiente in cui viene installato. Con l'introduzione di Unix e dei mini computer si diffonde l'uso dei terminali alfanumerici che consentono una comunicazione più facile ed efficiente tra l'utente ed il sistema (fino ad allora si erano usate quasi esclusivamente le schede perforate). Si fa largo anche il concetto di *sistema aperto* e di *elaborazione distribuita*: Unix è predisposto per comunicare facilmente con altri sistemi e ad essere interconnesso con altre macchine per operare su risorse non più centralizzate su un unico grosso sistema, ma distribuite su più macchine di media potenza.

Entrando solo per un attimo in dettagli più tecnici, possiamo accennare al fatto che Unix è basato su un nucleo (il *kernel*) che gestisce la comunicazione *di basso livello* con la macchina, l'esecuzione dei programmi e l'uso e la condivisione o la protezione delle risorse del sistema. Questa è la parte del sistema operativo più strettamente legata all'hardware del computer. Il resto del sistema è costituito da una serie di moduli aggiuntivi finalizzati alla gestione dei vari sottosistemi che completano l'ambiente operativo (il riconoscimento degli utenti, la visualizzazione a video, l'input da tastiera, l'esecuzione dei comandi impostati dall'utente, la posta elettronica, la stampa, ecc.); questi moduli sono spesso "portabili", cioè è possibile adattarli quasi senza nessuna modifica sostanziale per poterli utilizzare su macchine completamente diverse, corredate del loro specifico kernel. Questo fatto rende Unix un sistema operativo aperto e facilmente trasportabile su hardware diversi. È proprio per questa grande lungimiranza dei suoi progettisti iniziali che oggi possiamo disporre di una grande varietà di Unix differenti, utilizzabili su piattaforme diverse. D'altra parte la possibilità da parte di chiunque di sviluppare moduli aggiuntivi per il sistema operativo ha fatto sì che oggi non esista uno Unix, ma ne esistano diverse versioni che hanno finito per essere anche incompatibili tra di loro.

Le nozioni che esporremo in questa breve guida sono talmente generali che non andranno a scontrarsi con queste sottili differenze. Basti sapere che oggi esistono due standard di riferimento per gli sviluppatori di sistemi Unix: BSD, Berkeley System Distribution, e UNIX System V; quest'ultimo (noto anche come SVR4) è probabilmente più diffuso del BSD.

Ci accorgeremo presto che Unix è in un certo senso un sistema operativo assai sobrio ed elegante: tutto ciò che fa del DOS/Windows un sistema coloratissimo e di grande effetto visivo, in ambiente Unix è guardato con sospetto. Qui regna la sintesi, tutto ciò che è inutile o ridondante è bandito. I messaggi del sistema vengono visualizzati solo quando è strettamente necessario: spesso un programma viene eseguito senza che l'utente legga sullo schermo nessuna comunicazione da parte della macchina; se l'esecuzione termina senza che sia stato visualizzato alcun messaggio vorrà dire che l'esecuzione del programma è andata a buon fine, mentre nel caso contrario il sistema visualizzerà un breve messaggio di errore. I comandi della *shell* sono compatti, spesso costituiti di soli due caratteri, ma hanno un'infinità di opzioni perché l'utente si deve sentire libero di modificare a proprio piacimento la modalità operativa del software che utilizza. Questo fa di Unix un sistema operativo estremamente duttile e versatile.

Come vedremo meglio nella prossima sezione, una delle caratteristiche fondamentali di Unix è quella di essere un sistema operativo *multiutente*: più persone possono usare il sistema contemporaneamente o in tempi diversi. Questo fa sì che sia necessario imparare anche un certo "galateo" che deve contraddistinguere gli utenti di un sistema multiutente, in cui le risorse sono condivise tra più persone. Il fatto stesso che lo scambio di messaggi dalla macchina all'utente sia ridotta al minimo indispensabile è un segno del fatto che ogni operazione che possa in qualche modo appesantire il sistema, rallentando il lavoro di un altro utente, è evitata accuratamente, in modo da lasciare libere quante più risorse è possibile all'elaborazione vera e propria dei programmi.

1.2 Multiutenza e multitasking

Un sistema operativo *multitasking* permette di eseguire più programmi contemporaneamente: se ad esempio viene chiesto al sistema di eseguire contemporaneamente due *processi*, A e B, la CPU eseguirà per qualche istante il processo A, poi per qualche istante il processo B, poi tornerà ad eseguire il processo A e così via. È il sistema operativo a controllare che la CPU ripartisca equamente le sue prestazioni tra tutti i processi attivi; è sempre il sistema operativo a far sì che quando un processo va in *crash*, quando si blocca in seguito al verificarsi di un errore, i restanti processi e l'intero sistema non subiscano alcun danneggiamento e possano proseguire senza conseguenze i compiti loro assegnati.

Un sistema *multiutente* può essere utilizzato contemporaneamente da utenti diversi. Sotto Unix ad ogni utente del sistema viene assegnato uno *user name* che lo identifica univocamente: quando si inizia una sessione di lavoro si deve "entrare" nel sistema tramite una procedura di *login* durante la quale dovremo farci riconoscere dal sistema mediante l'introduzione del nostro username pubblico e della nostra *password* segreta.

Dopo essere entrati nel sistema potremo lanciare i nostri processi (le applicazioni) che verranno eseguiti in multitasking insieme ai processi lanciati dagli altri utenti collegati in quel momento sulla nostra stessa macchina.

1.3 Console, terminali e terminali grafici

Può essere utile imparare a distinguere il sistema hardware mediante cui si accede alla macchina Unix su cui operiamo. Infatti, mentre quando lavoriamo con il nostro personal in ambiente DOS, siamo gli unici utenti di quel sistema, ed il computer che stiamo usando è quello che abbiamo fisicamente davanti a noi, la situazione può essere molto diversa nel caso in cui si stia utilizzando un sistema Unix.

La *console* è la coppia tastiera/video collegata direttamente alla macchina. In ambiente DOS la tastiera ed il monitor del nostro PC sono in un certo senso la console del PC stesso. Visto che però ad una stessa macchina Unix possono accedere contemporaneamente più utenti, deve essere possibile il collegamento di più tastiere e video allo stesso computer. Ed infatti di solito ad una macchina Unix sono collegati numerosi *terminali*.

Un terminale è costituito da una tastiera, un video, ed una piccolissima unità di elaborazione locale, che si occupa esclusivamente di gestire la comunicazione tra il terminale stesso e l'elaboratore a cui è collegato. In sostanza il terminale si limita a visualizzare sullo schermo i messaggi del sistema e ad inviare i comandi digitati dall'utente sulla tastiera. Spesso come terminali vengono usati dei normali personal computer, dotati di un opportuno software di *emulazione di terminale*. In questo caso è del tutto ininfluente la potenza del personal utilizzato come terminale, visto che verrà usato esclusivamente per introdurre l'input e visualizzare l'output, mentre ogni elaborazione verrà eseguita dal computer a cui il terminale è collegato.

Esistono dei terminali più evoluti, i cosiddetti *terminali grafici* (in ambiente Unix si parla spesso di *X Terminal*) che permettono di utilizzare un'*interfaccia grafica* (GUI) per eseguire le operazioni di input/output e quindi consentono anche di visualizzare un output di tipo grafico (immagini, disegni, grafici).

1.4 Diritti ed attributi

Gli utenti di un sistema Unix non sono tutti uguali e soprattutto non hanno tutti gli stessi diritti. Questa affermazione un po' perentoria potrebbe suonare male a qualche sincero democratico e far pensare che Unix sia un sistema operativo illiberale. Naturalmente nulla potrebbe essere più falso: Unix garantisce la libertà di azione di tutti gli utenti facendo in modo che nessun altro utente non autorizzato possa in qualche modo violare la nostra privacy o distruggere o manomettere le nostre informazioni. In questo modo viene anche garantita una certa sicurezza dell'intero sistema: nessun utente "normale" potrà manometterlo compromettendone il corretto funzionamento; ma non solo: nessun utente "normale" potrà commettere errori talmente gravi nell'uso del sistema tanto da danneggiare altri utenti o il sistema stesso.

Cosa vuol dire più esattamente tutto questo? Abbiamo visto che ogni utente è individuato univocamente all'interno del sistema mediante uno username. Gli utenti del sistema sono distribuiti in più *gruppi*; ogni utente fa parte di un gruppo. Ad esempio nel sistema del Dipartimento di Matematica, alcuni utenti fanno parte del gruppo denominato "teograf", che raccoglie coloro che si occupano di Teoria dei Grafi, altri fanno parte del gruppo "algegeo", che raccoglie coloro che si occupano di Algebra e Geometria, e così via.

Esiste poi un utente privilegiato, il cui username è *root*, che viene assegnato all'amministratore del sistema, il cosiddetto *system manager*. Questi è una figura assai importante nella gestione di un sistema Unix. È colui che può modificare la configurazione dell'intero sistema e che ha la possibilità di verificare ogni cosa all'interno del sistema stesso. È il caso di dire che è anche l'unico utente che possa combinare dei guai seri su una macchina Unix!

Come si traduce tutto questo in pratica? Innanzi tutto ad ogni utente viene assegnata una propria *home directory* nel filesystem della macchina. Ad utenti differenti corrispondono home directory differenti; questa directory è di proprietà dell'utente a cui è assegnata e di solito ha lo stesso nome dell'utente. Anche ogni altro file o directory nel filesystem ha un proprietario che, mediante un apposito comando, ne stabilisce le modalità d'uso (i diritti di accesso) per gli altri utenti del sistema. È ad esempio possibile fare in modo che un certo file possa modificarlo solo il proprietario, che possano leggerlo tutti gli utenti del gruppo del proprietario e che gli altri utenti non possano né leggerlo, né modificarlo o cancellarlo. Lo stesso è possibile fare con le directory, di cui si possono stabilire i diritti di lettura, scrittura ed accesso, e per i *file binari* contenenti le applicazioni (i programmi), di cui è possibile stabilire i diritti di lettura, scrittura ed esecuzione.

L'utente root non ha questi vincoli, ma può accedere in qualsiasi modo a qualunque file o directory presente nel filesystem.

1.5 Uno sguardo al filesystem

Abbiamo usato più volte il termine *filesystem*, ma cosa significa esattamente? Con questa parola si indica l'insieme dei supporti di memorizzazione, fisici o virtuali, collegati al sistema (in gergo: *montati* sul sistema). Chi ha avuto modo di usare un PC con il DOS sa bene che ogni unità a disco è identificata da una lettera dell'alfabeto: A è il dischetto magnetico, C è il primo disco rigido, D è il secondo disco rigido e così via. In ambiente Unix la situazione cambia radicalmente. Esiste una unità (un disco) principale (*root*, radice, da non confondersi però con il system manager) a cui vengono "agganciate" come sottodirectory tutte le altre unità. L'insieme di tutte le unità di memorizzazione, accorpate in un'unica grande struttura ad albero, costituiscono il filesystem.

Generalmente, su quasi ogni sistema Unix, sono presenti alcune directory che rivestono una certa importanza all'interno del sistema e che hanno quasi sempre lo stesso nome. A titolo di esempio consideriamo la seguente struttura ad albero che rappresenta parte di un ipotetico filesystem (assai ridotto, per la verità):

```
\ ---+-- bin
     |
     +-- dev
     |
     +-- etc
     |
     +-- home --+-- elisa
     |          |
     |          +-- marco
     |          |
     |          +-- root
     +-- lib
     |
     +-- proc
     |
     +-- tmp
     |
     +-- usr --+-- X11
               |
               +-- bin
               |
               +-- include
               |
               +-- lib
               |
               +-- local --+-- bin
               |           |
               |           +-- etc
               |           |
               |           +-- lib
               +-- man
               |
               +-- spool
```

Diamo una rapidissima scorsa al contenuto delle directory elencate:

`/bin` Contiene molti file binari (eseguibili).

`/dev` È una directory molto importante che contiene i *device drivers* delle unità hardware installate sul nostro sistema. Sono alcune di quelle estensioni del kernel di cui parlavamo nelle pagine precedenti, che permettono al sistema di gestire le unità ad esso collegate. Ad esempio il file `/dev/ttyS0` gestisce l'input/output attraverso il primo terminale collegato al sistema, mentre `/dev/console` gestisce la console del sistema. `/dev/null` è l'unità "nulla", che risulta assai utile in alcune situazioni, come vedremo più dettagliatamente in seguito.

`/etc` Contiene una serie di file che non trovano collocazione migliore in altre directory; sono per lo più file di configurazione del sistema.

`/home` Contiene le home directory degli utenti del sistema.

`/lib` Contiene le *shared libraries*, dei file che contengono parte di codice eseguibile e che vengono condivisi da più applicazioni. Questo consente di ridurre la dimensione dei programmi, inserendo nelle librerie parti di codice comuni a più applicazioni.

`/proc` È una directory piuttosto particolare: i file che contiene non sono memorizzati su disco, ma direttamente nella memoria dell'elaboratore; contengono i riferimenti ai vari processi attivi nel sistema e le informazioni utili per potervi accedere.

`/tmp` È la directory temporanea di default. Spesso le applicazioni devono scrivere dei dati su un file temporaneo, che al termine dell'esecuzione verrà cancellato; in questi casi spesso usano la directory `/tmp`, che è sempre presente sui sistemi Unix.

`/usr` Contiene numerose sottodirectory molto importanti per il sistema: nulla di ciò che è contenuto sotto la directory `/usr` è di vitale importanza per il funzionamento della macchina, ma spesso è proprio sotto `/usr` che vengono collocate tutte quelle cose che rendono *utile* il sistema.

`/usr/X11` Contiene tutto ciò che riguarda l'interfaccia grafica X Window.

`/usr/bin` Altri eseguibili (file binari).

`/usr/include` Contiene i file *include* per i programmi in linguaggio C.

`/usr/lib` Altre librerie a collegamento dinamico (shared libraries).

`/usr/local` Contiene files tipici della nostra macchina; tipicamente applicazioni installate successivamente al sistema operativo.

`/usr/man` Contiene le *pagine di manuale* (il sistema di *help on-line* del sistema Unix).

`/usr/spool` Contiene i files in attesa di essere elaborati da altri programmi, ad esempio contiene la coda di stampa ed i messaggi di posta elettronica giacenti e non ancora letti dai rispettivi destinatari.

Capitolo 2

Comandi fondamentali

In questo capitolo diamo un rapido sguardo ai comandi principali della *shell* del sistema operativo Unix. Questi comandi ci consentono di muoverci nel filesystem attraverso le directory e gestire facilmente i nostri files e le funzioni di base del sistema.

La *shell* è il programma che di solito viene eseguito automaticamente dal sistema quando un utente effettua il *login*. È l'interprete dei comandi del sistema operativo, in altre parole è un programma che ci permette di usare i comandi che ci apprestiamo ad illustrare. Dobbiamo osservare, che i comandi sono molti di più di quelli riportati nel seguito, e che la shell, oltre a questi, è in grado di eseguire dei piccoli programmi, chiamati *shell script*, scritti nel linguaggio della shell (un po' come i files *batch* del DOS). La shell svolge più o meno la stessa funzione del programma `COMMAND.COM` del DOS, ma è molto più potente e versatile. Esistono diverse shell sotto Unix: tra queste citiamo la `sh`, la prima in oridine cronologico, la `bash`, Bourne again shell, la `ksh`, Korn shell, la `csh`, C shell, molto amata dai programmatori in quanto ha una sintassi del tutto simile a quella del linguaggio C; le differenze risultano evidenti solo scrivendo degli script, perché per il resto le varie shell sono del tutto simili fra loro.

È importante osservare che in ambiente Unix le lettere maiuscole e minuscole sono completamente diverse: sotto DOS scrivendo `DIR` e `dir` ci si riferisce allo stesso comando, come pure è possibile riferirsi ad un certo file chiamandolo `pippo.txt` o `PIPPO.TXT` indifferentemente. Sotto Unix non vale la stessa cosa: i nomi dei comandi devono essere digitati rispettando le maiuscole e le minuscole e lo stesso vale per i nomi dei file e delle directory. Inoltre non esistono le limitazioni imposte dal DOS sui nomi dei file: questi possono essere anche molto lunghi e non è necessaria l'"estensione"; ad esempio potremmo chiamare il file che contiene il testo di questo capitolo `guida.capitolo_2`.

2.1 Accesso al sistema e chiusura della sessione

Abbiamo già detto che per accedere al sistema si deve effettuare una procedura di "riconoscimento" detta *login*. In sistema ci chiede innanzi tutto di inserire il nostro username e poi la password. Per capire meglio vediamo un esempio:

```
woodstock login:  marco
Password:
Last login: Fri Apr 21 10:27:08 on ttyS2
$ _
```

La password non viene visualizzata a video quando la inseriamo, per impedire a chi ci sta guardando di scoprirla e poter quindi accedere al sistema a nome nostro. Il messaggio "Last login..." ci comunica la data e l'ora dell'ultima volta che siamo entrati nel sistema ed il nome del terminale da cui ci siamo collegati, in questo caso il terminale `ttyS2`.

Il simbolo "`$`" che viene visualizzato davanti al cursore è il *prompt*, ossia l'indicazione che la shell è pronta ad accettare un nostro comando immesso da tastiera; quello riportato in queste pagine è solo un esempio: il prompt potrebbe variare da sistema a sistema o essere configurato diversamente dall'utente.

Quando abbiamo terminato di lavorare, prima di andarcene, dobbiamo scollegarci, effettuare cioè la procedura di *logout* che serve proprio per comunicare al sistema che abbiamo terminato di lavorare e che quindi può rimettersi in attesa del prossimo utente. Per scollegarci possiamo usare il comando `exit`, ma andrà bene anche il comando `bye` o `logout`. Effettuato il logout il sistema presenterà nuovamente il messaggio di login in attesa che qualcun'altro si colleghi dal nostro terminale.

Di norma nei grandi centri di calcolo i terminali vengono lasciati sempre accesi, ma, salvo esplicita controindicazione, è possibile anche spegnerli dopo che si è effettuato il logout. È fondamentale invece non spegnere mai la macchina che ospita il sistema: solo root può effettuare la procedura di *shutdown* al termine della quale sarà possibile spegnere il sistema. Non si deve mai spegnere un sistema Unix senza prima aver completato la procedura di shutdown, pena la perdita irreparabile dei dati del filesystem.

2.2 Muoversi nel filesystem

Appena entrati nel sistema ci troviamo all'interno della nostra home directory. Per verificare il nome della directory possiamo usare il comando `pwd` (*print work directory*).

```
$ pwd
/home/marco
$ _
```

Di sicuro la prima cosa che ci viene in mente è quella di visualizzare il contenuto della nostra directory. Per far questo si deve usare il comando `ls` (*list*), equivalente al comando `dir` del DOS. Ad esempio potremmo ottenere il seguente output:

```
$ ls
lettera    mail     pippo.c     progetto   tesi
libro      pippo    pippo.zip   src
$ _
```

Vengono visualizzati nove nomi, ma non è chiaro se siano nomi di files di dati, di sottodirectory o di programmi. Abbiamo accennato precedentemente alle numerose opzioni che generalmente consentono di modificare opportunamente l'esecuzione di un certo comando o programa. Ad esempio potremmo provare la seguente variante del comando `ls`:

```
$ ls -F
lettera    mail/    pippo.c     progetto@  tesi/
libro/     pippo*   pippo.zip   src/
```

```
$ _
```

Accanto ad alcuni nomi compare un simbolo che non fa parte del nome, ma che serve ad indicare di che tipo di file si tratta: lo *slash* "`/`" indica che è una directory, l'asterisco "`*`" indica che si tratta di un file eseguibile, mentre la chiocciola "`@`" sta ad indicare che quel file (o directory) non è fisicamente presente nella nostra directory, ma è un *link*, un rimando, ad un file (o ad una directory) che si trova da un'altra parte nel filesystem.

Un'altra versione del comando `ls` si ottiene aggiungendo l'opzione "`-l`"; visto che è possibile specificare più opzioni su uno stesso comando, vediamo quale potrebbe essere l'output del comando "`ls -lF`" (che equivale anche ad "`ls -l -F`", visto che in molti casi le opzioni possono essere "raggruppate"):

```
$ ls -lF
-rw-r--r--  1 marco users    937 Apr 23 12:43 lettera
drwxr-xr-x  2 marco users   1024 Apr 10 16:04 libro/
drwx------  2 marco users   1024 Feb 01 09:32 mail/
-rwxr-x---  1 marco users  37513 Mar 10 11:55 pippo*
-rw-r--r--  1 marco users  18722 Mar 10 11:30 pippo.c
-rw-r--r--  1 marco users  23946 Mar 10 12:03 pippo.zip
lrwxrwxr--  1 marco users      8 Apr 04 18:16 progetto -> /home/elisa/proj
drwxrwx---  2 marco users   1024 Mar 10 08:47 src/
drwxr--r--  2 marco users   1024 Feb 12 15:29 tesi/
$ _
```

Illustriamo brevemente il significato delle numerose informazioni presentate dal comando "`ls -l`". Il primo carattere di ogni riga può essere "`d`" per indicare che si tratta di una directory, "`l`" per indicare un link o "`-`" per indicare che si tratta di un normale file. I successivi nove caratteri rappresentano in forma sintetica le proprietà dei files; devono essere letti raggruppandoli a tre a tre. I primi tre caratteri indicano i diritti del proprietario di tale file sul file stesso; i successivi tre caratteri indicano i diritti degli altri utenti facenti parte dello stesso gruppo del proprietario del file, mentre gli ultimi tre caratteri rappresentano i diritti di tutti gli altri utenti del sistema. Una "`x`" sta ad indicare il diritto di esecuzione di tale file (mentre è chiaro cosa significa eseguire un programma, è opportuno chiarire che "eseguire" una directory significa poterci entrare dentro). Il carattere "`r`" sta ad indicare il diritto di leggere tale file (read), mentre "`w`" indica il diritto di poterci scrivere sopra (write), modificandolo o cancellandolo.

Di seguito viene riportato lo username del proprietario del file (es.: `marco`) ed il nome del gruppo di appartenenza (es.: `users`). Viene poi visualizzata la dimensione del file, la data e l'ora in cui è stato creato ed infine il nome.

Nell'esempio il file `pippo.zip` può essere letto e modificato dal proprietario (marco), mentre può essere soltanto letto dagli altri utenti del sistema; è stato creato il 10 Marzo dell'anno in corso alle 12:03. Il file `progetto` è un link alla directory `/home/elisa/proj` e può essere utilizzato in lettura, scrittura ed esecuzione solo dal proprietario e dagli utenti del suo stesso gruppo, ma non dagli altri utenti del sistema che possono solo accedervi in lettura.

Delle numerosissime opzioni del comando `ls` ci limitiamo ad illustrarne solo un'altra, che ci permette di introdurre anche qualche interessante novità; cominciamo con il solito esempio, osservando l'output prodotto dal comando "`ls -a`":

```
$ ls -aF
./          .bashrc    lettera     pippo*      progetto@
../         .exrc      libro/      pippo.c     src/
.Xdefaults  .newsrc    mail/       pippo.zip   tesi/
$ _
```

Da dove spuntano questi strani file preceduti da un punto e perché fino ad ora non li avevamo visti? Per convenzione sotto Unix (ma anche sotto DOS) il nome "`.`" (punto) sta ad indicare la directory corrente, mentre con "`..`" si indica la directory "padre" nell'albero del filesystem[1]. Gli altri file preceduti dal punto sono dei normali file (o directory, ma non nel nostro caso) che però non vengono visualizzati dal comando `ls` proprio perché hanno un nome che comincia con il punto. In questo modo possiamo evitare di visualizzare alcuni file che devono essere presenti nella nostra directory (ad esempio i files di configurazione), ma che non utilizzeremo direttamente quasi mai e dei quali dimenticheremo presto la presenza. Visualizzarli ogni volta renderebbe l'output del comando `ls` eccessivamente ridondante e quindi tali file vengono, in un certo senso, "nascosti".

Per spostarsi attraverso le directory del filesystem si usa il comando `cd` (*change directory*). Ad esempio per "scendere" nella directory `src` si dovrà digitare il comando "`cd src`", mentre per risalire nella directory padre (la nostra home directory) dovremo digitare il comando "`cd ..`". A proposito della home directory è utile osservare che ci si può riferire ad essa mediante l'uso del simbolo "`~`" (tilde), mentre per riferirsi alla home directory dell'utente "elisa" si potrà usare la stringa "`~elisa`" (ad esempio si potrà digitare "`cd ~elisa`" per spostarci nella sua directory: questo modo di impostare il comando è sicuramente più sintetico del canonico "`cd /home/elisa`"). Visto che si presenta frequentemente la necessità di rientrare nella propria home directory, il comando `cd` dato senza nessun parametro assolve efficacemente tale scopo.

2.3 Gestione di files e directory

Abbiamo visto nella sezione precedente che ad ogni file sono associati degli attributi per limitarne l'accesso ad altri utenti del sistema. Per modificare gli *attributi* di un file si deve utilizzare il comando `chmod`, che ha la seguente sintassi:

`chmod` *attributi file*

Il parametro *attributi* può avere varie forme, ma forse la più semplice di tutte è quella numerica, secondo la seguente regoletta. Si associa valore 100 al diritto di eseguire il file per il proprietario, 10 per gli utenti del gruppo e 1 per tutti gli altri; si associa 200 al diritto di accesso in scrittura al file per il proprietario, 20 per gli utenti del gruppo e 2 per tutti gli altri; infine si associa 400 al diritto di accesso in lettura al file per il proprietario, 40 per gli utenti del gruppo e 4 per gli altri. Sommando questi numeri si ottiene l'attributo da assegnare a quel file. Così, ad esempio, il numero `700` equivale a stabilire che l'unico a poter leggere, scrivere ed eseguire il file è il proprietario (100+200+400=700), mentre con `644` si indica il fatto che il file può essere letto e modificato dal proprietario, ma può essere soltanto letto dagli altri utenti;

[1] Riferendoci all'esempio riportato a pag. 5, possiamo dire che `/usr` è la directory padre di `/usr/bin` e che quest'ultima è la directory "figlio" di `/usr`.

in quest'ultimo caso, volendo associare tale attributo al file `pippo.c` daremo il comando

```
$ chmod 644 pippo.c
$ _
```

Per creare una directory useremo il comando **`mkdir`** (*make directory*); ad esempio con il comando

```
$ mkdir esempio
$ _
```

possiamo creare una nuova directory nella directory di lavoro.

Per rimuovere una directory vuota (che non contiene alcun file) si deve usare invece il comando **`rmdir`** (*remove directory*), ad esempio:

```
$ rmdir esempio
$ _
```

che viene prontamente eseguito dal sistema senza chiederci ulteriore conferma.

Per copiare un file da una directory ad un'altra si deve usare il comando **`cp`** (*copy*); ad esempio volendo copiare il file `pippo.c` nella directory `src` dovremo dare il comando:

```
$ cp pippo.c src
$ _
```

Sintassi analoga ha anche il comando **`mv`** (*move*) che serve a spostare i file da una directory all'altra, ad esempio per spostare tutti i files il cui nome termina con "`.c`" dalla directory `src` alla directory `tesi` potremo dare il seguente comando:

```
$ mv src/*.c tesi
$ _
```

Il comando **`mv`** è anche utile per cambiare il nome ad un file, ad esempio se volessimo modificare il nome del file `pippo.c` e chiamarlo `pluto.c`, potremmo usare il comando:

```
$ mv pippo.c pluto.c
$ _
```

Infine per cancellare un file si usa il comando **`rm`** (*remove*); si deve porre particolare attenzione nell'uso di questo comando, perché se non si specifica l'opzione "`-i`" (*interactive*), il sistema eseguirà la cancellazione dei files senza chiedere ulteriori conferme. Ad esempio il seguente comando cancella ogni file contenuto nella directory `tesi`:

```
$ rm tesi/*
$ _
```

Usando l'opzione "`-i`" il sistema ci chiede conferma dell'operzione:

```
$ rm -i pippo.c
rm: remove 'pippo.c'? y
```

```
$ _
```

2.4 Visualizzazione e stampa di files

È molto utile poter visualizzare e stampare il contenuto di un file di testo. A questo scopo sono stati implementati diversi comandi Unix: ne vedremo solo alcuni, quelli che riteniamo veramente indispensabili.

Il primo, ed il più elementare, è il comando `cat`, che serve per concatenare due file, ma che può essere usato anche per visualizzare il contenuto di un file di testo. Ad esempio il seguente comando

```
$ cat pippo.c
#include <stdio.h>
main()
{
  printf("Hello.\n");
  return;
}
$ _
```

visualizza sul terminale il contenuto del file `pippo.c` (un banale programmino in C). Se il file è molto lungo e non può essere visualizzato all'interno di una schermata del terminale, l'output scorrerà sullo schermo fino all'ultima riga del file, senza che possiamo leggerne il contenuto. Per ovviare a questo problema si usa il comando `more`, che, come il `cat`, visualizza il contenuto del file sullo schermo, ma alla fine di ogni schermata rimane in attesa che l'utente batta un tasto, prima di visualizzare la schermata successiva. Il comando `more` si comporta diversamente a seconda del tasto che viene battuto:

- `spazio` l'output scorre in avanti di una schermata;
- `b` l'output scorre indietro (*back*) di una schermata;
- `q` il comando `more` viene interrotto (*quit*);
- `g` va all'inizio del file;
- `G` va alla fine del file;
- `v` richiama l'editor di default (quello specificato nella variabile di ambiente EDITOR) per modificare il file;
- `/` effettua una ricerca sull'intero file per individuare la stringa digitata subito dopo aver battuto il tasto $\boxed{/}$.

`more` può anche essere utilizzato per filtrare verso il terminale l'output proveniente da un altro comando, attraverso la possibilità di effettuare il *piping* dei comandi messa a disposizione dalla shell. Ad esempio se l'output del comando "`ls -l`" dovesse essere troppo lungo potremmo utilizzare il comando "`ls -l | more`" che prima di inviare l'output al terminale lo filtra attraverso il `more`, visualizzandolo una schermata alla volta.

L'uso della stampante è una tipica operazione "*site dependent*", che varia a seconda del tipo di stampante e dei cosiddetti *filtri di stampa* installati sul proprio sistema.

La differenza sostanziale è tra file di testo (scritti con un normale editor, senza fare uso di particolari caratteri di controllo o di formattazione del testo) e file PostScript (che di solito sono riconoscibili perché il nome termina con "`.ps`"). Se la stampante è una stampante generica sarà più semplice stampare i primi, se è una stampante PostScript sarà invece assai semplice stampare il secondo tipo di file. In ogni caso, come per quasi ogni altra operazione sotto Unix, nulla è impossibile e quindi saremo anche in grado di stampare normali file di testo su stampanti Postscript e file Postscript su stampanti generiche.

Il comando fondamentale in questo caso è `lpr`. Come il `more` anche questo può essere usato sia come comando che come filtro per l'output di un altro programma. Così ad esempio i due comandi "`lpr pippo.c`" e "`cat pippo.c | lpr`" sono assolutamente equivalenti e producono entrambi la stampa del file `pippo.c`. In questo caso abbiamo inviato alla *coda di stampa* un file di testo che potrà essere stampato correttamente se il sistema dispone di una stampante generica o se sono stati predisposti degli opportuni filtri automatici per la stampante PostScript; in modo del tutto analogo possiamo stampare il file PostScript "`tesi/cap1.ps`", ad esempio con il comando "`lpr tesi/cap1.ps`", se la nostra stampante è una unità PostScript o se il sistema dispone di un opportuno filtro di conversione per il formato PostScript (ad esempio il `gs`, anche noto come *GhostScript*). Se si dispone di un terminale grafico un modo molto comodo per visualizzare e stampare un file PostScript è quello di utilizzare il programma `ghostview`, che fornisce una semplice interfaccia al GhostScript, guidata mediante l'uso dei menù.

2.5 Le pagine di manuale

La fonte di informazione principale sui comandi Unix e sui programmi installati sul proprio sistema è costituita dalle cosiddette *man pages*, o pagine di manuale, che formano, in unione ad un comodo programma di consultazione, una vera e propria biblioteca di manuali *on-line*, sempre pronti ad essere consultati. L'insieme delle man pages è suddiviso per argomento in nove sezioni:

1. User command, dove sono riportati tutti i comandi utili per l'utente;
2. System calls, vengono descritte tutte le funzioni standard del linguaggio C per effettuare delle chiamate alle funzioni del sistema (utile soprattutto ai programmatori);
3. Subroutines, vengono descritte le *subroutines* e le funzioni del sistema di sviluppo (linguaggio C);
4. Devices, dove sono riportate le descrizioni dettagliate dei devices installati sul sistema;
5. File Formats, vengono riportati i formati dei principali file di configurazione del sistema;
6. Games, descrizione dei giochi installati sul sistema;
7. Miscellaneous, altre descrizioni che non trovano collocazione migliore nelle altre sezioni;

8. System administration, descrizione dei comandi per l'amministratore del sistema (root);

n. New, nuove pagine di manuale ancora non inserite nelle rispettive sezioni;

Ogni pagina di manuale è contenuta in un file diverso ed i file appartenenti ad una stessa sezione sono contenuti nella stessa directory; ad esempio i files con le descrizioni dei comandi della prima sezione possono trovarsi nella directory `/usr/man/man1`.

Se è presente nel sistema la pagina di manuale relativa ad un certo programma, potremo visualizzarla mediante il comando `man` seguito dal nome del programma; ad esempio con "`man mkdir`" si visualizza la pagina di manuale relativa al comando `mkdir`:

```
$ man mkdir
MKDIR(1L)                                          MKDIR(1L)
NAME
      mkdir - make directories
SYNOPSIS
      mkdir [-p] [-m mode] [--parents] [-mode=mode] [--help]
      [--version] dir...
DESCRIPTION
      This manual page  documents  the GNU version of mkdir.
      mkdir  creates a directory  with  each  given name. By
      default, the mode of created directory is  0777  minus
      the bits set in the umask.
   OPTIONS
      -m, --mode mode
             Set the mode of created  directories  to  mode,
             which  is  symbolic  as in chmod  and  uses the
             default mode as the point of departure.
      -p, --parents
             Ensure that each given path  exists.  Make  any
             missing parent directories for  each  argument.
...
$ _
```

La visualizzazione mediante il comando `man` è filtrata automaticamente attraverso il `more` e quindi potremo scorrere facilmente il testo della pagina, anche se questa dovesse risultare molto lunga. Se invece di operare mediante un terminale alfanumerico stessimo lavorando su un X Terminal, potremmo usare, invece di `man`, il comando `xman` che ci permette di "sfogliare" le pagine del manuale mediante una interfaccia a menù utilizzabile con il mouse.

Le man pages hanno tutte un formato piuttosto simile: viene sempre riportato il nome del comando illustrato seguito da una brevissima descrizione del comando stesso; viene poi descritto in modo sintetico la sintassi del comando (*synopsis*) ed infine vengono elencate e descritte dettagliatamente le opzioni che è possibile specificare insieme al comando. Di solito alla fine della pagina è riportato l'elenco dei files di configurazione del programma stesso (*files*), l'elenco di eventuali comandi correlati (*see also*), eventuali problemi noti, riscontrati in situazioni particolari, nell'uso del programma (*bugs*) ed il nome dell'autore del programma (*authors*).

2.6 Posta elettronica e comunicazione fra utenti

Anche se il nostro sistema non è collegato alla rete Internet o ad una rete locale, dispone del servizio di posta elettronica (*e-mail*) per consentire agli utenti di comunicare fra di loro. Vengono anche resi disponibili alcuni strumenti di comunicazione diretta (*chat*) fra utenti.

Per usare questi strumenti bisogna essere in grado di reperire alcune utili informazioni sugli utenti del sistema; ad esempio, potremmo cominciare da... noi stessi! Il comando `whoami` ci comunica il nostro username; visto che è impossibile accedere al sistema senza conoscere il proprio username, il comando risulterà più utile quando, trovando un terminale libero, con un utente attivo, ma non presente fisicamente al suo posto di lavoro, vogliamo informarci su chi sia lo sprovveduto che ha dimenticato di effettuare il logout prima di andarsene. Un altro comando, sicuramente più utile, è `who` che visualizza l'elenco degli utenti collegati in quel momento al sistema:

```
$ who
root     tty1     Apr 25 12:11
elisa    tty5     Apr 25 19:15 ( )
marco    ttyp0    Apr 25 18:05 ( )
marco    ttyp1    Apr 25 18:32 ( )
$ _
```

L'esempio precedente indica che al sistema sono collegati tre utenti, due dei quali (root ed elisa) accedono al sistema da terminali alfanumerici (`tty1` e `tty5`), mentre il terzo (marco) accede al sistema mediante un terminale grafico ed ha aperto due finestre diverse (`ttyp0` e `ttyp1`). Il sistema ci comunica anche l'ora in cui è stato effettuato il login.

Un output simile al precedente, ma più dettagliato lo possiamo ottenere con il comando `finger`:

```
$ finger
Login   Name              Tty  Idle  Login Time    Office        Phone
elisa   Elisa Masiello     5   0:02  Apr 25 19:15 [Stanza 12]   2212
marco   Marco Liverani    p0         Apr 25 18:05 [Stanza 37]   2237
marco   Marco Liverani    p0   1:07  Apr 25 18:32 [Stanza 37]   2237
root    Amministratore     1   0:12  Apr 25 12:11
$ _
```

In particolare ci viene comunicato anche il vero nome di ogni utente collegato, eventualmente il nome del suo ufficio ed il numero di telefono; viene anche visualizzato il tempo di inattività (*idle*) dell'utente, cioè da quanto tempo non sta più intergendo con il sistema mediante il mouse o la tastiera.

Lo stesso comando `finger` può essere utilizzato per reperire informazioni ancora più dettagliate su ogni utente del sistema:

```
$ finger elisa
Login: elisa                        Name: Elisa Masiello
Directory: /home/elisa              Shell: /bin/bash
Last login Tue Apr 25 19:15 (  ) on tty5
No Mail.
```

```
No Plan.
$ _
```

In questo caso, oltre ad alcuni dati già visti, ci viene comunicato anche il nome della home directory dell'utente, la shell che utilizza e la data dell'ultimo accesso al sistema; il messaggio "No Mail." ci informa che l'utente non ha posta elettronica giacente non ancora letta; se invece ci fossero stati dei nuovi messaggi da leggere, il sistema ci avrebbe informato sulla data e l'ora in cui l'utente ha letto per l'ultima volta la posta. Il messaggio "No Plan." ci informa invece che l'utente non ha predisposto un file per comunicare agli altri delle informazioni sul proprio lavoro o su altri aspetti della propria attività. Se nella home directory di Elisa si fosse trovato il file "`.plan`" (un normale file di testo), il contenuto di tale file sarebbe stato visualizzato al posto del messaggio "No Plan.".

Ora che sappiamo come verificare se un certo utente è collegato o meno al nostro sistema, possiamo fare il passo successivo: è possibile fare in modo di visualizzare un messaggio sul suo terminale. Il comando da usare è `write` seguito dallo username dell'utente a cui si vuole scrivere ed, eventualmente, anche dal terminale su cui si vuole visualizzare il messaggio; una volta dato il comando si può scrivere il messaggio, anche su più righe; per terminarlo si deve battere Ctrl-d. Immediatamente sul terminale dell'utente specificato verrà visualizzato il messaggio, con tanto di indicazione del mittente.

Un modo un po' più sofisticato (ed utile) di comunicare con gli altri utenti del sistema è offerto dal comando `talk`, che permette di stabilire una comunicazione bidirezionale con il nostro interlocutore, una specie di telefonata via terminale. La sintassi del comando è come al solito assai semplice: "`talk` *username* [*terminale*]" (come per il `write`, specificare il nome del terminale su cui effettuare il collegamento, non è indispensabile). Se l'utente specificato è effettivamente collegato, il sistema visualizza sul suo terminale un messaggio del tipo:

```
Message from TalkDaemon...
talk: connection requested by marco.
talk: respond with: talk marco
```

Se il nostro interlocutore accetta di iniziare il "talk" dovrà rispondere con un "`talk marco`" e a quel punto la sessione di *chat* avrà inizio: lo schermo dei due terminali viene diviso a metà, nella parte superiore vengono riportate le parole scritte dal proprietario del terminale, mentre in basso sono riportate quelle scritte dal suo interlocutore; per terminare il `talk` si deve battere Ctrl-c.

Questi due tipi di comunicazione (`write` e `talk`) hanno il limite di essere "volatili" quanto una telefonata: terminata la comunicazione di essa non rimarrà traccia da nessuna parte. Richiedono inoltre la presenza contemporanea di entrambi gli interlocutori. Viceversa la posta elettronica è uno strumento di maggiore utilità proprio perchè non è necessario che il destinatario del messaggio sia collegato al sistema nel momento in cui avviene la spedizione: i messaggi rimarranno giacenti nella "*mailbox*" dell'utente e potranno essere comodamente letti quando l'utente stesso si collegherà al sistema; in un certo senso è una sorta di potente servizio di segreteria telefonica. Ma non solo: i messaggi di posta elettronica non svaniscono nel nulla dopo che li si è letti, è infatti possibile salvarli su un file o stamparli su carta per poterli rileggere o riutilizzare in seguito.

Il programma più elmentare per la gestione della posta elettronica è **`mail`**, presente su quasi ogni sistema Unix, ma se siete un utente alle prime armi forse sarà meglio utilizzare un programma più sofisticato e semplice da usare come **`elm`** o **`pine`**. Descriviamo, in estrema sintesi, le principali operazioni da compiere per usare **`mail`**.

Per scrivere ed inviare un messaggio di posta elettronica ad un altro utente dovremo semplicemente digitare il comando **`mail`** seguito dallo username dell'utente; il sistema ci chiederà di inserire l'oggetto del messaggio, una sorta di titolo del messaggio stesso, quindi potremo iniziare a digitare il testo. Terminato il messaggio digiteremo all'inizio di una linea vuota un punto, che sta ad indicare la fine del messaggio stesso. Vediamo un esempio:

```
$ mail elisa
Subject: appuntamento a cena
Cara Elisa,
volevo solo ricordarti del nostro appuntamento al ristorante
cinese, per questa sera alle 20.
Ciao,
   Marco
.
EOT
$ _
```

Se Elisa è collegata al sistema vedrà comparire sul proprio terminale il messaggio "You have new mail.", altrimenti questo messaggio sarà visualizzato al momento del login, al prossimo collegamento. Per leggere i messaggi giacenti nella *mailbox* Elisa non dovrà fare altro che digitare il comando **`mail`**:

```
$ mail
Mail version 5.5 6/1/90.  Type ? for help.
"/var/spool/mail/elisa": 1 message 1 new
>N  1 marco         Wed Apr 26 17:53  21/450  "appuntamento a cena"
& 1
Message 1:
From marco Wed Apr 26 17:53:52 1995
Date: Wed, 26 Apr 95 17:53 GMT+0100
From: marco (Marco Liverani)
To: elisa
Subject: appuntamento a cena

Cara Elisa,
volevo solo ricordarti del nostro appuntamento al ristorante
cinese, per questa sera alle 20.
Ciao,
   Marco

& q
Saved 1 message in mbox
$ _
```

Se ci sono messaggi giacenti non ancora letti, digitando il comando `mail` si entra in un programma gestito mediante dei comandi formati da una sola lettera che devono essere digitati al prompt (che in questo caso è costituito dal carattere “`&`”). Nell'esempio Elisa si limita a leggere il messaggio, digitandone il numero progressivo corrispondente, e poi ad uscire dal programma digitando $\boxed{\texttt{q}}$ (*quit*). Uscendo dal programma i messaggi giacenti, letti ma non cancellati, vengono archiviati nella *mailbox* dell'utente e possono essere recuperati per essere riletti successivamente, con il comando “`mail -f`”. Per cancellare un messaggio si deve digitare $\boxed{\texttt{d}}$ (*delete*) ed il numero del messaggio; per replicare ad un messaggio si deve digitare $\boxed{\texttt{r}}$ (*reply*) ed il numero del messaggio. Battendo $\boxed{\texttt{?}}$ si ottiene una lista dei comandi principali del programma `mail`, mentre informazioni più dettagliate le fornisce la pagina di manuale (`man mail`).

Esiste un altro modo, non interattivo, di spedire messaggi di posta elettronica mediante il programma `mail` e consiste nell'usare le numerose opzioni su linea di comando. La sintassi è la seguente:

`mail -s "`*oggetto*`"` *destinatario* `-c` *altri indirizzi* `<` *file*

dove *oggetto* è l'oggetto del messaggio, racchiuso tra virgolette, *destinatario* è lo username dell'utente a cui si intende spedire il messaggio, *altri indirizzi* è una lista di username di altri destinatari a cui inviare il messaggio “per conoscenza” (*carbon copy*); l'opzone “`-c`”, con gli indirizzi che seguono, può anche essere omessa, visto che è una possibilità in più offerta dal programma, non indispensabile; alla fine della riga si digita il simbolo “`<`” (minore) seguito dal nome del file contenente il testo del messaggio. In questo modo è possibile preparare in precedenza il messaggio con il nostro editor preferito e poi, dopo averlo letto e corretto opportunamente, lo invieremo con questo comando.

Se desideriamo inserire alla fine di ogni nostro messaggio una firma piuttosto elaborata con cui personalizzare le nostre *mail*, è possibile prepararne il testo con un editor e salvarla nella nostra home directory con il nome “`.signature`”. Il programma di posta elettronica la aggiungerà automaticamente alla fine di ogni messaggio in partenza.

Lavorare con il programma `mail` è comunque abbastanza laborioso e di certo poco intuitivo; molto più semplice e guidato è invece l'uso di `elm` o del suo diretto concorrente `pine` (pine = pine is not elm). Quest'ultimo in particolare è guidato completamente mediante dei menù molto chiari e dispone di un editor (`pico`) estremamente comodo da usare.

2.7 Gestione dei processi

In ambiente Unix si parla di *processo* per indicare un programma in esecuzione. Sappiamo che Unix è un sistema operativo multitasking, ma fino ad ora abbiamo sfruttato questa caratteristica solo grazie alla multiutenza, che ci permetteva di “toccare con mano” il fatto che la macchina, avendo più utenti collegati contemporaneamente, stava effettivamente elaborando più di un programma.

Per sfruttare pienamente e con comodità il multitasking si deve disporre di un terminale grafico che ci permetta di aprire sullo schermo più finestre in cui lanciare i diversi processi; è possibile fare altrettanto su un normale terminale alfanumerico, ma anche in questo caso si deve disporre di un programma (ad esempio `screen`) che ci consenta di simulare un ambiente con più finestre. Vedremo in maggiore dettaglio l'ambiente X Window in seguito, per ora ci limiteremo a dire che è possibile lanciare delle applicazioni che lavorano autonomamente ed indipendentemente dalle altre in una regione dello schermo (finestra) riservata ad ognuna di esse.

Per lanciare un'applicazione, come al solito è necessario dare un comando al prompt della shell, ma così facendo, come abbiamo visto fino ad ora, perdiamo temporaneamente l'uso della shell, fino a quando non saremo usciti dal programma in esecuzione. Esiste un modo per sganciare il processo "figlio" (il programma da eseguire) dal processo "padre" (la shell da cui si lancia il programma) rendendo i due processi indipendenti ed autonomi. Se aggiungiamo una "`&`" (e commerciale) alla fine della riga che invoca un certo comando, tale comando sarà eseguito in una "sessione" separata e quindi potremo utilizzare contemporaneamente la shell (da cui possiamo nel frattempo lanciare altri processi) ed il programma che abbiamo lanciato.

Vediamo un esempio molto semplice che possiamo provare anche su un normale terminale alfanumerico. Il comando `yes` visualizza una serie di "y" sullo schermo fino a quando non viene interrotto dall'utente. Per interrompere l'esecuzione del programma dobbiamo battere Ctrl-c (*break*). Se invece di interromperlo volessimo solo sospenderne temporaneamente l'esecuzione, invece di break dovremmo battere Ctrl-z (*stop*):

```
$ yes
y
y
y
...
(l'utente batte Ctrl-z)
[1]+  Stopped          yes
$ _
```

Il sistema ci informa che il processo numero 1, che è stato lanciato con il comando `yes`, è stato momentaneamente interrotto. Il comando `jobs` ci permette di visualizzare la lista dei processi lanciati da quella shell; nel caso dell'esempio precedente avremmo il seguente output:

```
$ jobs
[1]+  Stopped          yes
$ _
```

In questo momento abbiamo due processi attivi: la shell ed il programma `yes` che è momentaneamente interrotto. In particolare diremo che il processo attivo, la shell, è in *foreground*, mentre l'altro processo è in *background*; in uno stesso momento può esserci un solo programma in foreground, ma anche molti programmi in background. Per riportare in foreground il programma `yes` (e mandare quindi in background la shell) si deve usare il comando `fg` (*foreground*) seguito dal numero del processo:

```
$ fg 1
y
y
y
...
```

Avevamo citato precedentemente tra le unità disponibili su un sistema Unix, anche il device "nullo" `/dev/null`. Ora potrebbe tornarci utile. È possibile redirigere l'output di una applicazione verso una unità diversa dal device di output standard (il video del terminale) mediante l'uso del carattere "`>`" (maggiore). Proviamo a redirigere l'output del programma **`yes`** verso l'unità nulla, dando il comando "**`yes > /dev/null`**". Il programma è in esecuzione, ma sullo schermo non appare nulla, neanche il prompt della shell per poter lanciare altri programmi nel frattempo. Interrompiamo l'esecuzione di **`yes`** battendo Ctrl-c e proviamo a riavviarlo con il comando "**`yes > /dev/null &`**":

```
$ yes > /dev/null &
[1] 143
$ jobs
[1] 143  Running          yes >/dev/null &
$ ps
  PID TTY STAT  TIME COMMAND
   67  1  S     1:32 bash
  143  1  R     0:00 yes
  152  1  R     0:00 ps
$ _
```

Con l'aggiunta del simbolo "**`&`**" alla fine della linea di comando, abbiamo lanciato l'applicazione in background, mantenendo l'uso della shell per impostare altri comandi. Con il messaggio "**`[1] 143`**" il sistema ci comunica che l'applicazione **`yes`** è il primo processo lanciato da questa shell e, nella tabella di tutti i processi del sistema, gli è stato assegnato il numero 143 (questo non vuol dire che ci sono 143 processi attivi). Con il comando **`jobs`** verifichiamo gli stessi dati ed in più il sistema ci comunica che l'applicazione è attualmente in esecuzione (*running*). Il comando **`ps`** ci fornisce delle informazioni su tutti i nostri processi attivi, non solo quelli lanciati attraverso una certa shell; l'esempio mostra che nel nostro caso sono attivi tre processi, tutti sul terminale **`tty1`**: **`bash`**, la shell è attiva da un'ora e 32 minuti ed è momentaneamente sospesa ("`S`") perchè sta eseguendo il comando **`ps`**, che è stato appena attivato ed è in esecuzione ("`R`"), come pure il programma **`yes`**, attivo anche questo solo da qualche istante.

Con il comando **`kill`** è possibile interrompere forzatamente l'esecuzione di un processo in background (come la pressione dei tasti Ctrl-c per i processi in foreground). Insieme al **`kill`** si deve specificare il PID (*Process ID*) dell'applicazione che vogliamo terminare o il suo *job number*, preceduto però dal simbolo di percentuale "**`%`**". Ad esempio per interrompere il programma yes dell'esempio precedente i due comandi "**`kill 143`**" e "**`kill %1`**" sono equivalenti.

2.8 Alcuni comandi utili

I comandi disponibili sul mio sistema Unix sono alcune centinaia: oltre ad essere impossibile riportarli tutti, sarebbe anche abbastanza inutile, visto che sono moltissimi i comandi che non ho mai usato e che probabilmente non userò mai. Ci limitiamo in questa sezione a descrivere per sommi capi alcuni comandi non indispensabili, ma che possono essere di una certa utilità. Per quanto riguarda gli altri, quelli che non hanno trovato spazio in queste pagine, il consiglio è il seguente: quando sentirete la necessità di un comando per svolgere un particolare compito, cercatelo, magari "sfogliando" le pagine di manuale, perché quasi sicuramente già esiste!

È forse il caso di chiarire che quando diamo un comando, la shell va a cercare il file eseguibile con quel nome in alcune directory del filesystem; queste directory sono quelle specificate nel *path*: è una variabile di ambiente della shell in cui è riportato l'elenco delle directory del nostro sistema che contengono file eseguibili. Per visualizzare l'elenco delle variabili di sistema si deve usare il comando `set`. Per sapere in quale directory è collocato fisicamente il file eseguibile di un certo programma si deve usare il comando "`which` *programma*".

Abbiamo spesso usato, negli esempi della pagine precedenti, gli operatori di redirezione dell'input/output. Più in dettaglio possiamo dire che il simbolo "`>`" serve per inviare l'output standard (quello che normalmente finisce sul video del terminale) su un file o su un altro device; il simbolo "`<`" serve invece per leggere l'input standard (quello che altrimenti sarebbe inserito manualmente dall'utente mediante la tastiera) da un file o da un altro device. Infine il simbolo "`|`" (*pipe*) serve ad indirizzare l'output di un programma verso l'input di un secondo programma. Il seguente esempio utilizza tutti gli operatori di redirezione dell'I/O:

```
$ cat < lista | sort > lista.ordinata
$ _
```

Il programma `cat` (che, come abbiamo visto, serve a visualizzare il contenuto di un file) riceve l'input dal file `lista`; l'output di `cat` viene inviato mediante il *piping* al programma `sort` (che serve ad ordinare i dati contenuti in una lista) che li invia in output al file `lista.ordinata`. Al termine dell'esecuzione di questo comando il file `lista.ordinata` conterrà gli stessi dati di `lista`, ma ordinati alfabeticamente. Riguardo al programma `sort` è forse opportuno aggiungere che i comandi "`sort < lista > lista.ordinata`" e "`sort lista -o lista.ordinata`" avrebbero svolto efficacemente lo stesso compito del comando riportato nell'esempio precedente al solo scopo di illustrare l'uso di tutti i simboli di redirezione dell'I/O.

Con il comando `date` il sistema ci fornisce la data e l'ora corrente, mentre con `cal` viene visualizzato un sintetico calendario del mese corrente; per avere il calendario completo di un intero anno basterà specificare l'anno desiderato di seguito al comando `cal`, ad esempio "`cal 1492`".

Spesso si desidera tenere un file sul disco, magari senza utilizzarlo spesso; in tal caso, per risparmiare spazio, potrebbe essere opportuno comprimere tale file, ricodificandolo in modo opportuno. A questo scopo esistono i programmi compattatori che si occupano di generare un file in formato compresso che, pur contenendo le stesse informazioni, occupa meno spazio del file originale. Naturalmente quando vorremo accedere alle informazioni contenute nel file originale, dovremo eseguire l'operazione inversa, scompattando il file compresso, per ottenere nuovamente il file originale. Su Unix esistono implementazioni di tutti i principali compattatori: `gzip` e `compress` sono sicuramente i più diffusi in questo ambiente. Con il comando "`gzip` *file*" si ottiene un file compresso, mentre per ottenere il file originale da quello compresso si deve usare il comando inverso "`gunzip` *file.gz*" (o anche "`gzip -d` *nome.gz*"). Sintassi del tutto analoga è quella del comando `compress` (e del suo inverso `uncompress`). I file compressi con `gzip` hanno estensione "`.gz`", mentre quelli trattati con `compress` hanno estensione "`.Z`".

Un po' diverso è il programma `zip`, assai simile al famoso `PKZIP` molto diffuso sui sistemi DOS. È possibile archiviare in formato compresso numerosi file all'interno di un unico file compresso. Ad esempio con il comando "`zip src.zip src/*`" si archiviano in formato compresso, nel file `src.zip`, tutti i files contenuti nella directory `src`. Per estrarre i file originali dal file "zippato" si deve usare il comando "`unzip src.zip`", mentre per vedere il contenuto del file compresso, senza però estrarre i file in esso contenuti, si può usare l'opzione "`-v`"; ad esempio "`unzip -v src.zip`".

Il comando `tar` (*tape archive*) è molto usato in ambiente Unix e di solito serve per includere (senza comprimere) in un file unico più files, magari sparsi in directory differenti. In particolare è molto usato per effettuare il *backup* dei files del sistema. I files archiviati con `tar` hanno estensione "`.tar`", mentre quelli con estensione "`.tgz`" devono essere prima decompressi con `gunzip`. Per visualizzare il contenuto di un file archiviato con `tar` si deve usare il comando "`tar tfv` *nomefile.tar*", mentre per estrarre effettivamente i files si usa il comando "`tar xfv` *nomefile.tar*".

Per visualzzare lo spazio su disco occupato dalla directory corrente e da tutte le sue sottodirectory si deve usare il comando `du` (*disk usage*). Per vedere invece quanta parte dei dischi montati nel filesystem è occupata si deve usare il comando `df` (*disk free*). Infine per visualizzare le risorse di memoria ancora libere si usa il comando `free`.

Come il comando `ps` visualizza i processi attivi sul sistema, il comando `lpq` (*line printer queue*) visualizza i "job pendenti" nella coda di stampa; se si desidera interrompere una stampa, bisogna eliminare il relativo job dalla coda di stampa con il comando `lprm` seguito dal *job-id* indicato dal comando `lpq`.

Capitolo 3

Editing di file di testo

In questo capitolo descriveremo sinteticamente tre programmi per comporre files di testo. Ulteriori e più approfondite informazioni su questi editor possono essere reperite, come al solito, sulle relative *man pages*. I primi due editor sono tra i più diffusi in ambiente Unix: `vi` ed `emacs`; il terzo è `pico`, l'editor del programma di posta elettronica `pine`, che può essere usato anche indipendentemente da quest'ultimo; `pico`, pur essendo meno diffuso dei primi due e soprattutto molto meno potente, è stato inserito in questa brevissima panoramica perché è uno degli editor più semplici da usare tra i tanti reperibili in ambiente Unix.

3.1 L'editor vi

`vi` (si pronuncia "*vuai*") è sicuramente l'editor più diffuso sotto Unix ed anche uno degli editor più potenti in assoluto. Con `vi` si può fare praticamente tutto, l'unico difetto è che è un po' ostico da utilizzare e completamente non standard rispetto agli altri editor (o meglio, vista la sua importanza, `vi` costituisce uno standard a sé).

Il programma è caratterizzato da due modalità operative completamente differenti: la modalità "comando" e la modalità "inserimento". Nella prima è possibile impostare i comandi generali per la gestione dell'intero file (in ambiente `vi` il file caricato si chiama *buffer*), mentre nella seconda è possibile scrivere e modificare il testo.

Appena caricato il `vi` ci troviamo in modalità comando, in cui possiamo muoverci all'interno del buffer utilizzando i tasti di spostamento del cursore o i tasti [h] per spostare il cursore verso sinistra, [j] per spostarlo in basso, [k] per muoverlo verso l'alto e [l] per muoverlo verso destra.

Dalla modalità comando il tasto [i] ci permette di passare alla modalità inserimento. Per tornare in modalità comando si deve battere il tasto [Esc] (che su alcuni sistemi è sostituito dai tasti [Ctrl-[]). Il programma non visualizza nessuna indicazione sulla modalità attiva, quindi si deve porre una certa attenzione quando si commuta da una modalità all'altra.

Entrati in modalità inserimento possiamo digitare il testo da inserire, proprio come su qualsiasi altro programma di videoscrittura. Ogni modifica, cancellazione o spostamento, avviene però sul singolo carattere: per operare su interi blocchi di testo ci si deve portare in modalità comando, dove si può, ad esempio, cancellare una intera linea del buffer battendo due volte il tasto [d], oppure un singolo carattere sotto al cursore, battendo [x]. Per tornare in modalità inserimento, invece del tasto [i] che ci permette di inserire il testo a partire dalla posizione attuale del cursore (*insert*), possiamo battere `a` (*add*) che ci permette di aggiungere il testo alla fine della linea su cui si trova il cursore.

La modalità comando mette a disposizione una vasta gamma di istruzioni, costituite da una o due lettere, con cui è possibile operare anche sull'intero buffer. Per utilizzarle si deve battere `:` (due punti) e poi digitare il comando. Vediamone alcuni:

`co` (*copy*) Per copiare il testo che va dalla riga n_1 alla riga n_2, dopo la riga n_3, si deve digitare il comando "`:`n_1, n_2`co`n_3";

`mo` (*move*) Per spostare dopo la riga n_3 il testo che va dalla riga n_1 alla riga n_2, si deve digitare il comando "`:`n_1, n_2`mo`n_3";

`d` (*delete*) Per cancellare il testo dalla riga n_1 alla riga n_2 si digiti il comando "`:`n_1, n_2`d`";

`r` (*read*) Per inserire il contenuto del file *nome* dopo la riga n, si digiti il comando "`:`n `r` *nome*";

`w` (*write*) Per salvare sul file *nome* il testo dalla riga n_1 alla riga n_2, si deve digitare il comando "`:`n_1, n_2 `w` *nome*"; per salvare l'intero file si digiti semplicemente "`:w`", oppure "`:w` *nome*";

`q` (*quit*) Per uscire dal programma si digiti "`:q`"; per uscire senza salvare su file le modifiche apportate al testo si digiti "`:q!`", mentre per uscire e salvare il testo su file si deve dare il comando "`:wq`".

Per spostarsi direttamente su una certa linea all'interno del testo si può digitare "`:`n", dove n è il numero di riga; per andare alla fine del file si può battere più semplicemente `G`.

Come in ogni editor che si rispetti, anche in `vi` è possibile cercare una certa stringa di caratteri all'interno del buffer: dalla modalità comando basta battere `/` (*slash*) seguito dalla stringa da cercare; una volta trovata la prima occorrenza della stringa, per cercarne un'altra occorrenza basterà battere nuovamente `/`, senza specificare nessuna stringa.

Per utilizzare i comandi di `vi` è estremamente utile poter identificare i numeri di linea del testo inserito; per questo è opportuno, appena avviato il programma, dare il comando "`:set number`" che attiva la numearzione automatica delle linee. Per far sì che `vi` abiliti automaticamente questa funzione, si deve inserire il comando "`set number`" nel file "`.exrc`" che contiene i comandi che devono essere eseguiti da `vi` all'inizio della sessione di lavoro; come tutti i file di configurazione, anche questo comincia con un punto e deve essere inserito nella nostra home directory. A titolo di esempio riporto il contenuto del mio "`.exrc`":

```
set autoindent
set tabstop=2 shiftwidth=2
set number
set wm=10
```

`vi` ha un gran numero di comandi, alcuni dei quali (pochissimi, a dire il vero) sono stati descritti in queste pagine; per avere un riferimento più dettagliato sulle varie possibilità offerte dal programma è opportuno riferirsi alla documentazione del proprio sistema.

3.2 Emacs

`emacs` è forse più sofisticato, ma altrettanto complesso e potente di `vi`. Supporta un linguaggio per la compilazione di *macro* molto simile al *Lisp*, un linguaggio funzionale che opera su liste. Questo linguaggio di macro è potente al punto che qualcuno si è anche divertito a sviluppare dei veri programmi che girano all'interno dell'editor `emacs`!

Come struttura il programma si avvicina maggiormente agli editor tradizionali, in questo caso non c'è la doppia modalità inserimento/comandi, ma tutto può essere fatto in un unico ambiente e le sequenze di tasti che non devono essere inserite nel testo, perché servono a dare dei comandi al programma, sono tutte composte premendo Ctrl in unione ad altri tasti.

Una caratteristica importante di `emacs` è il fatto di poter operare contemporaneamente su più file (che anche in questo caso vengono chiamati *buffer*).

Esiste anche una versione grafica di `emacs` che permette di lavorare in tutta comodità su un terminale grafico, utilizzando per i comandi, invece delle combinazioni di tasti, i consueti menù a tendina selezionabili con il mouse.

Spesso `emacs` è utilizzato proprio in questa modalità grafica guidata dai menù, che quindi non richiede ulteriori spiegazioni; ci limiteremo allora a descrivere i principali comandi che devono essere impartiti da tastiera nel caso si utilizzi il programma su un terminale alfanumerico. Generalmente si utilizzano i caratteri di spostamento del cursore per muoversi all'interno del buffer ed il tasto backspace per effettuare le cancellazioni, tuttavia non tutti i terminali prevedono l'uso di questi tasti; in tal caso è bene riferirsi alla seguente tabella:

Ctrl-V Avanti di una schermata;

Ctrl-v Indietro di una schermata;

Ctrl-p Indietro di una riga (*previous*);

Ctrl-n Avanti di una riga (*next*);

Ctrl-b indietro (a sinistra) di un carattere (*back*);

Ctrl-f avanti (a destra) di un carattere (*forward*).

Per la gestione dei files si deve fare riferimento alla seguente tabella:

Ctrl-x Ctrl-f *nomefile* Carica nel buffer il file specificato;

Ctrl-x Ctrl-s Salva su file il buffer corrente;

Ctrl-x Ctrl-c Termina l'esecuzione del programma; prima di uscire chiede se si desidera salvare su file eventuali buffer modificati ma non ancora salvati.

Ogni volta che si usa il comando Ctrl-x Ctrl-f per aprire un nuovo file, lo si carica nel buffer, ma non si eliminano i buffer precedentemente caricati: semplicemente avremo più buffer aperti e distinti, ognuno contiene il testo di un file. Si può passare da un buffer all'altro utilizzando le seguenti combinazioni di tasti che gestiscono le "finestre" di `emacs`:

Ctrl-x Ctrl-b Elenca i buffers attivi;

Ctrl-x Ctrl-f *nomebuffer* Se il buffer è già stato caricato, lo visualizza nella finestra attiva, altrimenti prima carica il file con quel nome, lo inserisce in un nuovo buffer e lo visualizza nella finestra attiva;

Ctrl-x 2 Divide lo schermo a metà creando una seconda finestra;

Ctrl-x o Se lo schermo visualizza più di una finestra (attivata con Ctrl-x 2) passa da una finestra all'altra (*other*);

Ctrl-x 1 Visualizza sullo schermo solo la finestra attiva (quella in cui si trova il cursore).

Per effettuare delle ricerche sul testo di un buffer si deve battere Ctrl-s (*search*) e poi digitare la stringa di testo da cercare; per ripetere la ricerca della stessa stringa sarà sufficiente battere di nuovo Ctrl-s.

Emacs è dotato di un potente sistema di *help on-line* e di un *tutorial*, una sorta di breve corso guidato sulle funzioni principali del programma. Per accedere all'help, ed avere istruzioni sull'uso di un certo comando, basta battere Ctrl-h c e poi la sequenza di caratteri su cui si vuole un aiuto. Ad esempio:

```
>> Ctrl-h c Ctrl-p
C-p runs the command previous-line
```

Per avere informazioni più complete su una certa sequenza di tasti si può usare il comando Ctrl-h k seguito dal comando di cui si vogliono le informazioni. `emacs` visualizza il testo di help in una finestra separata, quindi per ripristinare la finestra su cui stavamo lavorando, eliminando l'help, dovremo battere Ctrl-x 1.

Per eseguire il tutorial si deve battere Ctrl-h Ctrl-h t. Il tutorial viene visualizzato in una finestra separata e permette anche di sperimentare praticamente numerosi comandi.

3.3 Pico

Spenderemo soltanto poche parole per descrivere questo semplice editor che, come abbiamo già detto, viene anche utilizzato dal programma `pine` come editor per la compilazione dei messaggi di posta elettronica.

Ciò che distingue `pico` dagli altri due programmi visti in questo capitolo, è sicuramente il fatto che tutti i comandi che è possibile impostare vengono riportati sinteticamente in una specie di menù visualizzato sulle ultime due righe del terminale. Ogni comando viene impostato premendo Ctrl insieme ad un altro tasto. A differenza di `emacs`, `pico` dispone di un unico `buffer` e quindi può operare su un unico file alla volta. In ogni caso qualsiasi paragone tra questo programma ed uno degli altri due editor visti in precedenza è del tutto fuori luogo: si tratta di prodotti di classe completamente diversa, potenti e versatili `vi` ed `emacs`, estremamente limitato, ma assai intuitivo nell'uso, `pico`.

Riportiamo una breve spiegazione dei comandi principali:

Ctrl-c visualizza il numero di linea su cui si trova il cursore (*current position*);

Ctrl-g Visualizza le informazioni di *help* sul programma (*get help*);

$\boxed{\text{Ctrl-j}}$ "Giustifica", riallineando i margini destro e sinistro delle righe, il paragrafo corrente (*justify*);

$\boxed{\text{Ctrl-k}}$ Cancella la riga su cui si trova il cursore (*cut text*);

$\boxed{\text{Ctrl-o}}$ Salva su file il testo contenuto nel buffer (*write out*);

$\boxed{\text{Ctrl-r}}$ Inserisce il testo di un altro file nella posizione corrente del cursore (*read file*);

$\boxed{\text{Ctrl-u}}$ Annulla l'ultimo comando eseguito (*undo*);

$\boxed{\text{Ctrl-w}}$ Cerca una stringa di caratteri nel testo (*where is*);

$\boxed{\text{Ctrl-x}}$ Esce dal programma (*exit*).

Per cancellare o spostare un intero blocco di testo è possibile utilizzare il comando di "selezione estesa": ci si posiziona all'inizio del testo da selezionare e si batte $\boxed{\text{Ctrl-\^{}}}$; muovendosi poi con il cursore, si estende la selezione: il testo marcato viene visualizzato in *reverse*. A questo punto è possibile tagliare l'intera selezione con il comando $\boxed{\text{Ctrl-k}}$ e poi eventualmente incollarlo (con $\boxed{\text{Ctrl-u}}$) nella nuova posizione del cursore.

Pico non offre altre funzionalità avanzate: quindi solo lo stretto indispensabile per scrivere un breve file di testo (come un messaggio di posta elettronica, ad esempio) e poco altro. Se dovessimo avere la necessità di eseguire operazioni più complesse sul nostro file, dovremo rivolgeci alla maggiore potenza (e complessità) di **`emacs`** e **`vi`**.

Capitolo 4

L'interfaccia grafica X Window

Unix nasce nei primi anni '70, quando i grandi elaboratori centralizzati, i *mainframe*, lavoravano soprattutto mediante schede perforate, ed i terminali con video e tastiera, così come li conosciamo oggi, erano appannaggio di pochi fortunati. L'intero sistema operativo è stato sviluppato quindi in modo tale da prescindere completamente dal tipo di terminale con cui l'utente avrebbe utilizzato il sistema. Negli ultimi dieci anni hanno avuto un grosso sviluppo le tecnologie per la gestione del video e della cosiddetta *interfaccia utente*, ossia quell'insieme di dispositivi hardware e software che permettono all'utente di comunicare (in gergo, di *interfacciarsi*) con la macchina. Grazie al lavoro svolto da un gruppo di ricercatori "visionari" (almeno per quei tempi, oggi possiamo dire lungimiranti) del Palo Alto Research Center (PARC) della Xerox, hanno avuto un particolare sviluppo quelle che oggi chiamiamo *GUI*, Graphical User Interface, che ci permettono di operare su un terminale grafico, interagendo col sistema principalmente mediante il mouse e l'uso estensivo di rappresentazioni grafiche. Ci stiamo riferendo a quello che sicuramente ognuno di noi ha visto almeno una volta sul proprio personal computer: l'ambiente Microsoft Windows, il Finder del Macintosh e la WorkPlace Shell di OS/2, solo per citare alcuni esempi.

Anche Unix, ormai da diversi anni, è dotato di una propria interfaccia grafica standard, denominata *X Window*. In questo capitolo cercheremo di farci un'idea piuttosto sommaria ed un po' superficiale, di cosa ci offre in più l'uso di questo ambiente, rispetto al terminale alfanumerico.

4.1 X Window e i window manager

X Window è un insieme di funzioni di "basso livello" (cioè, che comunicano molto da vicino con l'hardware del sistema) per la gestione di alcune funzionalità di base dell'interfaccia utente grafica (GUI). Per alcuni aspetti, il rapporto che c'è tra Unix ed X Window, è abbastanza simile a quello che c'è tra il DOS e Microsoft Windows: si tratta in entrambi i casi di una interfaccia grafica che si appoggia sul sistema operativo e ne sfrutta le funzionalità ed i servizi. Nel caso di OS/2 la situazione è un po' diversa, perché nel *Presentation Manager* sono incluse alcune funzionalità fondamentali che in modalità "a carattere" non sono previste; nel Macintosh, l'interfaccia utente costituisce parte integrante del sistema operativo da cui è inscindibile, anzi, alcune funzioni di base dell'interfaccia grafica sono addirittura incluse su ROM e fanno parte del *firmware* dei computer Apple Macintosh.

Non dobbiamo però confondere X Window con il *window manager* che si occupa della presentazione a video delle finestre e della gestione di alcune funzionalità che non sono incluse nel set di base fornito da X Window stesso. Infatti, mentre con Microsoft Windows l'aspetto delle finestre è sempre lo stesso, sopra ad X Window si appoggia un altro programma, il window manager, appunto, che definisce l'aspetto esteriore delle finestre e gestisce il *desktop*. Uno stesso sistema può quindi essere configurato in modo tale da consentire ai propri utenti di scegliere il window manager preferito, con cui lavorare durante le sessioni al terminale grafico.

Più in dettaglio possiamo dire che il window manager definisce l'aspetto della *barra del titolo* della finestra, degli eventuali *bottoni* collocati su di essa, delle *icone* e dei *menù*, che solitamente vengono visualizzati clickando sullo sfondo dello schermo (*desktop*). Si occupa anche della modalità operativa che l'utente deve adottare per operare sulle finestre: i tre pulsanti del mouse, ad esempio, svolgono funzionalità diverse a seconda del window manager adottato.

A differenza di quanto avviene con Microsoft Windows, per selezionare una finestra e renderla attiva (per poter operare al suo interno), non è necessario effettuare un *click* del mouse su di essa, ma (in generale) basta che il puntatore del mouse si trovi sulla finestra stessa; l'input digitato sulla tastiera sarà diretto all'applicazione eseguita nella finestra selezionata.

Alcuni window manager hanno la cosiddetta gestione del desktop "virtuale": lo schermo a disposizione dell'utente viene reso virtualmente molto più grande di quello fisicamente visibile attraverso il monitor; mediante una finestra particolare (chiamata *pager*) che rappresenta in scala ridotta l'intero grande schermo, è possibile selezionare la zona da "inquadrare" nel video del terminale.

I window manager più diffusi sono quattro, ma molti altri ne esistono o sono in via di sviluppo o di definizione:

- *Tab Window Manager* (twm), il primo e forse il più "spartano" fra i window manager; non è molto sofisticato e tuttavia spesso è apprezzato proprio per la sobrietà con cui visualizza le finestre. Twm si limita a visualizzare una barra del titolo sopra ad ogni finestra con un bottone per "iconizzare" la finestra stessa ed un'altro per modificarne la dimensione; per spostare la finestra la si deve trascinare con il mouse clickando sulla barra del titolo.

- *F(?) Virtual Window Manager* (fvwm), è uno dei più diffusi window manager per X Window e nasce come evoluzione di twm. Oltre alla barra del titolo, fvwm visualizza anche un bordo intorno alle finestre e dà al tutto un aspetto tridimensionale. È possibile inserire diversi bottoni sulla barra del titolo a cui associare funzionalità diverse: rendere massima la dimensione della finestra, ridurla ad icona, ecc. Il bottone nell'angolo in alto a sinistra permette di accedere ad un menù per la gestione della finestra stessa. Con fvwm è disponibile la gestione dello schermo "virtuale" più grande di quello reale.

- *Open Look Window Manager* (olwm), è il window manager che Sun Microsystems ha sviluppato inizialmente solo per le proprie workstation, ma che ultimamente ha anche reso disponibile, almeno in parte, per ogni altro sistema. Personalmente ritengo che sia il window manager più elegante tra quelli disponibili; sicuramente ha un livello di sofisticazione superiore a quello offerto dagli altri prodotti. Degni di nota sono i menù a tendina "staccabili" che è possibile fissare sul desktop nella posizione più comoda per l'utente. Esiste anche una versione di Open Look dotata di desktop virtuale (olvwm).

4.2 Xterm

Mediante l'uso di X Window si può sperimentare e trarre profitto con maggiore facilità dalla grande potenza del multitasking di Unix. Spesso utilizzeremo il terminale grafico per compiti che avremmo potuto svolgere ugualmente su un normale terminale alfanumerico, ma che grazie ad X Window possiamo svolgere con maggiore comodità ed efficienza. Il programma che di solito è più usato sotto X è `xterm`, una finestra di emulazione di terminale, dove viene eseguita la shell da cui è possibile lanciare altri programmi.

In pratica `xterm` riproduce in una finestra lo schermo di un terminale alfanumerico, aggiungendo però numerose funzionalità rese disponibili dall'ambiente X Window. Aprendo più finestre di `xterm` sullo schermo, potremo operare contemporaneamente, grazie al multitasking, su più applicazioni. Dalla shell attivata nella finestra di `xterm` è possibile lanciare anche applicazioni grafiche indipendenti dalla finestra stessa.

Quando si accede ad X Window (spesso chiamato più semplicemente X) di solito viene attivata automaticamente una finestra `xterm`. In alcuni casi la prima finestra aperta è la cosiddetta "shell di login": chiudendo questa sessione di `xterm` si effettua il logout dal sistema.

La shell che ci viene resa disponibile mediante `xterm` sarà il nostro programma chiave per controllare il sistema durante la sessione di lavoro al terminale grafico. Cerchiamo di entrare un po' più in dettaglio cominciando con qualche esempio. Per sperimentare subito il multitasking potremmo provare ad aprire un altro `xterm`, dando semplicemente il comando "`xterm`" al prompt della shell. Dopo pochi istanti il sistema apre una seconda finestra con un'altra shell attiva al suo interno. Ci accorgiamo subito però che la shell della prima finestra non è più attiva, e questo perché è stata "congelata" temporaneamente per poter eseguire l'applicazione che da essa è stata lanciata (il secondo `xterm`). Dove è finito il multitasking? La risposta è semplice: non l'abbiamo sfruttato a dovere. Portando il mouse all'interno della finestra `xterm` appena attivata, digitiamo il comando "`exit`" e vediamo ricomparire il prompt della shell che fino ad ora era rimasta inattiva. Come avevamo visto nel paragrafo 2.7 è necessario aggiungere il simbolo "`&`" alla fine della linea di comando per attivare il processo "figlio" in una sessione separata da quella del processo "padre". Per lanciare un secondo `xterm` è quindi necessario dare il comando "`xterm &`": in questo modo viene aperta una seconda finestra e viene attivata una shell al suo interno. Ogni comando che digiteremo verrà indirizzato alla finestra attiva (quella in cui si trova il mouse), evidenziata in modo opportuno dal window manager. Ogni volta che da un `xterm` vorremo lanciare un'applicazione che deve essere eseguita in una sessione separata, dovremo ricordarci di aggiungere il simbolo "`&`" alla fine della linea di comando.

Non sempre però è opportuno eseguire un'applicazione in una sessione separata: ad esempio se volessimo usare l'editor `vi` all'interno della nostra finestra `xterm`, non dovremo aggiungere la "`&`" alla fine del comando, proprio perché questa volta desideriamo interrompere momentaneamente l'esecuzione della shell, che deve lasciare il posto, all'interno della stessa finestra, all'editor. Sperimentare direttamente queste differenze aiuterà a capire questi meccanismi molto più di quanto non possa fare io con le mie intricate spiegazioni.

È possibile lanciare il programma `xterm` specificando una serie di opzioni tipiche delle applicazioni che operano sotto X. Vediamo quelle principali, che in alcuni casi potremo anche utilizzare con altri programmi:

`geometry` serve per specificare la dimensione e la posizione iniziale della finestra che si sta aprendo; ad esempio si il comando "`xterm -geometry 90x30+100-20 &`" apre un `xterm` di 90 colonne per 30 righe, distante 100 punti dal margine sinistro dello schermo e 20 dal margine inferiore;

`bg` specifica il colore di *background* della finestra che si sta aprendo, ad esempio: "`xterm -bg skyblue4 &`"; l'elenco dei colori di default, generalmente è contenuto nel file "`/usr/X11/lib/X11/rgb.txt`";

`fg` specifica il colore di default per le scritte visualizzate nella finestra (*foreground*), ad esempio: "`xterm -bg skyblue4 -fg white &`";

`cr` specifica il colore del cursore, ad esempio: "`xterm -bg skyblue4 -fg white -cr yellow &`";

`fn` specifica il nome del *font* (tipo di carattere) da utilizzare, ad esempio: "`xterm -fn lucidasanstypewriter-12 &`";

`sb` abilita la *scroll bar*, ad esempio: "`xterm -sb &`".

Si possono modificare alcune opzioni del programma `xterm` anche quando questo è già in esecuzione (*run time*). Tenendo premuto il tasto Ctrl è possibile attivare tre diversi menù facendo click con i tre pulsanti del mouse sulla finestra:

- *Main Options* – si abilita questo menù premendo il pulsante sinistro del mouse; serve fondamentalmente per dirigere l'input da tastiera esclusivamente a quella finestra, anche quando non è la finestra attiva, per ridisegnare il contenuto della finestra, per uscire dal programma (ma è sempre meglio usare il comando `exit` impostato da tastiera);

- *VT Options* – si attiva mediante il pulsante centrale del mouse; serve principalmente per abilitare/disabilitare la scroll bar, o per visualizzare la finestra in reverse (con i colori del background e del foreground invertiti);

- *VT Fonts* – si attiva con il pulsante destro del mouse e serve per stabilire la dimensione del font utilizzato.

L'ultima funzionalità offerta da `xterm`, ma forse anche una delle più interessanti, è il *copy & paste*. È possibile selezionare una parte del testo visualizzato in una finestra `xterm` e poi incollarlo in un altro punto della stessa finestra o di una finestra differente, nella posizione occupata dal cursore. La selezione (*copy*) avviene mediante il mouse, tenendo premuto il pulsante sinistro; l'inserimento del testo evidenziato (*paste*) avviene cliccando il pulsante di centro del mouse: il testo selezionato sarà inserito a partire dalla posizione occupata dal cursore, come se venisse digitato da tastiera. Questa funzionalità arricchisce e semplifica notevolmente l'uso di un editor (ad esempio `pico`, ma anche `vi`) utilizzato all'interno della finestra `xterm`.

4.3 Alcune utility

Lavorare in un ambiente grafico come X Window ci permette di utilizzare tante comode utility, che, pur non essendo indispensabili per svolgere il nostro lavoro, lo semplificano e lo rendono più piacevole. In questa sezione descriviamo brevemente alcuni di questi comandi; come al solito, per ottenere informazioni più dettagliate è bene riferirsi alle pagine di manuale, magari mediante il programma `xman`.

Con `xsetroot` si possono impostare alcuni parametri di configurazione per lo sfondo dello schermo (la cosiddetta *root window*). In particolare il parametro "`-solid`" permette di impostare il colore dello sfondo; ad esempio il comando "`xsetroot -solid steelblue`" imposta un colore azzurro chiaro. Sui terminali monocromatici potranno invece essere utili le opzioni "`-gray`" e "`-def`" che impostano due diversi sfondi in bianco e nero.

Esistono numerosi orologi visualizzabili in una finestra sotto X: il più diffuso è `xclock`, ma spesso si trovano anche `oclock` e `clock`, quest'ultimo sotto Open Look. Sono numerose le opzioni che possiamo specificare per modificare l'aspetto degli orologi. Con `xclock` ad esempio l'opzione "`-update 1`" visualizza anche la lancetta dei secondi, mentre "`-chime`" abilita il segnale acustico emesso ogni ora; con `oclock` l'opzione "`-transparent`" rende trasparente lo sfondo dell'orologio, provocando un divertente effetto visivo.

La calcolatrice, sempre presente sotto X, può essere richiamata con il comando `xcalc`; se viene richiamata con l'opzione "`-rpn`" viene abilitata la "notazione polacca inversa" (*reverse polish note*), tipica delle calcolatrici tascabili HP.

Alcuni programmini divertenti, ma completamente inutili, sono `xeyes`, che visualizza due occhi che seguono con lo sguardo i movimenti del puntatore del mouse, `xlogo`, che visualizza il logo di X Window ed `xmelt` che provoca uno spettacolare quanto innocuo effetto sullo schermo del terminale: se è presente sul vostro sistema potrà esservi utile... per stupire gli amici!

X Window è dotato di una funzionalità di *screen saver* che annerisce completamente lo schermo del terminale dopo diversi minuti di inattività dell'utente; tuttavia può spesso essere utile attivare immediatamente uno screen saver, magari bloccandolo con la password, quando ci allontaniamo per qualche minuto dal nostro posto di lavoro. Il comando `xlock` abilita questa funzionalità visualizzando sullo schermo coloratissimi "effetti speciali"; se viene specificata l'opzione "`-nolock`" non sarà necessario digitare la password dell'utente per sbloccare il terminale (basterà un semplice click del mouse). Con l'opzione "`-mode`" è possibile scegliere l'animazione da visualizzare che altrimenti viene selezionata a caso tra quelle disponibili (hop, life, qix, image, swarm, rotor, pyro, flame, worm, random). Ad esempio si può attivare lo screen saver digitando "`xlock -nolock -mode flame`".

Oltre a queste divertenti utility, di solito ne vengono fornite altre un po' più sofisticate ed utili, come ad esempio gli editor `xedit` e `textedit`. Il primo è assai semplice e sfrutta la funzionalità di "copy & paste" offerta da X Window. È possibile scorrere il testo del file caricato mediante la scoll bar ed è possibile posizionare il cursore nella finestra anche mediante il mouse; tuttavia è un editor molto primitivo in cui mancano numerose funzionalità fondamentali. `textedit` è invece l'editor di sistema dell'ambiente Open Look. È un programma piuttosto comodo e sofisticato, meno potente di `emacs` e `vi`, ma pur sempre un utile strumento di lavoro. Dispone delle classiche funzionalità di *cut, copy & paste*, e di ricerca e sostituzione di stringhe di caratteri all'interno del testo; tutte queste funzionalità sono accessibili mediante i menù a tendina ed alcune *dialog box*.

4.4 Alcune applicazioni grafiche

In questa sezione diamo un rapido sguardo ad alcune applicazioni molto diffuse ed assai utili, che necessitano dell'ambiente grafico X Window per poter essere eseguite. Iniziamo con **`ghostview`**, già citato nel paragrafo 2.4, che serve per visualizzare e stampare files in formato PostScript (il cui nome, in genere, termina con "`.ps`"). Il programma è guidato mediante dei menù selezionabili con il mouse e permette sostanzialmente di scorrere le pagine del testo visualizzandole in una finestra grafica, selezionarle tutte o solo una parte, stampare l'intero documento o solo le pagine selezionate. Il formato PostScript riveste una particolare importanza in ambiente Unix perché è un sofisticato formato standard utilizzabile su piattaforme hardware/software completamente diverse, ma sufficientemente potente tanto da includere diversi stili del testo e figure in formato grafico. Tramite la rete Internet si possono reperire facilmente numerosi documenti e manuali tecnici realizzati in PostScript.

Oltre a questo formato, ne esiste anche un altro, denominato *DVI* (Device Independent), che garantisce la "portabilità" di documenti dotati di una formattazione ricca e complessa su piattaforme diverse. È questo il formato in cui vengono prodotti i documenti scritti e compilati con il programma TEX, un sofisticato strumento di composizione tipografica molto usato in ambito scientifico ed universitario. Per visualizzare i files DVI è spesso utile il programma **`xdvi`**, che però, al contrario di **`ghostview`**, non permette di stampare il file, ma soltanto di mostrarlo a video e di scorrerne le pagine. Se non è stato predisposto un apposito filtro di stampa per convertire automaticamente i files DVI in un formato adatto alla nostra stampante, con il programma **`dvips`** potremo convertire il file in formato DVI in un file equivalente in formato PostScript. Il programma **`xtex`**, utile soprattutto a chi utilizza il TEX per scrivere i propri documenti, al pari di **`xdvi`** consente di visualizzare i files DVI, ma in più gestisce anche la stampa.

Oltre al PostScript ed al DVI, orientati per lo più al testo, esistono numerosissimi formati standard per la memorizzazione di immagini grafiche di qualità fotografica. I più diffusi sono il *GIF*, il *BMP* ed il *JPEG*. Per visualizzare in una finestra del nostro terminale grafico un'immagine codificata in uno di questi formati, su molti sistemi è disponibile il programma **`xv`**. Mediante le numerose funzioni offerte dal pannello di controllo attivabile clickando col pulsante destro del mouse sulla finestra principale del programma, è possibile anche modificare e rielaborare l'immagine caricata, per poi salvarla, magari in un altro formato. Una interessante funzione offerta da **`xv`** è quella che permette di "catturare" il contenuto di un'altra finestra visualizzata sul desktop per poi salvarla o rielaborarla mediante **`xv`**.

L'ultima applicazione descritta in questa brevissima panoramica è **`gnuplot`**, un famoso programma per la visualizzazione di grafici di funzioni in una o due variabili (grafici di curve o di superfici) e per la rappresentazione grafica (mediante interpolazione lineare) di dati provenienti da elaborazioni esterne (ad esempio una collezione di dati raccolti per via sperimentale). **`gnuplot`** opera su due finestre distinte: quella in cui è possibile impartire i comandi, che è la finestra **`xterm`** da cui si è lanciata l'applicazione (senza "**`&`**" commerciale, quindi) e quella in cui avviene la visualizzazione dei grafici. Sono numerosi i comandi che è possibile impartire da tastiera all'interno di **`gnuplot`**, riporto brevemente solo i principali, rimandando come al solito per una descrizione più approfondita alla documentazione e all'help in linea che è possibile richiamare con il comando **`help`**.

`plot [`x_1`:`x_2`]` $f_1(x), f_2(x), \ldots$ Per visualizzare il grafico delle funzioni $f_1(x), f_2(x), \ldots$ nell'intervallo $[x_1, x_2]$;

`plot "`*file*`"` Per visualizzare il grafico per punti, prendendo i valori delle ordinate da un file di dati; con il comando "`plot "`*file*`" with lines`" si rappresenta lo stesso grafico, ma i punti vengono anche collegati mediante dei segmenti;

`splot` `[`x_1`:`x_2`]` `[`y_1`:`y_2`]` $f(x, y)$ Per visualizzare il grafico della superficie rappresentata dalla funzione di due variabili $f(x, y)$ nel rettangolo $[x_1, x_2] \times [y_1, y_2]$;

`hidden3d` Per visualizzare il grafico delle superfici in modo non trasparente, ma "solido", nascondendo le linee nascoste;

`nohidden3d` Per visualizzare il grafico delle superfici in modo trasparente, visualizzando anche le linee nascoste.

`quit` Per uscire da `gnuplot` e tornare alla shell.

Capitolo 5

Alcuni strumenti per l'uso della rete Internet

Spesso i sistemi Unix vengono utilizzati per gestire la connessione dell'ufficio o del centro di calcolo in cui sono inseriti, con la rete Internet. In effetti un sistema Unix si integra alla perfezione in un ambiente di connettività vasta ed eterogenea come quello di Internet, dove devono comunicare tra loro, in modo del tutto trasparente agli utenti, macchine e sistemi operativi anche molto diversi.

5.1 La Rete delle reti

Originariamente Internet nasce con il nome di *ARPA-net*, un progetto del Ministero della Difesa americano, che doveva definire un modello di rete di telecomunicazioni inattaccabile dal nemico. Siamo in periodo di piena guerra fredda, l'esercito americano è ossessionato dal pensiero di come ci si possa difendere da un eventuale attacco nucleare sovietico: il compito della rete ARPA-net è quello di collegare i principali centri di comando degli Stati Uniti, in modo tale che se uno di questi centri viene colpito, la rete non venga distrutta, ma possa comunque continuare a funzionare (trasmettere messaggi).

Inizialmente, quindi, esistevano solo pochi nodi sperimentali di questa rete militare, ma ben presto il progetto, che rivestiva anche un grande interesse dal punto di vista teorico, si è allargato a numerose università e centri di ricerca degli Stati Uniti, così che dai pochi nodi iniziali si è passati rapidamente a qualche centinaio di nodi. Col passare degli anni il progetto ha perso il suo significato militare e strategico (nel frattempo è finita anche la guerra fredda) ed ha acquistato un grandissimo interesse pratico. Oggi i nodi collegati alla rete Internet sono milioni e sono sparsi in tutto il mondo.

Non esiste nessun ente, né in Italia, né in nessun altro Paese, che gestisce Internet: la Rete è costituita da una miriade di reti di calcolatori sparse in tutto il mondo; ognuna di queste reti ha una o più macchine collegate con l'esterno (*gateway*) che permette alle macchine della rete locale di essere anche in rete Internet. Ogni università o azienda che decide di collegare le proprie macchine alla Rete, sostiene le spese della propria connessione fisica con il nodo più vicino ed autogestisce la piccola porzione di Internet che risiede sulle proprie macchine. Esistono però dei consorzi che volontariamente si fanno carico della regolamentazione dell'uso della rete; è a questi enti che bisogna fare capo perché vengano assegnati i nomi ai nodi della rete per fare in modo che due macchine qualisiasi connesse ad Internet possano comunicare fra loro (in Italia questo ente è il *GARR*: Gruppo di Armonizzazione delle Reti per la Ricerca).

5.2 IP address e routing

Ogni singola macchina su Internet ha un indirizzo univoco, chiamato *Internet number* o anche *IP address*; è rappresentato come una quaterna di numeri compresi tra 0 e 255. Ad esempio un numero valido può essere 147.31.254.2. I primi tre numeri rappresentano la rete locale di calcolatori a cui tale macchina è collegata, l'ultimo numero rappresenta invece l'indirizzo della macchina all'interno della sotto-struttura locale. Esistono poi dei meccanismi per associare a queste serie di numeri delle stringhe mnemoniche più semplici da ricordare e da utilizzare. Ad esempio il Dipartimento di Matematica dell'Università di Roma 1 ha il seguente *domain name*: "mat.uniroma1.it"; una delle macchine del dipartimento si chiama "venere", e quindi l'indirizzo Internet (IP address) di quella macchina è "venere.mat.uniroma1.it".

Per tentare di decifrare il nome dell'organizzazione che "si cela" dietro ad un certo indirizzo Internet, dovremo leggerlo da destra verso sinistra: in questo modo viene letto per primo il dominio più grande, per poi specificare via, via i vari sottodomini a cui tale macchina appartiene, come in un gioco di scatole cinesi. In particolare il primo elemento partendo da destra è un identificativo standard del dominio principale: ad esempio "it" significa Italia, come pure "uk" sta per United Kingdom, "au" per Australia, "ca" per Canada, "jp" per Giappone e "fr" per Francia. Negli Stati Uniti non viene specificato il nome della nazione, ma il tipo di organizzazione corrispondente a quel dominio Internet: "com" per le organizzazioni commerciali, "edu" per le università e le scuole, "gov" per gli enti governativi, "mil" per gli enti militari, "net" per quei consorzi che si occupano esclusivamente della gestione della rete, "org" per altre organizzazioni private.

La rete Internet basa il proprio funzionamento su un *protocollo software* chiamato *TCP/IP* (Transfer Control Protocol - Internet Protocol): ogni macchina collegata in Internet utilizza tale protocollo per comunicare con le altre. L'attività principale di una rete di calcolatori è lo scambio di messaggi tra le macchine; in una rete grande come Internet, che tra l'altro si sviluppa di ora in ora con un ritmo incredibile, è impossibile fare in modo che ogni macchina della rete conosca l'indirizzo di tutte le altre. Per ovviare a questo fatto il protocollo TCP/IP effettua una procedura, detta *routing*, che permette comunque di instradare un certo messaggio, se non direttamente alla macchina di destinazione, almeno ad un'altra macchina che possa farglielo arrivare. Se ad esempio la macchina A deve mandare un messaggio alla macchina B, che si trova dall'altra parte del mondo, non conoscendo esattamente il percorso che tale messaggio deve seguire sulla rete, lo invierà ad un altro sistema che probabilmente è in grado di "risolvere" l'indirizzo della macchina di destinazione, cioè è in grado di inviare direttamente il messaggio al destinatario, ovvero di inviarlo a sua volta ad un altro *router* più "informato". Questo meccanismo, che apparentemente può sembrare molto instabile ed inefficiente, è in realtà la chiave di volta del protocollo TCP/IP che permette di mantenere attiva la connessione tra due macchine, anche se magari alcuni dei nodi intermedi della rete non sono raggiungibili (teniamo sempre a mente lo scopo originario del progetto ARPA-net).

5.3 La posta elettronica

È sicuramente il più utilizzato tra i servizi offerti da Internet. Per un utente di un sistema Unix sono pochi i concetti nuovi da apprendere, perché per il resto, come abbiamo visto nel paragrafo 2.6, il servizio di posta elettronica è possibile utilizzarlo all'interno del proprio sistema, anche a prescindere dal collegamento in rete.

Ogni utente di una macchina collegata ad Internet ha un proprio indirizzo di posta elettronica, univoco, mediante cui è possibile essere raggiunti da ogni angolo della rete. L'indirizzo è costituito da due parti: *username@domain*. Lo *username* è il nome identificativo dell'utente sulla propria macchina (per gli utenti di un sistema Unix è il proprio username). Il *domain name* è l'*IP address* della macchina su cui opera l'utente. Ad esempio il mio indirizzo di posta elettronica al Dipartimento di Matematica è "liverani@venere.mat.uniroma1.it" perché "liverani" è il mio username su quella macchina e "venere.mat.uniroma1.it" è il domain name della macchina di cui sono utente. È ammesso anche l'indirizzo "liverani@mat.uniroma1.it" (anche se non viene specificato "venere", il nome di una particolare macchina del dipartimento) perché comunque l'ente di appartenenza è ben individuato (mat.uniroma1.it), quello che avviene –per ragioni tecniche o amministrative– all'interno di tale dominio (quindi lo specificare o meno il nome della singola macchina all'interno di quel dipartimento), non interessa l'esterno della rete e viene gestito internamente (la macchina che svolge il compito di *gateway* verso Internet sa che l'utente "liverani" ha un *account* sulla macchina chiamata "venere").

Usando il nostro indirizzo, e conoscendo quello del destinatario, possiamo ricevere e spedire messaggi in tutto il mondo; non è necessario essere fisicamente di fronte al computer per poter ricevere il messaggio; se non siamo presenti al momento dell'arrivo, questo sarà memorizzato nella nostra *mailbox* in modo da poterlo leggere al successivo collegamento.

Non mi dilungo ulteriormente nell'illustrazione del sistema di posta elettronica, perché, come ho già detto, è sostanzialmente lo stesso che usiamo per scambiare messaggi con gli utenti del nostro sistema e che abbiamo già visto nel paragrafo 2.6. Mi limito ad aggiungere che il parallelo tra ciò che è possibile fare per comunicare con gli utenti del nostro sistema e ciò che è possibile fare per comunicare con gli utenti dell'intera rete Internet, non si ferma alla sola posta elettronica. Ad esempio il comando `finger` può essere usato per ottenere informazioni su un particolare utente della Rete o sugli utenti collegati ad una certa macchina. Con il comando "`finger liverani@venere.mat.uniroma1.it`" si visualizzano le informazioni sul mio account al Dipartimento di Matematica, come pure con il comando "`finger @venere.mat.uniroma1.it`" si visualizza l'elenco degli utenti collegati in quel momento su quella macchina. Lo stesso vale per il comando "`talk`" che può essere usato per comunicare in tempo reale con utenti distanti da noi anche migliaia di chilometri. Anche per quanto riguarda questi comandi è opportuno riferirsi a quanto riportato nel paragrafo 2.6.

È opportuno tenere sempre presente però che non tutte le macchine connesse ad Internet sono delle macchine Unix; questo comporta il fatto che non tutte prevedono la possibilità di usufruire di tutti i servizi che invece offre un sistema Unix; ad esempio il `finger` o il `talk` spesso non sono previsti o sono disabilitati per motivi di sicurezza su alcuni sistemi. Un'altra osservazione importante è la seguente. Internet non è di proprietà di nessuno, è una struttura piuttosto "anarchica" e priva di un ente superiore di gestione e supervisione. Tuttavia, proprio al fine di mantenere questa grande libertà, è una struttura dotata di grandissimo autocontrollo esercitato dagli utenti stessi che mal tollerano un uso scorretto o un abuso della Rete stessa, isolando ed inibendo le proprie risorse a coloro che persistono in un abuso della Rete o in un uso al di fuori delle regole non scritte che la comunità degli uenti si è data.

Il meccanismo che gestisce la rete si è rivelato piuttosto stabile, ma non certo infallibile. Il fatto stesso che i messaggi che viaggiano sulla rete passino per numerosi nodi intermedi prima di arrivare a destinazione, rende il tutto potenzialmente soggetto ad errori di trasmissione o a violazione della privacy. Non stupiamoci più di tanto se un nostro messaggio inviato ad una certa ora impiega talvolta pochi secondi ed altre volte diverse ore (o giorni) per giungere a destinazione; non stupiamoci neanche se talvolta capita che un messaggio si perda nel mare delle connessioni Internet. Il consiglio è quindi di usare al meglio la Rete, sfruttandone al meglio le risorse, nei limiti imposti dalla *netiquette* (il galateo non scritto che gli utenti di Internet devono osservare), ma senza farle assumere un ruolo determinante per il nostro lavoro: mai affidare ad un messaggio di *e-mail* una comunicazione determinante per la nostra vita od una informazione estremamente riservata (come il numero della carta di credito).

5.4 Le News Usenet

La posta elettronica è un sistema privato di comunicazione tra utenti. Teoricamente nessuno, oltre al mittente ed al destinatario del messaggio, può leggere il contenuto del messaggio stesso. Spesso può essere utile far sì che un proprio messaggio su un determinato tema venga condiviso con molti altri utenti della Rete, anche sconosciuti. A questo scopo, su Internet sono stati creati dei gruppi di discussione (*newsgroup*) a tema, che raccolgono ognuno gli articoli dei propri lettori su un determinato argomento. Chiunque può inviare un articolo ad un newsgroup che verrà letto da tutti gli utenti che seguono (leggono periodicamente) i nuovi messaggi di quel gruppo.

Oggi i newsgroup attivi sono più di 4.000, ma questo numero è destinato a crescere di giorno in giorno, vista la frequenza con cui vengono creati nuovi gruppi di discussione. Per orientarsi in questo mare di articoli, sarà bene chiarire che i newsgroup sono organizzati in una struttura ad albero in cui le varie aree (i rami dell'albero) sono chiamate *gerarchie*. Ci sono dieci gerarchie principali:

`alt` È la gerarchia "alternativa", in cui si ritrovano i newsgroup più impensabili ed in cui ogni argomento è trattato in modo assai particolare;

`biz` Gruppi di tipo "business";

`comp` È la gerarchia dei newsgroup riguardanti argomenti tecnici di Computer Science;

`gnu` Gruppi concernenti il software e le attività dello GNU Project e della Free Software Foundation;

`news` Gruppi riguardanti la gestione della rete delle news;

`rec` Gruppi orientati agli hobby e ad argomenti di tipo ricreativo;

`sci` Gerarchia dei gruppi riguardanti le scienze ufficiali;

`soc` Newsgroup riguardanti aspetti sociali e culturali delle diverse nazioni del mondo;

`talk` Gruppi orientati alla creazione di dibattito (una sorta di "talk" in differita) tra gli utenti.

Per gli utenti di lingua italiana è utile indicare l'esistenza dei gruppi della gerarchia "`it`" e del newsgroup "`soc.culture.italian`".

Alcuni gruppi sono "moderati", ossia esiste un utente (o un gruppo di utenti) che svolgono la funzione di moderatore della discussione e vagliano a priori l'inserimento o la cancellazione dei messaggi nel newsgroup.

Andremmo ben oltre le finalità di questa breve introduzione descrivendo le numerose "buone maniere" che è necessario adottare per non attirare su di noi le maledizioni degli altri utenti; ci limitiamo quindi a ricordare che non sempre sono gradite "firme" eccessivamente lunghe o "artistiche" alla fine dei messaggi e che è bene evitare di entrare in sterili polemiche (*flames*) con gli altri utenti del newsgroup o di intraprendere inutili guerre di religione (*holy war*) a favore di questo o quell'argomento. In poche parole è bene discutere pacatamente ed in un certo senso anche con umiltà, visto che i nostri interlocutori sono migliaia e sparsi in tutto il mondo (è difficile pensare di avere sempre ragione o di essere il migliore in una situazione di questo genere...).

Per accedere in lettura ed in scrittura ai newsgroup delle News Usenet, si deve utilizzare un *news reader*, come i programmi `tin` o `rtin`. Entrambi questi programmi permettono di selezionare (evidenziandoli con il cursore all'interno di una lista) il gruppo in cui "entrare", e all'interno del gruppo, i nuovi messaggi che non abbiamo ancora letto. Con il news reader è possibile, come su un normale programma di post elettronica, leggere i nuovi messaggi, ed inviare al newsgroup le nostre risposte o i nostri articoli originali. La gestione del newsgroup avviene su una macchina remota (*news server*) a cui il newsreader si collega mediante il protocollo *NNTP* (Network News Transport Protocol). Ulteriori dettagli sulle modalità operative del programma utilizzato per leggere le news è possibile reperirle sulla documentazione del proprio sistema.

5.5 Connessione a sistemi remoti: Telnet

Il comando `telnet` consente di trasformare il nostro terminale nel terminale di una macchina remota connessa alla rete Internet. Tutto ciò che noi faremo in una sessione *telnet*, non verrà elaborato sulla nostra macchina locale, ma impegnerà le risorse (CPU, memoria, dischi, ecc.) della macchina remota.

Ad esempio da *MC-Link* o *Agorà* (due sistemi telematici commerciali italiani) posso accedere mediante il `telnet` alla macchina del Dipartimento di Matematica semplicemente impostando il comando "`telnet venere.mat.uniroma1.it`". Naturalmente devo avere un *account* sulla macchina a cui mi collego, altrimenti mi verrà rifiutata la connessione; vediamo un esempio:

```
$ telnet venere.mat.uniroma1.it
Trying 141.108.5.85...
Connected to venere.mat.uniroma1.it.
Escape character is '^]'.

SunOS UNIX (venere)
login: liverani
Password:

Last login: Wed May 3 11:22:11 from ax433.mclink.it
SunOS Release 4.1.3_U1 (GENERIC) #1: Wed Oct 13 17:48:35 PDT 1993
Venere (Sun4/60 del Dipartimento di Matematica)
```

```
IP 141.108.5.85
TERM = (vt100)

$ _
```

I messaggi che il sistema ci presenta a video sono gli stessi di quando ci connettiamo al sistema stesso da un terminale direttamente collegato ad esso. Alla fine il sistema ci presenta il prompt da cui, come al solito, potremo impartire i comandi necessari, che però, invece di essere seguiti sulla nostra macchina, saranno eseguiti sul sistema remoto. Per sconnettersi dal sistema si userà il consueto comando `exit` o `logout`.

Il messaggio "Escape character is..." indica che, in questo esempio, la sequenza di tasti CTRL-] ci permette di rientrare nel sistema locale per chiudere la connessione, sospenderla, ecc. Premendo CTRL-] ci viene presentato il prompt "`telnet>`" del programma `telnet`; ad esempio per chiudere la sessione a questo punto possiamo usare il comando `close`. È opportuno riferirsi alla documentazione del proprio sistema per maggiori informazioni sui comandi specifici che possono essere utilizzati.

5.6 Scambio di files con sistemi remoti: FTP

FTP significa *File Transfer Protocol* e serve appunto per trasferire file da una macchina all'altra attraverso la rete Internet. Anche in questo caso, connettendosi con un computer remoto per prelevare o depositare files mediante FTP, viene richiesto all'utente di farsi riconoscere mediante il suo username e la sua password. Spesso però è utile attingere alle risorse di un archivio pubblico residente su un sistema su cui non abbiamo un account; per questo motivo, in alcuni casi, è ammesso l'uso dello username `anonymous` che deve essere usato in unione alla password costituita dal nostro indirizzo di posta elettronica. In questo modo possiamo accedere ugualmente ad un certo server FTP, prelevando o depositando files in un'area pubblica in cui chiunque può avere accesso. È questo uno dei principali canali di scambio di informazioni, guide, manuali e software di *pubblico dominio*, che arricchisce e rende per certi versi unica la comunità degli utenti Internet.

All'interno di una sessione FTP si possono fare poche cose: sostanzialmente spostarsi all'interno del filesystem della macchina remota, prelevare o depositare files. Per far questo si deve utilizzare un ristretto set di comandi; riportiamo di seguito solo i principali, rimandando alla documentazione del proprio sistema per una illustrazione più chiara ed esauriente di tutti gli altri.

`cd` ha lo stesso uso e significato dell'omonimo comando della shell Unix: serve per cambiare la directory corrente;

`pwd` serve per visualizzare il nome della directory corrente (*print work directory*);

`get` serve per trasferire un file dal sistema remoto al nostro computer; la sintassi è "get *fileremoto filelocale*", dove *fileremoto* indica il nome del file che si trova sul server FTP remoto, e *filelocale* (che può anche essere omesso) indica il nome da assegnare al file quando sarà memorizzato nel disco del nostro sistema locale;

`put` svolge la funzione inversa del comando `get`, trasferendo sul sito remoto un file residente sul disco del nostro sistema locale; la sintassi è "put *filelocale fileremoto*";

`dir` elenca i files contenuti nella directory corrente.

Un esempio di una sessione FTP è il seguente, che riporta un collegamento con il server FTP "nic.switch.ch", uno dei più grandi siti europei:

```
$ ftp nic.switch.ch
Connected to nic.switch.ch
220- Hello user at venere.mat.uniroma1.it [141.108.5.85]
220-
220- Welcome to the SWITHinfo ftp archive.
220-
220- ***  Access to this FTP service is exclusively allowed for  ***
220- ***   -Swiss universities, schools & organisations with a   ***
220- ***    SWITCH service contract                              ***
220- ***   -foreign education & research organisation            ***
220-
...
220- nic.switch.ch FTP server (Version 4.1259 switchinfo@switch.ch) ready.
220- There are 26 (max 50) archive users in your class at the moment.
220- Local time is Wed May 3 17:12:39 MET
220
Name (nic.switch.ch:anonymous): anonymous
331 Guest login ok, give your e-mail address (user@domain) as password.
Password: liverani@mat.uniroma1.it
230- Guest 'liverani@mat.uniroma1.it' login ok
...
Remote system type is UNIX
Using binary mode to transfer files.
ftp> cd mirror/os2
250 CWD command successful.
ftp> dir
200 PORT command successful.
150 opening ASCII mode data connection for .
-rw-rw-r--  1 24     16     906 Mar 31 06:35 .message
-rw-rw-r--  1 488    16  235980 Jul 23 06:20 00index.txt
-rw-rw-r--  1 488    16    5742 Jul 22 19:27 0readme
drwxrwxr-x 11 24     16     512 Jul 24 01:10 1_x
...
-rw-rw-r--  1 24     16     377 Mar 31 06:32 cdrom.txt
...
226 Transfer complete.
ftp> get cdrom.txt
200 PORT command successful.
150 Opening BINARY mode data connection for /mirror/os2/cdrom.txt
226 Transfer complete.
377 bytes received in 0.3 seconds (1.2 Kbytes/sec)
ftp> quit
221 Goodbye.
$ _
```

Nell'esempio ci siamo collegati con il sistema entrando come utenti "anonimi", siamo entrati nella directory "`/mirror/os2`", contenente files e programmi riguardanti il sistema operativo OS/2 ed abbiamo prelevato il file `cdrom.txt`. Come al solito sperimentare questi comandi di persona aiuterà a capire il funzionamento di FTP molto più di qualsiasi altra spiegazione.

5.7 Uno strumento per la ricerca delle informazioni: Gopher

Il *gopherspace* è un insieme di basi di dati disseminate in giro per il mondo, ed interconnesse tra loro grazie ad Internet; si accede alle informazioni contenute in queste banche dati mediante il programma `gopher` che, con dei semplici menù, permette di "navigare" quasi senza accorgercene nel mare del gopherspace, saltando da un *gopher server* all'altro all'inseguimento dell'informazione cercata. Tramite `gopher` è possibile accedere a testi, immagini, programmi, ma anche stabilire connessioni in FTP o in Telnet con server remoti.

Tipicamente dopo aver lanciato il programma `gopher` ci si ritrova di fronte ad un menù simile a quello riportato in questo esempio:

```
            Internet Gopher Information Client v1.03

            Root gopher server: gopher.micro.umn.edu

-->  1.  Information About Gopher/
     2.  Computer Information/
     3.  Discussion Groups/
     4.  Fun & Games/
     5.  Internet file server (ftp) sites/
     6.  Libraries/
     7.  News/
     8.  Other Gopher and Information Servers/
     9.  Phone Books/
     10. Search lots of places at the U of M <?>
     11. University of Minnesota Campus Information/

Press ?  for Help, q to Quit, u to go up a menu        Page 1/1
```

Utilizzando i tasti di spostamento del cursore possiamo muovere la freccia "-->" sulle diverse voci del menù; per selezionare la voce puntata dalla freccia è sufficiente battere il tasto Return. In questo modo si accede ad un altro menù o all'informazione finale cercata (vedremo tra poco come capire in anticipo cosa si "nasconde" dietro una certa voce di menù). Il gopherspace, dal punto di vista dell'utente, è infatti strutturato ad albero: si parte dal menù iniziale del proprio *gopher server* e si scende sempre più in profondità selezionando un menù dopo l'altro. Per risalire di un livello, per tornare indietro al menù precedente, si deve battere il tasto u (*up*).

Alla fine di ogni voce di menù viene visualizzato un simbolo per indicare che tipo di *item* si "nasconde" dietro a quella voce; riportiamo alcuni esempi di questa simbologia:

. l'*item* è un file di testo;

/ accesso ad un altro menù;

`<TEL>` attivazione di una sessione Telnet;

`<FTP>` attivazione di una sessione FTP;

`<?>` ricerca su un indice;

`<BIN>` l'*item* è un file binario;

Quando si seleziona una voce (*item*) del menù ciò che si ottiene dipende dal tipo di *item* che è stato selezionato. Se è un file di testo, questo viene semplicemente presentato a video, in modo da poterne leggere il contenuto; al termine della lettura possiamo anche decidere di trasferirlo sul nostro sistema (memorizzandolo in un file) o farcelo inviare per posta elettronica dal *gopher server* al nostro indirizzo. Se l'item selezionato è un altro menù, questo viene visualizzato sullo schermo; è bene capire che accedere ad un menù di gopher può anche voler dire connettersi con un sistema lontano migliaia di chilometri dal nostro, anche se questo fatto è del tutto invisibile a noi utenti. Quando l'*item* è di tipo "`<TEL>`" o "`<FTP>`", selezionandolo si inizia una sessione Telnet o FTP, rispettivamente, mentre se l'*item* è di tipo "`<BIN>`", l'effetto della selezione è il trasferimento del file binario sulla nostra macchina. Infine gli *item* di tipo "`<?>`" costituiscono uno dei punti di forza di `gopher` e consentono di effettuare delle ricerche su degli indici di argomenti, per accedere direttamente ad una certa informazione, senza percorrere un lungo cammino nel gopherspace.

5.8 Navigazione nel World Wide Web

Per certi aspetti il *WWW* (*World Wide Web*) è un'evoluzione del gopherspace. È il risultato di studi condotti negli ultimi anni da alcuni ricercatori del Cern, che con questo nuovo potente strumento hanno reso accessibile a milioni di nuovi utenti "non tecnici" l'uso della rete Internet.

WWW si basa fondamentalmente su *HTML* e *HTTP*. Il primo (*Hyper Text Markup Language*) è un sistema convenzionale di marcatura di un testo per dotarlo di collegamenti ipertestuali e di link con altri documenti sparsi sulla rete. Il secondo (*Hyper Text Transfer Protocol*) è il protocollo che permette di sfruttare i link contenuti nei documenti HTML, per poterli collegare ad altri documenti (che chiameremo *pagine HTML*) residenti su altri host su Internet. Per utilizzare questi potenti strumenti, sono disponibili vari software, tra i quali è bene ricordare *Mosaic*, *Netscape* e *Lynx*. Il primo è il capostipite di questo tipo di applicazione, il secondo è il più potente prodotto di questo tipo, una sorta di campione di riferimento, mentre l'ultimo differisce dai primi due perché gira su un terminale alfanumerico, senza fare uso della grafica.

Lanciando il programma `netscape` (o anche `xmosaic`) da un X Terminal, ci troveremo di fronte ad un software grafico, che mediante il mouse, ci permetterà di selezionare i collegamenti ipertestuali presenti nella pagina visualizzata, insieme ad immagini e riferimenti a sequenze video o audio. Quando avremo acquisito una certa pratica nel maneggiare questi programmi, potremo eseguire delle ricerche tematiche ed individuare in pochi minuti moli notevoli di dati ed informazioni sparse un po' ovunque nel mondo.

Spiegare in dettaglio il funzionamento di un software complesso come `netscape` è un arduo compito che esula dall'obiettivo di questa breve introduzione. Ci limiteremo a dire che le pagine WWW sono individuate da una *URL* (*Universal Resource Location*) che è una espressione del tipo "`http://`*domain*"; ad esempio la *home page* di MC-Link (un buon punto per iniziare la navigazione del Web) è "`http://www.mclink.it`"; un'altra ottima home-page in lingua italiana, ideale, ad esempio, per avere collegamenti di vario tipo con le pagine di numerosi quotidiani, è la pagina della *Città Invisibile*, un'associazione senza scopo di lucro per la diffusione di un uso democratico della rete Internet: la URL è "`http://www.citinv.it`".

Il WWW ha avuto il grande merito di rendere possibile la vera esplosione della rete Internet che è sotto gli occhi di tutti in questi ultimi mesi. Fino ad ora infatti gli strumenti necessari per accedere ed utilizzare la rete, per la loro complessità e difficoltà d'uso, erano appannaggio solo di pochi tecnici specializzati. Oggi, invece, con software tipo Mosaic o Netscape, chiunque può essere in grado di navigare sulla rete con pochi click del mouse e così, come spesso accade in questi casi, un gran numero di nuovi utenti si sono affacciati sul mondo Internet richiedendo sempre più insistentemente un accesso economico ed efficiente alla rete. Questo fenomeno ha fatto sì che si moltiplicassero, anche in Italia, i *fornitori di connettività* Internet, società private che vendono a prezzi sempre più bassi la possibilità di connettersi ad un POP Internet (*Point Of Presence*) via modem o con un collegamento su cavo dedicato, per poter aprire in casa propria o nel proprio ufficio, una finestra verso il resto del mondo telematizzato. I rischi come al solito sono molti; primo fra tutti quello che si crei un nuovo monopolio anche in questo particolare settore delle telecomunicazioni, la qual cosa creerebbe non poche difficoltà alla libera circolazione delle idee, e snaturerebbe in modo irreversibile quella che è stata fino ad oggi la rete; da non sottovalutare anche il rischio che in Italia si diffonda solo la cultura dell'*arrembaggio* alle risorse messe a disposizione da altri, senza invece effettuare quel salto di qualità che ci permetta di diventare noi stessi fornitori di informazioni e non soltanto fruitori di informazioni "pubblicate" da altri.

IBM X Station

Appendice A

Sintesi dei comandi principali

`bash` Bourne-Again Shell è un interprete di comandi compatibile con `sh` che esegue i comandi letti da tastiera o da un file. `bash` incorpora anche alcune delle caratteristiche della Korn shell e della C shell.

`bg` Continua in background l'esecuzione di un processo precedentemente interrotto battendo Ctrl-z. come argomento del comando `bg` può essere specificato il *job number* (preceduto dal simbolo "`%`") o il *process ID* del processo da mandare in background. Ad esempio: "`bg %3`".

`bye` Termina l'esecuzione della shell di login chiudendo anche la sessione di lavoro dell'utente (logout).

`cal` Visualizza un calendario. Senza alcun parametro visualizza il calendario del mese corrente; si può anche specificare un particolare mese ed anno. Ad esempio: "`cal 7 1492`".

`cat` Concatena i files specificati come argomento del comando e li invia allo *standard output*. Ad esempio: "`cat file1 file2 file3`".

`chmod` Cambia i permessi di accesso ai file. Ad esempio: "`chmod 644 pippo`".

`clock` In ambiente grafico X Window visualizza una finestra con un orologio; `clock` è una utility dell'ambiente OpenWindows.

`compress` Compatta e scompatta un file. I file compressi con `compress` hanno estensione "`.Z`". Con il comando "`compress pippo`" si genera il file compattato `pippo.Z`; per scompattarlo, ottenendo il file originale si usi il comando "`compress -d pippo.Z`".

`cp` Copia i file. Il formato del comando è "`cp` [*opzioni*] *sorgente destinazione*", dove *sorgente* è il nome del (dei) file da copiare, mentre *destinazione* è il nome del file o della directory in cui copiare il (i) file *sorgente*. Ad esempio: "`cp pippo.* ./src`".

`csh` C Shell. Come `bash` e `ksh` è un interprete di comandi compatibile con `sh` che esegue i comandi letti da *standard input* o da un file.

`date` Visualizza la data e l'ora corrente.

`df` Visualizza lo spazio ancora libero nel filesystem.

`du` Visualizza l'occupazione (in Kbyte) della directory corrente e delle sue sottodirectory.

`dvips` Converte un documento dal formato DVI al PostScript. Ad esempio: "`dvips tesi.dvi -o tesi.ps`".

`elm` Programma interattivo di gestione della posta elettronica.

`emacs` Editor per file di testo.

`exit` Termina l'esecuzione della shell. Se la shell è una shell di login, `exit` provoca il logout dal sistema.

`fg` Porta in *foreground* il processo specificato mediante il suo *process ID* o il suo *job number* (preceduto dal simbolo "`%`"). Ad esempio: "`fg 143`".

`finger` Programma per la visualizzazione di informazioni sugli utenti. Ad esempio: "`finger liverani`".

`free` Visualizza il totale della memoria libera ed utilizzata presente sul sistema.

`ftp` È il programma che permette di fruttare il File Transfer Protocol che consente all'utente di trasferire files sulla rete dalla propria macchina ad un sito remoto e viceversa. Ad esempio: "`ftp ftp.cnr.it`".

`ghostview` Programma per la visualizzazione in una finestra grafica sotto X Window, di un documento PostScript utilizzando il programma GhostScript (`gs`).

`gnuplot` Programma per la visualizzazione di grafici di funzione in una o due variabili; consente anche di rappresentare in forma grafica dati numerici contenuti in file di testo.

`gopher` Programma a menù per la "navigazione" nel *gopherspace*; consente di effettuare ricerche tematiche su un gran numero di basi di dati sparse su Internet.

`gs` È il comando con cui si richiama il programma GhostScript, un interprete PostScript per visualizzare e stampare documenti in quel formato. Spesso è più comodo utilizzare GhostScript tramite `ghostview`, invece che utilizzare direttamente i comandi di `gs`.

`gunzip` Programma per scompattare files compressi con `gzip`. Ad esempio: "`gunzip pippo.gz`".

`gzip` Compatta e scompatta un file. I file compressi con `gzip` hanno estensione "`.z`" o "`.gz`". Con il comando "`gzip pippo`" si genera il file compattato `pippo.gz`; per scompattarlo, ottenendo il file originale si usi il comando "`gunzip -d pippo.gz`".

`jobs` Visualizza la tabella dei processi attivi nella shell corrente.

`kill` Provoca l'interruzione dell'esecuzione del processo specificato mediante il suo *Process ID* o mediante il *job number* (preceduto dal simbolo "`%`"). Ad esempio: "`kill %2`".

`ksh` Korn Shell. Come `bash` e `csh` è un interprete di comandi compatibile con `sh` che esegue i comandi letti da *standard input* o da un file.

`latex`	Sistema di editoria elettronica. Converte un file di testo scritto in formato LaTeX in un file DVI (device independent). Ad esempio: "`latex tesi.tex`".
`logout`	Termina l'esecuzione della shell di login e conclude la sessione di lavoro dell'utente, scollegandolo dal sistema.
`lpq`	Visualizza la coda di stampa; ad ogni elemento della coda di stampa è associato un *job number* ed un proprietario.
`lpr`	Invia il file specificato alla coda di stampa; tramite il *piping* può anche essere utilizzato come filtro dell'output di un altro programma. Ad esempio: "`lpr pippo.txt`" oppure "`cat pippo.txt | lpr`".
`lprm`	Rimuove dalla coda di stampa un file ancora non stampato; si deve specificare come parametro il *job number* del file da eliminare, ottenuto con il comando `lpq`. Ad esempio: "`lprm 127`".
`ls`	Visualizza i files contenuti nella directory specificata.
`lynx`	Programma ipertestuale per navigare sul WWW; non gestisce la visualizzazione di immagini grafiche, quindi può anche essere utilizzato su un terminale alfanumerico.
`mail`	Programma per la gestione della posta elettronica.
`man`	Visualizza la pagina di manuale del comando specificato; ad esempio: "`man mail`".
`mkdir`	Crea la directory specificata. Ad esempio: "`mkdir tesi`".
`more`	Visualizza il file specificato, introducendo una pausa alla fine di ogni schermata; mediante il *piping* può anche essere usato come filtro dell'output di un altro programma. Ad esempio: "`more pippo.txt`" oppure "`cat pippo.txt | more`".
`mv`	Sposta o rinomina il file specificato. Il formato del comando è "`mv` [*opzioni*] *sorgente destinazione*", dove *sorgente* è il nome del (dei) file da spostare, mentre *destinazione* è il nome del file o della directory in cui spostare il (i) file *sorgente*. Ad esempio: "`mv pippo.* ./src`".
`netscape`	Programma per la navigazione grafica ed ipertestuale sul WWW.
`oclock`	Orologio per X Window.
`pico`	Editor di file di testo.
`pine`	Sistema per la gestione della posta elettronica.
`ps`	Visualizza l'elenco dei processi attivi.
`pwd`	Visualizza il nome della directory corrente.
`rm`	Cancella i file specificati. Ad esempio: "`rm ./src/*`".
`rmdir`	Rimuove la directory specificata, che deve essere vuota (non deve contenere file o subdirectory). Ad esempio: "`rmdir src`".

rtin Programma per la lettura delle news Usenet.

screen Consente di aprire più finestre virtuali su uno stesso terminale alfanumerico.

sh È la shell di riferimento. Come `bash` , `ksh` e `csh` è un interprete di comandi che esegue i comandi letti da *standard input* o da un file.

sort Ordina alfabeticamente i dati contenuti in una lista contenuta in un file o letta dallo *standard input*; mediante il *piping* può essere usato anche come filtro per l'output di un altro comando. Ad esempio: "`sort lista.txt > lista.ord.txt`".

talk Permette di comunicare in modo interattivo con un altro utente. Il formato del comando è il seguente: "`talk` *username* [*ttynn*]", dove *username* è il nome dell'utente ed il parametro opzionale *ttynn* è il nome del terminale a cui l'utente con cui si desidera comunicare è collegato. Ad esempio: "`talk liverani tty2`".

tar Tape Archive. Consente di archiviare in un unico file (che di default viene memorizzato su nastro) più file e directory. Con `tar` è anche possibile effettuare l'operazione inversa, cioè estrarre da un file di archivio (in genere con estensione "`.tar`") i files originali. Ad esempio: "`tar xfv pippo.tar`".

telnet È il programma con cui è possibile sfruttare il protocollo TELNET per utilizzare un sistema remoto collegato in rete (ad esempio su Internet). Ad esempio: "`telnet mclink.mclink.it`".

tex È il comando per eseguire il sistema di editoria elettronica TeX. Converte un file di testo in formato plain TeX in un file in formato DVI (device independent). Ad esempio: "`tex tesi.tex`".

textedit Editor di file di testo in ambiente grafico X Window. È una delle utility di OpenWindows.

tin Programma per la lettura delle news Usenet.

unzip Scompatta il file specificato in formato zip; ad esempio: "`unzip pippo.zip`"; con il comando "`unzip -v pippo.zip`" si visualizza l'elenco dei files archiviati nel file `pippo.zip` in formato compresso.

vi Editor di file di testo.

who Visualizza l'elenco degli utenti collegati al sistema.

whoami Visualizza lo username dell'utente.

write Consente di comunicare con altri utenti collegati al sistema, visualizzando sullo schermo del loro terminale il messaggio digitato. Ad esempio: "`write liverani ttyp1`".

xcalc Calcolatrice in ambiente X Window.

xclock Orologio classico per X Window.

xdvi Programma in ambiente X Window per visualizzare documenti in formato DVI. Ad esempio: "`xdvi tesi.dvi &`".

`xedit` Semplice editor in ambiente X Window.

`xeyes` Gli occhi che seguono il mouse sotto X Window.

`xlock` Screen saver con possibilità di bloccare il terminale sotto X Window; ad esempio "`xlock -nolock -mode pyro`".

`xlogo` Visualizza il logo di X Window in una finestra grafica.

`xman` Permette di sfogliare le pagine di manuale sotto X Window mediante una comoda interfaccia grafica con menù selezionabili con il mouse.

`xmelt` "Squaglia" lo schermo.

`xmosaic` Programma per la navigazione grafica ed ipertestuale sul WWW.

`xsetroot` Imposta alcuni parametri di configurazione dell'ambiente X Window; ad esempio con il comando "`xsetroot -solid cadetblue`" si imposta il colore dello sfondo.

`xterm` Emulazione di terminale alfanumerico sotto X Window.

`xtex` Permette di visualizzare e stampare un documento in formato DVI sotto X Window. Ad esempio: "`xtex tesi.dvi &`".

`xv` Programma per la visualizzazione e l'elaborazione di immagini grafiche in ambiente X Window.

`yes` Visualizza indefinitamente il carattere "`y`".

`zip` Compattatore in formato zip. Permette di archiviare in formato compresso più file e directory in un unico file compattato. La sintassi del comando è la seguente: "`zip` [*opzioni*] *file.zip* file1, file2, ...", dove *file.zip* è il nome del file compresso di destinazione e *file1*, *file2*, ... sono i file da archiviare in *file.zip*. Ad esempio: "`zip src.zip ./src/*`".

Appendice B

Elenco alfabetico delle sigle

ANSI	American National Standard for Information Systems
ASCII	American Standard Code for for Information Interchange
BMP	Bitmap
BSD	Berkeley System Distribution
CPU	Central Processing Unit
DEC	Digital Equipment Corporation
DOS	Disk Operating System
DVI	Device Independet
EFF	Electronic Fronteer Foundation
FSF	Free Software Foundation
FTP	File Transfer Protocol
FVWM	F(?) Virtual Window Manager
GIF	Graphic Interchange Format
GNU	Gnu's Not Unix
GUI	Graphical User Interface
HP	Hewlett Packard
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IBM	International Business Machines
JPEG	Join Photographic Expert Group
MS-DOS	Microsoft Disk Operating System
NNTP	Network News Transport Protocol

OLVWM	Open Look Virtual Window Manager
OLWM	Open Look Window Manager
OS/2	Operating System 2
OSF	Open Software Foundation
PARC	Palo Alto Research Center
PC	Personal Computer
PID	Process ID
POP	Point Of Presence
POP3	Post Office Protocol versione 3
RISC	Reduced Istruction Set Computer
SGI	Silicon Graphics Inc.
SUN	Stanford University Network
SVR4	System V Release 4
TCP/IP	Transfer Control Protocol - Internet Protocol
TWM	Tab Window Manager
URL	Universal Resource Locator
WPS	Work Place Shell
WWW	World Wide Web

Bibliografia

[1] Rachel Morgan, Henry McGilton, *Il sistema operativo Unix System V*, Mc Graw-Hill, Milano, 1988

[2] Grace Todino, John Strang, *Learning the UNIX Operating System*, O'Reilly and Associates, 1987

[3] Peter Norton, Harley Hahn, *Unix*, Mondadori Informatica, Vicenza, 1992

[4] John J. Valley, *La grande guida Unix*, Jackson Libri, Milano, 1993

[5] Brian Fox, *Bash Features*, Free Software Foundation, 1991, (reperibile su Internet[1])

[6] G. Anderson, P. Anderson, *The Unix C shell field guide*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986

[7] Matt Welsh, *Linux Installation and Getting Started*, Agosto 1993 (reperibile su Internet)

[8] Olaf Kirch, *The Linux Network Administrators' Guide*, 1993 (reperibile su Internet)

[9] Mohamed el Lazy, *Editing in a Unix environment*, Prentice-Hall, Englewood Cliffs, New Jersey, 1985

[10] Brian W. Kernighan, Dennis M. Ritchie, *Linguaggio C*, Gruppo Editoriale Jackson, Milano, 1985

[11] M. J. Rochkind, *Advanced Unix programming*, Prentice-Hall, Englewood Cliffs, New Jersey, 1985

[12] Brendan P. Kehoe, *Zen and the Art of the Internet*, Gennaio 1992 (reperibile su Internet)

[13] John R. Levine, Carol Baroudi, *Usare Internet senza fatica*, Mc Graw-Hill, Milano, 1994

[14] Claudio Beccari, *LaTeX Guida a un sistema di editoria elettronica*, Editore Ulrico Hoepli, Milano, 1991

[15] D. E. Knuth, *The TeXbook*, Addison-Wesley Publishing Company, Reading, Mass., 1986

[1] Si intende dire che una copia tel testo in formato PostScript o ASCII può essere reperita presso numerosi siti Internet mediante FTP anonimo.

[16] Leslie Lamport, *LaTeX A Document Preparation System*, Addison-Wesley Publishing Company, Reading, Mass., 1986

[17] D. E. Knuth, *The Metafont book*, Addison-Wesley Publishing Company, Reading, Mass., 1986

Indice analitico