1993, 1994, 1996

Linux Guida dell'Utente

Larry Greenfield

Tutto il necessario per iniziare ad usare Linux, una versione di Unix liberamente distribuibile. Questo manuale descrive i comandi base di Unix e quelli specifici di Linux. Questo manuale è rivolto ai nuovi utenti Linux, anche se può essere utile per consultazione agli utenti più esperti.

permesso creare e distribuire copie testuali di questo manuale lasciando integri l'avviso di copyright e questi permessi su tutte le copie.

permesso copiare e distribuire versioni modificate di questo manuale sotto le stesse condizioni delle copie testuali, fornendo anche le sezioni che riportano la "Licenza Pubblica Generale GNU", la "Licenza Pubblica Generale Librerie GNU". Le altre sezioni che sono chiaramente mantenute sotto altri copyright possono essere riprodotte sotto le condizioni ad esse allegate, e provvedendo che l'intero lavoro risultante sia distribuito sotto i termini di un avviso identico a questo.

permesso copiare e distribuire traduzioni di questo manuale in altre lingue sotto le condizioni per le versioni modificate. La "Licenza Pubblica Generale GNU" e la "Licenza Pubblica Generale Librerie GNU" possono essere incluse in traduzioni approvate dalla Free Software Foundation al posto dell'originale inglese.

A vostra scelta potete distribuire copie testuali e versioni modificate di questo documento sotto i termini della Licenza Pubblica Generale GNU, eccetto per le sezioni per cui sia indicato chiaramente che sono sottoposte ad un copyright diverso.

Eccezioni a queste regole possono essere fatte per usi vari: si chieda a Larry Greenfield, via posta all'indirizzo sovrastante, o via e-mail a `leg+@andrew.cmu.edu`. richiesto (ma non imposto) che avvisiate l'autore quando stampate in larga scala e/o commerciate questo documento. Percentuali e donazioni sono accettate ed incoraggeranno future edizioni.

Queste sono alcune delle convenzioni tipografiche usate in questo libro.

grassetto Usato per evidenziare **concetti nuovi**, **AVVERTIMENTI** e **parole chiave** in un linguaggio.

corsivo Usato per *enfatizzare* il testo. anche usato per indicare comandi per l'utente da scrivere quando sono mostrate le interazioni con lo schermo (vedere sotto).

inclinato Usato per evidenziare **meta-variabili** nel testo, specialmente nelle rappresentazioni delle righe di comando. Per esempio,

`ls -l` *pippo*

dove *pippo* indica un nome di file, come `/bin/cp`.

`Dattilografico` Usato per rappresentare le interazioni con lo schermo.

Usato anche per esempi di codice "C", shell script o altri tipi di codice, e per mostrare file comunemente usati, come quelli di configurazione. Quando necessario per motivi di chiarezza, questi esempi o figure saranno racchiusi in piccoli riquadri.

Tasto Rappresenta un tasto da premere. Lo si vedrà spesso in questa forma:

Premere Invio per continuare.

◇ Un rombo nel margine, evidenzia "pericoli" o "accortezze". Leggete quei paragrafi attentamente.

X Questa X nel margine indica istruzioni speciali per gli utenti del Sistema X Window.

Ringraziamenti

L'autore vuole ringraziare le seguenti persone per il loro inestimabile aiuto nell'uso di Linux o nella stesura della *Linux Guida dell'Utente*:

Linus Torvalds per aver fornito qualcosa su cui scrivere questo manuale.

Karl Fogel che mi ha dato un grande aiuto con la scrittura di questo documento e ha scritto il capitolo 8 e il capitolo 9. Non penso di potergli dare sufficiente onore.

Maurizio Codogno ha scritto gran parte del capitolo 11.

David Channon ha scritto l'appendice sull'editor `vi`. (Appendice A)

La [Yggdrasil Computing, Inc.] per il loro contributo generoso (e volontario) nel supporto a questo manuale.

La [Red Hat Software] per il loro (più recente e ancora volontario!) supporto

Indice

Capitolo 1

Introduzione

1.1 Chi dovrebbe leggere questo Libro

Siete fra quelli che dovrebbero leggere questo libro? Rispondiamo con un'altra domanda: avete appena prelevato Linux da qualche parte, l'avete installato e volete sapere cosa farci? Oppure siete utenti di computer non-Unix che stanno considerando Linux ma vogliono vedere cosa può fare?

Se avete questo libro, la risposta a queste domande è probabilmente "sì". Chiunque ha Linux, la versione di Unix liberamente distribuibile scritta da Linux Torvalds, nel proprio PC ma non sa cosa farci dovrebbe leggere questo libro. Qui tratteremo la maggior parte dei comandi Unix semplici e alcuni di quelli più avanzati. Parleremo anche di GNU Emacs, un potente editor, e di parecchie altre grosse applicazioni Unix.

1.1.1 Cosa bisogna aver fatto prima di leggere questo libro

Questo libro assume che possiate accedere ad un sistema Unix. (Sfortunatamente è un po' arduo imparare senza!) Più precisamente, questo sistema Unix dovrebbe essere un PC Intel con Linux. Questo requisito non è necessario, ma dove le versioni di Unix differiscono tra loro, qui si parla esclusivamente del comportamento di Linux.

Linux è disponibile in vari formati, chiamati distribuzioni. Si spera che abbiate trovato una distribuzione completa, come la Slackware, la Debian o la MCC-Interim, e l'abbiate installata. Ci sono delle differenze tra le varie distribuzioni di Linux, ma sono perlopiù piccole e poco importanti. Potrete trovare delle differenze dagli esempi proposti in questo libro. Per la maggior parte saranno differenze minime, di cui non ci si deve preoccupare. Se notate una differenza notevole tra questo libro e la vostra esperienza reale, informate me, l'autore.

Se siete il superutente (l'installatore e amministratore) del sistema, dovreste anche aver creato un account normale per voi stessi. Consultate i manuali di installazione per informazioni su come fare. Se non siete il superutente, dovreste avere ottenuto un account da lui/lei.

Dovete avere anche tempo e pazienza. Imparare Linux non è semplice—molti hanno trovato più semplice imparare il Sistema Operativo Macintosh, ma una volta imparato Linux diventa tutto molto più facile. Unix è un sistema molto potente ed è facilissimo portare avanti dei compiti molto complessi.

Oltre a ciò, questo libro presume che abbiate un po' di familiarità con alcuni termini informatici. Sebbene questo requisito non sia necessario, rende più semplice la lettura del libro. Dovreste conoscere termini come 'programma' ed 'esecuzione'; altrimenti sarebbe meglio che aveste qualcuno che vi aiuti ad imparare lo Unix.

1.2 Come evitare di leggere questo libro

Il modo migliore per imparare un programma è usando il proprio computer. Molta gente ha trovato che leggere un libro senza usare il programma non è utile, quindi il modo migliore per imparare Unix e Linux è usarli. Usate Linux più che potete. Sperimentate. Non preoccupatevi—è *possibile* sbagliare qualcosa, ma si può sempre reinstallare tutto. Tenete copie di sicurezza e divertitevi!

Unix non è intuitivo come molti altri sistemi operativi, quindi molto probabilmente finirete con il leggere almeno i capitoli 4, 5 e 6.

Il primo modo per evitare di usare questo libro è di usare la documentazione disponibile in linea. Imparate ad usare il comando `man`—è descritto nella sezione 4.2.

1.3 Come leggere questo libro

Il suggerimento per imparare Unix è di leggere un po', poi provare. Continuate a provare finché non comincerete ad avere dimestichezza con i concetti, e poi iniziate a girare all'interno del libro. Troverete una varietà di argomenti, alcuni dei quali vi interesseranno, altri vi annoieranno. Dopo un po' avrete abbastanza confidenza per iniziare ad usare comandi senza sapere quello che fanno esattamente, che è una cosa molto utile.

Quando molta gente si riferisce ad Unix, in realtà si riferisce alla shell di Unix, un programma speciale che interpreta i comandi. È il programma che controlla l'"apparenza" di Unix. In pratica la si può vedere così, ma bisogna sapere che in realtà Unix consiste di molte cose in più, o molte in meno. Questo libro spiega come usare la shell, i programmi contenuti in Unix ed alcuni programmi che non sempre si trovano in Unix (ma in Linux di solito sì).

Questo capitolo è particolare—tratta di questo libro e come usarlo per imparare a lavorare. Gli altri capitoli contengono:

Capitolo 2: spiega da dove provengono Unix e Linux e dove sono diretti. Parla anche della Free Software Foundation e del Progetto GNU.

Capitolo 3: parla di come avviare e fermare il computer, e cosa succede in questi momenti. Per la maggior parte tratta argomenti non necessari per l'uso di Linux, ma comunque utili ed interessanti.

Capitolo 4: introduce la shell di Unix, che è dove si lavora e si eseguono i programmi. Parla dei comandi e programmi base che bisogna conoscere per usare Unix.

Capitolo 5: spiega il Sistema X Window. X è l'interfaccia grafica primaria di Unix e alcune distribuzioni lo impostano per default.

Capitolo 6: spiega alcune parti più avanzate della shell di Unix. Imparare le tecniche descritte in questo capitolo renderà più efficiente il vostro modo di lavorare.

Capitolo 7: è una breve descrizione di molti comandi Unix. Più strumenti un utente sa usare, più velocemente completa il suo lavoro.

Capitolo 8: descrive l'editor di testi Emacs. Emacs è un grosso programma che integra molti strumenti di Unix in un'unica interfaccia.

Capitolo 9: descrive come adattare il sistema Unix ai vostri gusti personali.

Capitolo 10: studia i modi in cui un utente Unix può comunicare con altre macchine sparse per il mondo, compresa la posta elettronica e il WWW.

Capitolo 11: descrive alcuni dei comandi più grossi e difficili da usare.

Capitolo 12: parla dei modi semplici per evitare errori in Unix e Linux.

1.4 Documentazione su Linux

Questo libro, *Linux Guida dell'Utente*, è rivolto ai principianti di Unix. Fortunatamente, il Linux Documentation Project sta scrivendo anche libri per gli utenti di maggiore esperienza.

1.4.1 Altri libri su Linux

Gli altri libri comprendono *Installation and Getting Started*, una guida su come prelevare ed installare Linux, *The Linux System Administrator's Guide*, come organizzare e amministrare un sistema Linux, e *The Linux Kernel Hackers' Guide*, un libro su come modificare Linux. *The Linux Network Administration Guide* parla di come installare, configurare ed usare una connessione di rete.

1.4.2 HOWTO

In aggiunta ai libri, il Linux Documentation Project ha creato una serie di brevi documenti che descrivono come impostare un particolare aspetto di Linux. Per esempio, lo `SCSI-HOWTO` descrive alcune delle complicazioni dell'uso dello `SCSI`—uno standard per la comunicazione tra dispositivi—con Linux.

Questi `HOWTO` sono disponibili in diversi formati: in un libro rilegato come la *The Linux Bible* o *Dr. Linux*, nel newsgroup `comp.os.linux.answers` o in svariati siti sulla World Wide Web. Uno dei siti centrali per le informazioni su Linux è `http://www.linux.org`.

1.4.3 Cos'è il Linux Documentation Project?

Come quasi tutte le organizzazioni che riguardano Linux, il Linux Documentation Project è un gruppo di persone che lavorano sparse in giro per il mondo. Originariamente organizzato da Lars Wirzenius, il Progetto è ora coordinato da Matt Welsh con l'aiuto di Michael K. Johnson.

Si spera che il Linux Documentation Project fornirà in futuro libri che vadano incontro a tutti i bisogni della documentazione di Linux. Diteci se ci siamo riusciti o come dovremmo migliorare. Potete contattare l'autore all'indirizzo `greenfie@gauss.rutgers.edu` e/o Matt Welsh a `mdw@cs.cornell.edu`.

1.5 Sistemi Operativi

Lo scopo principale di un sistema operativo è supportare i programmi che eseguono realmente il lavoro a cui si è interessati. Per esempio, state usando un editor per creare un documento. Questo editor non può lavorare senza l'aiuto del sistema operativo— necessita di questo aiuto per interagire con il terminale, i file ed il resto del computer.

Se tutto quello che fa il sistema operativo è supportare le applicazioni, perché è necessario un libro intero solo per poterlo usare? Ci sono varie attività di manutenzione (senza considerare i problemi più grossi) che è necessario fare. Nel caso di Linux, il sistema operativo contiene anche una serie di "mini-applicazioni" per aiutare ad eseguire il lavoro in modo più efficiente; conoscere il sistema operativo può essere utile quando non si lavora con grosse applicazioni.

I sistemi operativi (in breve OS) possono essere semplici e minimalisti, come il DOS, o grandi e complessi, come OS/2 o VMS.[1] Unix tenta di essere una via di mezzo: mentre fornisce più risorse e fa più cose dei primi sistemi operativi, non tenta di fare *tutto*. Unix è stato originariamente progettato come una semplificazione di un sistema operativo chiamato Multics.

La filosofia originale di Unix fu di distribuire le funzionalità in piccole parti, i programmi.[2] In questo modo si possono aggiungere nuove funzionalità e caratteristiche combinando le piccole parti (programmi) in nuovi modi. E se appaiono nuove utility (e succede), si possono integrare con i vecchi strumenti. Sfortunatamente i programmi tendono con il tempo a crescere e acquistare nuove caratteristiche, ma la flessibilità rimane. Mentre sto scrivendo questo documento, per esempio, sto usando attivamente questi programmi: `fvwm` per gestire le mie "finestre", `emacs` per editare il testo, LaTeX per formattarlo, `xdvi` per vedere l'anteprima, `dvips` per preparare la stampa e `lpr` per stamparlo. Se un domani mi procurassi un visualizzatore dvi migliore, lo potrei usare al posto di `xdvi` senza cambiare il resto della mia configurazione. In questo momento, il mio sistema ha trentotto programmi che girano simultaneamente (la maggior parte sono programmi di sistema che "dormono" finché non hanno qualcosa di specifico da fare).

Quando si usa un sistema operativo, si vuole minimizzare la quantità di lavoro necessaria per i ottenere lo scopo. Unix fornisce molti strumenti per aiutarvi, ma solo se sapete cosa fanno. Passare ore tentando di fare qualcosa e alla fine arrendersi non è molto produttivo. Questo libro vi insegnerà quali strumenti usare nelle varie situazioni e come collegarli tra loro.

La parte chiave di un sistema operativo è chiamata "kernel" (nucleo). In molti sistemi operativi, come Unix, OS/2 o VMS, il kernel fornisce ai programmi le funzioni da usare e gestisce la sequenza della loro esecuzione. Semplicemente dice che il programma A può avere un certo tempo a disposizione, il programma B un altro tempo, e così via. C'è sempre il kernel che gira: è il primo programma ad essere inizializzato quando si accende il sistema e l'ultimo a fare qualsiasi cosa quando lo si ferma.

---

[1] Apologia per gli utenti DOS, OS/2 e VMS. Ho usato tutti e tre, e ognuno ha i suoi lati positivi.

[2] Questo fu in realtà determinato dall'hardware in cui originariamente funzionava Unix. Per qualche strana ragione, il sistema operativo risultante era molto pratico su altri hardware. Il progetto di base viene ancora usato dopo venticinque anni.

Capitolo 2

Cos'è Unix?

2.1 Storia di Unix

Nel 1965, i laboratori della Bell Telephone (Bell Labs, una divisione della AT&T) stavano lavorando con la General Electric e il Project MAC del MIT per scrivere un sistema operativo chiamato Multics. Per fare la storia un po' più breve, i laboratori della Bell decisero che il progetto non sarebbe andato in porto e sciolsero il gruppo. Questo, comunque, lasciò i laboratori della Bell senza un buon sistema operativo.

Ken Thompson e Dennis Ritchie decisero di abbozzare un sistema operativo che doveva incontrare le necessità dei Laboratori della Bell. Quando a Thompson servì un ambiente di sviluppo (1970) che funzionasse su un PDP-7, implementò le loro idee. Come gioco di parole su Multics, Brian Kernighan, un altro ricercatore della Bell Labs, diede al sistema il nome Unix.

In seguito, Dennis Ritchie inventò il linguaggio di programmazione "C". Nel 1973, Unix fu riscritto in C, invece che nell'originale linguaggio assembly.[1] Nel 1977, Unix fu adattato ad una nuova macchina usando un processo chiamato porting , e fu tolto dalla macchina PDP su cui girava precedentemente. Ciò fu aiutato dal fatto che Unix era stato scritto in C, dato che non si dovette riscrivere molto codice, ma bastò ricompilarlo.

Alla fine degli anni '70, alla AT&T fu vietato di competere nell'industria informatica, quindi fornì a prezzo molto basso le licenze di Unix a svariati college ed università. Prese piede lentamente all'esterno delle istituzioni accademiche, ma alla fine divenne popolare anche nell'ambiente commerciale. Lo Unix di oggi è diverso dallo Unix del 1970: ha due grandi

---

[1] Il "linguaggio assembly" è un linguaggio di programmazione a livello molto basso legato ad un determinato tipo di computer. Programmare in assembly è spesso considerato una sfida.

versioni: la System V, degli Unix System Laboratories (USL), dei consociati della Novell,[2] e la Berkeley Software Distribution (BSD). La versione USL è arrivata alla quarta revisione, o SVR4,[3] mentre l'ultima versione BSD è la 4.4. Comunque ci sono molte diverse versioni di Unix oltre a queste due. La maggior parte delle versioni commerciali di Unix derivano da uno di questi due gruppi. Le versioni di Unix attualmente utilizzate incorporano caratteristiche di entrambi.

Le versioni attuali di UNIX per PC Intel costano tra i $500 e i $2000.

2.2 Storia di Linux

L'autore primario di Linux è Linus Torvalds. Dalla sua versione originale, è stato perfezionato da innumerevoli persone da tutto il mondo. È un clone, scritto interamente da zero, del sistema operativo Unix. Né USL né l'Università della California, Berkeley, sono stati coinvolti nella stesura di Linux. Uno dei fatti più interessanti di Linux è che il suo sviluppo avviene simultaneamente in tutto il mondo. Gente dall'Australia alla Finlandia ha contribuito a Linux, e continuerà felicemente a farlo.

Linux iniziò con un progetto di esplorazione del chip 80386. Uno dei primi progetti di Linus era un programma che doveva stampare alternativamente `AAAA` e `BBBB` (due processi funzionanti in multitasking). Questo poi è diventato Linux.

Linux è sottoposto al copyright della GNU General Public License (GPL). Si tratta di una licenza scritta dalla Free Software Foundation (FSF), progettata per prevenire la gente dal restringere la distribuzione del software. In breve essa dice che anche se si può far pagare quello che si vuole per una copia, non si può impedire alla persona a cui la copia è stata venduta di distribuirla liberamente. Significa anche che il codice sorgente[4] deve essere disponibile, cosa molto utile per i programmatori. Chiunque può modificare Linux e rendere disponibili le sue modifiche, sempre che mantengano il codice sotto lo stesso copyright.

Linux supporta molto del software Unix più famoso, compreso il Sistema X Window. Il sistema X Window è stato creato al Massachusetts Institute of Technology. È stato scritto per permettere ai sistemi Unix di creare finestre grafiche ed interagire facilmente l'uno con l'altro. Oggi, il sistema X Window viene usato su tutte le versioni di Unix disponibili.

In aggiunta alle due versioni di Unix, System V e BSD, c'è anche un insieme di documenti di standardizzazione, pubblicati dalla IEEE, intitolati **POSIX**. Linux è principalmente

---

[2] È stato recentemente venduto alla Novell. Precedentemente, USL era di proprietà della AT&T.

[3] Un modo criptico di dire "System five, release four".

[4] Il codice sorgente di un programma è ciò che il programmatore scrive e legge. Viene in seguito trasformato in codice macchina illeggibile che viene interpretato dal computer.

conforme a POSIX-1 e POSIX-2. Il suo aspetto è molto simile a BSD in alcune cose, e a System V in altre. È un insieme (secondo la maggior parte delle persone, ottimo) di tutti e tre gli standard.

Molte delle utility incluse nelle distribuzioni di Linux provengono dalla Free Software Foundation e sono parte del Progetto GNU. Il Progetto GNU è un'opera di scrittura di un sistema operativo portabile ed avanzato che somigli molto a Unix. "Portabile" significa che funzionerà su una varietà di macchine, non solo PC Intel, ma anche Macintosh o altri. Il sistema operativo GNU si chiama Hurd. La differenza principale tra Linux e Hurd non sta nell'interfaccia utente, ma nell'interfaccia di programmazione—Hurd è un sistema operativo moderno mentre Linux prende in prestito più cose dallo Unix originale.

La storia di Linux riportata qui sopra è carente perché non menziona nessuno *oltre a* Linus Torvalds. Ad esempio, H. J. Lu ha curato la manutenzione del `gcc` e la libreria C di Linux (due elementi necessari per la programmazione su Linux) fin da molto presto nella storia di Linux stesso. Potete trovare un elenco di persone che meritano di essere nominate in ogni sistema Linux nel file `/usr/src/linux/CREDITS`.

2.2.1 Linux oggi

La prima cifra del numero di versione di Linux indica revisioni veramente enormi, che cambiano molto lentamente; al momento in cui sto scrivendo (febbraio 1996) è disponibile solo la versione "1". La seconda cifra indica revisioni un po' più piccole. Una seconda cifra pari indica le versioni più stabili ed affidabili, mentre le versioni con la seconda cifra dispari sono versioni di sviluppo, più facilmente bacate. L'ultima cifra è il numero della versione—ogni volta che viene rilasciata una nuova versione, che può semplicemente risolvere dei piccoli problemi o aggiungere caratteristiche di piccola entità, questa cifra viene aumentata di uno. Oggi, febbraio 1996, siamo arrivati alla versione stabile 1.2.11, ed alla versione di sviluppo 1.3.61.

Linux è un grosso sistema operativo e sfortunatamente contiene bug (errori) che devono essere trovati e corretti. Sebbene alcune persone trovino regolarmente degli errori, normalmente avviene a causa di un hardware insolito o difettoso; i bug che interessano molte persone sono pochissimi.

Certamente, questi sono solo i bug del kernel. I bug possono essere presenti anche in ogni aspetto del sistema, e gli utenti inesperti possono avere problemi a distinguere programmi diversi l'uno dall'altro. Per esempio, un problema che potrebbe presentarsi è che i caratteri diventano illeggibili—è un bug o una "caratteristica"? Sorprendentemente, questa è una caratteristica—i caratteri illeggibili sono causati da alcune sequenze di controllo che a volte possono apparire. Speriamo che questo libro insegni a distinguere le varie situazioni.

2.2.2 Domande comuni su Linux

Prima di partire per il nostro lungo viaggio, mettiamo in chiaro le cose veramente importanti:

Domanda: Come si pronuncia Linux?

Risposta: Per volontà di Linus, dovrebbe essere pronunciato con una breve suono *ih*, come prInt, mInImo ecc. Linux dovrebbe far rima con Minix, un'altro clone di Unix. La pronuncia è *LIH-nucks*.

Domanda: Perché lavorare su Linux?

Risposta: Perché no? Linux è in genere meno caro degli altri sistemi operativi, e spesso meno problematico di molti sistemi commerciali. Può non essere il sistema migliore per le vostre particolari applicazioni, ma per chi è interessato ad usare le applicazioni Unix disponibili per Linux, è un sistema molto performante.

2.2.3 Software commerciale per Linux

Esiste molto software commerciale disponibile per Linux. Ad esempio si può cominciare con Motif, un'interfaccia utente per il sistema X Window che somiglia vagamente a Microsoft Windows. Linux si sta guadagnando sempre più software commerciale. In questi giorni si può comprare di tutto per Linux, da Word Perfect, un famoso word processor, a Maple, un complesso pacchetto di manipolazione simbolica.

Per i lettori interessati agli aspetti legali di Linux, ciò è permesso dalla licenza Linux. Mentre la Licenza Pubblica Generale GNU (riprodotta nell'appendice B) copre il kernel di Linux e sembra non permettere lo sviluppo di software commerciale, la Licenza Pubblica Generale Librerie (riprodotta nell'appendice C) copre molto del codice su cui le applicazioni dipendono. Ciò permette agli sviluppatori di software commerciale di vendere le loro applicazioni e non distribuire il codice sorgente.

Da notare che questi due documenti sono avvisi di copyright, non licenze per l'uso. Essi *non* regolano come si dovrebbe usare il software, ma semplicemente in quali circostanze si può copiare il software e ogni lavoro derivato. Per la Free Software Foundation, questa è una distinzione importante: Linux non ha nessuna licenza d'uso ma è semplicemente protetto dalla stessa regolamentazione che non permette di fotocopiare un libro.

Capitolo 3

Iniziare

3.1 Iniziare ad usare il proprio computer

Qualcuno potrebbe avere esperienze precedenti con l'MS-DOS o con altri sistemi operativi per utenti singoli, come OS/2 o Macintosh. In questi sistemi operativi non c'era bisogno di identificarsi al computer prima di usarlo; si presumeva di essere i soli utenti del sistema e quindi di poter accedere a tutte le risorse. Bene, Unix è un sistema operativo multi-utente—non solo può essere usato da più di una persona alla volta, ma diverse persone possono essere trattate in modo diverso.

Per distinguere i vari utenti, Unix ha bisogno di identificarli tramite un processo chiamato **login**. Quando si accende il computer avviene un processo abbastanza complesso, prima che il computer sia pronto per essere usato. Siccome questa guida è orientata a Linux, spiegherò cosa succede durante la sequenza di avvio di Linux (boot-up).

Se state usando Linux su un tipo di computer che non sia un PC Intel, alcune parti di questo capitolo non sono applicabili: la maggior parte dovrebbero trovarsi nella sezione 3.1.1

3.1.1 Accendere il Computer

La prima cosa che succede quando si accende un PC Intel è l'esecuzione del BIOS. BIOS sta per **B**asic **I**nput/**O**utput **S**ystem: normalmente è un programma memorizzato permanentemente nel computer su chip a sola lettura, che esegue alcuni test di base e poi cerca un floppy nella prima unità disco. Se ne trova uno cerca un "boot sector" (settore di avvio) su quel disco e inizia ad eseguirne il codice, se c'è. Se c'è un disco, ma senza settore di avvio, il BIOS scriverà un messaggio del genere:

```
Non-system disk or disk error
```

(Disco non di sistema o errore di disco) Togliendo il disco e premendo un tasto si farà continuare il processo di avvio.

Se non c'è un floppy nel drive, il BIOS cerca un master boot record (MBR) nel disco fisso. Inizierà l'esecuzione del codice trovato lì, che carica il sistema operativo. Su un sistema Linux, LILO, il **LI**nux **LO**ader, può occupare il MBR e caricare Linux. Per adesso assumiamo che succeda questo, e che Linux inizi a caricarsi. (La vostra distribuzione potrebbe avere dei meccanismi di avvio dal disco fisso diversi. Controllate la documentazione inclusa nella distribuzione. Un altro buon riferimento è la documentazione di LILO, [1].)

3.1.2 Linux parte

Dopo che il BIOS ha passato il controllo a LILO, LILO passa il controllo al kernel di Linux. Il kernel è il programma principale del sistema operativo, che controlla tutti gli altri programmi. La prima cosa che Linux fa quando parte, è commutare il sistema in modalità protetta. Il processore 80386,[1] che controlla il computer, ha due modalità, chiamate reale e protetta. Il DOS funziona in modalità reale, come il BIOS, ma alcuni sistemi operativi avanzati devono girare in modalità protetta. Per questo, quando Linux parte, viene abbandonato il BIOS.

Altri tipi di processore compiranno questo stadio in maniera diversa: nessun altro processore ha bisogno di cambiare modalità, e pochi devono avere una costruzione sulla procedura di avvio pesante come LILO e il BIOS. Una volta che il kernel è partito, Linux funziona nello stesso modo.

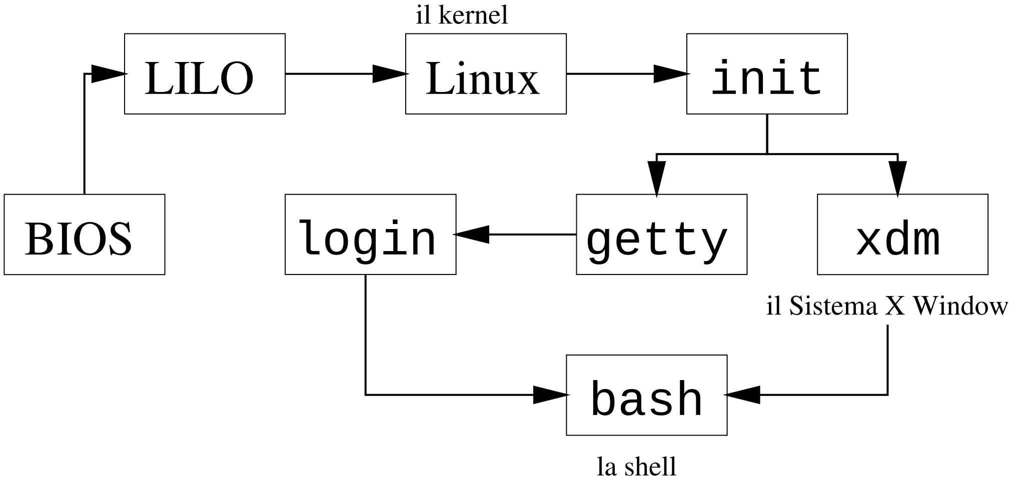
Linux controlla poi il tipo di hardware su cui sta girando: vuole sapere che tipo di disco fisso è presente, se c'è un bus mouse, se si lavora in rete e altre cose simili. Linux non può ricordarsi queste cose tra un avvio e l'altro, così le deve valutare ogni volta. Fortunatamente non le chiede *a voi*—ma all'hardware! Durante l'avvio, il kernel di Linux stampa un elenco di messaggi, che si possono leggere nella Sezione 3.3. Questo processo di richiesta di informazioni sull'hardware può causare qualche problema con il sistema, ma se lo fa, probabilmente è quando si installa Linux per la prima volta. Se avete problemi, consultate la documentazione della vostra distribuzione.

Il kernel gestisce semplicemente gli altri programmi quindi, una volta controllato che tutto sia a posto, deve avviarne uno per fare qualcosa di utile. Il programma avviato dal kernel si chiama `init` (notare la differenza dei caratteri: le parti di testo evidenziato con `questo tipo di carattere` normalmente sono nomi di programmi, file, directory o termini infor-

---

[1] Quando mi riferisco all'80386, parlo anche dell'80486, dei Pentium e dei Pentium Pro, a meno che non specifichi il contrario. Il nome sarà in seguito abbreviato in 386.

Figura 3.1 Il percorso che un PC Intel fa per arrivare al prompt di shell. `init` può avviare o no il Sistema X Window; se lo fa viene avviato `xdm`, altrimenti viene avviato `getty`.



matici). Dopo l'avvio di `init` il kernel non avvia più nessun programma; diventa un gestore di risorse, non un programma attivo.

Così, per vedere cosa fa il computer dopo l'avvio del kernel, bisogna esaminare `init`. `init` esegue una complicata sequenza di avvio che non è la stessa per tutti i computer. Per Linux ci sono varie versioni di `init`, e ognuna lavora a modo proprio. È importante anche se il vostro computer è in rete e quale distribuzione avete usato per installare Linux. Alcune cose che possono succedere quando parte `init` sono:

- Controllo dei filesystem. Cos'è un filesystem? È l'organizzazione dei file in un disco fisso. In questo modo Unix sa quali parti del disco sono utilizzate e quali no (è come un indice di un grosso archivio o lo schedario di una biblioteca). Sfortunatamente alcuni fattori, come una caduta di tensione, possono fare in modo che quello che le informazioni nel filesystem pensano che stia accadendo nel resto del disco e lo stato reale del disco siano in conflitto. Un programma speciale, `fsck`, può scovare queste situazioni e correggerle.
- Avviamento di speciali programmi di routing per la rete. Questi programmi dicono al computer come fare per contattare altri computer.
- Rimozione di file temporanei lasciati da alcuni programmi.
- Aggiornamento dell'orologio del sistema. È più complicato di quanto possa sembrare, dato che Unix, per default, vuole l'ora in UCT (Universal Coordinated Time, anche noto come GMT - Greenwich Mean Time) e l'orologio del CMOS, un orologio interno al computer alimentato a batterie, è probabilmente impostato all'ora locale. Ciò

significa che alcuni programmi devono leggere l'ora dall'orologio hardware e portarlo in UCT.

Dopo che `init` ha finito i suoi compiti all'avvio, comincia la sua attività normale. `init` può essere considerato il padre di tutti i processi di un sistema Unix. Un processo è semplicemente un programma in esecuzione; siccome ogni programma può girare più di una volta, possono esserci due o più processi per ogni programma.

In Unix, un processo, cioè un'istanza di un programma, è creato da una chiamata di sistema—un servizio fornito dal kernel—chiamata `fork` (si chiama "fork" - "forchetta"- perché un processo si divide in due processi separati). `init` si divide in un paio di processi, che si dividono a loro volta. Nel vostro sistema Linux, `init` esegue parecchie istanze di un programma chiamato `getty`. `getty` è il programma che permette ad un utente di collegarsi, e che alla fine chiama un programma chiamato `login`.

3.1.3 Adesso tocca all'utente

La prima cosa da fare per usare una macchina Unix è identificarsi. Questo processo, conosciuto come "logging in", o "login", è il modo in cui Unix riconosce un utente autorizzato ad usare il sistema, e chiede un nome di account e una password. Un nome di account è normalmente simile al vostro nome; dovreste averne già ricevuto uno dall'amministratore del sistema, o averne creato uno personale se l'amministratore siete voi (informazioni su come fare dovrebbero essere disponibili in *Installation and Getting Started* o *The Linux System Administrator's Guide*).

Dopo la procedura di avvio dovreste vedere qualcosa simile al seguente (la prima linea è semplicemente un messaggio di benvenuto—può essere un disclaimer o qualsiasi altra cosa):

```
Welcome to the mousehouse. Please, have some cheese.

mousehouse login:
```

Comunque è possibile che il sistema *non* si presenti così. Invece di un noioso schermo testuale, potrebbe usare la grafica. In qualsiasi caso vi chiederà sempre di loggarvi e funzionerà più o meno nella stessa maniera. Se è questo il vostro caso, state usando il Sistema X Window, e vi si presenterà un sistema a finestre. Il capitolo 5 discuterà alcune delle differenze a cui vi troverete di fronte, comunque il login sarà simile, come le basi della maggior parte di Unix. Se state usando X, cercate una X gigante a margine del testo del libro.

Questo è l'invito per voi a **loggarvi**. In questo manuale useremo l'utente fittizio (o reale, dipende dalla vostra macchina) `larry`. Dove trovate `larry`, dovreste sostituire il vostro nome di account. Il nome di account è spesso basato sul nome reale; i più grossi sistemi Unix seri hanno gli account formati dal cognome, da alcune combinazioni tra nome e cognome, o anche da alcuni numeri. Possibili account per Larry Greenfield potrebbero essere: `larry`, `greenfie`, `lgreenfi`, `lg19`.

`mousehouse` è il "nome" della macchina che sto usando. possibile che quando installate Linux, vi sia richiesto di inventare un nome buffo per la vostra macchina. Non è molto importante, ma userò `mousehouse`, o raramente `lionsden` quando dovrò usare un nome per un secondo sistema per chiarezza o per contrapporlo al primo.

Dopo aver inserito `larry` e aver premuto Invio, apparirà questo:

```
mousehouse login: larry
Password:
```

Quello che Linux sta chiedendo è la vostra **password**. Quando inserite la password, non riuscirete a vedere quello che digitate. Fate attenzione: è possibile cancellare, ma non riuscirete a vedere quello che state cambiando. Non digitate troppo lentamente se altra gente sta guardando—potrebbero capire la vostra password. Se sbagliate, vi verrà presentata un'altra possibilità di loggarvi.

Se avete inserito correttamente il nome di login e la password, apparirà un breve messaggio, chiamato il messaggio del giorno (`/etc/motd`); può dire qualsiasi cosa—lo sceglie l'amministratore del sistema, dopodiché appare un **prompt**. Un prompt è solamente un messaggio che appare per chiedere il prossimo comando da dare al sistema, e dovrebbe essere simile a:

```
/home/larry$
```

Se state già usando X Window, probabilmente vedrete un prompt simile a questo in una "finestra" in qualche parte dello schermo (una "finestra" è semplicemente una zona rettangolare dello schermo). Per scrivere un comando, spostare il cursore del mouse (probabilmente sarà una "x" o una freccia) dentro la finestra.

3.2 Spegnere il Computer

Non spegnere il computer! Rischiate di perdere dati importanti!

Diversamente da molte versioni del DOS, è una brutta cosa premere il pulsante di alimentazione quando avete finito di usare il computer. anche errato riavviare la macchina (con il tasto di reset) senza aver prima preso le opportune precauzioni. Linux, per aumentare l'efficienza, fa la cache del disco, cioè memorizza temporaneamente parte dei dati permanenti del computer in RAM[2]. L'idea di come Linux pensa che il disco dovrebbe essere e quello che il disco contiene realmente vengono sincronizzati ogni 30 secondi. Per spegnere o riavviare il computer, bisogna passare attraverso una procedura che comunichi a Linux di fermare il processo di cache del disco.

Quando avete finito con il computer, ma siete ancora loggati (avete inserito il nome di utente (**username**) e password), per prima cosa dovete uscire (fare logout). Per farlo, inserite il comando `logout`. Tutti i comandi vengono inviati battendo il tasto chiamato "Invio", "Enter" oppure "Return". Finché non premete Invio, non succede niente, e potete cancellare quello che avete scritto e ripartire.

```
/home/larry$ logout

Welcome to the mousehouse. Please, have some cheese.

mousehouse login:
```

Adesso può loggarsi un altro utente.

3.2.1 Spegnere il Computer

Se il sistema è a utente singolo, potete spegnere il computer appena avete finito di usarlo[3]; per farlo bisogna loggarsi con un account speciale chiamato `root`. L'account di `root` è l'account dell'amministratore del sistema e permette l'accesso a tutti i file. Se state per spegnere il computer, chiedete la password all'amministratore (in un sistema ad utente singolo, siete *voi*! Assicuratevi di sapere la password di `root`). Loggatevi come `root`:

```
mousehouse login: root
Password:
Linux version 1.3.55 (root@mousehouse) #1 Sun Jan 7 14:56:26 EST 1996
/# shutdown now
Why? end of the day
```

---

[2] La differenza tra la "RAM" e un disco fisso è come la differenza tra memoria a breve termine e menoria a lungo termine. Spegnere il computer è come dargli una botta in testa—si dimenticherà tutto quello che è nella memoria a breve termine, ma quello che è nella memoria a lungo termine verrà salvato. Il disco è migliaia di volte più lento della RAM.

[3] Per evitare possibili indebolimenti di alcuni componenti hardware, spegnete il computer solamente se avete finito di usarlo per quel giorno. Spegnere e riaccendere il computer una volta al giorno è probabilmente il miglior compromesso tra il consumo di energia ed il consumo delle parti hardware del sistema.

```
URGENT: message from the sysadmin:
System going down NOW

          ... end of the day ...

Now you can turn off the power...
```

Il comando `shutdown now` prepara il sistema per essere resettato o spento. Attendete il messaggio di conferma che tutto è a posto e poi riavviate o spegnete il sistema. (Quando il sistema vi chiede "Why?", cioè "perché?", è solo per poterlo comunicare agli altri utenti. Dato che quando spegnete non c'è nessuno collegato, potete scrivere quello che volete, o non scrivere niente).

Un rapido suggerimento ai pigri: un'alternativa al logout/login è di usare il comando `su`. Da utente normale, dal prompt inserire `su` e Invio. Il sistema chiede la password di root e poi vi dà i privilegi di root. Adesso potete eseguire lo shutdown del sistema con il comando `shutdown now`.

3.3 Messaggi del kernel

Quando avviate il computer, passano sullo schermo una serie di messaggi che descrivono l'hardware collegato al vostro computer. Questi messaggi vengono scritti dal kernel di Linux. In questa sezione cercherò di descriverli e spiegarli.

Naturalmente, i messaggi stampati dal kernel variano da macchina a macchina. Io descriverò quelli della mia. L'esempio seguente contiene tutti i messaggi standard ed alcuni messaggi specifici. (In generale, la macchina di cui sto parlando è una configurata al minimo, e non ci saranno molti messaggi specifici dell'hardware). L'esempio è stato fatto con la versione 1.3.55 di Linux—una delle più recenti nel momento in cui sto scrivendo.

1. La prima cosa che fa Linux è decidere che tipo di scheda video e schermo avete, in modo da utilizzare la giusta dimensione dei caratteri (più piccoli sono i caratteri, più se ne possono vedere nello schermo contemporaneamente). Linux potrebbe chiedervi se volete un carattere speciale, o può averne alcuni compilati al suo interno[4].

   ```
   Console: 16 point font, 400 scans
   ```

---

[4] "Compilazione" è il processo con cui un programma scritto da una persona viene tradotto in qualcosa che il computer può capire. Una caratteristica "compilata" in un programma è stata inclusa nel programma.

```
Console: colour VGA+ 80x25, 1 virtual console (max 63)
```

In questo esempio, il proprietario della macchina aveva deciso di volere in fase di compilazione i normali caratteri grandi. Inoltre, notate l'errore ortografico nella parola "color". Linus ha evidentemente imparato la versione sbagliata dell'inglese.

2. La prossima cosa che fa il kernel è dire quanto il sistema è veloce, misurando la velocità in "BogoMIPS". Un "MIP" sta per un milione di istruzioni al secondo, e un "BogoMIP" è un "bogus MIP": quante volte il computer può non fare assolutamente niente in un secondo (dato che il loop non fa niente, il numero non è realmente una misura della velocità del sistema). Linux usa questo numero quando deve restare in attesa di un dispositivo hardware.

   ```
   Calibrating delay loop.. ok - 33.28 BogoMIPS
   ```

3. Il kernel di Linux comunica anche qualcosa sull'uso della memoria:

   ```
   Memory: 23180k/24576k available (544k kernel code, 384k reserved, 468k data)
   ```

   Questo significa che la macchina ha 24 megabyte di memoria, di cui una parte è riservata per il kernel, e il resto può essere usata dai programmi: si tratta della RAM temporanea che viene usata per l'immagazzinamento di dati a breve termine. Il computer ha anche una memoria a lungo termine chiamata hard disk (o disco fisso); il contenuto dell'hard disk viene salvato anche quando il computer viene spento.

4. Durante la procedura di boot, Linux fa dei test su diversi componenti hardware e stampa dei messaggi anche su questi test.

   ```
   This processor honours the WP bit even when in supervisor mode. Good.
   ```

5. Ora Linux si sposta alla configurazione della rete. Quello che segue dovrebbe essere spiegato nella *The Linux Networking Guide*, e va oltre gli scopi di questo libro.

   ```
   Swansea University Computer Society NET3.033 for Linux 1.3.50
   IP Protocols: ICMP, UDP, TCP
   ```

   Linux supporta la FPU, l'unità a virgola mobile, un chip speciale (o parte di un chip, nel caso della CPU 80486), che permette di fare calcoli matematici con numeri non interi. Alcuni di questi chip sono difettosi, e quando Linux tenta di identificare questi chip, la macchina va in "crash", cioè la macchina si ferma. Se questo succede, dovreste vedere:

```
Checking 386/387 coupling...
```

Altrimenti vedreste:

```
Checking 386/387 coupling... Ok, fpu using exception 16 error reporting.
```

se state usando un 486DX. Se state usando un 386 con un 387, vedreste:

```
Checking 386/387 coupling... Ok, fpu using irq13 error reporting.
```

6. Fa ora un altro test sull'istruzione di “halt”.

```
Checking 'hlt' instruction... Ok.
```

7. Dopo la configurazione iniziale, Linux stampa una linea che identifica se stesso. Dice quale versione è, con quale versione del compilatore C GNU è stato compilato, e quando.

```
Linux version 1.3.55 (root@mousehouse) (gcc version 2.7.0) #1 Sun Jan 7 14:56:26 EST 1996
```

8. Il driver delle porte seriali ha iniziato a chiedere informazioni sul tipo di hardware. Un driver è una parte del kernel che controlla un dispositivo, normalmente una periferica, ed è responsabile dei dettagli della comunicazione tra la CPU ed il dispositivo. Permette a chi scrive applicazioni utente di concentrarsi sull'applicazione, e non preoccuparsi di come il computer lavora realmente.

```
Serial driver version 3.95 with no serial options enabled
tty00 at 0x03f8 (irq = 4) is a 16450
tty01 at 0x02f8 (irq = 3) is a 16450
tty02 at 0x03e8 (irq = 4) is a 16450
```

Qui ha trovato 3 porte seriali. Una porta seriale è l'equivalente di una porta `COM` del DOS, ed è normalmente usata con modem e mouse.

Quello che sta dicendo è che la porta seriale 0 (`COM1`) ha l'indirizzo `0x03f8`. Quando questa interrompe il kernel, generalmente per comunicare che stanno arrivando dati, essa usa l'IRQ 4. Un IRQ è un segnale di una periferica che chiama il software. Ogni porta seriale ha il proprio chip di controllo. Un chip comune per una porta è il 16450; altri valori possibili sono 8250 e 16550.

9. Adesso tocca alle porte parallele. Una porta parallela normalmente è connessa ad una stampante, e i nomi delle porte parallele (in Linux) iniziano con `lp`. `lp` sta per **L**ine **P**rinter, anche se oggi sarebbe più adatto **L**aser **P**rinter (in ogni caso, Linux comunicherà tranquillamente con qualsiasi tipo di stampante parallela: ad aghi, a getto d'inchiostro o laser).

   ```
   lp0 at 0x03bc, (polling)
   ```

   Questo messaggio dice che è stata trovata una porta parallela e si usa il driver standard.

10. Il kernel controlla ora le unità floppy. In questo esempio, la macchina ha due drive. In DOS, il drive "A" è un floppy da 5 1/4 pollici, e il drive "B" è un floppy da 3 1/2 pollici. Linux chiama il drive "A" `fd0` e il drive "B" `fd1`.

    ```
    Floppy drive(s): fd0 is 1.44M, fd1 is 1.2M
    floppy: FDC 0 is a National Semiconductor PC87306
    ```

11. Il prossimo driver a partire nel mio esempio è il driver SLIP, che stampa un messaggio sulla sua configurazione.

    ```
    SLIP: version 0.8.3-NET3.019-NEWTTY (dynamic channels, max=256) (6 bit encapsulation enabl
    CSLIP: code copyright 1989 Regents of the University of California
    ```

12. Il kernel controlla anche la presenza di dischi fissi, controllando le partizioni presenti in ciascuno di essi. Una partizione è una suddivisione logica in un disco, che viene usata per tenere separati i sistemi operativi, in modo da evitare interferenze. In questo esempio, il computer ha due dischi fissi (`hda`, `hdb`) con rispettivamente quattro e una partizione.

    ```
    Partition check:
      hda: hda1 hda2 hda3 hda4
      hdb: hdb1
    ```

13. Infine Linux esegue il **mount** della partizione di root. La partizione di root è la partizione del disco dove risiede il sistema operativo Linux. Quando Linux "monta" questa partizione, questa viene marcata come partizione disponibile per l'uso.

    ```
    VFS: Mounted root (ext filesystem).
    ```

3.3.1 Messaggi a run-time

Il kernel di Linux manda talvolta dei messaggi allo schermo. Qui sotto c'è un elenco di alcuni di essi con il loro significato. Spesso questi messaggi indicano che qualcosa non va. Alcuni di essi sono **critici**, cioè significano che il sistema operativo (e tutti i programmi!) smettono di funzionare. Quando si presentano, dovreste scrivervi quello che dicono e quello che stavate facendo quando è successo, e mandarli a Linus. Vedere la sezione 12.3.2.

Per fortuna, alcuni di questi messaggi sono soltanto informazioni—si spera, dato che li vedrete più spesso!

-
  ```
  Adding Swap: 10556k swap-space
  lp0 on fire
  ******** OBVIOUSLY INCOMPLETE
  ```

Capitolo 4

La shell di Unix

4.1 Comandi di Unix

Quando vi loggate la prima volta in un sistema Unix, vi si presenta qualcosa del genere:

```
/home/larry$
```

Si chiama **prompt**. Come può suggerire il nome, vi chiede di inserire un comando. Ogni comando Unix è una sequenza di lettere, numeri e caratteri, senza spazi. Alcuni comandi validi sono `mail`, `cat` e `CMU_is_Number-5`. Alcuni caratteri non sono permessi—verranno elencati più avanti. Unix è anche **case-sensitive** (sensibile al maiuscolo/minuscolo); cioè `cat` e `Cat` sono comandi diversi[1].

Il prompt viene visualizzato da un programma speciale, la **shell**. La shell accetta comandi e li esegue; esistono anche programmi scritti in un linguaggio proprio della shell, e si chiamano "shell script".

In Unix ci sono due tipi principali di shell, le Bourne shell e le C shell. Le Bourne shell prendono il nome dal loro inventore, Steven Bourne. Steven Bourne ha scritto la shell Unix originale, `sh`, e da allora la maggior parte delle shell hanno nomi che finiscono in `sh` per indicare che sono estensioni dell'idea originale. Ci sono molte implementazioni della sua shell, e si chiamano tutte Bourne shell. È comune anche un'altra classe di shell, la C shell (originariamente implementata da Bill Joy). Tradizionalmente, le Bourne shell vengono

---

[1] La sensibilità al maiuscolo/minuscolo è una cosa particolare. Alcuni sistemi operativi, come OS/2 o Windows NT mantengono i nomi in maiuscolo o minuscolo, ma non fanno distinzioni. In pratica è difficile che due comandi Unix differiscano solo per una lettera maiuscola invece che minuscola. Non è normale avere due comandi diversi come `cat` e `Cat`.

usate per gli script di shell e per compatibilità con l'originale, e le C shell sono usate per l'uso interattivo (le C shell hanno il vantaggio di avere caratteristiche di interattività migliori, ma sono più difficili da programmare).

Linux viene fornito con una Bourne shell chiamata `bash`, scritta dalla Free Software Foundation. `bash` sta per **B**ourne **A**gain **Sh**ell, uno dei tanti brutti giochi di parole in Unix. È una Bourne shell "avanzata": contiene le caratteristiche standard di programmazione normalmente presenti nelle Bourne shell, insieme con molte caratteristiche di interattività normalmente presenti nelle C shell. `bash` è la shell di default che viene usata con Linux.

Quando vi loggate, il prompt viene mostrato da `bash`, e state eseguendo il vostro primo programma Unix, la `bash` shell. Finché sarete loggati, la bash shell continuerà a girare.

4.1.1 Un tipico comando Unix

Il primo comando da conoscere è `cat`. Per usarlo, digitate `cat` e premete Invio:

```
/home/larry$ cat
```

Se il cursore si trova da solo in una nuova linea, avete fatto tutto giusto. Ci sono diverse varianti che avreste potuto scrivere—alcune funzionano, altre no.

- Se aveste sbagliato a scrivere `cat`, avreste visto:

  ```
  /home/larry$ ct
  ct: command not found
  /home/larry$
  ```

  In questo modo la shell vi informa che non ha trovato un programma chiamato "`ct`" e propone un altro prompt per lavorare. Ricordatevi, Unix fa differenze tra maiuscolo e minuscolo: `CAT` è errato.

- Potreste aver anteposto alcuni spazi bianchi prima del comando, come qui[2]:

  ```
  /home/larry#␣␣␣␣␣cat
  ```

  Questo produce il risultato corretto ed esegue il programma `cat`.

- Potreste anche aver premuto Invio in una riga da solo. Andate avanti—non è successo assolutamente niente.

---

[2] Il carattere '␣' indica che l'utente ha inserito uno spazio.

Presumo che adesso siate in `cat`. Probabilmente vi state chiedendo che cosa fa. No, non è un gioco. `cat` è un potente strumento anche se per il momento non sembra. Scrivete qualcosa e premete Invio. Quello che dovreste vedere è:

```
/home/larry$ cat
Aiuto! Sono incastrato in un programma Linux!
Aiuto!  Sono incastrato in un programma Linux!
```

(Il testo *obliquo* indica quello che ha scritto l'utente.) Quello che sembra fare `cat` è copiare sullo schermo quello che l'utente scrive. A volte è utile, ma non ora. Usciamo da questo programma e passiamo ad altri comandi che abbiano un'utilità più evidente.

Per chiudere molti comandi Unix, digitate Ctrl-d[3]. Ctrl-d è il carattere di fine-file (End-Of-File), o EOF per abbreviare. In alternativa, sta per fine-del-testo, a seconda di quale libro avete letto. Mi riferirò ad esso come fine-file. È un carattere di controllo che comunica ai programmi Unix che voi (o un altro programma) avete finito di inserire dati. Quando `cat` vede che non state scrivendo niente altro, termina.

Per avere un'idea, provate il programma `sort`. Come indica il nome, è un programma di ordinamento: se inserite un paio di linee e poi premete Ctrl-d, le visualizzerà in ordine alfabetico. Questo tipo di programmi sono chiamati **filtri**, perché prendono un testo, lo filtrano e lo mandano fuori in modo diverso. Sia `cat` che `sort` sono filtri particolari: `cat` è particolare perché legge il testo per linee e non fa *nessuna* azione su di esso, `sort` perché legge per linee e non rende nessun output finché non ha visto il carattere EOF. Molti filtri agiscono linea per linea: leggono una linea, fanno dei calcoli, e rendono come output una linea diversa.

4.2 Come cavarsela da soli

Il comando `man` mostra delle pagine di riferimento per il comando[4] che specificate. Per esempio:

```
/home/larry$ man cat

cat(1)                                                    cat(1)
```

---

[3] Tenete premuto il tasto "Ctrl" e premete "d", poi lasciateli entrambi.

[4] man dà informazioni anche su chiamate di sistema, subroutine, formati di file ed altro. Nella versione originale di Unix dava esattamente le stesse informazioni della versione stampata. Per ora, probabilmente, sarete interessati soltanto alla documentazione sui comandi.

```
NOME
  cat - Concatena o visualizza file

SINTASSI
  cat [-benstuvAET] [--number] [--number-nonblank] [--squeeze-blank]
  [--show-nonprinting] [--show-ends] [--show-tabs] [--show-all]
  [--help] [--version] [file...]

DESCRIZIONE
  Questa pagina documenta la versione GNU del comando cat ...
```

C'è una pagina intera di informazioni su `cat`. Provate, ma non aspettatevi di capire tutto. Le pagine man assumono che voi conosciate diverse cose che ancora non sapete. Quando avrete letto la pagina, c'è probabilmente un rettangolino in basso sullo schermo, con la scritta `--more--`, `Line 1` o qualcosa di simile. È il prompt di `more`, ed imparerete ad amarlo.

Invece di lasciar scorrere il testo, `man` lo ferma alla fine di ogni videata, e aspetta. Se volete proseguire, premete Spazio e avanzerete di una videata. Se volete uscire (quit) dalla pagina che state leggendo, premete q: ritornerete al prompt della shell, pronta a ricevere un nuovo comando.

Si possono anche dare delle opzioni a `man`. Per esempio, poniamo che siate interessati ad un comando per interagire con il Postscript, il linguaggio di controllo delle stampanti di Adobe. Digitate `man -k ps` o `man -k Postscript`, otterrete una lista di comandi, chiamate di sistema e altre parti documentate di Unix che hanno la parola "ps" (o "Postscript") nel nome o nella descrizione breve; può essere molto utile quando state cercando un programma che faccia qualcosa, ma non sapete il suo nome—o neanche se esiste!

4.3 Memorizzare le informazioni

I filtri sono molto utili quando si è utenti esperti, ma hanno un piccolo problema: come si memorizzano le informazioni? Sicuramente non vorrete scrivere tutto ogni volta che utilizzate un programma! Certo che no: Unix fornisce **file** e **directory**.

Una directory è simile ad una cartella: contiene pezzi di carta, o file (archivi). Una grossa cartella può anche contenere altre cartelle—le directory possono essere contenute in altre directory. In Unix, la struttura delle directory e dei file è chiamata filesystem. Inizialmente, il filesystem consiste in una sola directory, chiamata directory di "root". Dentro questa directory ci sono altre directory, e dentro queste directory ci sono file e ancora altre directory.[5]

Ogni file e ogni directory ha un nome. Ha sia un nome breve, che può essere lo stesso di un altro file o directory in un altro punto del sistema, e un nome esteso, che è univoco. Un nome breve per un file potrebbe essere `joe`, mentre il suo "nome esteso" potrebbe essere `/home/larry/joe`. Il nome esteso è chiamato comunemente **path** (percorso). Il path può essere decodificato in una sequenza di directory. Per esempio, vediamo come viene letto `/home/larry/joe`:

`/home/larry/joe`
Per prima cosa, siamo nella directory root.
Questo sta per la directory chiamata `home`. Si trova dentro la directory root.
Questa e la directory `larry`, che sta dentro `home`.
`joe` e dentro `larry`. Un percorso si puo riferire sia ad una directory che ad un file, quindi `joe` puo essere sia l'uno che l'altro. Tutto quello che precede l'ultimo elemento sono directory.

Un modo semplice per visualizzare questa struttura è usare un diagramma ad albero. Per vedere un diagramma di un tipico sistema Linux, guardate la figura 4.1. Notate che questo diagramma non è completo—un sistema Linux reale ha più di 8000 file!—e mostra solo alcune delle directory più comuni. Così ci possono essere alcune directory nel diagramma che non esistono nel vostro sistema, e certamente il vostro sistema ha directory non elencate qui.

4.3.1 Vedere le directory con `ls`

Adesso che sapete che esistono i file e le directory, ci sono vari modi per manipolarli: vediamo quali sono. Il comando `ls` è uno dei più importanti: fa una **lis**ta di file. Se provate `ls` come comando, vedrete:

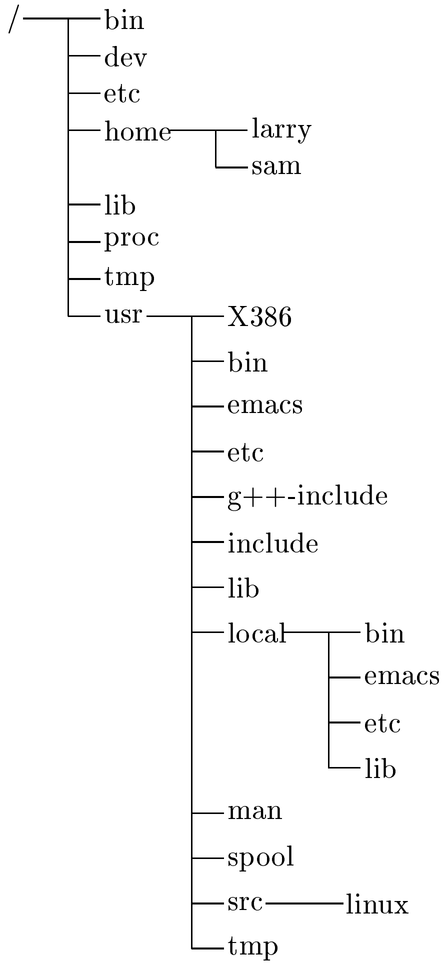
```
/home/larry$ ls
/home/larry$
```

È giusto, non vedete niente. Unix è intenzionalmente conciso: se non esistono file non dà niente, invece di dire "nessun file". Così l'assenza di dati è il modo in cui `ls` dice che non ha trovato nessun file.

---

[5] Può esserci o non esserci un limite di "profondità" in un filesystem (io non l'ho mai raggiunto—si possono facilmente avere directory nidificate 10 volte.

Figura 4.1 Un tipico albero di directory Unix.



Ma allora, se ho detto che possono esserci 8000 o più file in giro, dove sono? Arriviamo quindi al concetto di directory "corrente". Potreste vedere nel prompt che la vostra directory corrente è `/home/larry`, dove non avete ancora nessun file. Se volete la lista dei file di una directory più importante, provate con la directory di root:

```
/home/larry$ ls /
bin      etc      install  mnt      root     user     var
dev      home     lib      proc     tmp      usr      vmlinux
/home/larry$
```

Nel precedente comando, "`ls /`", la directory ("`/`") è un **parametro**. La prima parola del comando è il nome del comando, e tutto il resto sono parametri. I parametri generalmente modificano l'oggetto del comando—per `ls`, i parametri indicano di quale directory si vuole l'elenco. Alcuni comandi hanno speciali parametri chiamati **opzioni** o **switch**. Per vederlo, provate:

```
/home/larry$ ls -F /
bin/      etc/      install/  mnt/      root/     user/     var@
dev/      home/     lib/      proc/     tmp/      usr/      vmlinux
/home/larry$
```

`-F` è un'opzione, cioè un tipo speciale di parametro che comincia con un trattino e modifica il modo di agire del programma, ma non il suo oggetto. Per `ls`, `-F` è un'opzione che permette di vedere quali sono le directory, quali i file speciali e quali i file normali. Tutti quelli che terminano con una barra sono directory. Parleremo ancora delle caratteristiche di `ls` più avanti: è un programma sorprendentemente complesso!

Ora, ci sono due lezioni da imparare qui. Per prima cosa dovete imparare quello che fa `ls`. Provate un po' di directory mostrate nella figura 4.1, e guardate cosa contengono. Naturalmente alcune saranno vuote, e alcune avranno molti, molti file. Suggerisco di provare `ls` sia con sia senza l'opzione `-F`. Per esempio, `ls /usr/local` sarà più o meno:

```
/home/larry$ ls /usr/local
archives  bin       emacs     etc       ka9q      lib       tcl
/home/larry$
```

La seconda lezione è più generale: molti comandi Unix sono simili a `ls`, cioè hanno opzioni, che generalmente consistono di un trattino seguito da un carattere, e parametri. Diversamente da `ls`, alcuni comandi *richiedono* determinati parametri e/o opzioni. Per mostrare come un comando usa opzioni e parametri, useremo la seguente forma:

`ls` [-arF] [*directory*]

Questo è un modello di comando e lo vedrete ogni volta che viene introdotto un nuovo comando. La prima parola è il comando stesso (in questo caso `ls`). Dopo il comando ci sono i parametri; quelli opzionali vengono racchiusi tra parentesi quadre ("[" e "]"). Le meta-variabili sono *inclinate*—sono parole che vanno sostituite con i parametri reali (ad esempio, qui sopra vedete *directory*, che va sostituito con il nome di una directory reale).

Le opzioni sono un caso speciale. Vengono racchiuse tra parentesi, ma se ne può usare una sola senza usarle tutte. Ad esempio, con le tre sole opzioni date qui sopra per `ls`, si hanno otto diversi modi per dare il comando: con o senza ciascuna delle opzioni. (Paragonate `ls -R` con `ls -F`.)

4.3.2 La directory corrente e `cd`

`pwd`

Usare le directory sarebbe scomodo se doveste scrivere il percorso completo ogni volta che voleste accedere ad una directory. Invece, le shell Unix hanno una caratteristica chiamata directory “corrente” o “di lavoro”. Il vostro setup probabilmente vi mostra la directory corrente nel prompt: `/home/larry`; se non lo fa, provate il comando `pwd`, per **p**resent **w**orking **d**irectory (visualizza la directory di lavoro). Talvolta il prompt vi mostrerà il nome della macchina: una cosa del genere è utile solo in un ambiente di rete con molte macchine diverse.

```
mousehouse>pwd
/home/larry
mousehouse>
```

`cd` [*directory*]

Come potete vedere, `pwd` mostra la vostra directory corrente[6]—un comando molto semplice. Molti comandi agiscono, per default, nella directory corrente. Per esempio, `ls` senza nessun parametro ne mostra il contenuto. Possiamo cambiare la directory corrente usando il comando `cd`: per esempio, provate:

```
/home/larry$ cd /home
/home$ ls -F
larry/    sam/      shutdown/  steve/    user1/
/home$
```

Se omettete il parametro opzionale *directory*, ritornerete alla vostra home directory, o directory di origine, altrimenti `cd` vi porterà nella directory specificata. Per esempio:

```
/home$ cd
/home/larry$ cd /
/$ cd home
```

---

[6] Troverete tutti i termini in questo libro: directory corrente, directory di lavoro. Preferisco “directory corrente”, sebbene a volte verranno usate le altre forme per motivi stilistici.

```
/home$ cd /usr
/usr$ cd local/bin
/usr/local/bin$
```

Come potete vedere, `cd` vi permette di usare percorsi sia assoluti sia relativi: un percorso "assoluto" inizia con `/` e specifica tutte le directory prima della directory cercata, un percorso relativo è preso in relazione alla directory corrente. Nell'esempio precedente, quando ero in `/usr`, ho fatto uno spostamento relativo a `local/bin`—`local` è una directory dentro `usr`, e `bin` è una directory dentro `local`.

Ci sono due directory usate *solamente* nei percorsi relativi: "`.`" e "`..`". La directory "`.`" si riferisce alla directory corrente e "`..`" è la directory madre, cioè quella che contiene la directory corrente. Sono directory "scorciatoia": esistono in *ogni* directory, ma non seguono il concetto di "cartella in cartella". Anche la directory di root ha una directory madre—è la directory di root stessa!

Il file `./chapter-1` è il file chiamato `chapter-1` nella directory corrente. Occasionalmente è necessario inserire il "`./`" per far funzionare alcuni comandi, sebbene questo sia raro. In molti casi, `./chapter-1` e `chapter-1` sono la stessa cosa.

La directory "`..`" è molto utile per tornare indietro nell'albero:

```
/usr/local/bin$ cd ..
/usr/local$ ls -F
archives/  bin/       emacs@     etc/       ka9q/      lib/       tcl@
/usr/local$ ls -F ../src
cweb/      linux/     xmris/
/usr/local$
```

In questo esempio, sono passato nella directory madre usando `cd ..` e ho fatto l'elenco della directory `/usr/src` da `/usr/local` usando `../src`. Notate che se fossi stato in `/home/larry`, inserire `ls -F ../src` non sarebbe andato bene!

La directory `~/` è equivalente alla vostra home directory:

```
/usr/local$ ls -F ~/
/usr/local$
```

Potete vedere che non c'è niente nella vostra home directory. `~/` diverrà più utile quando impareremo come manipolare i file.

4.3.3 Creare e distruggere le directory

`mkdir` *directory1* [*directory2* ... *directoryN*]

Creare directory è estremamente semplice sotto Unix, e può essere un ottimo strumento di organizzazione. Per creare una nuova directory, usate il comando `mkdir`. `mkdir` sta per **make dir**ectory (crea directory).

Vediamo un piccolo esempio per vedere come lavora questo comando:

```
/home/larry$ ls -F
/home/larry$ mkdir report-1993
/home/larry$ ls -F
report-1993/
/home/larry$ cd report-1993
/home/larry/report-1993#
```

`mkdir` può gestire più di un parametro, e ne interpreta ognuno come una directory da creare. Si possono specificare sia percorsi assoluti sia relativi; nell'esempio precedente, `report-1993` è un percorso relativo.

```
/home/larry/report-1993$ mkdir /home/larry/report-1993/chap1 ~/report-1993/chap2
/home/larry/report-1993$ ls -F
chap1/  chap2/
/home/larry/report-1993$
```

L'opposto di `mkdir` è `rmdir` per **re**mo**v**e **dir**ectory (rimuovi directory). `rmdir` funziona esattamente come `mkdir`.

Un esempio di `rmdir` è:

```
/home/larry/report-1993$ rmdir chap1 chap3
rmdir: chap3: No such file or directory
/home/larry/report-1993$ ls -F
chap2/
/home/larry/report-1993$ cd ..
/home/larry# rmdir report-1993
rmdir: report-1993: Directory not empty
/home/larry$
```

Come potete vedere, `rmdir` si rifiuta di rimuovere directory che non esistono, come anche directory che contengono qualcosa (ricordatevi che `report-1993` ha una subdirectory all'interno, `chap2`!). C'è un'altra cosa interessante su `rmdir`: cosa succede se tentate di rimuovere la vostra directory corrente? Proviamo:

```
/home/larry$ cd report-1993
/home/larry/report-1993$ ls -F
chap2/
/home/larry/report-1993$ rmdir chap2
/home/larry/report-1993$ rmdir .
rmdir: .: Operation not permitted
/home/larry/report-1993$
```

Un'altra situazione che potreste considerare è cosa succede se provate a rimuovere la madre della directory corrente. In effetti questo non è un problema: la madre della directory corrente non è vuota, e quindi non può essere rimossa!

4.4 Spostare Informazioni

Tutte queste directory sono bellissime, ma non ci servono a niente, se non come posto dove immagazzinare i dati. Gli Dei dello Unix hanno visto questo problema, e l'hanno risolto dando agli utenti i file.

Impareremo a creare e modificare i file nei prossimi capitoli.

I principali comandi per manipolare i file in Unix sono `cp`, `mv` e `rm`. Rispettivamente questi stanno per **c**o**p**y (copia), **m**o**v**e (sposta) e **r**e**m**ove (rimuovi).

4.4.1 `cp` come un amanuense

`cp` [-i] *origine destinazione*
`cp` [-i] *file1 file2 ... fileN directory di destinazione*[7]

`cp` è uno strumento molto utile sotto Unix, ed estremamente potente: permette ad una persona di copiare più informazioni in un secondo di un monaco amanuense in un anno.

[7] `cp` nel suo modello ha due linee perché il significato del secondo parametro può essere diverso a seconda del numero di parametri.

Fate attenzione con `cp` se non avete molto spazio nel disco. Nessuno vuole vedere il messaggio `Error saving--disk full` mentre sta copiando dei dati importanti. `cp` può anche sovrascrivere i file esistenti senza avvisare—parlerò di questi pericoli più avanti.

Per prima cosa parleremo della prima linea del modello del comando: il primo parametro di `cp` è il file da copiare, il secondo è dove copiarlo. Si può copiare con un nuovo nome o su una diversa directory. Proviamo alcuni esempi:

```
/home/larry# ls -F /etc/passwd
/etc/passwd
/home/larry# cp /etc/passwd .
/home/larry# ls -F
passwd
/home/larry# cp passwd frog
/home/larry# ls -F
frog  passwd
/home/larry#
```

Con il primo `cp` ho copiato il file `/etc/passwd`, che contiene i nomi di tutti gli utenti del sistema Unix e le loro password (criptate), nella mia home directory. `cp` non elimina il file sorgente; in questo modo non ho fatto niente che possa danneggiare il sistema. Così adesso esistono due copie di `/etc/passwd` nel mio sistema, entrambe chiamate `passwd`, una nella directory `/etc` e una in `/home/larry`.

Poi ho creato una *terza* copia di `/etc/passwd` quando ho scritto `cp passwd frog`—le tre copie sono: `/etc/passwd`, `/home/larry/passwd` e `/home/larry/frog`. Il contenuto di questi tre file è lo stesso, anche se hanno nomi diversi.

`cp` può copiare file tra directory se il primo parametro è un file e il secondo una directory; in questo caso, il nome breve del file resta lo stesso.

Può copiare un file e cambiarne il nome se entrambi i parametri sono nomi di file. Questo è un pericolo di `cp`: se io avessi scritto `cp /etc/passwd /etc/group`, `cp` avrebbe creato un nuovo file con contenuto identico a `passwd` e l'avrebbe chiamato `group`, ma se `/etc/group` esisteva già, `cp` avrebbe distrutto il vecchio file senza darvi la possibilità di salvarlo! (Non avrebbe nemmeno stampato un messaggio facendovi presente che stavate distruggendo un file copiandocene un altro sopra!)

Vediamo un altro esempio di `cp`:

```
/home/larry# ls -F
frog  passwd
```

```
/home/larry# mkdir passwd_version
/home/larry# cp frog passwd passwd_version
/home/larry# ls -F
frog        passwd       passwd_version/
/home/larry# ls -F passwd_version
frog  passwd
/home/larry#
```

Come ho usato `cp`? Evidentemente `cp` accetta *più* di due parametri (è la seconda linea nel modello del comando). Quello che ha fatto il comando precedente è stato copiare tutti i file elencati (`frog` e `passwd`) e metterli nella directory `passwd_version`. In effetti `cp` può prendere un numero qualsiasi di parametri, interpretando i primi $n-1$ parametri come file da copiare, e l'n^{mo} come directory in cui copiarli.

Non si possono rinominare i file quando se ne copiano più di uno alla volta—mantengono sempre il loro nome. Questo porta ad una domanda interessante: cosa succede se scrivo `cp inittab pippo pluto`, dove `inittab` e `pippo` esistono e `pluto` non è una directory? Provate e vedrete.

4.4.2 Eliminare file con `rm`

`rm` [-i] *file1 file2 . . . fileN*

Adesso che abbiamo imparato a creare milioni di file con `cp` (e credetemi, troverete presto nuovi modi per creare altri file), è utile imparare anche come eliminarli. È molto semplice, il comando che state cercando è `rm`, e funziona proprio come state pensando: qualsiasi file che date come parametro a `rm` viene cancellato.

Per esempio:

```
/home/larry$ ls -F
inittab     pippo     rc_version/
/home/larry$ rm inittab pippo pluto
rm: pluto: No such file or directory
/home/larry$ ls -F
rc_version/
/home/larry$
```

Come potete vedere, `rm` non è molto amichevole: non solo non chiede conferma, ma elimina file anche se l'intera riga non è corretta, che può essere pericoloso. Considerate la differenza tra questi due comandi:

```
/home/larry$ ls -F
pippo  pluto/
/home/larry$ ls -F pluto
pippo
/home/larry$ rm pluto/pippo
/home/larry$
```

e questo

```
/home/larry$ rm pluto pippo
rm: pluto is a directory
/home/larry$ ls -F
pluto/
/home/larry$
```

Come potete vedere, cambiare *un solo* carattere comporta un esito molto diverso del comando. È vitale controllare il comando inserito prima di battere Invio.

4.4.3 Una scorciatoia molto utile

`mv` [-i] *vecchio-nome nuovo-nome*
`mv` [-i] *file1 file2 ... fileN nuova-directory*

Infine, l'altro comando che dovrete conoscere è `mv`. `mv` è simile a `cp`, tranne che elimina il file originale dopo averlo copiato. È quindi molto simile all'uso di `cp` e `rm` insieme. Vediamo cosa possiamo fare:

```
/home/larry$ cp /etc/inittab .
/home/larry$ ls -F
inittab
/home/larry$ mv inittab pippo
/home/larry$ ls -F
pippo
/home/larry$ mkdir report
```

```
/home/larry$ mv pippo report
/home/larry$ ls -F
report/
/home/larry$ ls -F report
pippo
/home/larry$
```

Come potete vedere, `mv` rinomina un file se il secondo parametro è un file. Se il secondo parametro è una directory, `mv` sposta il file nella nuova directory mantenendo lo stesso nome breve.

Dovete fare molta attenzione con `mv`—non controlla per vedere se il file di destinazione esiste già, e rimuoverà tutti i file che si trova tra i piedi. Per esempio, se ho già un file chiamato `pippo` nella mia directory `report`, il comando `mv pippo report` eliminerà il file `~/report/pippo` e lo sostituirà con `~/pippo`.

In effetti, c'è un modo per far sì che `rm`, `cp` e `mv` chiedano conferma prima di eliminare file. Tutti e tre i comandi accettano l'opzione `-i`, con la quale chiedono conferma all'utente prima di cancellare qualsiasi file. Se usate un **alias**, potete fare in modo che la shell faccia `rm -i` automaticamente quando digitate `rm`. Impareremo più su questo argomento nella Sezione 9.1.3 a pagina 116.

Capitolo 5

Il sistema X Window

Questo capitolo si applica solamente a chi usa il sistema X Window. Se vi trovate con uno schermo pieno di finestre, a colori o con un cursore che si sposta solo con il mouse, state usando X11 (se il vostro schermo consiste di caratteri bianchi su sfondo nero, non state usando X. Se lo volete avviare, date un'occhiata alla Sezione 5.1).

5.1 Avviare e fermare il sistema X Window

5.1.1 Avviare X

Anche se X non parte automaticamente quando vi loggate, è possibile avviarlo dal normale prompt della shell testuale. Ci sono due comandi possibili per avviare X, `startx` o `xinit`; provate prima `startx`: se la shell dice che non trova il comando, provate ad usare `xinit` e guardate se viene avviato X. Se nessuno dei due funziona, probabilmente X non è installato nel vostro sistema—consultate la documentazione locale della vostra distribuzione.

Se il comando funziona ma alla fine vi ritrovate con lo schermo nero con il prompt della shell, X è installato ma non configurato. Consultate la documentazione della vostra distribuzione per sapere come fare.

5.1.2 Uscire da X

A seconda di come X è configurato, ci sono due possibili modi per uscirne. Il primo si applica se il vostro gestore di finestre controlla se X viene o non viene eseguito; se è così, dovrete uscire da X usando un menù (vedere la Sezione 5.4.8 a pagina 60). Per visualizzare i menù, cliccate con un pulsante del mouse sullo sfondo.

La voce del menù che ci interessa è “Exit Window Manager” o “Exit X” o qualcosa di simile. Provate a cercarla (ricordatevi, può esserci più di un menù—provate i vari pulsanti del mouse!) e sceglietela.

L’altro metodo funziona se è un particolare `xterm` a controllare X; se è questo il caso, c’è probabilmente una finestra con titolo “login” o “system xterm”: per uscire da X, spostate il cursore in quella finestra e digitate “exit”.

Se X si è avviato automaticamente quando vi siete loggati, uno di questi metodi dovrebbe riuscire a chiudere completamente la vostra sessione. Per riaprirla, fate di nuovo il login. Se avete avviato X manualmente, questi metodi vi dovrebbero far tornare al prompt del modo testo (se volete fare logout, digitate `logout` al prompt testuale).

5.2 Cos’è il sistema X Window?

Il sistema X Window è un metodo di lavoro grafico distribuito, sviluppato originariamente al Massachusetts Institute of Technology. Da allora è stato passato ad un’associazione di venditori (appunto, l’“X Consortium”) e viene mantenuto da loro.

Il sistema X Window (da qui in poi abbreviato con “X”[1]) ha nuove versioni ogni pochi anni, chiamate release. Nel momento in cui scrivo, l’ultima versione è la X11R6, o release 6. L’undici in X11 è il numero di versione ufficiale, ma non c’è stata una nuova versione in parecchi anni, né ce n’è una in programma.

Ci sono due termini quando si parla di X che dovrebbero esservi familiari: il **client** è un programma X: per esempio, `xterm` è il client che mostra la shell quando vi loggate; il **server** è un programma che fornisce servizi ai programmi client, per esempio, il server disegna la finestra di `xterm` e comunica con l’utente.

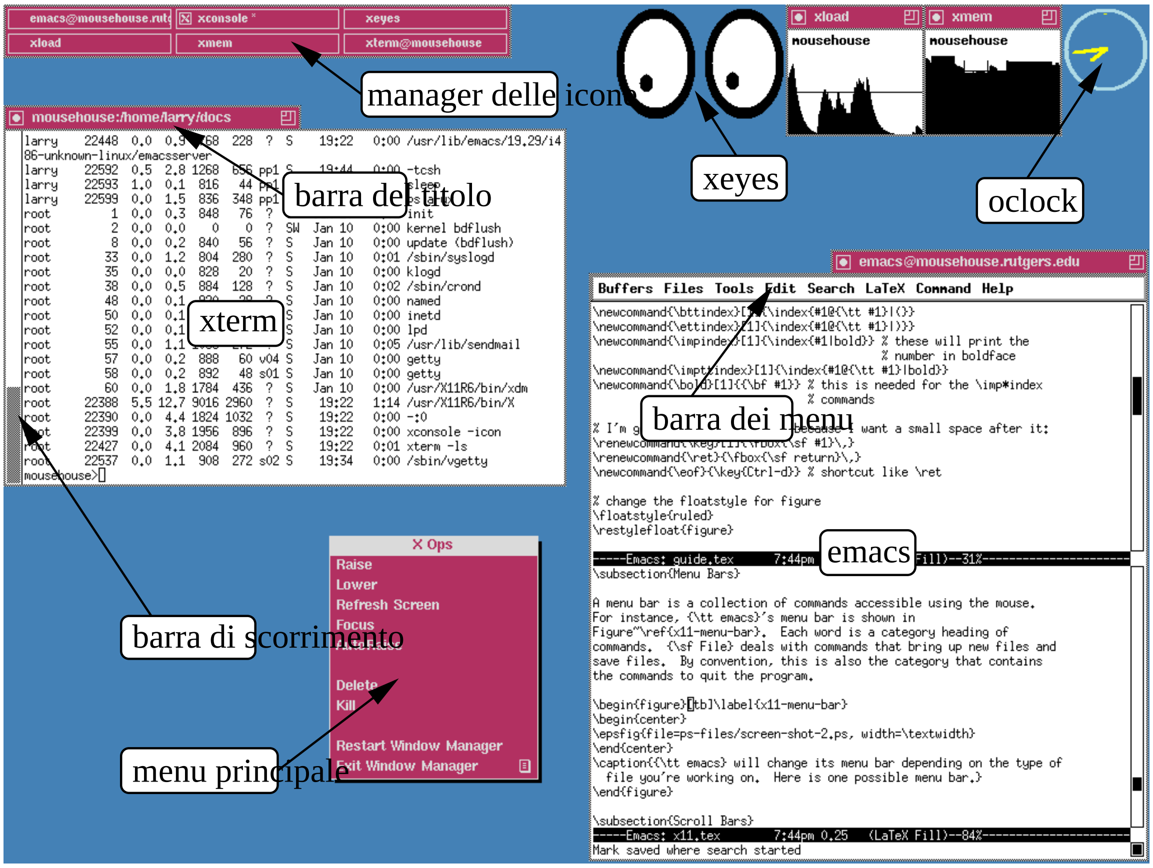
Siccome il client e il server sono due programmi separati, è possibile eseguire il client e il server *su due macchine fisicamente separate.* Oltre a fornire metodi standard per eseguire grafica, potete anche eseguire un programma su una macchina remota (anche oltreoceano, se volete!) e averlo visualizzato sulla stazione di lavoro che avete di fronte.

Un terzo termine che dovrebbe esservi familiare è il **gestore delle finestre**, o **window manager**. Il gestore delle finestre è uno speciale client che comunica al server dove posizionare le varie finestre e fornisce all’utente i mezzi per spostarle sullo schermo. Il server, da solo, non fa niente per l’utente, ma fornisce solamente un appoggio tra l’utente e il client.

---

[1] Ci sono parecchi altri modi per riferirsi al sistema X Window. Un modo comune, anche se *non* corretto, è “X Windows”.

Figura 5.1 Un esempio di una schermata standard di X. In questo caso, l'utente usa `twm`. L'orologio standard è stato sostituito da uno trasparente, `oclock`.



```
larry    22448  0.0  0.9 [illegible]68  228  ?  S    19:22   0:00 /usr/lib/emacs/19.29/i4
86-unknown-linux/emacsserver
larry    22592  0.5  2.8 1268  656 pp1 S    19:44   0:00 -tcsh
larry    22593  1.0  0.1  816   44 pp1 [illegible] sleep
larry    22599  0.0  1.5  836  348 pp1 [illegible] ps [illegible]
root         1  0.0  0.3  848   76  ?  [illegible] init
root         2  0.0  0.0    0    0  ?  SW  Jan 10  0:00 kernel bdflush
root         8  0.0  0.2  840   56  ?  S   Jan 10  0:00 update (bdflush)
root        33  0.0  1.2  804  280  ?  S   Jan 10  0:01 /sbin/syslogd
root        35  0.0  0.0  828   20  ?  S   Jan 10  0:00 klogd
root        38  0.0  0.5  884  128  ?  S   Jan 10  0:02 /sbin/crond
root        48  0.0  0.1 [illegible] S   Jan 10  0:00 named
root        50  0.0  0.1 [illegible] S   Jan 10  0:00 inetd
root        52  0.0  0.1 [illegible] S   Jan 10  0:00 lpd
root        55  0.0  1.1 [illegible] S   Jan 10  0:05 /usr/lib/sendmail
root        57  0.0  0.2  888   60 v04 S   Jan 10  0:00 getty
root        58  0.0  0.2  892   48 s01 S   Jan 10  0:00 getty
root        60  0.0  1.8 1784  436  ?  S   Jan 10  0:00 /usr/X11R6/bin/xdm
root     22388  5.5 12.7 9016 2960  ?  S    19:22   1:14 /usr/X11R6/bin/X
root     22390  0.0  4.4 1824 1032  ?  S    19:22   0:00 -:0
root     22399  0.0  3.8 1956  896  ?  S    19:22   0:00 xconsole -icon
root     22427  0.0  4.1 2084  960  ?  S    19:22   0:01 xterm -ls
root     22537  0.0  1.1  908  272 s02 S    19:34   0:00 /sbin/vgetty
mousehouse>
```

```
\newcommand{\bttindex}[1]{\index{#1@{\tt #1}|(}}
\newcommand{\ettindex}[1]{\index{#1@{\tt #1}|)}}
\newcommand{\impindex}[1]{\index{#1|bold}} % these will print the
                                           % number in boldface
\newcommand{\impttindex}[1]{\index{#1@{\tt #1}|bold}}
\newcommand{\bold}[1]{{\bf #1}} % this is needed for the \imp*index
                                % commands

% I'm g[illegible] because I want a small space after it:
\renewcommand{\key}[1]{\fbox{\sf #1}\,}
\renewcommand{\ret}{\fbox{\sf return}\,}
\newcommand{\eof}{\key{Ctrl-d}} % shortcut like \ret

% change the floatstyle for figure
\floatstyle{ruled}
\restylefloat{figure}

-----Emacs: guide.tex      7:44pm [illegible]Fill)--31%--------------------
\subsection{Menu Bars}

A menu bar is a collection of commands accessible using the mouse.
For instance, {\tt emacs}'s menu bar is shown in
Figure~\ref{x11-menu-bar}.  Each word is a category heading of
commands.  {\sf File} deals with commands that bring up new files and
save files.  By convention, this is also the category that contains
the commands to quit the program.

\begin{figure}[tb]\label{x11-menu-bar}
\begin{center}
\epsfig{file=ps-files/screen-shot-2.ps, width=\textwidth}
\end{center}
\caption{{\tt emacs} will change its menu bar depending on the type of
  file you're working on.  Here is one possible menu bar.}
\end{figure}

\subsection{Scroll Bars}
-----Emacs: x11.tex        7:44pm 0.25   (LaTeX Fill)--84%--------------------
Mark saved where search started
```

5.3 Cos'è questa roba sul mio schermo?

Quando avviate X, vengono avviati parecchi programmi: per primo viene avviato il server, poi normalmente diversi client. Sfortunatamente questo passaggio non è standardizzato nelle varie distribuzioni. È probabile che tra questi client ci sia un gestore di finestre, `fvwm` o `twm`, un prompt, `xterm`, e un orologio, `xclock`.

5.3.1 XClock

`xclock` [-digital] [-analog] [-update *secondi*] [-hands *colore*]

Parlerò prima del più semplice: `xclock` funziona esattamente come vi aspettate. Segna i secondi, i minuti e le ore in una piccola finestra.

Nessun tipo di interazione con mouse e tasti nella finestra di `xclock` ha effetto su esso—questo è *tutto* quello che fa. In effetti ci sono varie opzioni che si possono dare al programma per farlo funzionare in modi diversi: per esempio, `xclock -digital` crea un orologio digitale, `xclock -update 1` crea una lancetta dei secondi che si muove ogni secondo, e `-update 5` crea una lancetta dei secondi che si muove ogni 5 secondi.

Per maggiori informazioni sulle opzioni di `xclock`, consultate la sua pagina man—`man xclock`. Se volete provare ad eseguire un po' di `xclock`, dovreste probabilmente leggere la sezione 6.4 (Multitasking) per imparare come farli girare in aggiunta ai vostri normali programmi (se avviate un `xclock` in primo piano—il modo normale di avviare un programma—e ve ne volete sbarazzare, digitate ctrl-c).

5.3.2 XTerm

Una finestra con un prompt all'interno (qualcosa probabilmente simile a: `/home/larry#`) è controllata da un programma chiamato `xterm`. `xterm` è un programma stranamente complicato. A prima vista non sembra fare granché, ma in realtà svolge moltissimi compiti: `xterm` emula un terminale in modo che le applicazioni Unix in modo testo vi possano lavorare correttamente, e mantiene un buffer di informazioni, in modo che ci si possa riferire ai comandi dati in precedenza (per scoprire come usarlo, guardate la Sezione 5.6.3).

Per la maggior parte di questo libro impareremo ad usare Unix da riga di comando, e questo si può fare da dentro la finestra `xterm`. Per interagire con `xterm`, *normalmente* dovete spostare il puntatore del mouse (generalmente a forma di "X" o di freccia) nella finestra di `xterm`, ma questo comportamento dipende dal gestore delle finestre.

Uno dei modi per avviare un programma da dentro X è attraverso un `xterm`. Dato che i programmi X sono programmi Unix standard, anche loro possono essere avviati da dentro un normale prompt di comandi come gli `xterm`. Far girare un programma a lungo termine da un `xterm` bloccherebbe l'`xterm` finché il programma non finisce, quindi normalmente i programmi X si avviano in background. Per altre informazioni su questo, vedere la Sezione 6.4.

5.4 I gestori di finestre

In Linux vengono comunemente usati due diversi gestori di finestre: il primo, `twm`, che è l'abbreviazione di "Tab Window Manager", è più grande dell'altro che viene generalmente usato, `fvwm` (`fvwm` sta per "F(?) Virtual Window Manager"—l'autore ha trascurato di indicare per cosa stia esattamente la f). Sia `twm` sia `fvwm` sono altamente configurabili, e questo significa che non posso dirvi esattamente cosa fanno i vari tasti nella vostra particolare configurazione.

Per imparare la configurazione di `twm`, guardate la sezione 9.2.1. La configurazione di `fvwm` è spiegata nella sezione 9.2.2.

5.4.1 Quando vengono create nuove finestre

Un gestore di finestre ha tre possibilità quando viene creata una nuova finestra: lo si può configurare in modo che sia mostrato il contorno della finestra stessa e si possa posizionarla sullo schermo ("posizionamento manuale"), oppure è possibile che posizioni da solo la nuova finestra da qualche parte sullo schermo ("posizionamento casuale"); infine, alcune applicazioni richiedono un posizionamento particolare sullo schermo, oppure è possibile che il gestore di finestre sia configurato per mostrare determinate applicazioni sempre nello stesso posto dello schermo (per esempio, posso specificare che `xclock` appaia sempre nell'angolo in alto a destra).

5.4.2 Mettere a fuoco

Il gestore di finestre controlla alcune cose importanti; la prima cosa che vi dovrebbe interessare è il "focus" (messa a fuoco). La finestra messa a fuoco è quella che riceve i comandi che inserite da tastiera; generalmente in X è determinata dalla posizione del cursore del mouse: se questo si trova sopra una delle finestre di `xterm`[2], quell'`xterm` riceverà tutto quello che inserirete da tastiera. Notate che ciò è diverso da quello che succede con altri sistemi operativi a finestre, come Macintosh, OS/2 o Microsoft Windows, dove per mettere a fuoco una finestra dovete cliccarci con il mouse: di solito, sotto X, se il cursore del mouse finisce fuori dalla finestra, il fuoco viene perso e non potrete più digitarci dentro.

Notate, comunque, che è possibile configurare sia `twm` sia `fvwm` per fare in modo che dobbiate cliccare su una finestra per metterla a fuoco, e cliccare altrove per perdere il fuoco, proprio come per Microsoft Windows. Scoprite come è configurato il vostro gestore di finestre provando o consultate la documentazione locale.

---

[2] Potete avere più di una copia di `xterm` in esecuzione nello stesso tempo!

5.4.3 Spostare le finestre

Un'altra cosa molto configurabile in X è come spostare una finestra. Nella mia configurazione personale di `twm`, ci sono tre modi diversi per spostare una finestra: quello più ovvio è spostare il cursore del mouse sulla **barra del titolo**, e trascinare la finestra per lo schermo; sfortunatamente si può configurare il sistema per farlo con uno qualsiasi dei pulsanti (sinistro, destro e centrale)[3] (per trascinare, spostate il cursore del mouse sopra la barra del titolo, e tenete premuto il pulsante *mentre* spostate il mouse). Probabilmente, il vostro sistema è configurato per spostare le finestre usando il pulsante *sinistro*.

Un altro modo per spostare le finestre è di tener premuto un tasto mentre si trascina il mouse; per esempio, nella *mia* configurazione, se tengo premuto il tasto Alt e sposto il cursore su una finestra, la posso trascinare usando il pulsante sinistro del mouse.

Nuovamente, potete riuscire a capire come è configurato il gestore di finestre per tentativi o guardando la documentazione locale. Oltre a questo, se volete provare ad interpretare il file di configurazione, guardate la sezione 9.2.1 per `twm`, o la sezione 9.2.2 per `fvwm`.

5.4.4 Profondità

Siccome le finestre possono sovrapporsi, in X c'è il concetto di **profondità**. Anche se le finestre e lo schermo sono entrambi bidimensionali, una finestra può essere davanti ad altre, oscurandole parzialmente o completamente.

Ci sono diverse operazioni che permettono di gestire la profondità:

- **Raise** della finestra, per portarla in primo piano. Di solito lo si fa cliccando sulla barra del titolo con uno dei pulsante del mouse. A seconda di come è configurato il gestore di finestre, può essere uno qualsiasi dei tre (è anche possibile che si possa fare con più di un pulsante).
- **Lower** della finestra, per mandarla sullo sfondo. Si può fare con un diverso click sulla barra del titolo. È anche possibile configurare alcuni gestori in modo che un click mandi la finestra avanti se c'è qualcosa sopra, e la mandi dietro se è in primo piano.
- **Cycling** delle finestre. È un'altra operazione permessa da molti gestori: ogni finestra viene portata in primo piano in successione.

---

[3] Molti PC hanno mouse con solo due pulsanti. In questi casi è possibile simulare il tasto centrale usando contemporaneamente il destro e il sinistro.

5.4.5 Iconizzazione

Ci sono parecchie altre operazioni che possono oscurare o nascondere completamente le finestre; la prima è l'"iconizzazione"; a seconda del gestore, si può fare in vari modi: in `twm`, molti configurano un **gestore di icone** (**icon manager**): si tratta di una finestra speciale che contiene un elenco di tutte le altre finestre sullo schermo; cliccando su un nome (a seconda dell'impostazione, si può fare con un pulsante qualsiasi del mouse!) la finestra scompare—è iconificata; è ancora attiva, ma non potrete vederla. Un altro click nel gestore di icone ripristina la finestra sullo schermo.

Questo procedimento è abbastanza utile; per esempio, potreste avere `xterm` remoti su alcuni computer che usate solo occasionalmente; dato che raramente vengono usati tutti nello stesso momento, si possono tenere molte finestre di `xterm` iconificate, e lavorare solo con alcune. L'unico problema è che è semplice "perdersi" le finestre; si creano così sempre nuove finestre che duplicano le funzionalità di quelle già presenti, ma che ci si dimentica di avere iconificate.

Altri gestori di finestre possono creare icone lungo il margine inferiore dello schermo, o lasciarle sparse nella finestra principale.

5.4.6 Ridimensionare

Ci sono altri modi diversi per ridimensionare le finestre sotto X. Di nuovo, il modo usato dipende dal gestore, e da come è configurato esattamente. Il metodo più familiare agli utenti di Microsoft Windows è di cliccare e trascinare il bordo della finestra. Se il vostro gestore di finestre crea dei bordi spessi che fanno cambiare l'aspetto del cursore del mouse quando ci passa sopra, probabilmente questo è il metodo da usare per ridimensionare le finestre.

Un altro metodo è di creare un pulsante di "ridimensionamento" sulla barra del titolo. Nella Figura 5.3, si vede un pulsantino sulla destra di ogni barra del titolo; per ridimensionare le finestre, si sposta il mouse su questo pulsante e si preme il pulsante destro del mouse. A questo punto si può spostare il mouse fuori dei bordi della finestra per ridimensionarla, quando avete raggiunto la dimensione voluta, lasciate il pulsante del mouse.

5.4.7 Massimizzazione

Un'altra operazione supportata dalla maggior parte dei gestori di finestre è la massimizzazione. In `twm`, per esempio, si può massimizzare la larghezza o l'altezza, o entrambe le dimensioni della finestra: questa operazione è chiamata "zoom" nel linguaggio di `twm`, anche se io preferisco il termine massimizzazione. Diverse applicazioni rispondono in modo diverso al cambiamento della loro dimensione (per esempio, `xterm` non ingrandisce il carattere, ma rende disponibile uno spazio di lavoro maggiore!).

Sfortunatamente non è affatto sempre uguale il modo per ingrandire una finestra.

5.4.8 Menù

Un altro scopo del gestore di finestre è fornire menù all'utente per velocizzare i compiti ripetuti più frequentemente. Per esempio, si può creare una voce di menù che lancia automaticamente Emacs o un altro `xterm`: in questo modo non è più necessario inserire comandi in un `xterm`—ottimo specialmente se non ci sono `xterm` in esecuzione!

In generale, è possibile accedere a diversi menù cliccando nella finestra principale, che è la finestra immobile dietro a tutte le altre. Per default è colorata di grigio, ma potrebbe essere diversa.[4] Per provare a vedere un menù, cliccate e tenete premuto un pulsante sulla scrivania. Dovrebbe apparire un menù a tendina. Per fare una selezione, spostate (senza rilasciare il pulsante) il cursore del mouse su una voce e quindi lasciatelo.

5.5 Attributi di X

Ci sono vari programmi che usano X. Alcuni, come `emacs`, possono funzionare sia in modalità testo, *sia* creando la propria finestra. Comunque, la maggior parte dei programmi per X funzionano solo sotto X.

5.5.1 Geometria

Ci sono alcune cose comuni a tutti i programmi che funzionano sotto X. In X, la **geometria** di una finestra è la posizione e la dimensione della finestra stessa. La geometria di una finestra ha quattro componenti:

- La dimensione orizzontale, che di solito viene misurata in pixel (un pixel è la più piccola unità che può essere colorata. Molte configurazioni sui PC Intel hanno 1024 pixel in orizzontale e 768 pixel in verticale). Alcune applicazioni, come `xterm` e `emacs`, misurano le applicazioni in termini di numero di caratteri che entrano nella finestra (ad esempio, si può avere una larghezza di 80 caratteri).

- La dimensione verticale, anch'essa misurata in pixel. Anche questa può essere misurata in caratteri.

---

[4] Un programma divertente da provare si chiama `xfishtank`. Vi mette sullo sfondo un piccolo acquario.

- La distanza orizzontale da uno dei lati dello schermo. Ad esempio, `+35` posizionerà il lato sinistro della finestra a 35 pixel dal lato sinistro dello schermo. D'altra parte, `-50` posizionerà il lato destro della finestra a 50 pixel dal lato destro dello schermo. In genere è impossibile posizionare la finestra fuori dello schermo al suo avvio, anche se in seguito ci si può spostare (fanno eccezione le finestre molto grandi).
- La distanza verticale dal lato superiore o inferiore dello schermo. Una distanza positiva viene misurata dal lato superiore dello schermo, ed una negativa da quello inferiore.

Tutte le quattro componenti vengono unite in una stringa di geometria di questo genere: `503x73-78+0` (in questo caso la finestra è larga 503 pixel, alta 73 pixel, ed è posizionata vicino all'angolo in alto a destra dello schermo). Un altro modo di indicarla è: *dim-oriz*x*dim-vert*±*pos-oriz*±*pos-vert*.

5.5.2 Display

Ogni applicazione X ha associato un display. Il display è il nome dello schermo controllato dal server X, e consiste di tre componenti:

- Il nome della macchina su cui gira il server. Sulle macchine Linux stand-alone il server gira sullo stesso sistema dei client. In tali casi, il nome della macchina può essere omesso.
- Il numero del server che gira su quella macchina. Dato che ogni macchina può avere server X multipli che ci girano sopra (cosa strana per la maggior parte delle macchine Linux, ma possibile), ognuno deve avere un numero identificativo univoco.
- Il numero dello schermo. X supporta un server particolare che controlla più di uno schermo alla volta. Potete immaginare qualcuno che abbia bisogno di molto spazio schermo, e che abbia quindi due monitor uno accanto all'altro. Dato che per motivi di performance non è indicato avere due server sulla stessa macchina, un solo server controlla i due monitor.

Queste tre cose vengono messe insieme in questo modo: *macchina:numero-server.numero-schermo*.

Ad esempio, su `mousehouse`, tutte le mie applicazioni hanno il display impostato a `:0.0`, che sta per il primo schermo del primo server sul display locale. Comunque, se uso un computer remoto, il display sarà impostato a `mousehouse:0.0`.

Figura 5.2 Opzioni standard per i programmi per X.

Nome	Seguito da	Esempio
`-geometry`	geometria della finestra	`xterm -geometry 80x24+0+90`
`-display`	display su cui si vuole far apparire il programma	`xterm -display lionsden:0.0`
`-fg`	colore primario del primo piano	`xterm -fg yellow`
`-bg`	colore primario dello sfondo	`xterm -bg blue`

Per default, il display viene preso dalla variabile d'ambiente (vedere la Sezione 9.1.4) `DISPLAY`, che può essere superata usando un'opzione da linea di comando (vedere la Figura 5.2). Per vedere come impostare la variabile d'ambiente `DISPLAY`, provate il comando `echo $DISPLAY`.

5.6 Caratteristiche comuni

Anche se X è un'interfaccia grafica utente, è molto irregolare: èE impossibile sapere a priori come funziona uno qualsiasi dei suoi componenti, perché ognuno può essere riconfigurato, modificato e persino sostituito. Ne abbiamo già incontrato un caso: i vari gestori di finestre, ciascuno dei quali è altamente configurabile.

Un'altra causa di questa irregolarità è il fatto che le applicazioni per X sono costruite usando delle cose che si chiamano "insiemi di widget". Nelle distribuzioni standard sono inclusi gli "Athena widget" sviluppati all'MIT, che vengono comunemente usati nelle applicazioni liberamente distribuibili; hanno lo svantaggio che non sono particolarmente belli a vedersi e sono in qualche modo più difficili da usare di altri widget.

L'altro insieme di widget conosciuto è "Motif". Motif è un insieme di widget commerciale, simile all'interfaccia utente usata per Microsoft Windows. Molte applicazioni commerciali usano gli widget Motif, come anche alcune applicazioni liberamente distribuibili. Il noto Web browser `netscape` usa Motif.

Proviamo a vedere gli elementi più comuni che si incontrano.

5.6.1 Pulsanti

I pulsanti sono in genere le cose più facili da usare. Si richiama un pulsante posizionandoci sopra il cursore del mouse e cliccando (premendo e lasciando immediatamente il tasto del

Figura 5.3 `emacs` cambierà la sua barra dei menù a seconda del tipo di file su cui state lavorando. Ecco una possibile barra dei menù.

mouse) sul tasto sinistro del mouse. I pulsanti di Athena e di Motif sono funzionalmente la stessa cosa, anche se hanno un aspetto diverso.

5.6.2 Barre dei Menu

Una barra dei menù è un insieme di comandi accessibili usando il mouse. Come esempio, la barra dei menù di `emacs` è mostrata in Figura 5.3. Ogni parola è il titolo di una categoria di comandi. File comprende i comandi che aprono file nuovi e salvano i file. Per convenzione, è anche la categoria che contiene il comando per uscire dal programma.

Per accedere ad un comando, spostate il cursore del mouse su una determinata categoria (come File), premete il tasto sinistro del mouse e tenetelo premuto. Verrà mostrata una varietà di comandi. Per sceglierne uno, spostateci sopra il cursore e lasciate il tasto sinistro. Alcune barre dei menù vi lasciano cliccare su una categoria—se è questo il caso, cliccare su una categoria terrà aperto il menù finché non cliccate su un comando, su un altro menù o fuori della barra (indicando che non vi interessa avviare un comando particolare).

5.6.3 Barre di scorrimento

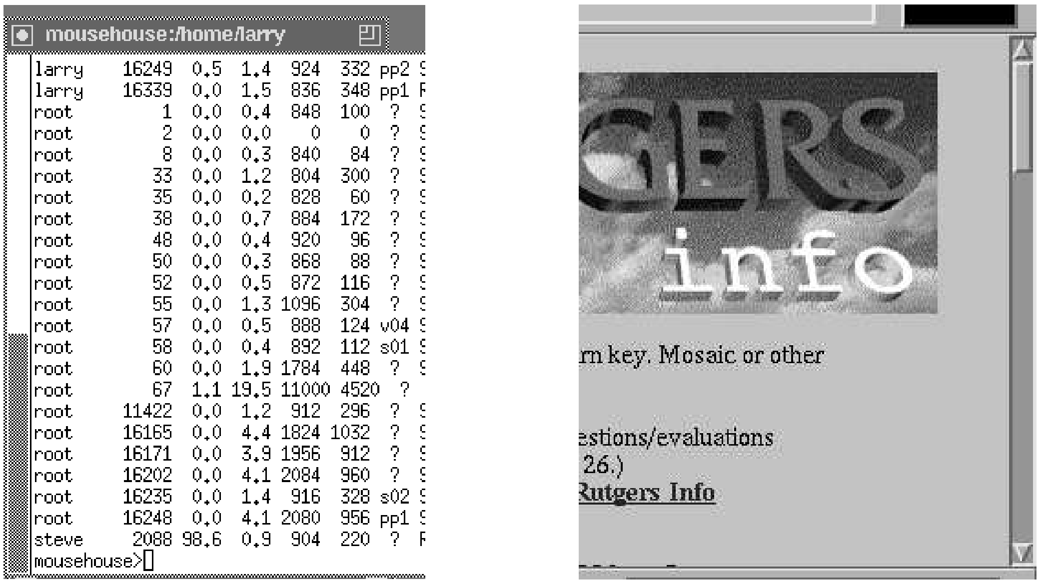
Una **barra di scorrimento** è un metodo che permette di visualizzare solo una parte del documento, mentre il resto rimane fuori dello schermo. Ad esempio, la finestra di `xterm` in Figura 5.4 mostra solo una terza parte del testo disponibile. È facile vedere quale parte del testo disponibile viene visualizzata: la parte scura della barra di scorrimento è relativa sia alla posizione che alla quantità di testo visualizzato. Se il testo viene visualizzato per intero, tutta la barra di scorrimento è scura. Se viene visualizzata la metà centrale del testo, è scura la metà centrale della barra di scorrimento.

Si può avere una barra di scorrimento verticale a sinistra o a destra del testo, e una orizzontale sopra o sotto, a seconda dell'applicazione.

Barre di scorrimento Athena

Le barre di scorrimento di tipo Athena operano in maniera diversa da quelle degli altri sistemi operativi a finestre. Per scorrere il testo verso l'alto (cioè per mostrare il materiale

Figura 5.4 A sinistra di questa finestra `xterm` è visibile una barra di scorrimento di tipo Athena. Sulla finestra di `Netscape` è visibile una barra di scorrimento di tipo Motif.



```
larry    16249  0.5  1.4  924  332 pp2
larry    16339  0.0  1.5  836  348 pp1
root         1  0.0  0.4  848  100  ?
root         2  0.0  0.0    0    0  ?
root         8  0.0  0.3  840   84  ?
root        33  0.0  1.2  804  300  ?
root        35  0.0  0.2  828   60  ?
root        38  0.0  0.7  884  172  ?
root        48  0.0  0.4  920   96  ?
root        50  0.0  0.3  868   88  ?
root        52  0.0  0.5  872  116  ?
root        55  0.0  1.3 1096  304  ?
root        57  0.0  0.5  888  124 v04
root        58  0.0  0.4  892  112 s01
root        60  0.0  1.9 1784  448  ?
root        67  1.1 19.5 11000 4520  ?
root     11422  0.0  1.2  912  296  ?
root     16165  0.0  4.4 1824 1032  ?
root     16171  0.0  3.9 1956  912  ?
root     16202  0.0  4.1 2084  960  ?
root     16235  0.0  1.4  916  328 s02
root     16248  0.0  4.1 2080  956 pp1
steve     2088 98.6  0.9  904  220  ?
mousehouse>
```

rn key. Mosaic or other

estions/evaluations
26.)
Rutgers Info

che sta sopra quello correntemente visibile) si può cliccare sul pulsante destro in qualsiasi punto della barra di scorrimento. Per scorrere verso il basso, cliccate con il pulsante sinistro in qualsiasi punto della barra.

Si può anche saltare ad un punto particolare del materiale mostrato cliccando con il pulsante centrale del mouse in un qualsiasi punto della barra di scorrimento. In questo modo nella finestra viene visualizzato il materiale che si trova a quel punto del documento.

Barre di scorrimento Motif

Le barre di scorrimento Motif funzionano più o meno come quelle del Macintosh o di Microsoft Windows. Un esempio si trova a destra in Figura 5.4. Notate che in aggiunta alla barra, ci sono delle frecce sopra e sotto la barra stessa; vengono usate per i piccoli movimenti: cliccando su una di esse con il pulsante destro o centrale del mouse si farà scorrere una piccola quantità di testo (come una linea); il pulsante sinistro non ha effetto.

Quello che succede cliccando all'interno della barra di scorrimento con le barre Motif è molto diverso rispetto alle barre Athena. Il pulsante destro non ha effetti. Cliccando con il sinistro sopra la posizione corrente il testo scorre verso l'alto. In modo simile, cliccando sotto la posizione corrente il testo scorre verso il basso. Cliccando e tenendo il pulsante sinistro *sulla* posizione corrente permette di spostare a piacere la barra; rilasciando il pulsante sinistro si posiziona la finestra.

Cliccare sul pulsante centrale in qualsiasi punto della barra farà saltare immediatamente a quella posizione, come per il pulsante centrale delle barre Athena, ma invece di iniziare a mostrare i dati alla posizione cliccata, quella posizione viene presa come *punto centrale* dei dati da mostrare.

Capitolo 6

Lavorare con Unix

Unix è un sistema molto potente per quelli che sanno come gestirlo. In questo capitolo proverò a descrivere vari modi di usare la shell di Unix, `bash`, in modo più efficiente.

6.1 Metacaratteri

Nel capitolo precedente avete imparato alcuni comandi di manutenzione dei file: `cp`, `mv` e `rm`. Può capitare che vogliate interagire con più di un file alla volta. Per esempio, potreste voler copiare tutti i file che cominciano con `data` nella directory `~/backup`: si può fare sia dando più comandi `cp`, sia elencando tutti i file in un'unica linea di comando; entrambi questi metodi, comunque, portano via molto tempo e avete molte possibilità di compiere errori.

Una via migliore per farlo è scrivere:

```
/home/larry/report$ ls -F
1993-1          1994-1          data1           data5
1993-2          data-new        data2
/home/larry/report$ mkdir ~/backup
/home/larry/report$ cp data* ~/backup
/home/larry/report$ ls -F ~/backup
data-new        data1           data2           data5
/home/larry/report$
```

Come potete vedere, l'asterisco dice a `cp` di prendere tutti i file che iniziano con `data` e copiarli in `~/backup`. Riuscite ad indovinare cosa farebbe `cp d*w ~/backup`?

6.1.1 Cosa succede *realmente*?

Buona domanda. In realtà ci sono due caratteri speciali che vengono intercettati dalla shell `bash`. Il carattere “`*`”, un asterisco, dice “sostituisci questa parola con tutti i file che verificano questa specifica”. In questo modo, il comando `cp data* ~/backup`, come nell'esempio precedente, viene cambiato in `cp data-new data1 data2 data5 ~/backup` prima di venire eseguito.

Per vedere come funziona introduciamo un nuovo comando, `echo`. `echo` è un comando estremamente semplice; rimanda sullo schermo o stampa ogni parametro. Cioè:

```
/home/larry$ echo Ciao!
Ciao!
/home/larry$ echo Come va?
Come va?
/home/larry$ cd report
/home/larry/report$ ls -F
1993-1          1994-1          data1           data5
1993-2          data-new        data2
/home/larry/report$ echo 199*
1993-1 1993-2 1994-1
/home/larry/report$ echo *4*
1994-1
/home/larry/report$ echo *2*
1993-2 data2
/home/larry/report$
```

Come potete vedere, la shell espande il metacarattere e passa tutti i file al programma che volete sia eseguito. Questo solleva una domanda interessante: cosa succede se *non* ci sono file che incontrano le specifiche del metacarattere? Provate `echo /rc/fr*og` e guardate cosa succede... `bash` passa testualmente la stringa con il metacarattere al programma.

Bisogna dire anche che altre shell, come `tcsh`, invece di passare testualmente il metacarattere, rispondono `No match`. Ecco lo stesso comando sotto `tcsh`:

```
mousehouse>echo /rc/fr*og
echo: No match.
mousehouse>
```

L'ultima cosa che chiederete è come fare se volete che sullo schermo compaia `data*`, al posto della lista dei nomi dei file? Beh, sia sotto `bash` che `tcsh`, basta racchiudere la stringa tra virgolette:

```
/home/larry/report$ echo "data*"
data*
/home/larry/report$
```

oppure

```
mousehouse>echo "data*"
data*
mousehouse>
```

6.1.2 Il punto interrogativo

Oltre all'asterisco, anche il punto interrogativo viene interpretato dalla shell come carattere speciale. Un punto interrogativo corrisponde ad uno, ed un solo carattere. Per esempio, `ls /etc/??` mostra tutti i nomi di file di due lettere nella directory `/etc`.

6.2 Risparmiare tempo con bash

6.2.1 Inserimento di linee di comando

Qualche volta vi capiterà di scrivere un lungo comando in `bash` e, prima di premere Invio, di accorgervi di un errore di battitura all'inizio della riga. Potreste cancellare tutto, tornare indietro e riscrivere tutto correttamente, ma questo richiede un bello sforzo! Potete invece premere i tasti freccia per tornare indietro, eliminare i caratteri errati, ed inserire quelli corretti.

Ci sono molti tasti speciali per modificare le linee di comando; molti sono simili ai comandi di GNU Emacs. Per esempio, Ctrl-T scambia due caratteri adiacenti.[1] Troverete molti di questi comandi nel capitolo di Emacs, il capitolo 8.

6.2.2 Completamento file e comandi

Un'altra caratteristica di `bash` è il completamento automatico della linea di comando. Diamo un'occhiata al seguente esempio di un tipico comando `cp`:

```
/home/larry$ ls -F
un-file-lunghissimo
/home/larry$ cp un-file-lunghissimo corto
/home/larry$ ls -F
corto                  un-file-lunghissimo
/home/larry$
```

---

[1] Ctrl-T significa tener premuto il tasto "Ctrl", premere il tasto "t" e quindi rilasciare il tasto "Ctrl".

È una gran fatica dover inserire ogni carattere di `un-file-lunghissimo` ogni volta che dovete accedervi. Create `un-file-lunghissimo` copiandolo da `/etc/passwd`,[2] e andiamo a ripetere lo stesso comando `cp` in un modo molto più veloce e con poche possibilità di errore.

Invece di inserire il nome per intero, digitate `cp un`, e premete e rilasciate il tasto Tab. Come per magia, il resto del nome viene mostrato nella riga di comando e potete inserire `corto`. Sfortunatamente, `bash` non può leggervi nel pensiero, e dovete inserire `corto` completamente.

Quando digitate Tab, `bash` guarda cosa avete scritto, e cerca un file che inizi nello stesso modo. Per esempio, se scrivo `/usr/bin/ema` e premo Tab, `bash` troverà `/usr/bin/emacs`, che è il solo file che comincia con `/usr/bin/ema` nel mio sistema. Comunque, se scrivessi `/usr/bin/ld` e premessi Tab, `bash` emetterebbe un suono di avviso, perché nel mio sistema ci sono tre file che cominciano con `/usr/bin/ld`: `/usr/bin/ld`, `/usr/bin/ldd` e `/usr/bin/ld86`.

Se provando il completamento automatico sentite un suono, potete premere di nuovo Tab per ottenere la lista di tutti i file che iniziano in quel modo. Così, se non siete sicuri del nome esatto di un file, potete iniziare a scriverlo e cercare in lista di nomi più ristretta.

6.3 Standard Input e Standard Output

Proviamo a risolvere un problema semplice: ottenere un listato della directory `/usr/bin`. Se facciamo `ls /usr/bin`, alcuni file scrolleranno fuori dallo schermo. Come possiamo vederli tutti?

6.3.1 Concetti di Unix

Il sistema operativo Unix rende molto semplice per i programmi usare il terminale. Quando un programma scrive qualcosa sullo schermo, sta usando una cosa che si chiama **standard output**. Lo standard output, abbreviato in stdout, è quello che usano i programmi per scrivere le informazioni all'utente. Il nome per quello che usate voi per comunicare con i programmi è **standard input** (stdin). È possibile che i programmi comunichino con l'utente senza usare standard input o output, ma la maggior parte dei comandi che descrivo in questo libro usano stdin e stdout.

Per esempio, il comando `ls` stampa il listato di una directory sullo standard output, che è normalmente "connesso" al vostro terminale. Un comando interattivo, come la shell, `bash`, legge i vostri comandi dallo standard input.

---

[2] `cp /etc/passwd un-file-lunghissimo`

È anche possibile per un programma scrivere nello **standard error**, dato che è molto semplice dirottare lo standard output altrove rispetto al terminale. Lo standard error, stderr, è quasi sempre connesso al terminale, in modo che si possano leggere i messaggi d'errore.

In questa sezione esamineremo tre modi per divertirsi con lo standard input e output: redirezione dell'input, dell'output e pipe.

6.3.2 Redirezione dell'output

Una caratteristica importante di Unix è la capacità di **redirigere** l'output. Questo vi permette, invece di vedere i risultati di un comando, di registrarli in un file o di mandarli direttamente alla stampante. Per esempio, per redirigere l'output del comando `ls /usr/bin`, metteremo un segno `>` alla fine della linea e diremo in quale file vogliamo che sia messo l'output:

```
/home/larry$ ls
/home/larry$ ls -F /usr/bin > lista
/home/larry$ ls
lista
/home/larry$
```

Come potete vedere, invece di scrivere i nomi di tutti i file, il comando crea un nuovo file nella home directory; proviamo a vederlo con il comando `cat`. Se tornate indietro, ricorderete che `cat` era un comando apparentemente inutile che copiava quello che scrivevate (lo standard input) sul terminale (standard output). `cat` può anche stampare un file sullo standard output se gli date il file come parametro:

```
/home/larry$ cat lista
...
/home/larry$
```

L'output esatto del comando `ls /usr/bin` appare nel contenuto di `lista`. Tutto bene, anche se questo non ha risolto il problema originale.[3]

Comunque, `cat` fa qualcosa di interessante quando il suo output viene rediretto. Cosa fa il comando `cat lista > nuovofile`? Normalmente, `> nuovofile` dice "prendere tutto

---

[3] Per i lettori impazienti, il comando che state cercando è `more`. Comunque ci sono ancora alcune cose da dire prima di arrivarci.

l'output del comando e metterlo in `nuovofile`". L'output del comando `cat lista` è il file `lista`. In questo modo abbiamo inventato un metodo nuovo (e non molto efficiente) per copiare i file.

E che dire del comando `cat > pippo`? `cat` da solo legge ogni linea inserita dal terminale (standard input) e stampa tutto nello standard output finché non legge Ctrl-d. In questo caso, lo standard output è stato rediretto nel file `pippo`. Adesso `cat` è servito come un rudimentale editor (programma per scrittura testi):

```
/home/larry$ cat > pippo
Oggi è una bella giornata.
premete Ctrl-d
```

Abbiamo creato quindi il file `pippo` che contiene la frase "Oggi è una bella giornata". Un altro uso del versatile comando `cat` è con**cat**enare più file insieme: `cat` stampa tutti i file che gli vengono passati come parametro, un file dopo l'altro. Così il comando `cat lista pippo` stampa il listato della directory `/usr/bin`, e poi la nostra frase. In questo modo, il comando `cat lista pippo > listaepippo` creerà un nuovo file contenente sia `lista` sia `pippo`.

6.3.3 Redirezione dell'input

Come per lo standard output, è anche possibile redirigere lo standard input. Un programma, invece di leggere dalla tastiera, legge da un file. Siccome la redirezione dell'input è collegata alla redirezione dell'output, sembra naturale che il carattere speciale per la redirezione dell'input sia `<`. Anche questo deve essere usato dopo il comando da eseguire.

La redirezione dell'input è utile generalmente se avete un file di dati e un comando si aspetta dell'input dallo standard input. La maggior parte dei comandi permettono di specificare un file su cui operare, così `<` non è usato nelle normali operazioni giornaliere come altre tecniche.

6.3.4 Soluzione: le pipe

Molti comandi Unix producono una grande quantità di informazioni: per esempio, non è strano per un comando come `ls /usr/bin` produrre più output di quello che potete vedere su una sola schermata. Per riuscire a vedere tutte le informazioni di un comando come `ls /usr/bin`, è necessario usare un altro comando Unix, chiamato `more`.[4] `more` si ferma ad ogni

---

[4] Si chiama `more` perché il prompt (messaggio) che stampava originariamente era `--more--`. In molte versioni di Linux, il comando `more` è identico ad un comando più avanzato che fa tutto quello

videata di informazioni. Per esempio, `more < /etc/rc` mostra il file `/etc/rc`, esattamente come farebbe `cat /etc/rc`, eccetto che vi permette di leggerlo. `more` permette anche la forma `more /etc/rc`, ed è questo il modo normale di invocarlo.

Comunque, questo non risolve il problema che `ls /usr/bin` mostra più informazioni di quelle che potete vedere. `more < ls /usr/bin` non funziona—la redirezione dell'input funziona solamente con i file, non con i comandi! *Potreste* fare questo:

```
/home/larry$ ls /usr/bin > temp-ls
/home/larry$ more temp-ls
...
/home/larry$ rm temp-ls
```

Unix dà un modo più pulito per fare la stessa cosa: si usa il comando `ls /usr/bin | more`. Il carattere "`|`" indica una **pipe** (conduttura). Come una conduttura dell'acqua, una pipe Unix controlla il flusso. Invece di acqua, stiamo controllando il flusso di informazioni!

Uno strumento utile con le pipe sono i programmi chiamati **filtri**. Un filtro è un programma che legge lo standard input, lo elabora in qualche modo e manda i risultati sullo standard output. `more` è un filtro—legge i dati che provengono dallo standard input e li visualizza sullo standard output una videata alla volta, permettendovi di leggere il file. `more` non è un granché come filtro, dato che il suo output non può essere usato come input per un altro programma.

Altri filtri sono i programmi `cat`, `sort`, `head` e `tail`. Per esempio, se volete leggere solo le prime dieci linee dell'output di `ls`, potete usare `ls /usr/bin | head`.

6.4 Multitasking

6.4.1 Le basi

Il **Job control** (controllo dei job) si riferisce alla possibilità di mettere i processi (essenzialmente la stessa cosa dei programmi) in background (sullo sfondo) e di riportarli in foreground (in primo piano), cioè poter eseguire un processo mentre voi state facendo qualcos'altro, ma mantenendone il controllo per potergli comunicare qualcosa o per poterlo fermare. In Unix, lo strumento principale per controllare i job è la shell—tiene traccia dei job per conto vostro, se imparate il suo linguaggio.

che fa `more` e molto di più. Come prova che i programmatori non hanno il senso dell'umorismo, l'hanno chiamato `less`!

Le due parole più importanti in questo linguaggio sono `fg`, per “foreground”, e `bg`, per “background”. Per vedere come funzionano, usiamo il comando `yes` al prompt.

```
/home/larry$ yes
```

Questo comando ha lo strano effetto di creare una lunga colonna di `y` nel bordo sinistro dello schermo, più veloce di quello che riuscite a vedere.[5] Per fermare il comando, normalmente dovete premere Ctrl-C, ma questa volta premete Ctrl-Z. Sembra che si sia fermato, e viene stampato un messaggio prima del prompt più o meno così:

```
[1]+  Stopped                    yes
```

Significa che il processo `yes` è stato **sospeso** in background. Potete farlo ripartire nuovamente con il comando `fg` al prompt, che lo riporterà in primo piano. Se volete, potete fare altro prima, mentre è sospeso. Provate alcuni `ls` o altro prima di rimetterlo in foreground.

Una volta ritornato in foreground, ricominceranno le `y`, veloci come prima. Non dovete preoccuparvi che il programma “immagazzini” delle `y` mentre il job è sospeso: quando un programma è sospeso, l’intero programma non viene più eseguito finché non lo fate ritornare in vita (e ora potete premere Ctrl-C per cancellarlo in modo definitivo, una volta che lo avete sopportato abbastanza).

Vediamo in dettaglio il messaggio lasciato dalla shell:

```
[1]+  Stopped                  yes
```

Il numero tra parentesi è il **numero del job** di questo processo, e sarà usato quando dovrete riferirvi in modo specifico a quel job (naturalmente, siccome il controllo dei job riguarda l’esecuzione di più processi, è necessario un metodo per distinguerli l’uno dall’altro). Il carattere `+` che segue le parentesi comunica che quello è il “job corrente”—cioè il job spostato per ultimo dal foreground al background. Se scrivete `fg`, metterete in foreground il job marcato con il `+` (ne parleremo ancora più avanti, quando parleremo dell’esecuzione di più job contemporaneamente). La parola `Stopped` significa che il job è sospeso: il job non è morto, ma non è più in esecuzione; Linux lo ha salvato in uno speciale stato di sospensione, pronto a tornare in azione appena qualcuno lo richieda. Infine, `yes` è il nome del processo che è stato fermato.

Prima di proseguire, cancelliamo questo job ed avviamolo in modo diverso. Il comando si chiama `kill` e può essere usato così:

---

[5] Ci sono delle buone ragioni per l’esistenza di questo strano comando. Talvolta i comandi chiedono conferma—chiedono di rispondere “yes” a delle domande. Il comando `yes` permette di automatizzare la risposta a queste domande.

```
/home/larry$ kill %1
[1]+  Stopped                 yes
```

Il messaggio che il job è sospeso è ingannevole. Per vedere se è ancora in vita (cioè in esecuzione o in stato sospeso), digitate `jobs`:

```
/home/larry$ jobs
[1]+  Terminated                 yes
```

Ecco qui! Il job è stato terminato! (È possibile che `jobs` non restituisca niente, per indicare che non ci sono jobs in esecuzione in background. Se avete appena cancellato un job, e digitando `jobs` non viene mostrato nulla, allora la cancellazione è stata compiuta con successo. Normalmente viene comunicato che il job è stato "terminato" (terminated) o "ucciso" (killed).)

Adesso avviamo `yes` nuovamente, così:

```
/home/larry$ yes > /dev/null
```

Se avete letto la sezione sulla redirezione dell'input e dell'output, saprete che questo sta mandando l'output di `yes` nel file speciale `/dev/null`. `/dev/null` è un buco nero che assorbe ogni output che gli si manda (potete immaginare che il flusso di `y` finisca fuori dal computer e scompaia in un buco nel muro, se questo vi rende più contenti).

Dopo aver digitato questo comando, non otterrete indietro il prompt, ma non vedrete nemmeno la colonna di `y`. Anche se l'output viene mandato a `/dev/null`, il job è ugualmente in esecuzione in foreground. Come al solito, potete sospendere il job premendo Ctrl-Z; fatelo adesso per ottenere di nuovo il prompt.

```
/home/larry$ yes > /dev/null
["yes" e' in esecuzione, e se premiamo ctrl-z adesso, lo sospenderemo
 e otterremo indietro il prompt. Immaginiamo che lo abbia appena fatto...]
[1]+  Stopped                 yes >/dev/null

/home/larry$
```

Hmm... c'è un modo per far sì che esso *funzioni* in background, così da lasciarci il prompt per altri lavori interattivi? Il comando da usare è `bg`:

```
/home/larry$ bg
[1]+ yes >/dev/null  &
/home/larry$
```

Adesso dovete avere fiducia in me: dopo che avete inserito `bg`, `yes > /dev/null` è tornato a funzionare nuovamente, ma questa volta in background. Infatti, se fate qualcosa al prompt, come `ls` e simili, noterete che la macchina è leggermente rallentata (convogliare un continuo flusso di caratteri richiede comunque del lavoro!). Oltre a questo, comunque, non ci sono altri effetti. Potete fare qualsiasi cosa vogliate al prompt, e `yes` continuerà a mandare il suo output nel buco nero.

Ci sono adesso due modi diversi per cancellarlo: con il comando `kill` che avete appena imparato, o rimettendo il job in foreground e premendo il tasto di interruzione (Ctrl-C). Proviamo il secondo metodo, solo per capire meglio le relazioni tra `fb` e `bg`:

```
/home/larry$ fg
yes >/dev/null

[adesso e' nuovamente in foreground. Immaginiamo che io abbia
 premuto ctrl-c per terminarlo]

/home/larry$
```

Ecco, è andato. Adesso avviamo un po' di job in contemporanea, così:

```
/home/larry$ yes  > /dev/null &
[1] 1024
/home/larry$ yes | sort > /dev/null &
[2] 1026
/home/larry$ yes | uniq > /dev/null
[e qui, premete ctrl-Z per sospenderlo]

[3]+  Stopped                 yes | uniq >/dev/null
```

La prima cosa che dovreste notare di questi comandi è il simbolo `&` alla fine delle prime due righe. Mettere un `&` dopo un comando comunica alla shell di avviarlo fin dall'inizio in background (è soltanto un modo per evitare di dover avviare il programma, premere ctrl-Z e digitare `bg`). Così abbiamo avviato questi due comandi in background. Il terzo, attualmente, è sospeso e inattivo. Potreste notare come adesso la macchina sia diventata un po' più lenta, in quanto ci sono due processi in esecuzione che richiedono un po' di tempo CPU.

Ogni processo vi ha comunicato il suo numero di job. I primi due hanno mostrato anche il loro numero di **Process IDentification**, o `PID`, subito dopo il numero del job. Normalmente non è necessario conoscere il PID dei processi, ma qualche volta può tornare utile.

Eliminiamo il secondo processo, dato che può rallentare la macchina. Potrei semplicemente scrivere `kill %2`, ma sarebbe troppo semplice. Invece, facciamo così:

```
/home/larry$ fg %2
yes | sort > /dev/null
[e poi premete ctrl-C per eliminarlo]
```

Come abbiamo appena dimostrato, `fg` prende anche i parametri che cominciano con `%`. In effetti, potevate anche digitare:

```
/home/larry$ %2
yes | sort >/dev/null
[e poi premete ctrl-C per eliminarlo]
```

Questo funziona perché la shell interpreta automaticamente un numero di job come una richiesta di mettere quel job in foreground. È possibile distinguere un numero di job da altri numeri anteponendo `%`. Adesso digitiamo `jobs` per vedere quali job sono ancora in esecuzione:

```
/home/larry$ jobs
[1]-  Running                  yes >/dev/null  &
[3]+  Stopped                  yes | uniq >/dev/null
```

Il "`-`" significa che il job numero 1 è il secondo job pronto ad essere messo in foreground, se digitate `fg` senza dare nessun altro parametro. Il "`+`" significa che il job in questione è il primo della lista—un `fg` senza parametri porterà in primo piano il job numero 3. Comunque potete farlo anche tramite il suo numero, se volete:

```
/home/larry$ fg %1
yes >/dev/null
[adesso premete ctrl-Z per sospenderlo]
```

```
[1]+  Stopped                 yes >/dev/null
```

Essere passati al job numero 1 ed averlo sospeso ha anche cambiato la priorità di tutti i vostri job. Potete vederlo con il comando `jobs`:

```
/home/larry$ jobs
[1]+  Stopped                 yes >/dev/null
[3]-  Stopped                 yes | uniq >/dev/null
```

Adesso sono entrambi sospesi (perché li avete sospesi con ctrl-z), e il job 1 è il prossimo job che viene messo in foreground per default; questo perché avete lo avete messo in foreground manualmente, e successivamente sospeso. Il "+" si riferisce sempre al job più recente che è stato sospeso dal foreground. Potete rimetterlo in esecuzione di nuovo:

```
/home/larry$ bg
[1]+ yes >/dev/null  &
/home/larry$ jobs
[1]-  Running                 yes >/dev/null
[3]+  Stopped                 yes | uniq >/dev/null
```

Notate che adesso è in esecuzione, e l'altro job è tornato indietro nell'elenco ed ha il `+`. Ora eliminiamo tutti questi job per poter usare meglio la macchina:

```
/home/larry# kill %1 %3
[3]   Terminated              yes | uniq >/dev/null
/home/larry# jobs
[1]+  Terminated              yes >/dev/null
/home/larry#
```

Dovreste vedere vari messaggi sulla terminazione dei job—nessuno muore tranquillamente, sembra. La Figura 6.1 fa un breve riassunto di tutto quello che dovreste sapere sul controllo dei job:

6.4.2 Cosa sta succedendo veramente?

È importante capire che il controllo dei job viene fatto dalla shell. Non esiste nel sistema un programma `fg`; piuttosto `fg`, `bg`, `&`, `jobs` e `kill` sono tutti comandi interni alla shell

Figura 6.1 Riassunto dei comandi e dei tasti usati nel controllo dei job.

`fg` *%job* È un comando di shell che riporta in foreground un job. Per scoprire a quale viene applicato per default, digitate `jobs` e cercate quello con il `+`.
Parametri: Numero di job opzionale. Il default è il processo identificato con il `+`.

`&` Quando viene aggiunta una `&` alla fine della linea di comando, il comando viene eseguito automaticamente in background; il processo è poi soggetto a tutti i metodi normali di controllo dei job.

`bg` *%job* È un comando di shell che fa girare in background un job sospeso. Per scoprire quale è di default, digitate `jobs` e cercate quello con il `+`.
Parametri: numero di job opzionale. Il default è il processo identificato con il `+`.

`kill` *%job PID* È un comando di shell che termina un job in background, sia sospeso che attivo. Si dovrebbe specificare sempre il numero di job, e se state usando il numero di job, ricordatevi di anteporvi un `%`.
Parametri: il numero di job (preceduto da `%`) o il PID (senza `%`). Su una stessa linea si possono specificare più job.

`jobs` Questo comando di shell elenca semplicemente alcune informazioni sui job attivi o sospesi al momento. Talvolta indica anche quelli che sono stati appena terminati.

ctrl-c È il generico carattere di interrupt. In genere, se lo digitate mentre c'è un programma che gira in primo piano, lo termina (qualche volta servono un paio di tentativi); non funziona con tutti i programmi.

ctrl-z Questa combinazione di tasti sospende un programma, anche se alcuni programmi la ignorano. Una volta che è stato sospeso, il job può essere avviato in background o ucciso.

(qualche volta `kill` è un programma indipendente, ma la shell `bash` usata da Linux ce l'ha interno). Questo è un modo logico di procedere: siccome ogni utente ha il proprio spazio per il controllo dei job, e ogni utente ha già la sua shell, è più semplice far controllare i job dell'utente alla shell. Quindi, ogni numero di job di un utente si riferisce soltanto a quell'utente: il mio job numero [1] e il tuo job numero [1] sono due processi totalmente differenti. In effetti, se siete loggati più di una volta, ogni vostra shell avrà dati diversi per controllare i job, così ogni utente può avere due job diversi con lo stesso numero in due shell diverse.

Il modo per riferirsi in modo sicuro ad un processo è mediante il numero di identificazione di processo, o Process ID (`PID`). Questo numero viene assegnato dal sistema—ogni processo ha un `PID` univoco. Due utenti diversi possono riferirsi ad un processo mediante il suo `PID` e sapere che si stanno riferendo allo stesso processo (assumendo che siano loggati nella stessa macchina!).

Diamo un'occhiata ad un altro comando per capite cosa sono i `PID`. Il comando `ps` elenca tutti i processi in esecuzione, compresa la vostra shell. Provate. Ha anche alcune opzioni: le più importanti (secondo molti) sono `a`, `u` e `x`. L'opzione `a` elencherà i processi di tutti gli utenti, non solo i vostri. L'opzione `x` mostrerà i processi che non hanno un terminale associato.[6] Infine, l'opzione `u` dà ulteriori informazioni sui processi, che spesso sono utili.

Per avere davvero un'idea di cosa sta facendo il sistema, mettete insieme tutte le opzioni: `ps -aux`. Potete vedere i processi che usano più memoria guardando la colonna `%MEM`, e più tempo di CPU sulla colonna `%CPU` (la colonna `TIME` elenca il tempo *totale* di CPU usato).

Un'altra nota veloce sui PID. `kill`, oltre ad accettare le opzioni nella forma %*numero-job*, accetta direttamente anche i PID. Così, mettete `yes > /dev/null` in background, eseguite `ps` e cercate `yes`. Quindi digitate `kill PID`.[7]

Se iniziate a programmare in C nel vostro sistema Linux, imparerete che il controllo dei job della shell è soltanto una versione interattiva delle chiamate `fork` e `execl`. È troppo complesso da vedere qui, ma potrebbe essere utile ricordarlo più avanti quando dovrete programmare e vorrete eseguire processi multipli di un singolo programma.

6.5 Console Virtuali: essere in più posti contemporaneamente

Linux supporta le **console virtuali**, che sono un modo per rendere una singola macchina simile a terminali multipli, tutti connessi ad un unico kernel Linux. Fortunatamente l'uso delle console virtuali è una delle cose più semplici di Linux: ci sono alcuni "tasti dedicati" per passare rapidamente da una console all'altra. Per provarlo, loggatevi nel sistema Linux, tenete premuto il tasto Alt sinistro e premete F2 (cioè il tasto funzione numero 2).[8]

Vi troverete di fronte ad un altro prompt di login. Niente panico: siete nella console virtuale (VC) numero 2! Loggatevi e fate qualcosa—un po' di `ls` o altro—per avere conferma che è una vera shell di login. Ora potete tornare nella VC numero 1, tenendo premuto il tasto Alt sinistro e premendo F1. Oppure potete passare ad una *terza* VC, in modo ovvio (Alt-F3).

---

[6] Principalmente sono programmi di sistema che non comunicano con l'utente attraverso la tastiera.

[7] In genere è più semplice eliminare il processo tramite il numero di job invece che con il PID.

[8] Assicuratevi di farlo dalle console in modo testo: se state eseguendo X Window o altre applicazioni grafiche, probabilmente non funzionerà, sebbene voci dicano che X Window permetta di commutare le console virtuali sotto Linux.

I sistemi Linux hanno generalmente quattro VC abilitate per default. Potete aumentarle fino a otto; come dovrebbe essere spiegato in *The Linux System Administrator's Guide*: basta editare uno o due file in `/etc`, anche se quattro dovrebbero essere sufficienti per la maggior parte delle persone.

Una volta che vi ci sarete abituati, le VC probabilmente diventeranno uno strumento indispensabile per fare più cose contemporaneamente. Per esempio, io uso spesso Emacs nella VC 1 (e faccio la maggior parte del mio lavoro qui); contemporaneamente ho un programma di comunicazione nella VC 3 (così posso scaricare file con il modem mentre lavoro, o portare avanti dei job su una macchina remota), e tengo una shell nella VC 2 in caso voglia eseguire qualcos'altro senza sospendere il lavoro sulla VC 1.

Capitolo 7

Piccoli programmi potenti

7.1 La potenza di Unix

La potenza di Unix è nascosta in piccoli comandi che non sembrano molto utili se usati da soli, ma che combinati con altri comandi (direttamente o indirettamente) rendono il sistema più potente e flessibile della maggior parte degli altri sistemi operativi. I comandi di cui vi parlerò in questo capitolo sono `sort`, `grep`, `more`, `cat`, `wc`, `spell`, `diff`, `head` e `tail`. Sfortunatamente quello che fanno non si capisce immediatamente dai loro nomi.

Vediamo ogni programma separatamente e poi alcuni esempi di come usarli insieme.[1]

7.2 Operazioni sui file

In aggiunta ai comandi come `cd`, `mv` e `rm` che avete imparato nel capitolo 4, ci sono altri comandi che operano sui file ma non sui dati in essi contenuti. Questi sono, tra gli altri, `touch`, `chmod`, `du` e `df`. Tutti questi comandi non si preoccupano di cosa c'è *nel* file—cambiano solamente alcune delle cose che Unix ricorda relativamente al file.

Ecco alcuni degli elementi che vengono manipolati da questi comandi:

- Il time stamp. Ogni file ha tre date ad esso associate:[2] la data di creazione (quando è stato creato il file), la data di ultima modifica (quando il file ha subìto l'ultimo

---

[1] Notare che il breve sommario dei comandi alla fine di questo capitolo non è completo. Consultate le pagine man dei comandi se volete sapere tutte le opzioni.

[2] I vecchi filesystem in Linux memorizzavano soltanto una data, in quanto derivavano da Minix. Se avete uno di questi filesystem, alcune delle informazioni potrebbero non essere disponibili—le operazioni saranno in genere le stesse.

aggiornamento) e la data dell'ultimo accesso (quando il file è stato letto l'ultima volta).

- Il proprietario. Ogni file in Unix è di proprietà di un utente del sistema.
- Il gruppo. Ogni file ha anche un gruppo di utenti a cui è associato: quello più comune per i file utente è chiamato `users`, ed è generalmente condiviso da tutti gli account utente del sistema.
- I permessi. Ogni file ha permessi (a volte chiamati "privilegi") associati ad esso, che comunicano a Unix chi può accedere a quel file o modificarlo o, in caso di un programma, eseguirlo. Ognuno di questi permessi può essere impostato separatamente per il proprietario, il gruppo e tutti gli altri utenti.

`touch` *file1 file2 ... fileN*

`touch` aggiorna i time stamp dei file elencati nella linea di comando all'ora corrente. Se un file non esiste, `touch` lo creerà vuoto. È anche possibile specificare la data che `touch` deve impostare per i file—consultate la pagina man di `touch`.

`chmod` [-Rfv] *modalità file1 file2 ... fileN*

Il comando usato per cambiare i permessi di un file è `chmod`, abbreviazione di **ch**ange **mod**e. Prima di andare a spiegare come usare il comando, parliamo di quali permessi ci sono in Unix: ogni file ha un gruppo di permessi associati ad esso, che dicono a Unix se il file può essere letto, modificato o eseguito come programma (nel prossimo paragrafo parlerò di come l'utente li può impostare. Ogni programma eseguito da un utente ha gli stessi permessi dell'utente stesso; questo può essere un problema di sicurezza se non sapete cosa fa un particolare programma).

Unix riconosce tre tipi di persone: prima il proprietario del file (e la persona autorizzata ad usare `chmod` su quel file). Secondo, il gruppo. Il gruppo di molti dei vostri file potrebbe essere "users", che indica gli utenti normali del sistema (per vedere il gruppo di un particolare file, usare `ls -l` *file*). Poi c'è chiunque altro non sia il proprietario e non sia membro del gruppo, appropriatamente chiamato "others" ("altri").

Così un file potrebbe avere i permessi di lettura e scrittura per il proprietario, di lettura per il gruppo e nessun permesso per gli altri. Oppure, per altre ragioni, un file potrebbe essere leggibile/scrivibile per il gruppo e gli altri, ma *non* per il proprietario!

Proviamo ad usare `chmod` per cambiare un po' di permessi. Prima, create un nuovo file usando `cat`, `emacs` o qualsiasi altro programma. Normalmente dovreste riuscire a leggere e scrivere questo file (i permessi dati agli altri utenti variano a seconda di come è impostato il sistema e il vostro account). Assicuratevi di poter leggere il file usando `cat`. Adesso proviamo a togliere il vostro permesso di lettura usando `chmod u-r` *nomefile* (il parametro `u-r` si legge "user meno read", cioè toglie all'utente – user – il permesso di lettura – read); se adesso tentate di leggere il file, otterrete l'errore di `Permission denied` (accesso negato)! Aggiungete il permesso di lettura con `chmod u+r` *nomefile*.

I permessi delle directory usano la stessa struttura: lettura, scrittura ed esecuzione, ma agiscono un po' diversamente. Il permesso di lettura consente all'utente (o gruppo o altri) di leggere la directory—cioè di elencare i nomi dei file. Il permesso di scrittura consente all'utente (o gruppo o altri) di aggiungere o rimuovere file. Il permesso di esecuzione consente all'utente di accedere ai file della directory o di qualsiasi sottodirectory (se un utente non ha il permesso di esecuzione in una directory, non può farci neanche `cd`!).

Per usare `chmod`, sostituite *modalità* con quello che volete modificare, **u**ser (proprietario), **g**roup (gruppo), **o**ther (altri) o **a**ll (tutti) e cosa farci (cioè usate il simbolo `+` per aggiungere il permesso, il simbolo `-` per toglierlo oppure il simbolo `=` per specificare esattamente i permessi da assegnare). I possibili permessi da aggiungere sono **r**ead (lettura), **w**rite (scrittura) ed e**x**ecute (esecuzione).

L'opzione `R` di `chmod` cambia i permessi di una directory e di tutti i file e sottodirectory di quella directory ('R' sta per ricorsiva). L'opzione `f` forza `chmod` a provare a cambiare i permessi, anche se l'utente non è il proprietario del file (se `chmod` ha l'opzione `f`, non stamperà messaggi di errore quando fallisce nel tentativo di modificare i permessi di un file). L'opzione `v` rende `chmod` prolisso—spiegherà tutto quello che sta facendo.

7.3 Statistiche del sistema

I comandi di questa sezione mostreranno statistiche sul sistema operativo o su una sua parte.

`du` [-abs] [*percorso1 percorso2 . . . percorsoN*]

`du` sta per **d**isk **u**sage (utilizzo del disco), e controlla la quantità di spazio disco che una data directory *e tutte le sue sottodirectory* occupano nel disco. `du` da solo rende una lista di quanto spazio usano le sottodirectory della directory corrente, e riporta alla fine quanto spazio sta usando la directory corrente (più tutte le sue sottodirectory già considerate). Se gli date alcuni parametri, conterà lo spazio usato da questi file o directory al posto della directory corrente.

L'opzione `a` mostra il conteggio per i file insieme alle directory. L'opzione `b` mostra i totali in byte invece che in kilobyte (1024 caratteri). Un byte è equivalente ad una lettera in un documento di testo. L'opzione `s` mostra solo le directory elencate nella riga di comando e *non* le loro sottodirectory.

`df`

`df` sta per "disk filling" ("riempimento del disco"): riassume la quantità di spazio disco usato. Per ogni filesystem (ricordatevi, si hanno diversi filesystem sia su diverse unità disco sia su diverse partizioni) mostra lo spazio disco totale, lo spazio usato, lo spazio disponibile e la capacità del filesystem usato.

Una strana cosa che si può incontrare è che è possibile che la capacità superi il 100%, oppure che lo spazio usato più quello disponibile non sia uguale allo spazio totale. Questo perché Unix riserva dello spazio su ogni filesystem per `root`. In questo modo, se un utente accidentalmente riempie il disco, il sistema avrà ancora una zona per continuare a funzionare.

Per la maggior parte delle persone, `df` non ha nessuna opzione utile.

`uptime`

Il programma `uptime` fa esattamente quello che si potrebbe pensare: stampa il tempo trascorso da quando il sistema è in funzione—il tempo dall'ultimo avvio di Unix.

`uptime` dà anche l'ora corrente e il carico medio, che è il numero medio di job eseguiti in un certo periodo di tempo. `uptime` mostra il carico medio dell'ultimo minuto, di cinque minuti e di dieci minuti. Un carico medio vicino allo zero indica che il sistema è relativamente scarico, uno vicino all'unità indica che il sistema è utilizzato pienamente ma non ancora sovraccarico. Alti carichi medi sono il risultato di parecchi programmi in esecuzione simultaneamente.

Sorprendentemente, `uptime` è uno dei pochi programmi Unix che sono *senza* opzioni!

`who`

`who` mostra gli utenti collegati al sistema e quando si sono loggati. Con i parametri `am i` (come in: `who am i`, "chi sono io?"), mostra l'utente corrente.

`w` [-f] [*nome-utente*]

Il programma `w` mostra gli utenti collegati al sistema e quello che stanno facendo (semplicemente combina le funzionalità di `uptime` e `who`. L'intestazione di `w` è esattamente la stessa di `uptime`, e ogni linea mostra un utente, quando si è loggato (e per quanto tempo è disoccupato – idle). `JCPU` è il totale del tempo di CPU usato da quell'utente, mentre `PCPU` è il tempo totale di CPU usato da quello che sta facendo in quel momento.

Con l'opzione `f`, `w` mostra il sistema remoto da cui sono collegati gli utenti, se c'è. I parametri opzionali restringono la visualizzazione di `w` al solo utente nominato.

7.4 Cosa c'è nel file?

Ci sono due comandi importanti di Unix che servono ad elencare file: `cat` e `more`. Ho parlato di entrambi nel capitolo 6.

`cat` [-nA] [*file1 file2 . . . fileN*]

`cat` non è un comando amichevole—non aspetta che voi leggiate il file, ed è spesso usato in congiunzione con le pipe; ha però alcune utili opzioni da linea di comando: per esempio, `n` numera tutte le linee nel file, e `A` mostra i caratteri di controllo come caratteri normali invece di mostrare (forse) strane combinazioni sullo schermo (ricordatevi, per vedere queste strane e forse "poco utili" opzioni, usate il comando `man`: `man cat`). `cat` accetta input dallo stdin se non viene specificato nessun file nella linea di comando.

`more` [-l] [+*numero-linea*] [*file1 file2 . . . fileN*]

`more` è molto più utile, ed è il comando che preferirete usare quando dovrete leggere un file di testo ASCII. L'unica opzione interessante è `l`, che dice a `more` che non siete interessati a trattare il carattere Ctrl-L come carattere di "nuova pagina". `more` partirà da uno specifico numero di linea.

Siccome `more` è un comando interattivo, ho riassunto i principali comandi da dargli:

Barra spaziatrice Mostra la prossima schermata di testo.

d Fa scorrere lo schermo di 11 linee, o circa metà di uno schermo normale (25 linee).

/ Cerca un'espressione regolare. Siccome un'espressione regolare può essere abbastanza complicata, potete digitare una stringa di testo da ricercare. Per esempio, `/rana` Invio ricerca la prossima parola "rana" nel file corrente. Una barra seguita da Invio ripete la ricerca dell'ultima espressione.

`n` Anche questo ricerca la successiva posizione dell'espressione.

`:` `n` Se specificate più di un file nella linea di comando, vi sposterà nel prossimo file.

`:` `p` Vi sposterà nel file precedente.

`q` Esce da `more`.

`head` [-*linee*] [*file1 file2 . . . fileN*]

`head` mostra le prime dieci linee dei file elencati, o le prime dieci linee dello stdin se non è specificato nessun file nella linea di comando. Ogni opzione numerica viene interpretata come il numero di linee da stampare, così `head -15 pippo` stampa le prime quindici linee del file `pippo`.

`tail` [-*linee*] [*file1 file2 . . . fileN*]

Come `head`, `tail` mostra solo una frazione del file. Naturalmente, `tail` mostra la fine del file, o le ultime dieci linee che arrivano dallo stdin. Anche `tail` accetta opzioni che specificano il numero di linee.

`file` [*file1 file2 . . . fileN*]

Il comando `file` prova ad identificare in quale formato è scritto un particolare file. Dato che non tutti i file hanno estensioni o altri marcatori che li identificano, il comando `file` esegue dei controlli rudimentali per cercare di indovinare cosa contengono.

Fare attenzione, comunque, perché è possibile che `file` faccia un'identificazione sbagliata.

7.5 Comandi di elaborazione

Questa sezione discute i comandi che alterano un file, ci eseguono una determinata operazione o mostrano statistiche su di esso.

`grep` [-nvwx] [-*numero*] *espressione* [*file1 file2 . . . fileN*]

Uno dei comandi più utili di Unix è `grep`, il **g**eneralized **r**egular **e**xpression **p**arser (analizzatore generalizzato di espressioni regolari). È un nome buffo per uno strumento che ricerca del testo in un file solamente. Il modo più semplice per usare `grep` è questo:

```
/home/larry$ cat animali
Gli animali sono creature molto interessanti. Il mio animale preferito
e' la tigre, un terrificante animale con grandi denti.
Mi piace anche il leone---non e' di certo un erbivoro!
/home/larry$ grep igre animali
e' la tigre, un terrificante animale con grandi denti.
/home/larry$
```

Uno svantaggio di questo comando è che, sebbene mostri tutte le linee contenenti una parola, non dice dove cercarla nel file—non dà nessun numero di riga: può essere utile secondo quello che state facendo. Per esempio, se state cercando gli errori nell'output di un programma, potete provare `a.out | grep error`, dove `a.out` è il nome del vostro programma.

Se siete interessati alla posizione delle corrispondenze, usate l'opzione `n` con `grep` per dirgli di stampare i numeri di riga. Usate l'opzione `v` se volete vedere tutte le linee che *non* contengono l'espressione specificata.

Un'altra caratteristica di `grep` è che verifica solo parti di una parola, come nel mio esempio precendente `igre` verifica `tigre`. Per comunicare a `grep` di verificare solo parole intere, usate l'opzione `w`; l'opzione `x` comunica a `grep` di controllare le corrispondenze di linee intere.

Se non specificate nessun file, `grep` esamina lo stdin.

`wc` [-clw] [*file1 file2 . . . fileN*]

`wc` sta per **w**ord **c**ount (conta parole). Conta semplicemente il numero di parole, linee e caratteri nel/nei file. Se non viene specificato nessun file nella linea di comando, opera sullo stdin.

I tre parametri, `clw`, stanno rispettivamente per **c**aratteri, **l**inee e **w**ord (parole), e comunicano a `wc` quali dei tre conteggiare. Così, `wc -cw` conta il numero di caratteri e di parole, ma non il numero di linee. Per default, `wc` conta tutto—parole, linee e caratteri.

Un uso semplice di `wc` è trovare quanti file sono presenti in una directory: `ls | wc -w`. Se volete vedere quanti file finiscono `.c`, provate `ls *.c | wc -w`.

`spell` [*file1 file2 . . . fileN*]

`spell` è un comando Unix di controllo ortografico molto semplice, generalmente per l'inglese americano.[3] `spell` è un filtro, come molti dei programmi di cui abbiamo parlato, che legge un file di testo ASCII e scrive tutte le parole che considera errate. `spell` opera sui file elencati nella linea di comando, o, se non ne vengono dati, sullo stdin.

Probabilmente nella vostra macchina è disponibile anche un programma di ortografia più sofisticato, `ispell`. `ispell` offre la possibilità di correggere l'ortografia ed ha una simpatica interfaccia a menù se viene specificato un file nella linea di comando, oppure funziona come filtro se non viene specificato nessun file.

Anche se operare con `ispell` dovrebbe essere abbastanza banale, consultate la pagina man se avete bisogno di maggiori informazioni.

`cmp` *file1* [*file2*]

`cmp` "**comp**ares" (confronta) due file. Il primo deve essere specificato nella linea di comando, mentre il secondo può essere specificato nella linea di comando o letto dallo standard input. `cmp` è molto semplice, e dice solamente dove i due file cominciano a non essere identici.

`diff` *file1 file2*

Uno dei comandi standard di Unix più complicati è chiamato `diff`. La versione GNU di `diff` ha più di venti opzioni possibili da linea di comando! È una versione di `cmp` molto più potente e mostra quali sono le differenze invece di comunicare semplicemente dove si trova la prima.

Dato che parlare in modo approfondito di `diff` esula dallo scopo di questo libro, ne mostrerò soltanto le operazioni di base. In breve, `diff` accetta due parametri (i file da confrontare) e mostra le differenze tra questi linea per linea. Per esempio:

```
/home/larry$ cat animali
Gli animali sono creature molto interessanti. Il mio animale preferito
e' la tigre, un terrificante animale con grandi denti.
Mi piace anche il leone---non e' di certo un erbivoro!
/home/larry$ cp animali pippo
/home/larry$ diff animali pippo
/home/larry$ cat cani
```

---

[3] Anche se ci sono versioni per parecchie altre lingue europee (n.d.t. anche per l'italiano), la copia nella vostra macchina Linux potrebbe essere per l'inglese americano e solamente per questo. Mi dispiace.

```
Gli animali sono creature molto nteressanti. Il mio animale preferito

e' la tigre, un terrificante animale con grandi denti.
Mi piace anche   il leone---non e' di certo un erbivoro!
/home/larry$ diff animali cani
1c1,2
< Gli animali sono creature molto interessanti. Il mio animale preferito
---
> Gli animali sono creature molto nteressanti. Il mio animale preferito
>
3c4
< Mi piace anche il leone---non e' di certo un erbivoro!
---
> Mi piace anche   il leone---non e' di certo un erbivoro!
/home/larry$
```

Come potete vedere, `diff` non stampa niente quando i due file sono identici; quando confronto due file diversi, ho un'intestazione di sezione, `1c1,2`, che indica che stava confrontando la linea 1 del file di sinistra, `animali`, con le linee 1–2 di `cani` e quali differenze ha notato; quindi confronta la linea 3 di `animali` con la linea 4 di `cani`. Mentre può sembrare strano a prima vista confrontare diversi numeri di riga, è molto più efficiente di elencare ogni singola linea se c'è soltanto un ritorno a capo in più all'inizio di un file.

`gzip` [-v#] [*file1 file2 . . . fileN*]
`gunzip` [-v] [*file1 file2 . . . fileN*]
`zcat` [*file1 file2 . . . fileN*]

Questi tre programmi vengono usati per comprimere e decomprimere dati. `gzip`, o GNU zip, è il programma che legge i file originali e rende file più piccoli. `gzip` cancella i file specificati dalla linea di comando e li sostituisce con file che hanno nome identico ma con un "`.gz`" finale.

`tr` *stringa1 stringa2*

Il comando di "traduzione di caratteri" opera sullo standard input—non accetta un nome di file come parametro; invece, i suoi due parametri sono stringhe arbitrarie. Sostituisce tutte le stringhe uguali a *stringa1* nell'input con *stringa2*. In aggiunta a dei comandi semplici come `tr pippo pluto`, `tr` accetta anche comandi più complicati. Ad esempio, ecco un modo veloce di convertire i caratteri minuscoli in caratteri maiuscoli:

```
/home/larry# tr [:lower:] [:upper:]
questa e' una frase STRANA.
QUESTA E' UNA FRASE STRANA.
```

`tr` è abbastanza complesso e di solito viene usato in piccoli programmi di shell.

Capitolo 8

Modificare file con Emacs

8.1 Cos'è Emacs?

Per poter fare qualsiasi cosa con un computer, avete bisogno di un modo per inserire del testo in un file, e un modo per modificare il testo già inserito nei file; si fa con dei programmi detti **editor**. `Emacs` è uno degli editor più popolari—anche perché è molto semplice per un principiante poter lavorarci bene (l'editor classico di Unix, `vi`, è spiegato nell'appendice A).

Per imparare `emacs`, avete bisogno di un file di testo semplice (lettere, numeri e simili); copiatelo nella vostra home directory[1] (non vogliamo modificare il file originale, se esso contiene informazioni importanti), e richiamate Emacs sul file:

```
/home/larry$ emacs README
```

(Naturalmente, se volete copiare `/etc/inittab` o un qualsiasi altro file, sostituite quel nome di file a `README`. Per esempio `cp /etc/inittab ~/inittab` e poi `emacs inittab`.)

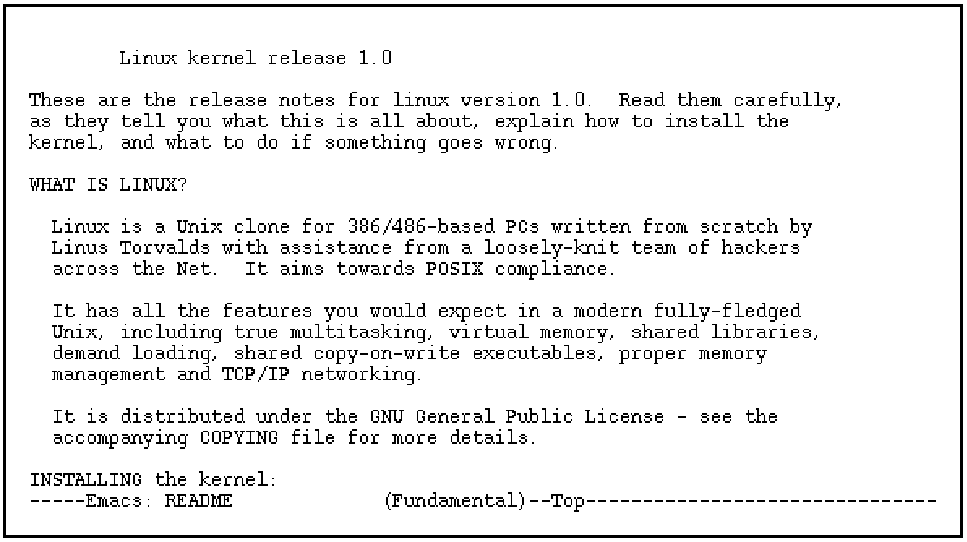
"Richiamare" Emacs può avere effetti diversi a seconda di quale ambiente state usando. Da una semplice console di testo, Emacs userà l'intera console. Se lo richiamate da X, Emacs creerà una sua propria finestra. Presumerò che lo facciate dalla console testo, ma è valido tutto anche nella versione X Window—basta sostituire la parola "schermo" con la parola "finestra". Ricordatevi anche che dovete muovere il puntatore del mouse nella finestra di Emacs per scriverci!

Lo schermo (o finestra, se state usando X) dovrebbe ora assomigliare alla figura 8.1. Gran parte dello schermo contiene il vostro documento, ma quello che vi interessa particolarmente, se volete imparare ad usare Emacs, sono le ultime due righe. La penultima linea (quella con una serie di trattini) si chiama **riga di stato**.

---

[1] Per esempio, `cp /usr/src/linux/README ./README`

Figura 8.1 Emacs appena avviato con `emacs README`



```
        Linux kernel release 1.0

These are the release notes for linux version 1.0.  Read them carefully,
as they tell you what this is all about, explain how to install the
kernel, and what to do if something goes wrong.

WHAT IS LINUX?

  Linux is a Unix clone for 386/486-based PCs written from scratch by
  Linus Torvalds with assistance from a loosely-knit team of hackers
  across the Net.  It aims towards POSIX compliance.

  It has all the features you would expect in a modern fully-fledged
  Unix, including true multitasking, virtual memory, shared libraries,
  demand loading, shared copy-on-write executables, proper memory
  management and TCP/IP networking.

  It is distributed under the GNU General Public License - see the
  accompanying COPYING file for more details.

INSTALLING the kernel:
-----Emacs: README             (Fundamental)--Top--------------------------------
```

Nella riga di stato vedrete “Top”. Potrebbe essere anche “All”, ed avere altre leggere differenze (molti hanno l’ora corrente visualizzata nella linea di stato). La riga immediatamente sotto alla riga di stato è chiamata **minibuffer**, o a volte **area eco**. Emacs usa quest’area per mandarvi dei messaggi e qualche volta, quando serve, per leggere il vostro input. Per adesso Emacs vi sta chiedendo “`For information about the GNU Project and its goals, type C-h C-p`”. Ignoratelo per adesso, non parleremo dell’uso del minibuffer per un po’.

Prima di effettuare qualsiasi modifica del testo nel file, dovete imparare come spostarvi nel testo. Il cursore dovrebbe trovarsi all’inizio del file, nell’angolo in alto a sinistra dello schermo. Per spostarvi in avanti, premete `C-f` (cioè, tenete premuto il tasto Control e premete f, per “forward”). Il cursore si sposterà di un carattere alla volta, e se mantenete premuti i tasti il sistema ripeterà automaticamente lo spostamento circa ogni mezzo secondo. Notate che quando arrivate alla fine della riga il cursore passa automaticamente nella riga sottostante. `C-b` (per “backward”) ha il significato opposto. E, visto che siamo in argomento, `C-n` e `C-p` spostano il cursore rispettivamente nella riga successiva e precedente.[2]

Usare i tasti di controllo è generalmente il modo più veloce di spostarsi mentre state scrivendo. Lo scopo di Emacs è di mantenere le vostre mani sui tasti alfanumerici della tastiera, dove viene fatta la maggior parte del lavoro. Comunque, se volete, i tasti freccia dovrebbero funzionare comunque.

---

[2] Nel caso non l’abbiate già notato molti dei comandi di spostamento di Emacs consistono nella combinazione del tasto Control con una singola lettera mnemonica.

In effetti, quando state usando X, dovreste riuscire a posizionare il puntatore del mouse e cliccare con il pulsante sinistro per spostare il cursore dove volete; questa operazione è però molto lenta—dovete portare la vostra mano sul mouse ogni volta! La maggior parte delle persone che usano Emacs usano principalmente la tastiera per spostarsi.

Usate `C-p` e `C-b` per tornare fino all'angolo superiore sinistro. Adesso mantenete premuto `C-b` per un po'; dovreste sentire il suono del cicalino e vedere il messaggio "`Beginning of buffer`" (inizio del buffer) apparire nel minibuffer. A questo punto potreste essere sorpresi: "Ma cos'è un buffer?"

Quando Emacs lavora su un file, non lavora proprio sul file, ma ne copia il contenuto in una speciale area di lavoro chiamata **buffer**, dove potete modificarlo. Quando avete finito di lavorare, comunicate ad Emacs di salvare il buffer—in altre parole, di scrivere il contenuto del buffer nel file corrispondente. Finché non lo fate, il file rimane invariato, e il contenuto del buffer esiste solamente dentro Emacs.

Con questo in mente, preparatevi ad inserire i vostri primi caratteri nel buffer. Fino ad ora, tutto quello che abbiamo fatto è stato "non-distruttivo", così questo è un grande momento. Potete scegliere qualsiasi carattere preferite, ma se volete farlo con stile, suggerisco di usare un bell'"X" maiuscolo. Mentre lo digitate, guardate l'inizio della riga di stato in fondo allo schermo. Quando modificate il buffer in modo che il suo contenuto non sia più uguale al file su disco, Emacs mostra due asterischi all'inizio della riga di stato, per farvi sapere che il buffer è stato modificato:

```
--**-Emacs: file.txt        (Fundamental)--Top-----------------------
```

Questi due asterischi vengono visualizzati non appena modificate il buffer, e rimangono visibili finché non lo salvate. Potete salvare più volte il buffer durante una stessa sessione di scrittura—il comando per farlo è `C-x C-s` (tener premuto il tasto Control e premere "x" e "s"... ok, probabilmente avrete già capito come funziona!). È volutamente semplice da digitare, perché salvare il buffer è una cosa che è meglio fare presto e spesso.

Elencherò ora alcuni comandi, insieme a quelli che avete già imparato, e che potete provare: suggerisco di familiarizzare con questi prima di andare avanti:

`C-f`	Muove avanti di un carattere.
`C-b`	Muove indietro di un carattere.
`C-n`	Va alla riga successiva.
`C-p`	Va alla riga precedente.
`C-a`	Va all'inizio della riga.
`C-e`	Va alla fine della riga.
`C-v`	Va alla pagina/schermata di testo successiva.
`C-l`	Ridisegna lo schermo, con la riga corrente al centro.
`C-d`	Elimina il carattere sotto il cursore (provate ad usarlo).
`C-k`	Elimina il testo dal cursore alla fine della riga.
`C-x C-s`	Salva il buffer nel file corrispondente.
Backspace	Elimina il carattere precedente (quello appena digitato).

8.2 Iniziare velocemente con X

Se volete solo modificare velocemente dei file ed usate X, potete farlo usando solo i menù in cima alla finestra:

Questi menù non sono disponibili in modalità testo.

Quando avviate Emacs per la prima volta avrete quattro menù in cima alla finestra: Buffers, File, Edit e Help. Per usare un menù, semplicemente spostate il puntatore del mouse sopra il nome (ad esempio File), premete e tenete premuto il pulsante sinistro del mouse, poi spostate il puntatore sul comando che volete usare e rilasciate il pulsante. Se cambiate idea, spostate il puntatore lontano dai menù e rilasciate il pulsante.

Il menù Buffers elenca i diversi file che state modificando in questa sessione di Emacs: il menù File comprende una serie di comandi per caricare e salvare file—molti di questi saranno descritti più avanti. Il menù Edit contiene alcuni comandi per modificare i buffer, e il menù Help dovrebbe dare la documentazione in linea.

Potete notare che i corrispondenti da tastiera sono elencati a destra delle voci nei menù. Dovreste cercare di impararli se contate di diventare più veloci; inoltre molte delle funzionalità di Emacs sono disponibili *solamente* attraverso la tastiera—quindi vi sarà utile leggere il resto di questo capitolo.

8.3 Modificare più file contemporaneamente

Emacs può lavorare su più di un file alla volta. In effetti, il solo limite alla quantità di buffer che Emacs può contenere dipende dalla memoria disponibile nella macchina. Il comando per aprire un nuovo file nel buffer di Emacs è `C-x C-f`. Quando lo digitate, vi verrà chiesto il nome di un file nel minibuffer:

```
Find file: ~/
```

La sintassi usata qui è la stessa usata per specificare i file dal prompt della shell; la barra rappresenta le subdirectory, `~` indica la vostra home directory. Potete ottenere anche il completamento automatico, nel senso che se quello che avete scritto al prompt è sufficiente per identificare univocamente un file, potete semplicemente premere Tab per completarlo (o per vedere i possibili completamenti, se ce ne sono più di uno). Anche Spazio ha un ruolo nel completamento dei nomi dei file, simile a Tab, ma vi lascio provare per vedere le differenze. Una volta che avete completato il nome del file nel minibuffer, premete Invio, ed Emacs vi presenterà un buffer contenente quel file. In Emacs, questo processo è noto come **finding** (ricerca) di un file. Andiamo avanti, troviamo qualche altro file di testo e carichiamolo in Emacs (fatelo dal buffer originale `file.txt`). Adesso avete un nuovo buffer; poniamo che si chiami `altro_file.txt`, dato che non posso vedere la vostra linea di stato.

Il vostro buffer originale sembra scomparso—vi starete probabilmente chiedendo dove sia finito. È ancora dentro Emacs, e potete attivarlo nuovamente con `C-x b`. Quando lo digitate, vedrete nel minibuffer una richiesta per sapere quale buffer attivare, e propone un nome di default. Il buffer di default è quello a cui passate se premete solamente Invio, senza scrivere un nome. Il buffer di default è sempre quello che avete lasciato per ultimo, così quando state lavorando molto tra due buffer, `C-x b` permette di passare per default all'altro buffer (facendovi risparmiare di scriverne il nome). Anche se il buffer di default è quello che volete, potete comunque tentare di scrivere il suo nome.

Notate che avete la possibilità di sfruttare il completamento automatico come quando cercate un file: premendo Tab viene completato quanto possibile. Ogni volta che viene chiesto qualcosa nel minibuffer, è una buona idea vedere se Emacs riesce a fare il completamento: sfruttare la possibilità del completamento quando viene offerta permette di risparmiare parecchie digitazioni. Emacs generalmente esegue il completamento quando state scegliendo una voce da una lista predefinita.

Tutto quello che avete imparato su come spostarvi e modificare il testo nel primo buffer si applica anche agli altri. Andate avanti e cambiate un po' di testo nel nuovo buffer, ma non salvatelo (cioè non premete `C-x C-s`). Assumiamo che voi vogliate annullare le vostre modifiche senza salvarle nel file; il comando per farlo è `C-x k`, che “uccide” (kill) il buffer. Digitatelo ora: per prima cosa vi viene chiesto quale buffer eliminare, ma quello di default è il buffer corrente, e di solito si elimina quello, quindi premete semplicemente Invio; poi vi viene chiesto se volete *realmente* eliminare il buffer—Emacs controlla sempre prima di eliminare un buffer che contiene modifiche non salvate. Digitate semplicemente “yes” e premete Invio, se volete eliminarlo.

Andate avanti e fate pratica caricando file, modificandoli, salvandoli e eliminando i loro buffer. Assicuratevi di non modificare nessun file importante di sistema, in modo da non provocare disastri,[3] ma cercate comunque di aprire almeno cinque buffer contemporaneamente, in modo da capire come commutare tra un buffer e un altro.

8.4 Terminare una sessione di Scrittura

Quando avete finito di lavorare con Emacs, assicuratevi che le modifiche in tutti i buffer siano state salvate, e uscite con `C-x C-c`. A volte `C-x C-c` vi pone un paio di domande nel minibuffer prima di lasciarvi—non allarmatevi, semplicemente rispondete in modo ovvio. Se pensate di dover ritornare in Emacs più tardi, non usate `C-x C-c`; usate `C-z`, che lo sospenderà. Potete tornarvi nuovamente con il comando della shell “`fg`”: è più efficiente di chiudere e far ripartire Emacs diverse volte, specialmente se dovete modificare di nuovo lo stesso file più tardi.

Sotto X, premere `C-z` iconizza semplicemente la finestra. Guardate la sezione sull'iconizzazione nel capitolo 5. Questo vi dà due modi per iconizzare Emacs, il modo solito offerto dal vostro gestore di finestre, e `C-z`. Ricordatevi, quando iconizzate, un semplice `fg` non fa riapparire la finestra—dovete ricorrere al gestore di finestre.

8.5 Il Tasto Meta

Avete già imparato un tasto “modificatore” in Emacs, il tasto Control: ne esiste un secondo, il tasto **Meta**, che è usato altrettanto frequentemente. Comunque, non tutte le tastiere hanno il tasto Meta nello stesso posto, e alcune non ce l'hanno proprio; la prima cosa che dovete fare è trovare dove si trova sulla vostra tastiera. È possibile che i tasti Alt della tastiera siano anche tasti Meta, se state usando un PC IBM o altre tastiere che hanno un tasto Alt.

Il modo per provarlo è tenere premuto un tasto che pensate possa essere un tasto Meta e

---

[3] Se non siete l'utente “root” (il supervisore) nella macchina, non dovreste avere la possibilità di danneggiare il sistema, ma fate lo stesso attenzione.

premere “x”. Se vedete apparire un piccolo prompt nel minibuffer (tipo: `M-x`) allora l’avete trovato. Per togliere il prompt e tornare al buffer di Emacs, premete `C-g`.

Se non riuscite a ottenere un prompt, allora c’è un’altra soluzione: potete usare come tasto Meta il tasto Esc, ma invece di tenerlo premuto mentre scrivete la lettera successiva, dovete premerlo e rilasciarlo rapidamente, e *successivamente* digitare la lettera. Questo metodo funziona sia che abbiate un vero tasto Meta sia che non lo abbiate, dunque è il metodo più semplice. Provate adesso a premere Esc e poi a premere “x”. Dovreste ottenere nuovamente il piccolo prompt. Usate semplicemente `C-g` per annullarlo. `C-g` è il modo più generale in Emacs per uscire da qualcosa che non volevate fare: generalmente fa un beep per avvisare che avete interrotto qualcosa, ma va bene, dato che è quello che volevate fare premendo `C-g`![4]

La notazione `M-x` è analoga a `C-x` (dove “x” è un carattere qualsiasi). Se avete trovato un tasto Meta, usate quello, altrimenti usate il tasto Esc. Scriverò semplicemente `M-x` e saprete che dovrete usare il vostro tasto Meta.

8.6 Tagliare, Incollare, Eliminare

Emacs, come tutti i buoni editor, permette di tagliare e incollare blocchi di testo. Per farlo avete bisogno di un modo per definire l’inizio e la fine del blocco: in Emacs si fa impostando due punti nel buffer, chiamati **mark** e **point**. Per impostare il mark (inizio), andate alla posizione in cui volete che inizi il blocco e premete `C-SPC` (“`SPC`” indica il tasto Spazio). Dovreste vedere il messaggio “Mark set” (Mark impostato) nel minibuffer.[5] Il mark è stato impostato in quel punto; non ci sono particolari evidenziazioni che lo indicano, ma voi sapete dove lo avete messo, ed è questo che importa.

Che dire di **point**? Beh, viene impostato ogni volta che muovete il cursore, in quanto “point” si riferisce semplicemente alla posizione corrente nel buffer. In termini più formali, point è il punto dove verrebbe inserito il testo quando digitate qualcosa. Impostando il mark e spostandovi alla fine del testo avete definito un blocco di testo, detto anche **regione**. La regione indica sempre l’area tra mark e point.

La semplice definizione di una regione non la rende disponibile per incollarla, ma dovete comunicare ad Emacs di copiarla. Per copiare la regione, assicuratevi che mark e point siano correttamente impostati e premete `M-w`. La regione è stata copiata in Emacs. Per incollarla altrove, andate in quella posizione e premete `C-y` (per “yanking”).

---

[4] Talvolta solo un `C-g` non è sufficiente per convincere Emacs che volete interrompere quello che state facendo. Ripremetelo, e Emacs vi darà ascolto.

[5] Su alcuni terminali, `C-SPC` non funziona, e dovete usare `C-@`.

Se volete spostare il testo della regione altrove, usate `C-w` al posto di `M-w`. Questo **eliminerà** la regione—tutto il testo sparirà. In effetti è stato salvato allo stesso modo di `M-w`. Potete riportarlo indietro con `C-y`, incollandolo come al solito. Il posto in cui Emacs salva tutto questo testo si chiama **kill-ring**. Alcuni editor lo chiamano "clipboard" (appunti) o "buffer di incolla".

C'è un altro modo per tagliare e incollare: ogni volta che usate `C-k` per eliminare fino alla fine della linea, il testo eliminato finisce nel kill-ring. Se eliminate più di una riga, queste vengono salvate tutte insieme nel kill-ring, in modo che la prossima volta che incollate, incollerete tutte le linee insieme. Grazie a questa caratteristica, spesso è più veloce usare ripetutamente `C-k` per eliminare del testo che impostare esplicitamente mark e point e usare `C-w`; comunque, entrambi i sistemi funzionano, ed è una preferenza personale quale dei due usare.

8.7 Cercare e Sostituire

Ci sono parecchi modi per cercare del testo in un file con Emacs; molti sono abbastanza complessi e non li vedremo qui. Il modo più semplice e divertente è usare **isearch**: "isearch" sta per "ricerca incrementale". Supponiamo che vogliate cercare la stringa "leone" nel seguente testo:

> *Gli animali sono creature molto interessanti. Il mio animale preferito è la tigre, un terrificante animale con grandi denti. Mi piace anche il leone—non è di certo un erbivoro!*

Dovete spostarvi all'inizio del buffer, o almeno in un punto che sapete precedere la prima ricorrenza della parola da cercare, "leone", e premere `C-s`: questo vi pone in modalità ricerca. Adesso iniziate a scrivere la parola che volete trovare, "leone". Appena scrivete "l", vedrete che Emacs salta alla prima ricorrenza di "l" nel testo; se il testo precedente è tutto il contenuto del buffer, allora questa è la lettera "l" della parola "Gli". Premete ora "e" di "leone", e Emacs arriverà ad "animale", che è la prima ricorrenza di "le". La "o" farà arrivare Emacs a "leone", senza farvi scrivere l'intera parola.

Quello che si fa durante una ricerca incrementale è definire una stringa da cercare. Ogni volta che aggiungete un carattere alla fine della stringa il numero di corrispondenze si riduce, finché non avrete inserito abbastanza caratteri da definire la stringa in modo univoco. Una volta trovata la corrispondenza che cercavate, potete uscire dalla modalità ricerca con il tasto Invio o qualsiasi dei tasti di movimento. Se pensate che la stringa da cercare sia indietro nel buffer, allora dovreste usare `C-r`, che esegue la ricerca nella direzione inversa.

Se incontrate una corrispondenza, ma non è quella che cercavate, premete `C-s` nuovamente mentre siete ancora in modalità ricerca: ogni volta che premerete questo tasto vi sposterete nella successiva corrispondenza completa. Se non ci sono più corrispondenze, Emacs dirà che la ricerca è fallita, ma se premete nuovamente `C-s`, la ricerca ripartirà dall'inizio del buffer. Vale anche per `C-r`—ripartirà dalla fine del buffer.

Provate a prendere un buffer di testo semplice ed eseguite la ricerca per la stringa "`gli`". Digitate innanzitutto quello che cercate, quindi usate ripetutamente `C-s` per andare a tutte le ripetizioni di "gli". Notate che questo corrisponde anche a parole che contengono la stringa "gli". Per cercare unicamente "gli", dovete aggiungere uno spazio alla fine della stringa da cercare. Potete aggiungere caratteri nuovi alla stringa in qualsiasi punto della ricerca, anche dopo aver premuto ripetutamente `C-s` per trovare le corrispondenze successive. Potete usare anche Backspace o Delete (Canc) per rimuovere caratteri dalla stringa di ricerca in qualsiasi momento; premendo Invio si esce dalla modalità ricerca, lasciandovi all'ultima corrispondenza trovata.

Emacs vi permette anche di sostituire tutte le ripetizioni di una stringa con qualche altra stringa—si chiama **cerca-sostituisci**. Per richiamarlo, digitate `M-x query-replace` e premete Invio. Il completamento viene fatto anche sul nome del comando, quindi una volta che avete inserito "query-re", potete premere Tab per completarlo. Diciamo che volete sostituire tutte le ricorrenze di "leone" con "tigre". Al prompt "`Query replace`" digitate "leone" e premete Invio; vi viene richiesto nuovamente dell'input, e dovete inserire "tigre". Emacs quindi scorrerà il testo, si fermerà ad ogni ricorrenza della parola "leone", e vi chiederà se volete sostituirla. Premete semplicemente "`y`" o "`n`" ogni volta, per "Yes" o "No", finché non finisce. Se non avete capito leggendolo, provateci.

8.8 Ma che succede veramente?

In realtà, le combinazioni di tasti che avete imparato sono scorciatoie per delle funzioni di Emacs. Per esempio, `C-p` è un modo breve per dire a Emacs di eseguire la funzione interna `previous-line` (linea precedente). Tutte queste funzioni interne possono essere richiamate anche per nome, usando `M-x`; se vi dimenticate che `previous-line` è associata a `C-p`, potete semplicemente digitare `M-x previous-line` Invio, e vi muoverete verso l'alto di una linea. Provatelo adesso, per capire come `M-x previous-line` e `C-p` sono realmente la stessa cosa.

Chi ha progettato Emacs è partito dalla base, definendo prima un intero set di funzioni interne, e poi associando delle combinazioni di tasti a quelle più utilizzate. A volte è più semplice richiamare esplicitamente una funzione con `M-x` invece di ricordarsi a quale tasto è associata. La funzione `query-replace`, per esempio, è associata in alcune versioni di Emacs a `M-%`. Ma come si fa a ricordarsi una combinazione così strana? A meno che non usiate molto spesso cerca-sostituisci, è più semplice richiamarla con `M-x`.

La maggior parte dei tasti che digitate sono lettere, che devono essere nel testo del buffer; quindi ognuno di questi tasti è **associato** alla funzione `self-insert-command`, che non fa nient'altro che inserire quella lettera nel buffer. Le combinazioni che usano il tasto Control con una lettera generalmente sono associate a funzioni che fanno altre cose, come lo spostamento. Per esempio, `C-v` è associato alla funzione chiamata `scroll-up`, che fa scorrere il buffer in alto di una schermata (in modo che la vostra posizione nel buffer si sposta in *basso*, naturalmente).

Se volete inserire un carattere Control nel buffer, come potete fare? Dopo tutto, i caratteri Control sono caratteri `ASCII`, anche se vengono usati raramente, e potreste doverli inserire in un file. C'è un modo per evitare che Emacs interpreti i caratteri Control come comandi: il tasto `C-q`[6] è associato ad una funzione speciale chiamata `quote-insert`. Tutto quello che fa `quote-insert` è leggere il tasto seguente ed inserirlo letteralmente nel buffer, senza tentare di interpretarlo come comando. Questo è il modo per inserire caratteri Control nei file con Emacs. Naturalmente per inserire un C-q bisogna premere `C-q` due volte!

Emacs ha anche molte funzioni che non sono associate a nessun tasto. Per esempio, se state scrivendo un messaggio lungo, non dovete premere Invio alla fine di ogni linea; potete farlo fare ad Emacs (gli si può far fare qualsiasi cosa)—il comando per farlo è `auto-fill-mode`, ma non è normalmente associato a nessun tasto. Per richiamare questo comando è sufficiente digitare "`M-x auto-fill-mode`". "`M-x`" è il tasto usato per richiamare le funzioni per nome. Potete usarlo anche per richiamare funzioni come `next-line` e `previous-line`, ma sarebbe molto poco efficiente, dato che già sono associate a `C-n` e `C-p`!

Ad ogni modo, se guardate la riga di stato dopo aver richiamato `auto-fill-mode`, noterete che è stata aggiunta la parola "Fill" sul lato destro. Fintanto che rimane lì, Emacs suddividerà automaticamente il testo in linee. Potete sospendere questa funzione premendo nuovamente "`M-x auto-fill-mode`"—è un comando di commutazione.

Il lato negativo di digitare lunghi nomi di funzioni nel minibuffer è minimo, dato che Emacs completa i nomi delle funzioni nello stesso modo che i nomi dei file, quindi raramente vi capiterà di dover scrivere per intero il nome della funzione. Se non siete sicuri se potete usare il completamento automatico, premete Tab. Non preoccupatevi, nel peggiore dei casi vi mostrerà l'elenco dei comandi possibili, altrimenti, se siete fortunati, completerà il comando voluto.

---

[6] Chiamiamo `C-q` "tasto", anche se è prodotto tenendo premuto Control e premendo "q", perché è un singolo carattere `ASCII`.

8.9 Chiedere aiuto ad Emacs

Emacs ha una guida molto esauriente—tanto esauriente, in effetti, che ne possiamo appena accennare il contenuto. Le caratteristiche basilari della guida sono accessibili premendo `C-h` e poi una singola lettera. Per esempio, `C-h k` fornisce la guida su un tasto (vi chiede di premere un tasto e vi dice cosa fa). `C-h t` vi porta in un breve manuale di Emacs. `C-h C-h` apre la guida della guida, per dirvi cosa è disponibile una volta premuto il tasto `C-h` la prima volta. Se sapete il nome di una funzione di Emacs (`save-buffer`, per esempio), ma non vi ricordate a quale sequenza di tasti è associata, allora usate `C-h w`, per "`where-is`" (dov'è), e digitate il nome della funzione. Oppure, se volete sapere cosa fa una funzione in dettaglio, usate `C-h f`, che vi chiede il nome di una funzione.

Ricordatevi, dato che Emacs esegue il completamento del nome delle funzioni, non dovete essere sicuri del nome di una funzione per richiedere la relativa pagina della guida: se pensate di poter indovinare come inizia il nome, digitatelo e premete Tab per vedere se il completamento fa qualcosa, altrimenti tornate indietro e riprovate con qualcos'altro. Lo stesso vale per i nomi dei file: anche se non vi ricordate tutto il nome che avete dato ad un file che non toccate da tre mesi, potete provare ad usare il completamento per trovare il nome giusto. Imparate ad usare il completamento anche per chiedere informazioni, non solo come un modo per risparmiare digitazioni.

Ci sono altri caratteri che potete digitare dopo `C-h`, e ognuno vi dà una guida di tipo diverso. Quelli che userete più spesso sono `C-h k`, `C-h w` e `C-h f`. Quando avrete più familiarità con Emacs, un altro da provare è `C-h a`, che chiede una stringa e vi comunica tutte le funzioni il cui nome contiene quella stringa ("a" sta per "apropos", a proposito di. . .).

Un'altra sorgente di informazioni è il lettore di documentazione **Info**. Info è un argomento troppo complesso da approfondire qui, ma se vi interessa esplorarlo da soli, digitate `C-h i` e leggete il paragrafo in cima allo schermo, che vi dirà come ottenere maggiore aiuto.

8.10 I buffer specializzanti: le modalità

I buffer di Emacs hanno delle **modalità** associate ad essi[7]; la ragione è che avete bisogno di cose diverse se state scrivendo una mail o, ad esempio, se state scrivendo un programma. Invece di cercare di fare un editor che venisse sempre incontro ai bisogni di tutti (che sarebbe impossibile), il progettista di Emacs[8] ha scelto di far comportare Emacs in modo diverso a seconda di cosa state facendo in ciascun buffer. Quindi, i buffer hanno delle modalità,

---

[7] Per renderla un po' più complicata, esistono delle "modalità maggiori" e delle "modalità minori", ma ancora non vi serve saperlo.

[8] Richard Stallman, noto anche come "`rms`", che è il suo nome di login.

ciascuna progettata per un'attività specifica. La caratteristica principale che distingue una modalità dall'altra sono i corrispondenti da tastiera, ma ci possono essere anche altre differenze.

La modalità più semplice è la `fondamentale`, che non ha nessun comando speciale. In effetti, ecco cosa dice Emacs della Modalità Fondamentale:

```
Fundamental Mode:

Major mode not specialized for anything in particular.
Other major modes are defined by comparison with this one.
```

Cioè:

```
(Modalita' fondamentale:

Principale modalita', non specializzata per niente in particolare.
Le altre modalita' maggiori vengono definite in confronto ad essa.)
```

Per avere queste informazioni ho fatto così: ho digitato `C-x b`, che corrisponde a `switch-to-buffer` (passa al buffer), ed ho digitato "pippo" quando mi ha chiesto il nome del buffer a cui passare. Dato che non esisteva un buffer che si chiamava "pippo", Emacs lo ha creato e ci è passato. Per default era in modalità fondamentale, ma se non lo fosse stato avrei potuto digtare "`M-x fundamental-mode`" per portarcelo. Tutti i nomi delle modalità hanno un comando "`<nome-modalità>-mode` che porta il buffer corrente in quella modalità. Poi, per avere le informazioni sulla modalità maggiore del buffer, ho digitato `C-h m`, che dà la guida sulla modalità maggiore corrente del buffer in cui vi trovate.

C'è una modalità leggermente più utile che si chiama `text-mode`, che ha i comandi speciali `M-S`, per `center-paragraph`, (centra-paragrafo), e `M-s`, che richiama `center-line` (centra-linea). `M-S`, in ogni caso, significa proprio quello che pensate: tenete premuti il tasto Meta ed il tasto Shift, e premete "S".

Non vi fidate di me—create un buffer nuovo, portatelo in modalità testo (`text-mode`), e digitate `C-h m`. Forse non capirete tutto quello che Emacs vi dirà, ma dovreste ricavarne delle informazioni utili.

Ecco un'introduzione ad alcune delle modalità usate più comunemente. Se le usate, assicuratevi di digitare `C-h m` in tutte, per avere altre informazioni su tutte le modalità.

8.11 Modlità di programmazione

8.11.1 Modalità C

Se usate Emacs per programmare in linguaggio C, gli potete far rientrare le linee automaticamente. I file con nomi che finiscono per "`.c` " o "`.h`" vengono aperti automaticamente in modalità `c-mode`, in cui sono disponibili alcuni comandi speciali, utili per scrivere programmi in C. In modalità C, Tab corrisponde al comando `c-indent-command`, cioè, premendo il tato Tab non si inserisce un carattere di tabulazione vero e proprio, ma la linea viene rientrata a seconda della sua posizione nel programma. Questo implica che Emacs conosce qualcosa della sintassi del C, ed è vero, anche se non sa niente della semantica—non può fare in modo che il vostro programma non contenga errori!

Per calcolare il rientro della linea `emacs` assume che le linee precedenti siano rientrate correttamente. Se nella linea precedente manca una parentesi tonda o graffa, un punto e virgola, o qualcos'altro del genere, Emacs rientrerà la linea in maniera sbagliata. Quando vedrete che fa così, saprete che dovete controllare la punteggiatura della linea precedente.

Potete usare questa caratteristica per controllare la punteggiatura dei programmi—invece di leggere il programma intero cercando gli errori, fate rientrare le linee dall'inizio del programma usando Tab, e quando vedete che i rientri sono strani, controllate le linee precedenti. In altre parole, fate lavorare Emacs!

8.11.2 Modalità Scheme

È una modalità maggiore che non vi servirà a niente se non avete un compilatore o un interprete per il linguaggio di programmazione Scheme. Averlo non è così frequente come, ad esempio, avere un compilatore C, ma sta diventando molto comune, quindi ne parleremo lo stesso. Molto di ciò che vale per la modalità Scheme vale anche per la modalità Lisp, se preferite scrivere in Lisp.

Beh, per farla complicata, Emacs ha due modalità Scheme diverse, perché ci sono due diverse correnti di pensiero. Quella che descriverò si chiama `cmuscheme`, e poi, nella sezione sulla personalizzazione di Emacs, parlerò di come ci possano essere due modalità Scheme diverse, e come usarle. Per adesso non vi preoccupate se quello che succede sul vostro Emacs non corriponde perfettamente a quello che dico qui. Un editor personalizzabile è per forza imprevedibile, e non c'è niente da fare!

In Emacs potete far girare un processo Scheme interattivo, con il comando `M-x run-scheme`. Questo comando crea un buffer con nome "`*scheme*`", che contiene il normale prompt di Scheme. Al prompt potete inserire espressioni Scheme, premere Invio, e Scheme le valuterà e mostrerà la risposta. Poi, per poter interagire con il processo Scheme, potete inserire tutte le definizioni di funzioni e le applicazioni che volete. Probabilmente avete già scritto del codice sorgente Scheme in qualche file, e sarebbe più facile lavorare su quel file e mandare le definizioni al buffer del processo di Scheme, se necessario.

Se i file sorgente finisce per "`.ss`" o "`.scm`", verrà aperto automaticamente in **modalità Scheme** quando lo trovate con `C-x C-f`. Se per qualche ragione non viene aperto in modalità Scheme, si può fare a mano con il comando `M-x scheme-mode`. Questa modalità `scheme-mode` non è la stessa cosa del buffer in cui gira il processo Scheme; piuttosto, il codice sorgente che si trova in `scheme-mode` ha dei comandi speciali per comunicare con il buffer del processo.

Se vi mettete all'interno di una definizione di funzione nel buffer del codice sorgente Scheme e digitate `C-c C-e`, quella definizione verrà "inviata" al buffer del processo — esattamente come se ce l'aveste digitata dentro. `C-c M-e` invia la definizione e poi vi porta nel buffer del processo per fare il lavoro interattivo. `C-c C-l` carica un file di codice Scheme (funziona sia per il buffer del processo che per il buffer del codice sorgente). Inoltre, come per le altre modalità di programmazione, digitare `Tab` in qualsiasi punto di una linea di codice fa rientrare correttamente la linea.

Se siete al prompt del buffer del processo, potete usare `M-p` e `M-n` per spostarvi tra i comandi che avete dato in precedenza (**storia di input**). Così, se state facendo il debug della funzione `'rotate'`, e l'avete già applicata a degli argomenti del buffer del processo, così:

```
> (rotate '(a b c d e))
```

potete richiamare questo comando digitando `M-p` al prompt. Non ci dovrebbe essere bisogno di ridigitare espressioni lunghe al prompt di Scheme — abituatevi ad usare la storia di input e risparmierete un sacco di tempo.

Emacs conosce un bel po' di linguaggi di programmazione: C, C++, Lisp e Scheme sono solo alcuni. In genere, sa come fare i rientri delle linee in modo intuitivo.

8.11.3 Modalità posta

Con Emacs si possono anche creare e mandare messaggi di posta elettronica. Per entrare in un buffer di posta, digitate `C-x m`. Dovete riempire i campi `To:` e `Subject:`, ed usare C-n per scendere sotto la linea separatrice e scrivere il corpo del messaggio (che inizialmente è vuoto). Non modificate o cancellate la linea separatrice, o Emacs non potrà spedire il messaggio—usa quella linea per distinguere l'intestazione del messaggio, che gli dice dove spedirlo, dal vero contenuto del messaggio stesso.

Sotto la linea separatrice potete scrivere quello che volete. Quando siete pronti a spedire, digitate `C-c C-c`, ed Emacs spedirà il messaggio e cancellerà il buffer della posta.

8.12 Come essere ancora più efficienti

Gli utenti esperti di Emacs sono fanatici dell'efficienza. In effetti, perderanno spesso un sacco di tempo per cercare dei modi per essere più efficienti! Anche se non voglio che succeda anche a voi, ecco alcuni facili suggerimenti per diventare un utente di Emacs più bravo. Talvolta gli utenti esperti fanno sentire i novellini degli scemi perché non conoscono questi trucchi—per qualche strana ragione, la gente diventa quasi "religiosa" per usare Emacs "correttamente"; condannerei questo tipo di elitarismo, se non ne fossi colpevole anche io. Ecco qui:

Quando vi spostate nel testo, usate il modo più veloce. Sapete che `C-f` sta per `forward-char` (avanti di un carattere)—riuscite ad indovinare che `M-f` sta per `forward-word` (avanti di una parola)? C-b sposta indietro di un carattere. Cosa farà `M-b`? E non è tutto: vi potete spostare in avanti di una frase alla volta con `M-e`, se alla fine di una frase mettete almeno due spazi dopo il punto finale (altrimenti Emacs non si accorge di dove finisce la frase e ne inizia un'altra). `M-a` sposta indietro di una frase.

Se vi ritrovate ad usare `C-f` multipli per arrivare alla fine della linea, vergognatevi e la prossima volta usate `C-e`, e `C-a` per andare all'inizio della linea. Se usate molti `C-n` per spostarsi in giù schermata per schermata, vergognatevi e d'ora in poi usate `C-v`. Se usate ripetutamente `C-p` per scorrere in alto di parecchie pagine, vergognatevi di farvi vedere in pubblico ed usate `M-v`.

Se siete quasi alla fine di una linea e vi accorgete che avete sbagliato qualcosa all'inizio della linea, *non usate* Backspace o Canc per tornare indietro... dovreste cancellare delle intere parole scritte bene! Invece, usate delle combinazioni di `M-b`, `C-b`, e `C-f` per spostarsi sull'errore, correggetelo, e poi usate `C-e` per spostarvi di nuovo alla fine della linea.

Quando dovete digitare il nome di un file, non inseritelo mai tutto. Digitate abbastanza caratteri per identificarlo univocamente, e fate fare il resto al completamento di Emacs premendo Tab o Spazio. Perché sprecare digitazioni se invece potete sprecare cicli di CPU?

Se state digitando del testo semplice, e il testo non vi va a capo, usate `M-q`, che corrisponde a `fill-paragraph` nelle normali modalità di testo. Vi "aggiusterà" il paragrafo come se foste andati a capo linea per linea a mano. `M-q` funziona solo da dentro il paragrafo, o dal suo inizio o dalla fine.

Qualche volta aiuta usare `C-x u`, (**undo**), che cercherà di annullare gli ultimi cambiamenti fatti. Emacs tira ad indovinare quanti passaggi annullare; di solito è molto intelligente. Richiamare il comando più volte annullerà sempre più comandi, finché Emacs non si ricorderà più quali cambiamenti sono stati fatti.

8.13 Personalizzare Emacs

Emacs è *così* grande e complesso, che ha addirittura il suo proprio linguaggio di programmazione! Non sto scherzando: per poter personalizzare veramente Emacs, dovrete scrivere dei programmi in questo linguaggio; si chiama Emacs Lisp, ed è un dialetto del Lisp; se avete precedenti esperienze in Lisp lo troverete abbastanza facile.

Altrimenti, non vi preoccupate: non entrerò molto nei particolari, perché è meglio imparare facendo. Per imparare veramente ad usare Emacs, dovreste consultare la documentazione Info su Emacs Lisp, e leggere molto codice sorgente Emacs Lisp.

La maggior parte delle funzionalità di Emacs vengono definite nei file di Emacs Lisp[9] La maggior parte di questi file vengono distribuiti con Emacs, e sono noti in generale come "libreria Emacs Lisp". Dove si trova questa libreria dipende da come Emacs è stato installato sul vostro sistema — posti comuni sono `/usr/lib/emacs/lisp`, `/usr/lib/emacs/19.19/lisp/`, eccetera. "`19.19`" è il numero di versione di Emacs, e sul vostro sistema può essere diverso.

Non avete bisogno di ricercare la libreria di Lisp nel vostro filesystem, perché Emacs ha questa informazione registrata al suo interno, nella variabile **load-path**. Per trovare il valore di questa variabile, bisogna **valutarla**, cioè bisogna far interpretare il suo valore dall'interprete Lisp di Emacs. In Emacs esiste una modalità speciale per valutare le espressioni Lisp, che si chiama **lisp-interaction-mode**. Di solito c'è un buffer che si chiama "`*scratch*`" che è già in questa modalità. Se non lo trovate, create un nuovo buffer qualsiasi, e digitateci dentro `M-x lisp-interaction-mode`.

Ora avete un'area di lavoro dove interagire con l'interprete Emacs Lisp. Digitate:

```
load-path
```

e quando avete finito premete `C-j`. Nella modalità di interazione Lisp, `C-j` è collegato a `eval-print-last-sexp`. Una "`sexp`" è un'"**espressione-s**", cioè un gruppo bilanciato di parentesi, che non ne include nessuna. Beh, così è un po' troppo semplificato, ma capirete di cosa si tratta lavorando con Emacs Lisp. Comunque, la valutazione di `load-path` dovrebbe essere una cosa del genere:

```
load-path C-j
```

---

[9] Talvolta chiamate non ufficialmente "Elisp".

```
("/usr/lib/emacs/site-lisp/vm-5.35" "/home/kfogel/elithp"
"/usr/lib/emacs/site-lisp" "/usr/lib/emacs/19.19/lisp")
```

Non sarà la stessa su tutti i sistemi, naturalmente, dato che dipende da come è stato installato Emacs. L'esempio qui sopra viene da un sistema PC 386 con Linux. Come indica il testo qui sopra, `load-path` è un elenco di stringhe; il nome di ogni stringa indica una directory che può contenere file di Emacs Lisp. Quando Emacs deve caricare un file di codice Lisp, lo cerca in tutte queste directory, in ordine; se nell'elenco c'è una directory che non esiste nel filesystem, Emacs la ignora.

Quando lo avviate, Emacs cerca automaticamente di caricare il file `.emacs` dalla vostra home directory, quindi, se volete personalizzare Emacs, dovreste mettere i comandi di personalizzazione in `.emacs`. Le personalizzazioni più comuni sono quelle dei corrispondenti da tastiera, quindi ecco come si fanno:

```
(global-set-key "\C-cl" 'goto-line)
```

`global-set-key` è una funzione di due argomenti: il tasto da collegare e la funzione a cui collegarlo. La parola "`global`" significa che il collegamento è valido in tutte le modalità maggiori (esiste un'altra funzione, `local-set-key`, che collega un tasto ad una funzione in un singolo buffer). Qui sopra, ho collegato `C-c l` alla funzione `goto-line`. Il tasto viene descritto usando una stringa. La sintassi speciale "`\C-<char>`" sta a significare che il tasto Control deve essere tenuto premuto mentre si preme il tasto `<char>`. Nello stesso modo, "`\M-<char>`" indica il tasto Meta.

Va tutto bene, ma come facevo a sapere che il nome della funzione era "`goto-line`"? Potevo sapere solo di voler far corrispondere il tasto `C-c l` ad una funzione che richiede il numero di linea e sposta il cursore su quella linea, ma come facevo a sapere come si chiamava la funzione?

È qui che ci aiuta la guida in linea di Emacs: una volta che avete deciso che tipo di funzione state cercando, potete usare Emacs per ritrovare il suo nome esatto. Ecco un modo veloce e un po' sporco per farlo: dato che Emacs usa il completamento dei nomi delle funzioni, digitate semplicemente `C-h f` (che sta per `describe-function`, e quindi dà la descrizione di una funzione), e poi digitate Tab senza scrivere niente. In questo modo chiedete ad Emacs di completare una stringa vuota, e quindi Emacs elenca tutte le funzioni! Ci vorrà un po' per fare la lista completa, dato che Emacs ha molte funzioni interne, ma quando è pronto ne mostrerà tante quante ne entrano nello schermo.

A questo punto, premete `C-g` per uscire dalla funzione di descrizione. Sarà stato creato un buffer `*Completions*`", che conterrà tutta la lista che avete generato. Passate a quel buffer. Ora potete usare `C-s` e `isearch` per cercare la funzione che vi interessa. Ad esempio, è probabile che una funzione che chiede il numero di una linea e passa a quella linea conterrà la stringa “`line`”. Quindi cercate la stringa “`line`”, e alla fine troverete quello che cercate.

Se volete usare un altro metodo, provate con `C-h a`, `command-apropos`, che mostra tutte le funzioni il cui nome contiene la stringa data. L’output di `command-apropos` è un po’ più complicato da leggere di una lista semplice, a mio parere, ma forse la penserete in modo diverso. Provate entrambi i metodi e vedete quale vi piace di più.

C’è sempre la possibilità che Emacs non abbia nessuna funzione predefinita che faccia quello che volete. In questo caso, dovrete scrivervi la funzione da soli. Non vi parlerò di come farlo in questo libro—dovreste dare un’occhiata alla libreria Emacs Lisp e cercare degli esempi di definizioni di funzioni, e leggere le pagine Info su Emacs Lisp. Se per caso conoscete qualcuno che conosce bene Emacs, chiedetegli come fare; definire le proprie funzioni di Emacs non è molto difficile—per darvi un’idea, ne ho scritte 131 nell’ultimo anno. Richiede un po’ di pratica, ma non è difficile da imparare.

Un’altra cosa che si fa spesso nel file `.emacs` è impostare alcune variabili a valori predefiniti. Per esempio, mettete questa linea nel vostro file `.emacs`, ed avviate una nuova copia di Emacs:

```
(setq inhibit-startup-message t)
```

Emacs controlla il valore della variabile `inhibit-startup-message` per decidere se mostrare o no all’avvio determinate informazioni sulla versione e sulla mancanza di garanzia. L’espressione Lisp qui sopra usa il comando `setq` per impostare questa variabile al valore ‘`t`’, uno speciale valore Lisp che significa **vero** (true). L’opposto di ‘`t`’ e ‘`nil`’, che è il valore predefinito per ‘**falso**’ nell’Emacs Lisp. Ed ecco due cosette che ho nel mio `.emacs` e che possono essere utili:

```
(setq case-fold-search nil) ; non considera le maiuscole nelle ricerche
;; fa rientrare i programmi C come piace a me:
(setq c-indent-level 2)
```

La prima espressione non fa considerare le maiuscole nell ricerche (compreso `isearch`); cioè, la ricerca darà entrambe le versioni, con lettere maiuscole o minuscole, di un carattere, anche se la stringa di ricerca contiene solo le lettere minuscole. La seconda espressione imposta i rientri per le frasi del linguaggio C un po’ più piccoli del normale—è solo un gusto personale; trovo che renda il codice C più leggibile.

Il carattere di commento nel Lisp è “`;`”. Emacs ignora qualsiasi cosa che segua un punto e virgola, così:

```
;; queste due linee vengono ignorate dall’interprete Lisp, ma
;; l’espressione seguente viene valutata appieno:
(setq una-stringa-letterale "Una strana pausa; per nessun motivo.")
```

È una buona idea commentare i vostri cambiamenti ai file Lisp, in modo che anche dopo sei mesi vi ricorderete di cosa stavate pensando mentre li modificavate. Se il commento appare da solo su una linea, anteponetevi due punti e virgola; aiuta Emacs a far rientrare correttamente i file Lisp.

Potete trovare del materiale sulle variabili interne di Emacs nello stesso modo che avete usato per le funzioni. Usate `C-h v` e `describe-variable` per creare una lista dei completamenti, oppure usate `C-h C-a apropos`. `Apropos` si distingue da `C-h a`, `command-apropos` nel fatto che mostra le funzioni e le variabili invece che solo le funzioni.

L'estensione di default per i file Emacs Lisp è "`.el`", come in "`c-mode.el`". Comunque, per rendere più veloce il codice Lisp, Emacs permette di compilarlo byte per byte; questi file di codice Lisp compilato hanno estensione "`.elc`" invece di "`.sl`". L'eccezione a questa regola è il file "`.emacs`", che non ha bisogno dell'estensione `.el` perché Emacs sa dove cercarlo all'avvio.

Per compilare un file di codice Lisp interattivamente, usate il comando `M-x load-file`. Vi chiederà il nome del file. Per caricare i file

```
(load "c-mode") ; forza Emacs a caricarlo in c-mode.el o .elc
```

Emacs aggiungerà per prima cosa l'estensione `.elc` al nome del file e proverà a cercarlo in qualche directory del `load-path`. Se non ci riesce, proverà con l'estensione `.sl`, oppure userà la stringa letterale che viene passata a `load`. Potete compilare byte per byte un file con il comando `M-x byte-compile-file`, ma se modificate il file spesso, probabilmente non vi conviene. Non dovreste mai compilare così il file `.emacs`, né dargli l'estensione `.el`.

Dopo che sarà stato caricato `.emacs`, Emacs cercherà di caricare un file con nome `default.el`. Di solito si trova in una directory, nel load-path, che si chiama `site-lisp` o `local-elisp` o qualcosa del genere (vedere l'esempio di `load-path` che ho dato qualche pagina indietro). Chi mantiene Emacs su sistemi multiutente usa il default.el per apportare cambiamenti alle versioni di Emacs di tutti, dato che l'Emacs di ciascuno si carica dal proprio file `.emacs` personale. Nemmeno `default.el` dovrebbe essere compilato, dato che normalmente viene modificato abbastanza spesso.

Se il `.emacs` di qualcuno contiene degli errori, Emacs non cerca di caricare `defaul.el`, ma si ferma, mandando un messaggio che dice "`Error in init file`" ("Errore nel file di inizializzazione"). Se vedete un messaggio del genere, probabilmente c'è qualcosa di sbagliato nel file `.emacs`.

C'è ancora un tipo di espressione che si ritrova spesso nei file `.emacs`. La libreria di Emacs Lisp spesso offre dei pacchetti multipli per fare la stessa cosa in modi diversi; bisognerà così specificare quale volete usare (o dovrete usare il pacchetto di default, che non è sempre il migliore per tutti gli scopi). Un'area in cui si ha questo problema sono le caratteristiche di interazione di Emacs con Scheme: con Emacs vengono distribuite due diverse interfacce Scheme (almeno nella versione 19): `xscheme` e `cmuscheme`.

```
prompt> ls /usr/lib/emacs/19.19/lisp/*scheme*
/usr/lib/emacs/19.19/lisp/cmuscheme.el
/usr/lib/emacs/19.19/lisp/cmuscheme.elc
/usr/lib/emacs/19.19/lisp/scheme.el
/usr/lib/emacs/19.19/lisp/scheme.elc
/usr/lib/emacs/19.19/lisp/xscheme.el
/usr/lib/emacs/19.19/lisp/xscheme.elc
```

A me l'interfaccia di `cmuscheme` piace molto di più di quella di default, che è `xscheme`. Come posso far usare ad Emacs quella che piace di più a me? Mettendo queste linee in `.emacs`:

```
;; notate come l'espressione può essere spezzata in più linee.  Il Lisp
;; in genere ignora gli spazi vuoti:
(autoload 'run-scheme "cmuscheme"
"Avvia uno Scheme inferiore, come lo voglio io." t)
```

La funzione `autoload` prende come argomento il nome di una funzione (tra apici "`'`", per ragioni che hanno a che fare con il Lisp) e dice ad Emacs che tale funzione è definita in un determinato file. Il file è il secondo argomento, una stringa (senza l'estensione "`.el`" o "`.elc`") che indica il nome del file da cercare nel `load-path`.

I restanti argomenti sono in genere opzionali, ma in questo caso necessari: il terzo è una stringa di documentazione per la funzione, in modo che se chiamate `describe-function` sulla funzione avete delle informazioni che vi possono servire. Il quarto dice ad Emacs che questa funzione autoavviabile può essere richiamata interattivamente (cioè usando `M-x`). In questo caso quest'ultimo argomento è molto importante, dato che bisogna poter digitare `M-x run-scheme` per avviare un processo di scheme sotto Emacs.

Adesso che `run-scheme` è stata definita come funzione autoavviabile, cosa succede quando digito `M-x run-scheme`? Emacs cerca la funzione `run-scheme`, vede che è impostata per venire autoavviata, e carica il file indicato dall'autoavviamento (in questo caso, "`cmuscheme`"). Il file compilato byte per byte `cmuscheme.elc` esiste, quindi Emacs lo caricherà. Questo file *deve* definire la funzione `run-scheme`, o ci sarà un errore nell'autocaricamento. Per fortuna lo fa, quindi tutto va liscio, ed io ottengo la mia interfaccia Scheme preferita[10].

---

[10] Tra l'altro, `cmuscheme` era l'interfaccia di cui stavo parlando più indietro, nella sezione su come lavorare con Scheme, quindi se volete usare il materiale di quel capitolo, dovete assicurarvi di stare usando `cmuscheme`.

Un “autocaricamento” è come una promessa fatta ad Emacs che, quando sarà il momento, potrà trovare la funzione specificata nel file in cui gli si dice di guardare; e voi potete controllare quello che viene caricato. Inoltre, l’autocaricamento vi aiuta a diminuire la memoria usata da Emacs, evitando di caricare determinate caratteristiche finché non vengono richieste. La maggior parte dei comandi non sono definiti come funzioni quando Emacs parte; piuttosto, sono semplicemente impostati come autocaricamento da determinati file. Se non invocate il comando, non viene mai caricato. Questo modo di salvare spazio è vitale per il funzionamento di Emacs: se caricasse tutti i file disponibili nella libreria Lisp, ci vorrebbero venti minuti per avviarlo, ed occuperebbe la maggior parte della memoria disponibile sul sistema. Non vi preoccupate, non dovete impostare tutti questi autocaricamenti nel file `.emacs`: sono stati preimpostati all’interno di Emacs stesso.

8.14 Scoprirne di più

Non vi ho detto tutto quello che c’è da sapere su Emacs. In effetti, non credo di avervene detto nemmeno l’1%. Anche se ne sapete abbastanza per andare avanti, ci sono ancora moltissimi trucchetti per salvare tempo e cose comode che dovreste trovare. Il miglior modo di farlo è aspettare finché non vi serve qualcosa, e poi cercare una funzione che la faccia.

Non potrò mai sottolineare abbastanza l’importanza di saper usare la guida in linea di Emacs. Ad esempio, supponiamo che vogliate inserire il contenuto di un file in un buffer già aperto su un file diverso, in modo che il buffer li contenga entrambi. Beh, se tiraste ad indovinare che c’è un comando che si chiama `insert-file`, ci indovinereste. Per controllare, digitate `C-h f`. Al prompt del minibuffer, digitate il nome della funzione su cui volete la guida. Dato che sapete che esiste il completamento dei nomi delle funzioni, e potete indovinare che il comando che state cercando inizia per “insert”, digitate insert e premete Tab; vi verranno mostrati tutti i nomi delle funzioni che cominciano per “insert”, tra cui anche “insert-file”.

Completate quindi il nome della funzione e leggete come funziona, e poi usate `M-x insert-file`. Se vi state chiedendo se è collegata ad un tasto, digitate `C-h w insert-file` Invio, e scopritelo. Più cose sapete della guida di Emacs, più facilmente gli potete porre domande. La capacità di farlo, unita ad uno spirito di esplorazione ed alla volontà di imparare nuovi modi di fare le cose vi porteranno a risparmiare un sacco di tempo.

Per ordinare una copia del manuale per l’utente di Emacs e/o del manuale della programmazione Lisp per Emacs, scrivete a:

Free Software Foundation
675 Mass Ave
Cambridge, MA 02139
USA

Entrambi questi manuali sono distribuiti elettronicamente con Emacs, in forma leggibile usando il lettore di documentazione Info (`C-h i`), ma può essere più facile usare la versione ad albero che quella in linea. Inoltre, il prezzo è ragionevole, e i soldi vanno per una giusta causa—software libero di qualità! Una volta o l'altra dovreste leggere le condizioni di copyright di Emacs con `C-h C-c`. È più interessante di quanto pensiate, e vi aiuterà a chiarirvi il concetto di software libero. Se pensate che il termine "software libero" ("free software") vuol dire semplicemente che il programma è gratis, per favore leggete il copyright appena ne avete il tempo!

Capitolo 9

Voglio essere me stesso!

9.1 Personalizzare `bash`

Una delle caratteristiche della filosofia di Unix è che l'impostazione del sistema non vuole prevedere tutte le necessità dell'utente, ma tenta di rendere semplice per ciascuno modificarsi l'ambiente a seconda delle proprie necessità. Per questo vengono usati principalmente dei **file di configurazione**, noti anche come "file init", "file rc" (per "run control", controllo dell'esecuzione) o anche "file punto", perché il loro nome di solito inizia con "`.`". Se vi ricordate, i file che hanno il nome che inizia con "`.`" non vengono normalmente visualizzati da `ls`.

I file di configurazione più importanti sono quelli utilizzati dalla shell. La shell di default di Linux è `bash`, ed è di questa che parleremo in questo capitolo. Prima di vedere come personalizzare `bash`, bisogna sapere quali file utilizza.

9.1.1 Avvio della shell

Ci sono diversi modi in cui può essere eseguita `bash`, ad esempio come **shell di login**, che è come viene eseguita al login. La shell di login dovrebbe essere la prima che vedete.

Un'altra modalità in cui si esegue `bash` è come **shell interattiva**: una shell qualsiasi che presenti un prompt all'utente ed aspetti dei comandi. Una shell di login è anche interattiva. Un modo per ottenere una shell interattiva non di login è per esempio una shell all'interno di un `xterm`. Qualsiasi shell che viene creata in modo diverso dal login è una shell non di login.

Infine ci sono le **shell non interattive**, che sono usate per eseguire dei file di comandi, più o meno come i file batch di MS-DOS—quelli che finiscono in `.BAT`. Questi **script di**

shell funzionano come dei mini-programmi; anche se sono generalmente molto più lenti di un normale programma compilato, è spesso vero che sono molto più semplici da scrivere.

A seconda del tipo di shell, al suo avvio vengono usati diversi file:

Tipo di shell	Azione
Login interattivo	Viene letto ed eseguito il file `.bash_profile`
Interattivo	Viene letto ed eseguito il file `.bashrc`
Non interattivo	Viene letto ed eseguito lo script di shell

9.1.2 File di avvio

Dato che la maggior parte degli utenti vogliono avere lo stesso ambiente in qualsiasi shell, interattiva, di login o no, iniziamo con un comando molto semplice da mettere nel file `.bash_profile`: “`source ~/.bashrc`”. Il comando `source` dice alla shell di interpretare l'argomento come uno script di shell. Per noi questo significa che ogni volta che viene eseguito `.bash_profile`, viene eseguito *anche* `.bashrc`.

Ora aggiungeremo semplicemente dei comandi al file `.bashrc`. Se volete che un comando sia valido solo per una shell di login, aggiungetelo a `.bash_profile`.

9.1.3 Aliasing

Quali sono le cose che volete personalizzare? Ecco qualcosa che credo si trovi nel `.bashrc` del 90% degli utenti di `bash`:

```
alias ll="ls -l"
```

Questo comando ha definito un **alias** di shell con nome `ll`, che, quando viene richiamato dall'utente, si “espande” nel normale comando di shell “`ls -l`”. Quindi, assumendo che `bash` abbia letto quel comando dal file `.bashrc`, potete digitare semplicemente `ll` per avere l'effetto di “`ls -l`” nella metà delle digitazioni. Quello che accade è che quando digitate `ll` e premete Invio, `bash` lo intercetta, dato che si aspetta degli alias, e lo sostituisce con “ls -l”. In realtà non c'è nessun programma che si chiama `ll` sul sistema, ma la shell traduce automaticamente l'alias in un programma valido.

Alcuni alias di esempio sono riportati in Figura 9.1.3. Li potete aggiungere al vostro `.bashrc`. Un alias particolarmente interessante è il primo; con quell'alias, quando qualcuno digita `ls`, automaticamente viene aggiunta l'opzione `-F` (l'alias non prova ad espandersi di nuovo). È un modo abbastanza comune per aggiungere delle opzioni che usate ogni volta che richiamate un programma.

Figura 9.1 Degli esempi di alias per `bash`.

```
alias ls="ls -F"          # da' i caratteri alla fine dell'elenco
alias ll="ls -l"          # ls speciale
alias la="ls -a"
alias ro="rm *~; rm .*~"  # elimina i file di backup creati da Emacs
alias rd="rmdir"          # risparmia battute!
alias md="mkdir"
alias pu=pushd            # pushd, popd, e dirs non sono stati spiegati in questo
alias po=popd             # manuale---forse volete guardare di che si tratta
alias ds=dirs             # nella manpage di bash
# questi sono solo scorciatoie da tastiera
alias to="telnet cs.oberlin.edu"
alias ta="telnet altair.mcs.anl.gov"
alias tg="telnet wombat.gnu.ai.mit.edu"
alias tko="tpalk kold@cs.oberlin.edu"
alias tjo="talk jimb@cs.oberlin.edu"
alias mroe="more"         # per correggere!
alias moer="more"
alias email="emacs -f rmail" # il mio lettore di posta
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""
                          # un modo di richiamare Emacs
```

Notate i commenti con il carattere `#` nella Figura 9.1.3. Quando appare un `#`, la shell ignora il resto della linea.

Potreste aver notato alcune strane cose su questi alias: ad esempio, in alcuni ho tralasciato le virgolette—come in `pu`. Rigorosamente parlando, le virgolette non sono necessarie quando avete solo una parola a destra del segno di uguale.

Non è che mettere le virgolette faccia male, quindi non prendete cattive abitudini: dovreste usarle di sicuro se state per fare un alias di un comando con delle opzioni e/o argomenti:

```
alias rf="refrobnicate -verbose -prolix -wordy -o foo.out"
```

Inoltre, l'alias finale ha dei caratteri strani:

```
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""
```

Come potete aver indovinato, volevo passare le virgolette nelle opzioni stesse, quindi le ho dovute far precedere da una barra rovesciata in modo da evitare che `bash` pensasse che volevo segnalare la fine dell'alias.

Infine, ho messo come alias due comuni errori di battitura, "mroe" e "moer", per il comando esatto, `more`. Gli alias non interferiscono con gli argomenti passati ad un programma. Funziona lo stesso se scrivete:

```
/home/larry# mroe hurd.txt
```

In effetti, sapere come creare gli alias è probabilmente almeno metà della personalizzazione della shell che farete mai. Sperimentate, scoprite quali comandi lunghi date spesso, e createne degli alias. Scoprirete che rende molto più piacevole lavorare al prompt della shell.

9.1.4 Variabili d'ambiente

Un'altra cosa importante che si può fare nel file `.bashrc` è impostare le **variabili d'ambiente**. E che cosa sono? Prendiamola da un'altra direzione: supponiamo che stiate leggendo della documentazione per il programma `pippo`, ed incontrate queste frasi:

> `pippo` normalmente cerca il suo file di configurazione, `.pipporc`, nella home directory dell'utente. Comunque, se la variabile d'ambiente `PIPPOPATH` è impostata ad un nome di file diverso, cercherà lì.

Ogni programma viene eseguito in un **ambiente** definito dalla shell che ha chiamato il programma[1]. Si può dire che l'ambiente esiste "all'interno" della shell. I programmatori hanno una routine speciale per porre delle domande all'ambiente, e il programma "pippo" usa questa routine: controlla il valore della variabile d'ambiente `PIPPOPATH`; se la variabile non è definita, userà semplicemente il file `.pipporc` nella home directory. Se è definita, comunque, `pippo` userà il valore della variabile (che dovrebbe essere il nome di un file che `pippo` può usare) al posto del default `.frugglerc`.

Ecco come si può cambiare l'ambiente in `bash`:

```
/home/larry# export PGPPATH=/home/larry/secrets/pgp
```

---

[1] Ecco perché le shell sono così importanti. Pensate se aveste dovuto passare l'intero ambiente a mano ogni volta che aveste richiamato un programma!

Si può pensare al comando `export` come ad un invito: “Per favore, esporta questa variabile all'ambiente, dove richiamerò i programmi, in modo che il suo valore sia a loro visibile”. Ci sono delle buone ragioni per chiamarlo `export`, come vedremo poi.

Questa particolare variabile viene usata dal famoso programma di criptazione a chiave pubblica di Phil Zimmerman `pgp`. Normalmente `pgp` cerca determinati file di cui ha bisogno (che contengono le chiavi di criptazione) nella home directory, e la usa anche per registrare i file temporanei che crea mentre è in funzione. Impostando la variabile `PGPPATH` a questo valore, gli ho detto di usare la directory `/home/larry/secrets/pgp`. Ho dovuto leggere il manuale di `pgp` per trovare il nome esatto della variabile e che cosa fa, ma è abbastanza standard l'uso del nome del programma in lettere maiuscole, seguito dal suffisso “PATH”.

È anche utile saper chiedere informazioni sull'ambiente:

```
/home/larry# echo $PGPPATH
/home/larry/.pgp
/home/larry#
```

Notate il “$”: si premette un segno di dollaro al nome di una variabile d'ambiente per estrarre il valore della variabile. Se aveste scritto il comando senza il segno di dollaro, `echo` avrebbe semplicemente stampato i suoi argomenti:

```
/home/larry# echo PGPPATH
PGPPATH
/home/larry#
```

Il “$” viene usato per *valutare* le variabili d'ambiente, ma lo fa solo nel contesto della shell—cioè, quando è la shell che interpreta i comandi. Quando è che lo fa? Beh, quando state digitando dei comandi al prompt, o quando `bash` sta leggendo dei comandi da un file come `.bashrc` si può dire che è la shell che “interpreta” i comandi.

C'è un altro comando molto utile per rivolgere domande all'ambiente: `env`. `env` elencherà semplicemente tutte le variabili d'ambiente. È possibile, specialmente se state usando X, che l'elenco scorra fuori dallo schermo; se succede, mandate l'output di `env`, con una pipe, a `more`: `env | more`.

Alcune di queste variabili sono piuttosto utili, quindi le descriveremo più dettagliatamente. Guardate la Figura 9.1.4. Queste quattro variabili sono definite automaticamente quando fate login: non c'è bisogno di impostarle nei file `.bashrc` o `.bash_login`.

Diamo un'occhiata più da vicino alla variabile `TERM`. Per capirla, torniamo un attimo alla storia di Unix: il sistema operativo ha bisogno di conoscere alcune cose sulla vostra

Figura 9.2 Alcune importanti variabili d'ambiente.

Variabile	Contiene	Esempio
`HOME`	La home directory	`/home/larry`
`TERM`	Il tipo di terminale	`xterm`, `vt100`, o `console`
`SHELL`	Il percorso per la shell	`/bin/bash`
`USER`	Il vostro nome di login	`larry`
`PATH`	Lista di directory in cui cercare i programmi	`/bin:/usr/bin:/usr/local/bin:/usr/bin/X11`

console, per poter attivare delle funzioni di base come scrivere un carattere sullo schermo, spostare il cursore sulla linea seguente, eccetera. Nei primi anni dell'informatica, i produttori di computer aggiungevano continuamente nuove caratteristiche ai terminali: per prima l'inversione, poi forse i caratteri europei, e alla fine magari anche delle primitive funzioni grafiche (ricordatevi, ancora non esistevano i sistemi a finestre ed i mouse!). Comunque, tutte queste nuove funzioni rappresentavano un problema per i programmatori: come potevano sapere quali funzioni aveva un terminale, e quali no? E come potevano supportare le nuove caratteristiche senza far diventare inutili i vecchi terminali?

In Unix, la risposta a queste domande è stata `/etc/termcap`. `/etc/termcap` è un elenco di tutti i terminali che il vostro sistema conosce, e come controllano il cursore. Se un amministratore di sistema comprava un nuovo tipo di terminale, tutto quello che doveva fare era aggiungere una voce per quel terminale in `/etc/termcap` invece di ricompilare tutto Unix. Talvolta è ancora più facile: lungo la strada, il vt100 della Digital Equipment Corporation è diventato quasi uno standard, e molti terminali nuovi sono stati costruiti in modo da poterlo emulare, o da potersi comportare come un vt100.

Sotto Linux, il valore di `TERM` è talvolta `console`, che è un terminale di tipo vt100 con delle caratteristiche aggiunte.

Anche un'altra variabile, `PATH`, è cruciale per il funzionamento corretto della shell. Ecco la mia:

```
/home/larry# env | grep ^PATH
PATH=/home/larry/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/TeX/bin
/home/larry#
```

Il vostro `PATH` è un elenco, con le voci separate da due punti, delle directory in cui la shell deve cercare i programmi, quando digitate il nome di un programma da eseguire. Quando digito `ls` e premo Invio, per esempio, `bash` guarda per prima cosa in `/home/larry/bin`, una directory che ho creato per i programmi scritti da me; ma io non ho creato `ls` (in effetti, credo che sia stato scritto prima che io nascessi!). Non trovandolo in `/home/larry/bin`, `bash` cerca in `/bin`—e lì lo trova. `/bin/ls` esiste ed è eseguibile, quindi `bash` interrompe la ricerca e avvia il programma. Ci poteva essere un altro programma con nome `ls` nella directory `/usr/bin`, ma `bash` non l'avrebbe mai avviato a meno che io non avessi specificato esplicitamente il percorso:

```
/home/larry# /usr/bin/ls
```

Lo scopo della variabile `PATH` è di non farci digitare i percorsi completi ogni volta che dobbiamo dare un comando. Quando digitate un comando, `bash` lo cerca nelle directory elencate in `PATH`, in ordine, e se lo trova lo avvia. Se non lo trova, vi dà un errore:

```
/home/larry# clubly
clubly: command not found
```

Notate che nel mio `PATH` non c'è la directory corrente, "`.`". Se ci fosse stata, sarebbe stato così:

```
/home/larry# echo $PATH
.:/home/larry/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/TeX/bin
/home/larry#
```

Questo è un problema su cui si dibatte nei circoli Unix (di cui siete membri adesso, che vi piaccia o no). Il problema è che avere la directory corrente nel percorso può essere un buco di sicurezza. Supponiamo che facciate `cd` in una directory in cui qualcuno abbia lasciato un programma "Trojan Horse" con nome `ls`, voi fate un `ls`, come sarebbe naturale entrando un una nuova directory. Dato che la directory corrente, "`.`", è la prima del vostro `PATH`, la shell troverebbe questa versione di `ls` e la eseguirebbe. Qualsiasi cattiveria ci fosse in quel programma, l'avreste appena eseguita (e può essere molto cattiva!). Non ci sarebbe bisogno di privilegi di root per farlo; basterebbe un permesso di scrittura sulla directory in cui si trova il "falso" `ls`. Potrebbe essere anche la loro home directory, se avessero saputo che ci sareste capitati dentro.

Sul vostro sistema è molto poco probabile che la gente si lasci delle trappole l'un l'altro. Tutti gli utenti sono probabilmente vostri amici o colleghi. Comunque, su grandi sistemi multi-utente (come molti computer delle università), ci possono essere programmatori poco amichevoli che non avete mai incontrato. Se volete o no che nel vostro percorso ci sia "`.`" dipende dalla situazione; non sarò dogmatico sull'una o sull'altra scelta, ma vorrei che aveste presenti i rischi[2]. I sistemi multi-utente sono come delle comunità, dove c'è gente che si può comportare in modo imprevedibile.

Il mio `PATH` è in realtà impostato in un modo che riassume la maggior parte delle cose che avete imparato finora sulle variabili d'ambiente. Ecco cosa c'è nel mio `.bashrc`:

```
export PATH=${PATH}:.:${HOME}/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/TeX/bin
```

Qui mi avvantaggio del fatto che la variabile `HOME` viene impostata prima che `bash` legga il file `.bashrc`, ed uso il suo valore nell'impostazione del `PATH`. Le parentesi graffe ("`{...}`") sono un ulteriore livello di quoting: delimitano l'estensione di quello che il "`$`" deve valutare, in modo che la shell non si confonda con il testo seguente (nel nostro caso "`/bin`"). Ecco un altro esempio del loro effetto:

```
/home/larry# echo ${HOME}foo
/home/larryfoo
/home/larry#
```

Senza le parentesi, non avrei niente, dato che non esiste una variabile d'ambiente che si chiama `HOMEfoo`.

```
/home/larry# echo $HOMEfoo

/home/larry#
```

Fatemi chiarire un'altra cosa sulla mia impostazione della variabile: il significato di "`$PATH`": quello che fa è includere il valore di qualsiasi variable `PATH` impostata *precedentemente*. Dove sarebbe impostata una variabile del genere? Il file `/etc/profile` fa da `.bash_profile` globale, comune a tutti gli utenti; avere un file centralizzato del genere aiuta il sistemista ad aggiungere nuove directory al `PATH` di tutti gli utenti, senza dover farlo uno per uno. Se includete il vecchio path nel vostro, non perderete nessuna directory utile.

Potete anche controllare l'aspetto del prompt: si fa impostando il valore della variabile d'ambiente **PS1**. Personalmente preferisco un prompt che mi faccia vedere il percorso della directory corrente—ecco come faccio:

```
export PS1='$PWD# '
```

---

[2] Ricordatevi che potete sempre eseguire i programmi nella directory corrente dandogli il percorso esplicito, come "`./foo`".

Come potete vedere, qui vengono usate *due* variabili. Quella che viene impostata è la variabile `PS1`, e viene impostata al valore della variabile `PWD`, che può significare sia “Print Working Directory” sia “Path to Working Directory”. La valutazione di `PWD` avviene all'interno degli apici. Gli apici servono per valutare l'espressione tra essi contenuta, che in questo caso viene valutata come la variabile `PWD`. Se aveste fatto solo `export PS1=$PWD`, il prompt avrebbe contenuto sempre il percorso alla directory corrente *al momento in cui avevate impostato* `PS1`, invece di aggiornarlo continuamente quando si cambia directory. Beh, è un po' confuso, e non è poi così importante. Tenete solo a mente che se volete che nel prompt sia mostrata la directory corrente vi servono gli apici.

Potreste preferire `export PS1='$PWD>'`, o anche il nome del sistema: `export PS1=‘hostname‘'>'`; fatemi sezionare quest'ultimo esempio un po' meglio.

Nell'ultimo esempio ho usato un *nuovo* tipo di quoting, quello con gli apici rovesciati: non protegge niente—in effetti, noterete che “hostname” non appare da nessuna parte nel prompt. Quello che accade è che il comando all'interno degli apici rovesciati viene valutato, e il suo output viene mostrato al posto degli apici e del nome del comando.

Provate a fare `echo ‘ls‘` o `wc ‘ls‘`. A mano a mano che acquistate esperienza nell'uso della shell, questa tecnica diventa sempre più potente.

Ci sono parecchie altre cose che si possono configurare nel `.bashrc`, e qui non c'è spazio per spiegarle tutte. Potete leggervi la pagina di manuale di `bash`, o chiedere a chi ne sa di più. Ecco un file `.bashrc` completo da studiare: è piuttosto standard, anche se il PATH è un po' lungo.

```
# varie:
ulimit -c unlimited
export history_control=ignoredups
export PS1='$PWD>'
umask 022

# percorsi specifici per le applicazioni:
export MANPATH=/usr/local/man:/usr/man
export INFOPATH=/usr/local/info
export PGPPATH=${HOME}/.pgp

# il PATH principale:
homepath=${HOME}:~/bin
stdpath=/bin:/usr/bin:/usr/local/bin:/usr/ucb/:/etc:/usr/etc:/usr/games
pubpath=/usr/public/bin:/usr/gnusoft/bin:/usr/local/contribs/bin
softpath=/usr/bin/X11:/usr/local/bin/X11:/usr/TeX/bin
```

```
export PATH=.:${homepath}:${stdpath}:${pubpath}:${softpath}
# Tecnicamente, le parentesi graffe non erano necessarie, perche' i due
# punti sono delimitatori validi; comunque, e' una buona abitudine abituarsi
# a mettere le parentesi, che non fanno mai male.

# alias
alias ls="ls -CF"
alias fg1="fg %1"
alias fg2="fg %2"
alias tba="talk sussman@tern.mcs.anl.gov"
alias tko="talk kold@cs.oberlin.edu"
alias tji="talk jimb@totoro.bio.indiana.edu"
alias mroe="more"
alias moer="more"
alias email="emacs -f vm"
alias pu=pushd
alias po=popd
alias b="~/.b"
alias ds=dirs
alias ro="rm *~; rm .*~"
alias rd="rmdir"
alias ll="ls -l"
alias la="ls -a"
alias rr="rm -r"
alias md="mkdir"
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\""

function gco
{
  gcc -o $1 $1.c -g
}
```

9.2 I file di inizializzazione di X Windows

La maggior parte delle persone lavorano all'interno di un ambiente grafico, e, per le macchine Unix, questo di solito significa usare X. Se siete abituati al Macintosh, o a Microsoft Windows, può essere un po' lungo abituarcisi, specialmente per quanto riguarda la personalizzazione.

In Macintosh o in Microsoft Windows l'ambiente si personalizza dall'*interno* dell'ambiente stesso: se volete cambiare lo sfondo, ad esempio, si clicca su un nuovo colore in qualche programma speciale per le impostazioni grafiche. In X, i default di sistema vengono controllati da file di testo, che si modificano direttamente—in altre parole, per cambiare il colore di sfondo si scrive il nome del colore in un file.

Non si può dire che questo metodo sia più elegante di qualche sistema a finestre commerciale. Credo che questa tendenza a rimanere legati al testo, anche in un ambiente grafico, sia legata al fatto che X è stata creata da dei programmatori che non cercavano di scrivere del software che potesse essere usato dai loro nonni. Questa tendenza dovrebbe cambiare nelle prossime versioni di X (almeno lo spero), ma per adesso dovete imparare a combattere con altri file di testo; in ogni caso questo vi dà un controllo molto flessibile e preciso sulla configurazione del sistema.

Ecco i file più importanti per configurare X:

`.xinitrc`	Uno script usato da X all'avvio.
`.twmrc`	Letto da un manager di finestre, `twm`.
`.fvwmrc`	Letto da un manager di finestre, `fvwm`.

Tutti questi file, se esistono, dovrebbero essere nella vostra home directory.

Il file `.xinitrc` è un semplice script di shell che viene avviato quando richiamate X. Può fare tutto quello che fa un normale script di shell, ma naturalmente ha senso se si usa per avviare dei programmi di X e per impostare dei parametri del sistema a finestre. L'ultimo comando del file `.xinitrc` è di solito il nome di un manager di finestre, ad esempio `/usr/bin/X11/twm`.

Che tipo di cose si mettono in un file `.xinitrc`? Forse delle chiamate al programma `xsetroot`, per impostare l'aspetto della finestra di root (lo sfondo) e del cursore del mouse, una chiamata a `xmodmap`, che dice al server[3] come interpretare i segnali da tastiera, e tutti gli altri programmi che volete avviare ogni volta che avviate X (ad esempio, `xclock`).

Ecco una parte del mio `.xinitrc`; il vostro sarà quasi sicuramente diverso, quindi questo è solo un esempio:

```
#!/bin/sh
# La prima linea dice al sistema operativo quale shell usare per
# interpretare lo script. Lo script stesso dovrebbe essere reso eseguibile:
# lo si puo' fare con "chmod +x ~/.xinitrc".
```

---

[3] Il "server" è semplicemente il processo X principale sulla vostra macchina, quello con cui tutti i programmi di X devono comunicare per usare lo schermo. Questi altri programmi si chiamano "client", e l'insieme si chiama sistema "client-server".

```
# xmodmap e' un programma per dire al server X come interpretare i segnali
# da tastiera. Vale *veramente* la pena imparare ad usarlo. Si puo' fare
# "man xmodmap", "xmodmap -help", "xmodmap -grammar", e cosi' via.
# Non garantisco che le espressioni qui sotto significhino qualcosa sul
# vostro sistema (non garantisco nemmeno che significhino qualcosa sul
# mio):
xmodmap -e 'clear Lock'
xmodmap -e 'keycode 176 = Control_R'
xmodmap -e 'add control = Control_R'
xmodmap -e 'clear Mod2'
xmodmap -e 'add Mod1 = Alt_L Alt_R'

# xset e' un programma per impostare qualche altro parametro del server X:
xset m 3 2 &       # parametri del mouse
xset s 600 5 &     # preferenze del salvaschermo
xset s noblank &   # come sopra
xset fp+ /home/larry/x/fonts # per cxterm
# Per scoprirne di piu', "xset -help".

# Dice al server X di sovrapporre fish.cursor su fish.mask, ed usare
# il disegno risultante come cursore per il mouse:
xsetroot -cursor /home/lab/larry/x/fish.cursor /home/lab/larry/x/fish.mask &

# un colore e un'immagine di sfondo che mi piacciono
xsetroot -bitmap /home/lab/larry/x/pyramid.xbm -bg tan

# da fare: xrdb? E il file .Xdefaults?

# Dovreste fare "man xsetroot" o "xsetroot -help" per avere altre
# informazioni sul programma usato qui sopra.

# Un programma client, l'orologio tondo a colori di Jim Blandy:
/usr/local/bin/circles &

# Forse vorrete avere un orologio sullo schermo in ogni momento?
/usr/bin/X11/xclock -digital &

# Permette ai programmi client di X che girano su occs.cs.oberlin.edu di
```

```
# usare questo come schermo, e lo stesso per juju.mcs.anl.gov:
xhost occs.cs.oberlin.edu
xhost juju.mcs.anl.gov

# Potete anche dire al server X di permettere a client che girano su
# qualsiasi host (in cui host e' una macchina remota) di usare lo schermo,
# mae'e un buco di sicurezza---questi client possono essere manovrati da
# qualcun altro, e chiunque potrebbe vedere le digitazioni della vostra
# password o cose del genere!
# In ogni caso, se volete farlo lo stesso, potete usare un "+" per
# significare tutti i possibili host, invece di uno specifico, cosi':
# xhost +

# E, infine, il manager di finestre:
/usr/bin/X11/twm
# Alcuni preferiscono altri manager. Io uso twm, ma spesso con Linux
# viene distribuito anche fvwm:
# /usr/bin/X11/fvwm
```

Notate che alcuni comandi vengono avviati in background (cioè sono seguiti da una “&”) e altri no. La distinzione è che alcuni programmi vengono avviati quando avviate X e continuano a funzionare finché non uscite—mettete questi in background. Altri programmi vengono eseguiti una sola volta e se ne esce immediatamente. `xsetroot` è uno di questi; imposta semplicemente la finestra di root o il cursore o qualcos'altro, e poi esce.

Una volta partito, il manager delle finestre leggerà il suo file di inizializzazione, che controlla cose come l'impostazione dei menù, o in quale posizione vengono aperte le finestre, il controllo delle icone e altre cose estremamente importanti. Se usate `twm`, il file è `.twmrc`, nella vostra home directory. Se usate fvwm sarà `.fvwmrc` e così via. Qui tratterò solo questi due, dato che sono quelli che incontrerete più facilmente con Linux.

9.2.1 Configurazione di Twm

Il file `.twmrc` non è uno script di shell—credetelo o no, è scritto in un linguaggio fatto proprio per `twm`![4] La cosa con cui la gente ama giocherellare nel file `.twmrc` è lo stile delle

---

[4] Questa è una delle caratteristiche negative dei file di inizializzazione: hanno generalmente il loro proprio linguaggio caratteristico; ciò significa che gli utenti diventano bravissimi ad imparare velocemente i linguaggi. Suppongo che sarebbe bello se i primi programmatori di Unix si fossero messi d'accordo su un formato standard per i file di inizializzazione, in modo che non dovessimo

finestre (i colori e cose di questo genere), e i menù, quindi ecco un file `.twmrc` di esempio che configura queste caratteristiche:

```
# Imposta i colori per le varie parti delle finestre. Fa molto
# effetto sulla personalizzazione dell'ambiente.
Color
{
    BorderColor "OrangeRed"
    BorderTileForeground "Black"
    BorderTileBackground "Black"
    TitleForeground "black"
    TitleBackground "gold"
    MenuForeground "black"
    MenuBackground "LightGrey"
    MenuTitleForeground "LightGrey"
    MenuTitleBackground "LightSlateGrey"
    MenuShadowColor "black"
    IconForeground "DimGray"
    IconBackground "Gold"
    IconBorderColor "OrangeRed"
    IconManagerForeground "black"
    IconManagerBackground "honeydew"
}

# Spero che non abbiate un sistema in bianco e nero, ma se e' cosi'...
Monochrome
{
    BorderColor "black"
    BorderTileForeground "black"
    BorderTileBackground "white"
    TitleForeground "black"
    TitleBackground "white"
}

# Ho creato beifnag.bmp con il programma "bitmap". Qui dico a twm di
# usarlo come disegno di default per le barre del titolo delle
# finestre.
```

imparare nuove sintassi ogni volta, ma ad essere realisti è difficile predire di quali informazioni avranno bisogno i programmi.

```
Pixmaps
{
    TitleHighlight "/home/larry/x/beifang.bmp"
}

# Non vi preoccupate di queste cose, sono solo per i fissati :-)
BorderWidth     2
TitleFont       "-adobe-new century schoolbook-bold-r-normal--14-140-75-75-p-87-iso8859-1"
MenuFont        "6x13"
IconFont        "lucidasans-italic-14"
ResizeFont      "fixed"
Zoom 50
RandomPlacement

# Questi programmi non avranno una barra del titolo per default:
NoTitle
{
  "stamp"
  "xload"
  "xclock"
  "xlogo"
  "xbiff"
  "xeyes"
  "oclock"
  "xoid"
}

# "AutoRaise" significa che la finestra viene messa in primo piano
# ogni volta che il puntatore del mouse ci passa sopra. Io lo trovo
# scomodo, quindi l'ho disabilitato. Come potete vedere, ho ereditato
# il mio file .twmrc da persone che avevano la mia stessa idea.
AutoRaise
{
  "nothing"    # Non mi piace autoraise # A me neanche # Nemmeno a me
}

# Qui vengono definite le funzioni dei pulsanti del mouse. Notate
# lo schema: un pulsante premuto sulla finestra di sfondo, senza
# premerci insieme nessun tasto modificatore, apre sempre un menu'.
```

```
# Premere un pulsante su altre zone porta ad una qualche
# manipolazione della finestra, ed i tasti modificatori vengono
# usati insieme ai pulsanti del mouse per avere le manipolazioni piu'
# sofisticate.
#
# Non c'e' bisogno che seguiate questo schema nel vostro .twmrc --
# la configurazione dell'ambiente e' completamente a vostro piacere.

# Pulsante = TASTI : CONTESTO : FUNZIONE
# ----------------------------------
Button1 =      : root    : f.menu "main"
Button1 =      : title   : f.raise
Button1 =      : frame   : f.raise
Button1 =      : icon    : f.iconify
Button1 = m    : window  : f.iconify

Button2 =      : root    : f.menu "stuff"
Button2 =      : icon    : f.move
Button2 = m    : window  : f.move
Button2 =      : title   : f.move
Button2 =      : frame   : f.move
Button2 = s    : frame   : f.zoom
Button2 = s    : window  : f.zoom

Button3 =      : root    : f.menu "x"
Button3 =      : title   : f.lower
Button3 =      : frame   : f.lower
Button3 =      : icon    : f.raiselower

# Potete scrivere delle funzioni personalizzate; questa viene usata
# nel menu' "windowops" quasi alla fine di questo file:
Function "raise-n-focus"
{
    f.raise
    f.focus
}

# Okay, qui sotto ci sono i menu' a cui ci si riferisce nella sezione
# dei pulsanti del mouse. Notate che molte delle voci
```

```
# richiamano a loro volta dei sottomenu'. Potete avere quanti livelli
# di menu' volete, ma state attenti a non farli ricorsivi: non
# funzionano, ci ho provato.

menu "main"
{
"Vanilla"       f.title
"Emacs"         f.menu "emacs"
"Logins"        f.menu "logins"
"Xlock"         f.menu "xlock"
"Misc"          f.menu "misc"
}

# Questo mi permette di invocare Emacs su macchine diverse. Date
# un'occhiata alla sezione sui file .rhosts per altre informazioni
# su come fare:
menu "emacs"
{
"Emacs"      f.title
"here"       !"/usr/bin/emacs &"
""           f.nop
"phylo"      !"rsh phylo \"emacs -d floss:0\" &"
"geta"       !"rsh geta \"emacs -d floss:0\" &"
"darwin"     !"rsh darwin \"emacs -d floss:0\" &"
"ninja"      !"rsh ninja \"emacs -d floss:0\" &"
"indy"       !"rsh indy \"emacs -d floss:0\" &"
"oberlin"    !"rsh cs.oberlin.edu \"emacs -d floss.life.uiuc.edu:0\" &"
"gnu"        !"rsh gate-1.gnu.ai.mit.edu \"emacs -d floss.life.uiuc.edu:0\" &"
}

# Mi fa aprire degli xterm su parecchie altre macchine. Guardate la
# sezione sui file .rhosts per altre informazioni.
menu "logins"
{
"Logins"     f.title
"here"       !"/usr/bin/X11/xterm -ls -T `hostname` -n `hostname` &"
"phylo"      !"rsh phylo \"xterm -ls -display floss:0 -T phylo\" &"
"geta"       !"rsh geta \"xterm -ls -display floss:0 -T geta\" &"
"darwin"     !"rsh darwin \"xterm -ls -display floss:0 -T darwin\" &"
```

```
"ninja"       !"rsh ninja \"xterm -ls -display floss:0 -T ninja\" &"
"indy"        !"rsh indy \"xterm -ls -display floss:0 -T indy\" &"
}

# I salvaschermo di xlock, richiamati con varie opzioni (ognuna
# delle quali da' un'immagine diversa):
menu "xlock"
{
"Hop"   !"xlock -mode hop &"
"Qix"   !"xlock -mode qix &"
"Flame" !"xlock -mode flame &"
"Worm" !"xlock -mode worm &"
"Swarm" !"xlock -mode swarm &"
"Hop NL"   !"xlock -mode hop -nolock &"
"Qix NL"   !"xlock -mode qix -nolock &"
"Flame NL" !"xlock -mode flame -nolock &"
"Worm NL" !"xlock -mode worm -nolock &"
"Swarm NL" !"xlock -mode swarm -nolock &"
}

# Programmi vari che uso ogni tanto:
menu "misc"
{
"Xload"          !"/usr/bin/X11/xload &"
"XV"             !"/usr/bin/X11/xv &"
"Bitmap"         !"/usr/bin/X11/bitmap &"
"Tetris"         !"/usr/bin/X11/xtetris &"
"Hextris"        !"/usr/bin/X11/xhextris &"
"XRoach"         !"/usr/bin/X11/xroach &"
"Analog Clock"  !"/usr/bin/X11/xclock -analog &"
"Digital Clock" !"/usr/bin/X11/xclock -digital &"
}

# Questo e' il menu' che ho collegato al pulsante centrale del mouse:
menu "stuff"
{
"Chores"       f.title
"Sync"         !"/bin/sync"
"Who"          !"who | xmessage -file - -columns 80 -lines 24 &"
```

```
"Xhost +"      !"/usr/bin/X11/xhost + &"
"Rootclear"    !"/home/larry/bin/rootclear &"
}

# Funzioni di X che talvolta sono comode:
menu "x"
{
"X Stuff"               f.title
"Xhost +"               !"xhost + &"
"Refresh"               f.refresh
"Source .twmrc"         f.twmrc
"(De)Iconify"           f.iconify
"Move Window"           f.move
"Resize Window"         f.resize
"Destroy Window"        f.destroy
"Window Ops"            f.menu "windowops"
""                      f.nop
"Kill twm"              f.quit
}

# Un sottomenu' del precedente:
menu "windowops"
{
"Window Ops"            f.title
"Show Icon Mgr"         f.showiconmgr
"Hide Icon Mgr"         f.hideiconmgr
"Refresh"               f.refresh
"Refresh Window"        f.winrefresh
"twm version"           f.version
"Focus on Root"         f.unfocus
"Source .twmrc"         f.twmrc
"Cut File"              f.cutfile
"(De)Iconify"           f.iconify
"DeIconify"             f.deiconify
"Move Window"           f.move
"ForceMove Window"      f.forcemove
"Resize Window"         f.resize
"Raise Window"          f.raise
"Lower Window"          f.lower
```

```
"Raise or Lower"        f.raiselower
"Focus on Window"       f.focus
"Raise-n-Focus"         f.function "raise-n-focus"
"Destroy Window"        f.destroy
"Kill twm"              f.quit
}
```

Phew! Credetemi, non è nemmeno il file .twmrc più complicato che abbia mai visto. Probabilmente avrete un esempio decente di `.twmrc` con la vostra versione di X. Guardate nella directory `/usr/lib/X11/twm/` o in `/usr/X11/lib/X11/twm` e vedete com'è.

Un bug a cui bisogna fare attenzione in `.twmrc` è scordarsi di mettere l'& dopo un comando da menù. Se notate che X si congela quando gli date determinati comandi, è probabile che sia questa la causa. Uscite da X con Control-Alt-Backspace, modificate `.twmrc` e riprovate.

9.2.2 Configurazione di Fvwm

Se state usando `fvwm` ci sono dei buoni file di esempio nella directory `/usr/lib/X11/fvwm/` (o `/usr/X11/lib/X11/fvwm/`).

[Gente: non so niente di fvwm, anche se credo di poter capire qualcosa dai file di configurazione di esempio. Comunque, lo stesso può fare il lettore :-). Inoltre, dato che esiste il file system.twmrc nella directory menzionata, mi chiedo se vale la pena di dare il lungo esempio di .twmrc qui sopra. Per ora ce lo lascio, ma non so se va bene o no. -Karl]

9.3 Altri file di inizializzazione

Altri file di inizializzazione notevoli sono:

`.emacs`	Letto dall'editor di testi Emacs al suo avvio.
`.netrc`	Dà i nomi di login e le password di default per ftp.
`.rhosts`	Rende l'account accessibile da remoto.
`.forward`	Per inoltrare automaticamente la posta.

9.3.1 Il file di inizializzazione di Emacs

Se usate `emacs` come editor primario, il file `.emacs` è piuttosto importante. Se ne tratta largamente nel Capitolo 8.

9.3.2 Impostazioni dell'FTP

Il file .netrc vi permette di impostare dei default per l'`ftp` prima di avviare `ftp` stesso. Ecco un piccolo esempio:

```
machine floss.life.uiuc.edu login larry password fishSticks
machine darwin.life.uiuc.edu login larry password fishSticks
machine geta.life.uiuc.edu login larry password fishSticks
machine phylo.life.uiuc.edu login larry password fishSticks
machine ninja.life.uiuc.edu login larry password fishSticks
machine indy.life.uiuc.edu login larry password fishSticks

machine clone.mcs.anl.gov login fogel password doorm@
machine osprey.mcs.anl.gov login fogel password doorm@
machine tern.mcs.anl.gov login fogel password doorm@
machine altair.mcs.anl.gov login fogel password doorm@
machine dalek.mcs.anl.gov login fogel password doorm@
machine juju.mcs.anl.gov login fogel password doorm@

machine sunsite.unc.edu login anonymous password larry@cs.oberlin.edu
```

Ogni linea dei file `.netrc` specifica il nome della macchina, il nome di login da usare per default su quella macchina, e una password. È molto comodo se fate molto spesso `ftp` e siete stanchi di digitare continuamente lo username e la password su diversi siti. Se fate `ftp` su una delle macchine elencate nel file, il programma proverà a collegarsi automaticamente usando le informazioni presenti nel file `.netrc`.

Si può dire ad `ftp` di ignorare il file .netrc e di non tentare di fare login automaticamente richiamandolo con l'opzione `-n`: "`ftp -n`".

Bisogna assicurarsi che il file `.netrc` sia leggibile *solo* da voi. Usate il comando `chmod` per impostare i permessi di lettura del file. Se altri possono leggerlo, potranno scoprire le vostre password sugli altri siti che vi sono nominati. Questo è una delle falle di sicurezza più gravi che si possono avere; per incoraggiarvi ad essere prudenti, `ftp` ed altri programmi che usano il file `.netrc` si rifiuteranno di funzionare se i permessi di lettura del file sono impostati male.

C'è di più sul file `.netrc` di quanto abbia detto: quando vi capita, fate "`man .netrc`" o "`man ftp`".

9.3.3 Come permettere l'accesso remoto al vostro account

Se avete un file `.rhosts` nella home directory, vi permetterà di far girare remotamente programmi su questa macchina; potrete cioè essere collegati alla macchina `cs.oberlin.edu`, e con un file `.rhosts` ben configurato su `floss.life.uiuc.edu` far girare un programma su `floss.life.uiuc.edu` e far arrivare l'output su `cs.oberlin.edu`, senza nemmeno dovervi collegare o digitare la password.

Un file `.rhosts` è fatto così:

```
frobnozz.cs.knowledge.edu jsmith
aphrodite.classics.hahvaahd.edu wphilps
frobbo.hoola.com trixie
```

Il formato è abbastanza chiaro: il nome di una macchina, seguito dal nome dell'utente. Supponiamo che questo esempio sia il mio file `.rhosts` su `floss.life.uiuc.edu`: ciò significherebbe che potrei far girare dei programmi su floss, con l'output visualizzato su una qualsiasi delle macchine elencate, fintanto che io fossi collegato con il nome di utente corrispondente.

Il meccanismo esatto con cui si fa girare un programma remoto è di solito il programma `rsh`, che sta per "remote shell", e quello che fa è inizializzare una shell su una macchina remota ed eseguire un comando specificato. Ad esempio:

```
frobbo$ whoami
trixie
frobbo$ rsh floss.life.uiuc.edu "ls ~"
foo.txt    mbox   url.ps    snax.txt
frobbo$ rsh floss.life.uiuc.edu "more ~/snax.txt"
[snax.txt viene visualizzato qui]
```

L'utente trixie su floss.life.uiuc.edu, che ha il file .rhosts di esempio riportato qui sopra, permette esplicitamente a trixie su frobbo.hoola.com di far girare programmi come trixie da floss.

Non c'è bisogno di avere lo stesso nome di utente su tutte le macchine per far funzionare bene un file `.rhosts`. Usate l'opzione "`-l`" con `rsh`, per dire alla macchina remota quale nome di utente usare per fare login. Se quel nome di utente esiste sulla macchina remota, ed ha un file `.rhosts` con la macchina ed il nome di utente su cui siete collegati, `rsh` funzionerà.

```
frobbo$ whoami
trixie
frobbo$ rsh -l larry floss.life.uiuc.edu "ls ~"
[Elenco della mia directory su floss]
```

Tutto questo funzionerà se l'utente `larry` su `floss.life.uiuc.edu` ha un file `.rhosts` che permette all'utente `trixie` da `frobbo.hoopla.com` di far girare dei programmi sul suo account. Se sono o meno la stessa persona è irrilevante: le uniche cose importanti sono i nomi degli utenti, i nomi delle macchine e le voci nel file `.rhosts` di larry su floss. Notate che il file `.rhosts` di trixie su frobbo non c'entra niente, importa solo quello sulla macchina remota.

Ci sono altre combinazioni che possono entrare in un file `.rhosts`—ad esempio, si può lasciare vuoto il nome dell'utente dopo il nome di una macchina remota, per permettere a tutti gli utenti di quella macchina di far girare programmi sulla macchina locale! Questo è, naturalmente, un rischio per la sicurezza: qualcuno potrebbe far girare un programma che rimuove i vostri file, solo avendo un account su una determinata macchina. Se fate cose come lasciare bianco il nome dell'utente, vi dovreste accertare che il file `.rhosts` è leggibile solo da voi.

9.3.4 Forwardare la posta

Potete anche avere un file `.forward`, che non è strettamente parlando un file di inizializzazione. Se contiene un indirizzo di posta elettronica, tutta la posta che vi arriva sarà forwardata a quell'indirizzo. È utile quando avete account su diversi sistemi, ma volete leggere la posta solo su uno di essi.

C'è un mare di altri possibili file di inizializzazione. Il numero esatto varierà da sistema a sistema, e dipende da quali software sono installati. Un modo di impararne di più è guardare i file nella vostra home directory il cui nome inizia per "`.`": non sono sempre file di inizializzazione, ma c'è una buona possibilità.

9.4 Alcuni esempi

L'esempio più calzante che vi posso fare è un sistema Linux che funziona. Quindi, se avete accesso ad Internet, potete tranquillamente fare telnet su `floss.life.uiuc.edu`. Fate login come "guest", con password "explorer", e date un'occhiata in giro. La maggior parte dei file di esempio che ho messo in questo libro si possono trovare in `/home/kfogel`, ma ci sono anche altre directory utente. Sentitevi liberi di copiare tutto quello che riuscite a leggere.

Per favore siate prudenti: floss non è estremamente sicuro, e potete di certo riuscire ad avere accesso come root, se ci provate abbastanza. Preferisco avere fiducia, invece che vigilare costantemente, per mantenere la sicurezza del sistema.

Capitolo 10

Comunicare con gli altri

I sistemi operativi Unix moderni sono molto bravi nel comunicare con altri computer, o nel formare delle reti. Due computer Unix distinti possono scambiarsi informazioni in moltissimi modi diversi; questo capitolo parlerà di come trarre utilità da questa grande abilità.

Cercheremo di coprire la posta elettronica, le news di Usenet e svariati programmi di utilità di Unix per le comunicazioni.

10.1 Posta elettronica

Una delle caratteristiche più popolari di Unix è la posta elettronica. Con essa si può evitare di litigare con la busta, la carta, la penna, il francobollo e la buca delle lettere, e al loro posto litigare con il computer.

10.1.1 Spedire la posta

Tutto ciò che si deve fare è digitare `mail` *nome utente* e scrivere il messaggio.

Ad esempio, supponiamo che io voglia scrivere un messaggio ad un utente con nome `sam`:

```
/home/larry# mail sam
Subject: La documentazione utente
Sto solo provando il sistema di posta.
EOT
/home/larry#
```

Il programma `mail` è molto semplice. Come `cat`, accetta input dall'input standard, una linea alla volta, finché riceve il carattere di fine testo da solo su una linea: Ctrl-d. Così, per inviare il mio messaggio dovrei digitare Invio e poi Ctrl-d.

`mail` è il modo più semplice per spedire la posta, ed è piuttosto utile quando viene usato con le pipe e con la redirezione. Ad esempio, se volessi inviare il file `report1` a "Sam", potrei fare "`mail sam < report1`", oppure addirittura "`sort report1 | mail sam`".

Comunque, il lato negativo dell'uso di `mail` per spedire la posta è che comporta l'uso di un editor veramente scarno: non si può modificare una linea una volta che si è premuto invio! Quindi, vi suggerisco di inviare la posta (quando non dovete usare le pipe o la redirezione) con la modalità mail di Emacs, che viene spiegata nella Sezione 8.10.

10.1.2 Leggere la posta

`mail` [*user*]

Il programma `mail` offre un modo abbastanza goffo di leggere la posta. Se scrivete `mail` senza parametri, vedrete:

```
/home/larry# mail
No mail for larry
/home/larry#
```

Ora mi manderò della posta, in modo da poter giocherellare con il lettore:

```
/home/larry# mail larry
Subject: Rane!
e rospi!
EOT
/home/larry# echo "pippo" | mail larry
/home/larry# mail
Mail version 5.5 6/1/90.  Type ? for help.
"/usr/spool/mail/larry": 2 messages 2 new
>N  1 larry                Tue Aug 30 18:11  10/211   "Rane!"
 N  2 larry                Tue Aug 30 18:12   9/191
&
```

Il prompt all'interno del programma di mail è una "e commerciale" ("`&`"); permette di dare un paio di semplici comandi, e visualizzerà una breve schermata di aiuto se digitate `?` e Invio.

I comandi base di `mail` sono:

`t` *lista di messaggi* Mostra (o `t`ype) i messaggi sullo schermo.

`d` *lista di messaggi* Cancella i messaggi.

`s` *lista di messaggi file* Salva i messaggi in *file*.

`r` *lista di messaggi* Risponde ai messaggi—cioè, comincia a comporre un nuovo messaggio a chiunque abbia spedito i messaggi nella lista.

`q` Esci salvando tutti i messaggi non cancellati in un file con nome `mbox` nella home directory.

Cos'è una *lista di messaggi*? È una lista di numeri interi separati da spazi, o un intervallo, come `2-4` (che è la stessa cosa di "`2 3 4`"). Si può anche immettere il nome di utente del mittente, in modo che il comando `t sam` stamperebbe a video tutta la posta arrivata da Sam. Se viene omessa la lista di messaggi, il comando viene applicato all'ultimo messaggio mostrato.

Ci sono diversi problemi con la lettura dei messaggi con il programma `mail`. Per prima cosa, se un messaggio è più lungo dello schermo, il programma non si ferma, ma lo fa scrollare fuori dallo schermo. Dovete salvarlo ed usare `more`. Secondo, non ha una buona interfaccia per la posta vecchia—cioè se volete salvare la posta e leggerla in un secondo tempo.

Emacs ha anche un programma accessorio per leggere la posta, `rmail`, che però non viene spiegato in questo libro. In aggiunta a ciò, la maggior parte dei sistemi Linux ha diversi altri programmi di lettura della posta disponibili, come `elm` o `pine`.

10.2 Troppe notizie

10.3 Cercare persone

10.3.1 Il comando `finger`

Il comando `finger` permette di acquisire informazioni sugli altri utenti del vostro sistema e di altri sistemi di tutto il mondo. Senza dubbio il comando `finger` è stato chiamato così grazie alla pubblicità della AT&T che esortava le persone ad "allungarsi e toccare qualcuno". Dato che Unix ha le sue radici nella AT&T, probabilmente questo ha divertito l'autore.

`finger` [-slpm] [*utente*][*@macchina*]

I parametri opzionali per `finger` possono confondere leggermente; in realtà non è così male: si possono chiedere informazioni su un utente locale ("sam"), su un'altra macchina ("@lionsden"), su un utente remoto ("sam@lionsden"), e informazioni sulla macchina locale (niente).

Un'altra caratteristica simpatica è che, se chiedete informazioni su un utente e non esiste un nome di account che sia precisamente quello che avete richiesto, proverà a far corrispondere il nome reale con quello che voi gli date. Ad esempio, se io digito `finger Greenfield`, mi verrà detto che per Sam Greenfield esiste un account `sam`.

```
/home/larry# finger sam
Login: sam                              Name: Sam Greenfield
Directory: /home/sam                    Shell: /bin/tcsh
Last login Sun Dec 25 14:47 (EST) on tty2
No Plan.
/home/larry# finger greenfie@gauss.rutgers.edu
[gauss.rutgers.edu]
Login name: greenfie                    In real life: Greenfie
Directory: /gauss/u1/greenfie           Shell: /bin/tcsh
On since Dec 25 15:19:41 on ttyp0 from tiptop-slip-6439
13 minutes Idle Time
No unread mail
Project: You must be joking!
No Plan.
/home/larry# finger
Login    Name                Tty  Idle  Login Time   Office     Office Phone
larry    Larry Greenfield     1   3:51  Dec 25 12:50
larry    Larry Greenfield    p0         Dec 25 12:51
/home/larry#
```

L'opzione `-s` dice a `finger` di mostrare sempre la forma breve (quello che ricevete normalmente quando fate finger su una macchina), e l'opzione `-l` invece rende sempre la forma estesa, anche quando fate finger su una macchina. L'opzione `-p` dice a `finger` che non volete vedere i file `.forward`, `.plan`, o `.project`, e `-m` gli dice che, se chiedete informazioni su un utente, volete le informazioni solo se c'è un nome di utente corrispondente—e non proverà a fare corrispondere la stringa che gli date con il nome reale.

10.3.2 Plan e progetti

Ora, cosa sono un `.plan` e un `.project`? Sono file immagazzinati nella home directory di un utente che vengono mostrati quando viene fatto finger sull'utente stesso. Potete creare i vostri file `.plan` e `.project`—l'unica restrizione è che viene mostrata solo la prima linea di un file `.project`.

Inoltre, chiunque deve avere privilegi di esecuzione nella vostra home directory (`chmod a+x ~/`) e di lettura sui file `.plan` e `.project` (`chmod a+r ~/.plan ~/.project`).

10.4 Usare i sistemi da remoto

`telnet` *sistema remoto*

Il modo principale di usare un sistema Unix remoto è attraverso `telnet`. `telnet` di solito è un programma piuttosto semplice da usare:

```
/home/larry# telnet lionsden
Trying 128.2.36.41...
Connected to lionsden
Escape character is '^]'.

lionsden login:
```

Come potete vedere, dopo che si manda un comando di `telnet`, ci si presenta un prompt di login per il sistema remoto. Posso digitare qualsiasi nome di utente (sempre che io sappia la password!) ed usare il sistema remoto quasi come se ci fossi seduto davanti.

Il modo normale di uscire da `telnet` è fare `logout` dal sistema remoto, ma si può anche digitare il carattere di Escape, che (come nell'esempio qui sopra) è di solito Ctrl-]. In questo modo mi si presenta un nuovo prompt: `telnet>`. Posso ora digitare `quit` e Invio e la connessione all'altro sistema verrà chiusa e si uscirà da `telnet` (se cambiate idea, premete semplicemente invio e tornerete al sistema remoto).

Se state usando X, create un nuovo `xterm` per l'altro sistema in cui state lavorando. Usate il comando "`xterm -title "lionsden" -e telnet lionsden &`": creerà una nuova finestra di `xterm` che automaticamente aprirà una connessione `telnet` (se fate spesso qualcosa del genere, vi conviene creare un alias o uno script di shell per farlo).

10.5 Scambiarsi file

`ftp` *sistema remoto*

Il modo normale di spedire file tra sistemi Unix è usare `ftp`, che sta per **file transfer protocol** (protocollo di trasmissione file). Dopo aver avviato il comando `ftp`, vi verrà chiesto di fare il login nel sistema remoto, più o meno come per `telnet`. Fatto questo, avrete un prompt speciale: un prompt `ftp`.

Il comando `cd` funziona normalmente, ma sul sistema remoto: cambia la directory corrente sull'*altro* sistema. Nello stesso modo, il comando `ls` elencherà i file sul sistema remoto.

I due comandi principali sono `get` e `put`. `get` trasferirà un file dal sistema remoto a quello locale, e `put` prenderà un file dal sistema locale e lo metterà in quello remoto. Entrambi i comandi agiscono sulla directory locale da cui si è avviato `ftp` e nella directory corrente nel sistema remoto (che si può cambiare usando `cd`).

Un problema comune con `ftp` è la distinzione tra file di testo e file binari. `ftp` è un protocollo piuttosto vecchio, e c'erano dei vantaggi ad assumere che i file trasferiti fossero file di testo. Alcune versione di `ftp` impostano il trasferimento di file di testo per default, il che significa che qualsiasi programma venga trasferito arriva corrotto. Per sicurezza, usate il comando `binary` prima di usare `get` o `put`.

Per uscire da `ftp` usate il comando `bye`.

10.6 Navigare in rete

La World Wide Web, o WWW, è il servizio più comunemente usato di Internet. Consiste di **pagine**, ognuna associata ad un URL—**uniform resource locator** (è un tipo di indirizzo). Gli URL sono la strana sequenza nella forma `http://www.rutgers.edu/`. Le pagine vengono in genere scritte in HTML (**hypertext markup language**).

L'HTML permette a chi scrive un documento di collegare determinate parole o frasi (o immagini) ad altri documenti in punti qualsiasi del Web. Quando un utente legge un documento, può passare rapidamente ad un altro cliccando su una parola chiave, o su un pulsante, e caricarne un altro—forse lontano migliaia di chilometri.

`netscape` [*url*]

Il più popolare software di navigazione (browser) per Linux è `netscape`, che è un browser commerciale venduto (o distribuito) dalla Netscape Communications Corporation. `netscape` gira solo sotto X.

`netscape` cerca di essere il più semplice possibile ed usa l'insieme di widget Motif per avere un'apparenza molto simile a Microsoft Windows. La strategia di base per usare `netscape` è che le parole sottolineate in blu sono collegamenti, come anche molte figure (si può determinare quali figure sono collegamenti cliccandoci sopra). Cliccando su queste parole con il pulsante sinistro del mouse, vi si presenterà una nuova pagina.

Linux supporta molti altri browser, incluso il browser web originale, `lynx`. `lynx` è un browser testuale—non mostrerà nessuna delle immagini a cui è di solito associato il Web—ma funzionerà senza X.

`lynx` [*url*]

È un po' più difficile imparare ad usare `lynx`, ma in genere giocherellare con i tasti freccia vi aiuterà. Le frecce in alto e in basso vi fanno spostare tra i collegamenti in una data pagina, e la freccia a destra segue il collegamento corrente (che è evidenziato). La freccia a sinistra ricaricherà la pagina precedente. Per uscire da `lynx`, premete `q`. `lynx` ha molti altri comandi—consultate la man page per saperne di più.

Capitolo 11

Comandi buffi

Beh, la maggior parte di chi ha avuto a che fare con i comandi UNIX trattati in questo capitolo non sarà d'accordo con questo titolo. "Che diamine! Mi hai appena fatto vedere che l'interfaccia di Linux è standard, e adesso abbiamo un sacco di comandi, ognuno che lavora in modo completamente diverso. Non mi ricorderò mai tutte queste opzioni, e mi stai dicendo che sono *buffi*?" Ebbene sì, ecco un esempio di senso dell'umorismo degli hackers. Comunque, guardate il lato positivo: non esiste un equivalente MS-DOS di questi comandi; se ne avete bisogno, li dovete comprare, e non saprete mai come sarà la loro interfaccia. Ecco un'utile – e gratuita – aggiunta, quindi divertitevi!

L'insieme di comandi coperto in questo capitolo comprende `find`, che permette all'utente di cercare nell'albero delle directory dei gruppi specifici di file, `tar`, utile per creare degli archivi da spedire o da salvare, `dd`, il copiatore a basso livello, e `sort`, che ...sì, ordina i file. Un'ultimo commento; questi comandi non sono affatto standardizzati, e, anche se troverete delle opzioni comuni a tutti i sistemi ∗IX, la versione GNU, che è quella spiegata qui e che troverete su tutti i sistemi Linux, ha di solito molte capacità aggiuntive. Quindi, se progettate di usare altri sistemi operativi di tipo UNIX, non dimenticate di controllare la pagina man del sistema in questione per imparare le differenze, che spesso non sono trascurabili.

11.1 `find`, per cercare i file

11.1.1 Generalità

Tra i vari comandi visti finora, ce ne sono alcuni che permettono all'utente di scendere ricorsivamente nell'albero delle directory per fare qualche azione: gli esempi canonici sono `ls -R` e `rm -R`. Bene. `find` è *il* comando ricorsivo. Ogni volta che pensate "Beh, devo fare questo e quest'altro su tutti questi tipi di file nella mia partizione", dovreste pensare di usare `find`. In un certo senso il fatto che `find` trovi file è un effetto secondario: il suo vero compito è valutare.

La struttura di base del comando è la seguente:

`find` *percorso* [. . .] *espressione* [. . .]

Questo almeno vale per la versione GNU; altre versioni non permettono di specificare più di un percorso, ed inoltre è molto raro che serva una cosa del genere. La spiegazione della sintassi del comando è piuttosto semplice: dite da dove volete che cominci la ricerca (la parte *percorso*; con GNU si può omettere e per default verrà presa in considerazione la directory corrente `.`), e che tipo di ricerca volete fare (la parte *espressione*).

Il comportamento standard del comando è piuttosto complicato, quindi vale la pena notarlo. Supponiamo che nella vostra home directory ci sia una directory `butta`, che contiene il file `pippo`. Digitate `find . -name pippo` (che come potete indovinare ricerca un file di nome `pippo`), e ottenete . . . niente altro che il prompt. Il problema sta nel fatto che `find` è per default silenzioso: rende 0 se la ricerca è stata completata (trovando qualcosa o meno) o un valore non-zero se ci sono stati dei problemi. Questo non accade con la versione che trovate su Linux, ma è comunque utile da ricordare.

11.1.2 Espressioni

La parte delle *espressioni* può essere divisa in quattro gruppi di parole chiave diversi: *opzioni*, *test*, *azioni* ed *operatori*; ognuno di questi può restituire un valore vero o falso, insieme ad un effetto secondario. Le differenze tra i gruppi vengono mostrate più avanti.

opzioni influiscono sulle operazioni generali di find, piuttosto che sul processo di un singolo file. Un esempio è -follow, che dice a `find` di seguire i link simbolici invece di dire semplicemente l'inode. Restituisce sempre il valore vero.

test sono veri e propri controlli (ad esempio, -empty controlla se il file è vuoto), e possono restituire i valori vero o falso.

azioni hanno anche un effetto secondario, il nome del file considerato. Possono restituire vero o falso anch'esse.

operatori non restituiscono un valore (possono essere considerati veri per convenzione), e vengono usati per costruire espressioni compresse. Un esempio è -or, che prende l'O logico delle due sottoespressioni ai suoi lati. Notate che quando si mettono due espressioni una dopo l'altra, è sottinteso -and.

Notate che `find` si basa sulla shell per leggere la linea di comando, ciò significa che le parole chiave devono essere messe tra spazi, e soprattutto che molti caratteri devono essere trattati in modo da non essere corrotti dalla shell. Si può fare sia con la barra rovesciata che con le virgolette, singole e doppie; negli esempi le parole chiave a carattere singolo verranno quotate con la barra rovesciata, perché (almeno secondo me, ma sono io che scrivo queste note!) è il modo più semplice.

11.1.3 Opzioni

Ecco la lista di tutte le opzioni note alla versione GNU di `find`. Ricordate che restituiscono sempre il valore vero.

- -daystart misura il tempo trascorso non da 24 ore fa ma dalla scorsa mezzanotte. Un vero hacker probabilmente non capirà l'utilità di questa opzione, ma un programmatore che lavora dalle otto alle cinque l'apprezzerà di sicuro.
- -depth processa il contenuto di ciascuna directory prima della directory stessa. Per dire la verità, non conosco molti usi di questa opzione, tranne che per un'emulazione del comando `rm -F` (naturalmente non potete cancellare una directory prima di aver cancellato tutti i file in essa contenuti ...).
- -follow dereferenzia (cioè, segue) i link simbolici. Implica l'opzione -noleaf; vedi sotto).
- -noleaf toglie un'ottimizzazione che dice "Una directory contiene due sottodirectory di meno del loro conteggio degli hard link". Se il mondo fosse perfetto, tutte le directory sarebbero referenziate da ognuna delle loro sottodirectory (grazie all'opzione `..`), come `.` dentro se stessa, e dal suo nome "reale" dalla sua directory madre.

  Ciò significa che ciascuna directory deve essere referenziata almeno due volte (una da se stessa, una dalla sua madre) e ogni referenza aggiuntiva proviene da una sottodirectory. In pratica, comunque, i link simbolici ed i filesystem distribuiti[1] possono rompere questa regola. Questa opzione fa girare `find` leggermente piú lento, ma può dare i risultati attesi.
- -maxdepth *livelli*, -mindepth *livelli*, dove *livelli* è un intero non negativo, che indica rispettivamente il numero massimo e minimo di livelli di directory in cui cercare. Sono obbligatori un paio di esempi: `-maxdepth 0` indica che il comando deve essere

---

[1] I filesystem distribuiti permettono ai file di apparire come locali su una macchina mentre sono in realtà da qualche altra parte.

applicato solo agli argomenti nella linea di comando, cioè senza entrare ricorsivamente nell'albero delle directory; `-mindepth 1` inibisce l'applicazione del comando agli argomenti nella linea di comando, mentre vengono processati tutti gli altri file piú in basso nell'albero.

- -version stampa semplicemente il numero di versione corrente del programma.
- -xdev, che è un nome che trae in inganno, dice a `find` di **non** incrociare i dispositivi, cioè di non cambiare filesystem. È molto utile quando si sta cercando qualcosa nel filesystem di root; in molte macchine è una partizione piuttosto piccola, ma un `find` normale cercherebbe in tutta la struttura!

11.1.4 Test

I primi due test sono molto semplici da capire: -false restituisce solo il valore falso, mentre -true restituisce solo il valore vero. Altri test che non hanno bisogno della specifica di un valore sono -empty, che restituisce il valore vero se il file è vuoto, e la coppia -nouser / -nogroup, che restituiscono il valore vero se nessuna voce di `/etc/passwd` o `/etc/group` corrisponde all'id di utente/gruppo del proprietario del file; è una cosa comune che accade nei sistemi multiutente: viene cancellato un utente, ma nei posti più strani del filesystem restano suoi file, e per la legge di Murphy occupano un sacco di posto.

Naturalmente, è possibile ricercare un utente o un gruppo specifico. I test sono -uid *nn* e -gid *nn*. Sfortunatamente non è possibile dare direttamente il nome dell'utente, ma bisogna usare l'id numerico, *nn*.

È possibile usare le forme $+nn$, che sta per "un valore strettamente maggiore di *nn*" e $-nn$, che sta per "un valore strettamente minore di *nn*"; è piuttosto stupido nel caso degli UID, ma sarà comodo con altri test.

Un'altra opzione utile è -type *c*, che restituisce il valore vero se il file è di tipo *c*. Le corrispondenze mnemoniche per le scelte possibile sono le stesse che in `ls`; quindi si ha **b** per i file block special, **c** per quelli character special, **d** per le directory, **p** per le pipe con nome, **l** per i link simbolici, e **s** per le socket. I file regolari sono indicati con una **f**. Un test correlato è -xtype, che è simile a -type tranne nel caso dei link simbolici. Se non è stato dato -follow, il file a cui si punta viene controllato al posto del link stesso. Completamente scorrelato è il test -fstype *tipo*; in questo caso viene controllato il filesystem. Credo di aver preso questa informazione dal file `/etc/mtab`, quello che indica i filesystem che vengono montati; sono sicuro che i tipi nfs, tmp, msdos ed ext2 vengono riconosciuti.

I test -inum *nn* e -links *nn* controllano se il file ha numero di inode *nn* o *nn* link, mentre -size *nn* rende vero se il file ha allocato *nn* blocchi da 512 byte (beh, non precisamente: per i file sparsi i blocchi non allocati vengono contati lo stesso). Dato che al giorno d'oggi i risultati di `ls -s` non vengono sempre misurati in parti da 512 byte (Linux per esempio usa unità di 1K), è possibile appendere a *nn* il carattere *b* per contare in byte, o *k* per contare in kilobyte.

I bit di permesso vengono controllati con il test -perm *modalità*. Se *modalità* non ha segno, i bit di permesso dei file devono combaciare perfettamente. Un − che precede i bit significa che tutti i bit di permesso devono essere impostati, ma non fa assunzioni sugli altri; un + è soddisfatto se uno qualsiasi dei bit è impostato. Ops! Mi dimenticavo di dire che la modalità è scritta in ottale o simbolicamente, come si fa in `chmod`.

Il prossimo gruppo di test è correlato all'ora di ultimo utilizzo del file; è comodo quando un utente ha riempito il suo spazio, e come al solito ci sono moltissimi file inutilizzati da anni, e di cui si è dimenticato il significato. Il problema è trovarli, e `find` è l'unica speranza. -atime *nn* rende vero se il file è stato utilizzato *nn* giorni fa – ad esempio, con un comando `chmod` – e -mtime *nn* se il file è stato modificato per l'ultima volta *nn* giorni fa. Talvolta si ha bisogno di un tempo più preciso; il test newer *file* è soddisfatto se il file considerato è stato modificato dopo *file*. Quindi, dovete semplicemente usare `touch` con la data desiderata, ed avete fatto. Il find della GNU aggiunge i test -anewer e -cnewer che si comportano in maniera simile, ed i test -amin, -cmin e -mmin, che contano il tempo in minuti invece che in periodi di 24 ore.

Per ultimo, il test che uso più spesso: -name *pattern* rende vero se il nome del file corrisponde esattamente a *pattern*, che è più o meno quello che si usa in un `ls` standard. Perché 'più o meno'? Perché naturalmente dovete ricordare che tutti i parametri sono processati dalla shell, e quei bei metacaratteri vengono espansi. Quindi, un test come `-name foo*` non renderà quello che volete, e dovreste scrivere o `-name foo` o `-name "foo*"`. Questo è probabilmente uno degli errori più comuni fatti dagli utenti disattenti, quindi scrivetelo a lettere GRANDI sul vostro schermo. Un altro problema è che, come con `ls`, i punti all'inizio non vengono riconosciuti; per questo potete usare il test -path *pattern*, che non si preoccupa dei punti e delle barre quando paragona il percorso del file considerato con *pattern*.

11.1.5 Azioni

Ho detto che le azioni fanno, appunto, un'azione. Beh, -prune invece non fa qualcosa, cioè non discende all'interno dell'albero delle directory (a meno che non si dia -depth). Di solito si trova insieme a -fstype, per scegliere tra i vari filesystem da controllare.

Le altre azioni possono essere divise in due grandi categorie:

- Azioni che *stampano* qualcosa. La più banale – e certo, l'azione di default di `find`

– è -print che stampa semplicemente il nome del/dei file che corrispondono alle altre condizioni nella linea di comando, e rende il valore vero. Una semplice variante di -print è -fprint *file*, che usa *file* invece dell'output standard, -ls elenca il file corrente nello stesso formato di `ls -dils`, -printf *formato* si comporta più o meno come la funzione del C printf(), in modo che potete specificare come formattare l'output, e -fprintf *file formato* fa la stessa cosa, ma scrivendo su *file*. Anche queste azioni rendono il valore vero.

- Azioni che *eseguono* qualcosa. La loro sintassi è un po' strana e sono largamente usate, quindi per favore dategli un'occhiata.

  -exec *comando* `\;` il comando è eseguito, e l'azione restituisce il valore vero se il suo stato finale è 0, cioè se l'esecuzione è regolare. La ragione per il `\;` è piuttosto logica: `find` non sa dove finisce il comando, e il trucco di mettere l'azione exec alla fine del comando non è applicabile. Beh, il modo migliore per segnalare la fine del comando è di usare il carattere che la shell stessa usa, cioè '`;`', ma naturalmente un punto e virgola da solo sarebbe mangiato dalla shell e non arriverebbe a `find`, quindi deve essere usato con un comando di escape. La seconda cosa da ricordare è come specificare il nome del fie corrente all'interno di *comando*, dato che probabilmente vi siete adoperati perché l'espressione faccia qualcosa, e non solo stampare `date`; si fa per mezzo della stringa `{}`. Alcune versioni vecchie di `find` richiedono che debba essere messa tra spazi bianchi – non molto comodo se ad esempio vi serve il percorso intero e non solo il nome del file – ma con il find della GNU può essere ovunque nella stringa che compone *comando*. E, chiederete di sicuro, non deve essere usato con un comando di escape o quotato? Meraviglioso: non l'ho mai dovuto fare né sotto tcsh né sotto bash (sh non considera `{` e `}` come caratteri speciali, quindi non è un problema). La mia idea è che la shell "sa" che `{}` non è un'opzione che ha senso, quindi non prova ad espanderla, fortunatamente per `find`, che la ottiene integra.

  -ok *comando* `\;` si comporta come -exec, con la differenza che per ogni file scelto viene chiesto all'utente di confermare il comando; se la risposta inizia per `y` o `Y` viene eseguito, altrimenti no, e l'azione restituisce il valore falso.

11.1.6 Operatori

Esistono numerosi operatori: eccone una lista, in ordine di precedenza decrescente.

\\(*espr* \\)

forza l'ordine precedente. Le parentesi devono naturalmente essere quotate, dato che hanno un significato anche per la shell.

! *espr*

-not *espr*
cambia il valore vero/falso dell'espressione, cioè se *espr* è vero diventa falso. Il punto esclamativo non ha bisogno di un carattere di escape, dato che è seguito da uno spazio.

espr1 espr2
espr1 -a *espr2*
espr1 -and *espr2*
corrispondono tutti all'operazione logica E, che è implicita nel primo e nel secondo caso. *espr2* non viene valutata, se *espr1* è falsa.

espr1 -o *espr2*
espr1 -or *espr2*
corrisponde all'operazione logica O. *espr2* non viene valutata, se *espr1* è vera.

espr1 , *espr2*
è l'indicazione dell'elenco; vengono valutate sia *espr1* che *espr2*, (e naturalmente anche tutti gli effetti secondari!) ed il valore finale dell'espressione è quello di *espr2*.

11.1.7 Esempi

Beh sì, `find` ha proprio troppe opzioni, lo so, ma ci sono un sacco di modi predefiniti di usarlo che vale la pena ricordare, dato che vengono usati molto spesso. Vediamone alcuni.

```
% find . -name foo\* -print
```

trova tutti i nomi di file che cominciano per `foo`. Se la stringa può trovarsi all'interno del nome, probabilmente è più indicato scrivere `"*foo*"` invece di `foo`.

```
% find /usr/include -xtype f -exec grep foobar \
          /dev/null {} \;
```

è un grep eseguito ricorsivamente a partire dalla directory /usr/include. In questo caso ci interessano sia il file regolare che i link simbolici che puntano a file regolari, da cui il test -xtype. Molte volte è più semplice evitare di specificarlo, specialmente se siamo piuttosto sicuri che non esistono file binary che contengono la stringa desiderata. E perché il /dev/null nel comando? È un trucco per forzare grep a scrivere il nome del file dove trova una corrispondenza. Il comando grep viene applicato a ciascun file in un'invocazione diversa, e quindi non pensa che sia necessario di dare come output il nome del file; ma ora ci sono *due* file, cioè il file corrente e `/dev/null`! Un'altra possibilità è fare una pipe del comando verso `xargs` e fargli fare il grep. L'ho appena provato ed ho distrutto del tutto il mio filesystem (anche queste note che sto cercando di recuperare a mano `:-(`).

```
% find /  -atime +1 -fstype ext2 -name core \
              -exec rm {} \;
```

è un tipico lavoro da crontab: cancella tutti i file `core` nei filesystem di tipo ext2 che non hanno avuto accessi nelle ultime 24 ore. È possibile che qualcuno voglia usare il file core per fare un dump post mortem, ma nessuno si può ricordare quello che stava facendo 24 ore prima...

```
% find /home -xdev -size +500k -ls > piggies
```

è utile sapere chi ha questi file che intasano il filesystem. Notate l'uso di -xdev: dato che ci interessa un solo filesystem, non è necesario scendere negli altri montati sotto `/home`.

11.1.8 Un'ultima parola

Tenete a mente che `find` è un comando che prende moltissimo tempo, dato che deve accedere a tutti gli inode del sistema per operare. È quindi saggio combinare tutte le operazioni che vi servono in un'unica invocazione di `find`, specialmente nei lavori di 'pulizia' fatti da crontab. Un esempio illuminante è questo: supponiamo di voler cancellare i file che finiscono con `.BAK` e cambiare le protezioni di tutte le directory a 771 e quelle dei file che finiscono con `.sh` a 755. E magari stiamo montando dei filesystem via NFS su un collegamento via modem, e non vogliamo controllare quei file. Perché scrivere tre comandi separati? Il modo più efficace di portare a termine il compito è questo:

```
% find . \( -fstype nfs -prune \) -o \
         \( -type d       -a -exec chmod 771 {} \; \) -o \
         \( -name "*.BAK" -a -exec /bin/rm {}   \; \) -o \
         \( -name "*.sh"  -a -exec chmod 755 {} \; \)
```

Sembra brutto (e che spreco di barre rovesciate!), ma guardando attentamente rivela che la logica che c'è sotto è molto lineare. Ricordate che quello che fa in realtà è una valutazione vero/falso; il comando compreso è solo un effetto secondario. Ma ciò significa che viene fatto solo se `find` deve valutare la parte exec dell'espressione, cioè solo se il lato sinistro della sottoespressione rende vero. Quindi, se per esempio il file considerato al momento è una directory viene valutato il primo exec ed i permessi dell'inode vengono cambiati a 771; altrimenti se ne dimentica e passa alla seconda sottoespressione. Probabilmente è più semplice vederlo in pratica che scriverlo, ma dopo un po' diventerà una cosa naturale.

11.2 `tar`, l'archiviatore a nastro

11.2.1 Introduzione

11.2.2 Le opzioni principali

11.2.3 Modificatori

11.2.4 Esempi

11.3 `dd`, il duplicatore di dati

La leggenda narra che all'inizio dei tempi, quando fu creato il primo UNIX, i suoi sviluppatori avevano bisogno di un comando a basso livello per copiare i dati tra dispositivi. Dato che avevano molta fretta, decisero di prendere in prestito la sintassi usata dalle macchine IBM-360, e di sviluppare in seguito un'interfaccia consistente con quella degli altri comandi. Il tempo passò, e tutti si erano talmente abituati allo strano modo di usare il comando `dd` che rimase com'era. Non so se è vero, ma è una bella storia.

11.3.1 Opzioni

A dire la verità, `dd` non è proprio diverso dagli altri comandi Unix: è un **filtro**, che legge per default dall'input standard e scrive sull'output standard. Così, se digitate semplicemente dd al terminale, resta in silenzio aspettando dell'input, e ctrl-C è l'unica cosa sensata da digitare.

La sintassi del comando è la seguente:

`dd` [if=*file*] [of=*file*] [ibs=*byte*] [obs=*byte*] [bs=*byte*] [cbs=*byte*] [skip=*blocchi*] [seek=*blocchi*] [count=*blocchi*] [conv={ascii, ebcdic, ibm, block, unblock, lcase, ucase, swab, noerror, notrunc, sync}]

Tutte le opzioni hanno la forma *opzione=valore*. Non sono permessi spazi né prima né dopo il segno di uguale; disturbava, dato che la shell in una situazione del genere non espandeva un nome di file, quindi non ve ne dovete preoccupare. È anche importante riordare che tutti i valori sotto forma di numeri (`byte` e `blocchi` qui sopra) possono essere seguiti da un moltiplicatore. Le possibili scelte sono **b** per blocchi, che moltiplica per 512, **k** per kilobyte (1024), **w** per parole (2) e **xm** moltiplica per **m**.

La spiegazione delle opzioni è data qui sotto.

- if=*filein* e of=*fileout* dicono a `dd` rispettivamente di leggere da *filein* e scrivere su *fileout*. Nell'ultimo caso, il file di output viene troncato al valore dato a `seek`, o, se la parola chiave non è presente, a 0 (cioè viene cancellato), prima di fare l'operazione. Ma date un'occhiata all'opzione `notrunc`.
- ibs=*nn* e obs=*nn* specificano quanti byte devono essere letti o scritti alla volta. Credo che il default sia 1 blocco, cioè 512 byte, ma non ne sono molto sicuro: quello che è sicuro è che funziona in quel modo con i file semplici. Questi parametri sono molto importanti quando si usano device speciali come input o output; ad esempio, leggere dalla rete dovrebbe impostare `ibs` a 10k, mentre un floppy da 3.5" ad alta densità ha la sua naturale grandezza di blocchi a 18k. Non impostare questi valori può non solo aumentare il tempo di compimento del comando, ma anche portare ad errori di timeout, quindi state attenti.
- bs=*nn* legge e scrive *nn* byte alla volta. Supera le parole chiave `ibs` e `obs`.
- cbs=*nn* imposta i buffer di conversione a *nn* byte. Questi buffer vengono usati quando si traduce da ASCII a EBCDIC o da un dispositivo senza blocchi ad uno che li ha. Ad esempio, i file creati sotto VMS hanno spesso blocchi di 512 byte, cosicché bisogna impostare `cbs` a 1b quando si legge un nastro VMS. Spero che non dobbiate mai lottare con questo tipo di cose!
- skip=*nbl* e seek=*nbl* dicono al programma di saltare *nbl* blocchi rispettivamente all'inizio dell'input ed all'inizio dell'output. Naturalmente il secondo caso ha senso se si dà la conversione `notrunc` (vedere sotto). Le dimensioni di ogni blocco sono il valore di `ibs` (`obs`). Attenzione: se non avete impostato `ibs` e scrivete `skip=1b` state in realtà saltando 512×512 byte, cioè 256KB. Non è precisamente quello che volevate, no?
- count=*nbl* copia solo *nbl* blocchi dall'input, ognuno delle dimensioni date da `ibs`. Questa opzione, insieme alla precedente, rimane utile se per esempio avete un file corrotto e volete recuperarne il più possibile: saltate le parti illeggibili e prendete quello che resta.
- conv=*conversione*,[*conversione*...] converte il file come specificato dall'argomento. Le conversioni possibili sono ascii, che converte da EBCDIC ad ASCII, ebcdic e `ibm`, che fanno entrambe una conversione inversa (sì, non c'è un'unica conversione da EBCDIC ad ASCII! La prima è quella standard, ma la seconda funziona meglio se dovete stampare i file su una stampante IBM), block, che porta linee che finiscono con un carattere di newline alla lunghezza di `cbs`, sostituendo la newline con degli spazi,

unblock, che fa l'opposto (elimina gli spazi in fondo alla linea, e li sostituisce con un carattere di newline), lcase e ucase, per convertire il testo da maiuscole a minuscole e viceversa, swab, che scambia ogni coppia di byte di input (ad esempio ne avete bisogno per usare un file contenente degli interi corti scritto su una macchina 680x0 con una macchina Intel), noerror, per continuare ad agire su un file anche dopo aver incontrato errori, e sync, che porta i blocchi di input alla dimensione di `ibs` aggiungendo dei caratteri NUL alla fine.

11.3.2 Esempi

L'esempio classico è quello in cui siete incappati quando avete provato a creare il primo dischetto Linux: come scrivere su un floppy senza un filesystem MS-DOS. La soluzione è semplice:

```
% dd if=disk.img of=/dev/fd0 obs=18k count=80
```

Ho deciso di non usare `ibs` perché non so qual'è la grandezza di blocchi migliore per un hard disk, ma in questo caso non avrei fatto nessun danno se al posto di `obs` avessi usato `bs` – potrebbe anche essere stato leggermente più veloce. Notate l'esplicitazione del numero di settori da scrivere (18KB vengono occupati da un settore, quindi `count` viene impostato a 80) e l'uso del nome a basso livello del dispositivo floppy.

Un'altra applicazione utile di `dd` è legata al backup su rete. Supponiamo di essere sulla macchina *alfa*, e che sulla macchina *beta* ci sia l'unità a nastro `/dev/rst0`, con un file tar che ci interessa prendere. Abbiamo gli stessi privilegi su entrambe le macchine, ma non c'è spazio su *beta* per mettere il file tar. In questo caso, possiamo scrivere

```
% rsh beta 'dd if=/dev/rst0 ibs=8k obs=20k' | tar xvBf -
```

per fare l'intera operazione in un solo passo. In questo caso, abbiamo usato le possibilità date da `rsh` di leggere dal nastro. Le dimensioni di input ed output sono impostate al default per questa operazione, cioè 8KB per leggere dal nastro e 20KB per scrivere sulla ethernet; dal punto di vista dell'altro estremo del tar, c'è lo stesso flusso di byte che si può prendere dal nastro, eccetto per il fatto che arriva in maniera non costante, ed è necessaria l'opzione B.

Dimenticavo: non credo proprio che `dd` sia un acronimo per "duplicatore dati", ma almeno è un buon modo per ricordarsi il suo significato…

11.4 `sort`, per ordinare i dati

11.4.1 Introduzione

11.4.2 Opzioni

11.4.3 Esempi

Capitolo 12

Errori, bug ed altre spiacevolezze

12.1 Evitare gli errori

Molti utenti si sentono in alcune occasioni frustrati con il sistema operativo Unix, spesso a causa del loro proprio comportamento. Una caratteristica del sistema operativo Unix, che molti utenti amano quando stanno lavorando bene, e odiano dopo una sessione a tarda notte è che molto pochi comandi chiedono conferma. Quando l'utente è sveglio ed ha la mente fresca, raramente ci pensa, ed è una cosa positiva perché il lavoro è più fluido.

Ci sono però degli svantaggi. `rm` e `mv` non chiedono mai conferma, e spesso questo porta a dei problemi. Vediamo quindi una piccola lista di ciò che vi può essere utile per evitare il disastro totale:

- Fate delle copie di riserva! Vale specialmente per i sistemi monoutente—tutti gli amministratori di sistema dovrebbero fare backup regolari del loro sistema! Una volta alla settimana è sufficiente per salvare molti file. Vedete *The Linux System Administrator's Guide* per ulteriori informazioni.
- Gli utenti individuali dovrebbero tenere i propri backup, se possibile. Se usate regolarmente più di un sistema, tentate di tenere copie aggiornate di tutti i vostri file su ciascun sistema. Se avete accesso ad un drive per floppy, dovreste fare dei backup su floppy del materiale più critico. Alla peggio, tenete delle copie aggiuntive del materiale più importante in giro per il vostro account *in una directory separata*!
- Pensate ai comandi, specialmente quelli "distruttivi" come `mv`, `rm`, e `cp` prima di agire. Attenzione alla la redirezione (`>`)—sovrascriverà i file se non state attenti. Anche il comando più innocuo può diventare sinistro.

  ```
  /home/larry/report# cp report-1992 report-1993 backups
  ```

può facilmente provocare un disastro.

```
/home/larry/report# cp report-1992 report-1993
```

- L'autore raccomanda anche, dalla sua esperienza personale, di non fare manutenzione dei file a tarda notte. La struttura delle vostre directory sembra un po' confusionaria alle 1:32 di notte? Lasciatela com'è—un po' di confusione non fa male al computer.

- Tenete conto della directory in cui siete. Talvolta il prompt che usate non mostra in quale directory state lavorando, ed il pericolo è pronto a colpire. È triste leggere dei messaggi su `comp.unix.admin`[1] di un utente `root` che era in `/` invece che in `/tmp`! Ad esempio:

```
mousehouse> pwd
/etc
mousehouse> ls /tmp
passwd
mousehouse> rm passwd
```

  La serie di comandi riportata qui sopra renderebbe qualsiasi utente molto infelice, dato che rimuove il file delle password del sistema. Senza questo file, non ci si può loggare!

12.2 Cosa fare quando qualcosa va storto

12.3 Non è colpa vostra

Sfortunatamente per i programmatori di tutto il mondo, non tutti i problemi sono causati da errori degli utenti. Unix e Linux sono sistemi complessi, e tutte le versioni note hanno dei bug. Talvolta questi sono difficili da scovare, ed appaiono solo in determinate circostanze.

Per prima cosa, cos'è un bug? Un esempio di bug è se chiedete al computer di calcolare "5+3" e lui dice "7". Anche se è un esempio banale, la maggior parte dei bug nei programmi di computer dipendono in qualche strano modo dalla matematica.

---

[1] Un gruppo di discussione internazionale su Usenet, che tratta dell'amministrazione dei computer Unix.

12.3.1 Quando c'è un bug

Se il computer dà una risposta sbagliata (verificate che sia veramente sbagliata!) o crasha, è un bug. Se un qualsiasi programma crasha o dà un errore di sistema, è un bug.

Se un comando non smette di girare può essere un bug, ma accertatevi che non gli abbiate chiesto di fare quello che deve fare in un tempo lungo. Chiedete aiuto se non sapete cosa doveva fare il programma.

Alcuni messaggi vi avviseranno della presenza di bug; altri messaggi non sono bug. Controllate la Sezione 3.3 e qualsiasi altra documentazione per controllare che non siano normali messaggi di informazione. A esempio, messaggi come "disk full" ("disco pieno") o "lp0 on fire" ("la porta lp0 va a fuoco") non sono problemi software, ma qualcosa che non va con l'hardware—non c'è sufficiente spazio su disco, o ci sono problemi con la stampante.

Se non riuscite a trovare niente su un programma, è un bug nella documentazione, e dovreste contattare l'autore del programma ed offrirvi di scriverla voi stessi. Se qualcosa non è corretto nella documentazione esistente[2], è un bug del manuale. Se qualcosa sembra incompleto o non chiaro nel manuale, è un bug.

Se non riuscite a vincere `gnuchess` a scacchi, è una pecca del vostro algoritmo scacchistico, non necessariamente un bug del vostro cervello.

12.3.2 Notificare un bug

Quando siete sicuri di aver trovato un bug, è importante accertarsi che questa informazione arrivi a chi di dovere. Cercate di scoprire quale programma sta causando il bug—se non ci riuscite, forse potete chiedere aiuto in `comp.os.linux.help` o in `comp.unix.misc`. Trovato il programma, leggete la pagina man per vedere chi l'ha scritto.

Il metodo preferito di spedire le notifiche di bug nel mondo Linux è via posta elettronica. Se non avete accesso ad una casella di posta elettronica, potete contattare chi vi ha fornito Linux—alla fine dovrete trovare qualcuno che ha la posta elettronica, o che vende Linux commercialmente e quindi vuole rimuovere quanti bug possibili. Ricordatevi però che nessuno è costretto ad eliminare bug a meno che non abbiate un contratto!

Quando mandate una notifica di bug, includetevi tutte le informazioni che vi vengono in mente, compreso:

- Una descrizione di quello che pensate sia sbagliato. Ad esempio, "Mi dà 5 quando faccio 2+2", e "Dice `segmentation violation -- core dumped`." È importante dire esattamente quello che accade, in modo che venga risolto il *vostro* bug!

---

[2] Specialmente questa!

- Includete qualsiasi variabile d'ambiente rilevante.
- La versione del kernel (vedere il file `/proc/version`) e delle librerie di sistema (vedere la directory `/lib`—se non riuscite a decifrarla, mandate un elenco del contenuto di `/lib`).
- Come facevate girare il programma in questione, o, se è un bug del kernel, quello che stavate facendo.
- **Tutte** le informazioni secondarie. Ad esempio, il comando `w` può non mostrare il processo corrente per alcuni utenti. Non dite semplicemente "`w` non funziona per un determinato utente". Il bug può avvenire perché il nome dell'utente è lungo più di 8 caratteri, o quando si collega attraverso la rete. Dite invece "`w` non mostra il processo corrente per l'utente `greenfie` quando si collega via rete".
- E, ricordatevi, siate educati. La maggior parte di chi lavora sul free software lo fa per il gusto di farlo, e perché hanno dei cuori grandi così. Non gli rovinate tutto—la comunità Linux ha già distrutto le illusioni di troppi sviluppatori, ed è ancora agli inizi!

Appendice A

Introduzione a Vi

`vi` (pronunciato “vi ai”) è l’unico editor che si trova veramente in quasi tutte le installazioni di Unix. Originariamente è stato scritto all’Università della California a Berkeley, e se ne possono trovare delle versioni in ciascuna edizione di Unix, compreso Linux. Inizialmente è un po’ ostico, ma ha molte caratteristiche potenti; in generale suggeriamo ad un utente alle prime armi di usare Emacs, che è in genere più facile da imparare. Comunque, chi usa più di una piattaforma o chi non ama Emacs può volere imparare ad usare `vi`.

Per capire perché il tasto `k` può significare di spostare il cursore in alto di una linea e perché ci sono tre modalità diverse è necessaria una breve introduzione storica di `vi`. Se siete impazienti di imparare ad usare l’editor, i due manuali vi guideranno dai primi comandi alla conoscenza di tutti i comandi di cui mai avrete bisogno. Il capitolo comprende anche una guida ai comandi, che serve come riferimento da tenere accanto al terminale.

Anche se `vi` non diventerà il vostro editor di testi usuale, non fa male sapere come funziona. È praticamente sicuro che il sistema Unix che usate abbia qualche variante di `vi`. Forse sarà necessario usare `vi` per installare qualche altro editor, come Emacs. Molti strumenti, applicazioni e giochi Unix usano un sottoinsieme dei comandi di `vi`.

A.1 Breve storia di Vi

I primi editor di testo erano editor di linea, e venivano usati tipicamente da terminali di stampa stupidi. Un editor tipico che operava in questa modalità è **Ed**: come editor era potente ed efficiente, usava pochissime risorse del computer, e funzionava bene con i monitor del tempo. `vi` offre un’alternativa visiva con un insieme dei comandi significativamente esteso in confronto ad `ed`.

`vi` come lo conosciamo è cominciato come editor di linea, e si chiamava `ex`. In effetti, `ex` si può vedere come una modalità particolare di `vi`, anche se in effetti è vero il contrario. La componente visiva di `ex` può essere avviata da linea di comando usando il comando `vi`, o da dentro `ex`.

L'editor `ex/vi` è stato sviluppato all'Università della California a Berkeley da William Joy. Originariamente era fornito come utility non supportata fino alla sua inclusione ufficiale nella distribuzione dello Unix System V della AT&T. Da allora è diventato sempre più popolare, anche dopo le sfide di editor più moderni a tutto schermo.

Grazie alla popolarità di `vi` ne esistono molte varianti e se ne possono trovare versioni sulla maggior parte dei sistemi operativi. Non è intenzione di questo capitolo includere tutti i comandi disponibili sotto `vi` o sotto le sue varianti; molti cloni hanno esteso e modificato il comportamento originale di `vi`, e la maggior parte non supportano tutti i comandi originari di `vi`.

Se avete una buona conoscenza di lavoro di `ed`, `vi` sarà facile da imparare. Anche se non avete intenzione di usare `vi` come normale editor, una sua conoscenza di base non può che essere utile.

A.2 Breve manuale di Ed

Lo scopo di questo manuale è di farvi cominciare ad usare `ed`. `ed` è progettato per essere facile da usare, e richiede poco esercizio per iniziare. Il modo migliore per imparare è esercitarsi, quindi seguite le istruzioni e provatelo prima di disprezzare i suoi vantaggi pratici.

A.2.1 Creare un file

`ed` è capace di modificare solo un file alla volta. Seguite il seguente esempio per creare il vostro primo file di testo con `ed`.

```
/home/larry# ed
a
Questo e' il mio primo file di testo con Ed.
E' veramente divertente.
.
w primo.txt
/home/larry# q
```

Potete verificare il contenuto del file usando il comando di concatenamento file di Unix:

```
/home/larry# cat primo.txt
```

L'esempio qui sopra illustra diversi punti importanti. Quando richiamate `ed` come qui sopra avrete un file vuoto. Il tasto `a` viene usato per aggiungere del testo al file. Per finire di inserire il testo, viene usato un punto `.` nella prima colonna del testo stesso. Per salvare il testo in un file, si usa il tasto `q` in combinazione con il nome del file, ed infine si usa il tasto `q` per uscire dall'editor.

L'osservazione più importante è che ci sono due modalità di operazione: inizialmente l'editor è in modalità comandi; un comando è definito da caratteri, quindi per capire le intenzioni dell'utente `ed` usa una **modalità testo** e una **modalità comandi**.

A.2.2 Modificare un file esistente

Per aggiungere una linea di testo ad un file esistente seguite il prossimo esempio.

```
/home/larry# ed primo.txt
a
Questa e' un'altra linea di testo.
.
w
q
```

Se controllate il file con `cat` vedrete che è stata aggiunta una nuova linea tra la prima e la seconda linea del file originario. Come faceva `ed` a sapere dove inserire la nuova linea di testo?

Quando `ed` legge il file tiene traccia della linea corrente; il comando `a` aggiungerà del testo dopo la linea corrente. `ed` può anche inserire il testo prima della linea corrente con il tasto di comando `i`: l'effetto sarà l'inserimento del testo prima della linea corrente.

Ora è facile vedere che `ed` opera sul testo, linea per linea. Tutti i comandi possono essere applicati alla linea scelta.

Per aggiungere una linea di testo alla fine del file:

```
/home/larry# ed primo.txt
        $a
        L'ultima linea di testo.
        .
        w
        q
```

Il modificatore di comandi `$` dice a `ed` di aggiungere la linea dopo l'ultima linea del testo. Per aggiungere una linea prima della prima linea il modificatore sarebbe `1`. Possiamo ora scegliere la linea a cui premettere o posporre una linea di testo da inserire.

Come facciamo a sapere che c'è sulla linea corrente? Il tasto di comando `p` mostrerà il contenuto della linea corrente. Se volete mettere la linea corrente alla linea 2 e vederne il contenuto, fate così:

```
/home/larry# ed primo.txt
        2p
        q
```

A.2.3 I numeri di linea in dettaglio

Avete visto come mostrare il contenuto della linea corrente, con l'uso del comando `p`; sapete anche che ci sono dei modificatori dei numeri di linea per i comandi. Per stampare il contenuto della seconda linea, digitate

```
        2p
```

Ci sono dei modificatori speciali che si riferiscono a posizioni che possono variare nella durata della sessione di modifica del file: il tasto `$` è l'ultima linea del testo. Per stampare l'ultima linea:

```
        $p
```

Il numero di linea corrente usa il modificatore speciale `.`. Per mostrare la linea corrente usando un modificatore, fate:

```
        .p
```

Questa procedura può sembrare inutile, anche se è molto utile quando si parla di intervalli di numeri di linea.

Per mostrare il contenuto del testo dalla linea 1 alla linea 2, bisogna dare a `ed` un intervallo.

```
        1,2p
```

Il primo numero si riferisce alla linea di inizio, e il secondo si riferisce alla linea di fine. La linea corrente sarà di conseguenza il secondo numero dell'intervallo di comando.

Se volete mostrare il contenuto del file dall'inizio alla linea corrente:

```
1,.p
```

Per mostrare il contenuto del file dalla linea corrente alla fine:

```
.,$p
```

Manca solo di mostrare il contenuto del file intero, che viene lasciato come esercizio a voi.

Per cancellare le prime 2 linee del file:

```
1,2d
```

Il tasto di comando `d` cancella il testo linea per linea; se volete cancellare l'intero contenuto dovete dare:

```
1,$d
```

Se avete fatto molti cambiamenti e non volete salvare il contenuto del file, l'opzione migliore è uscire dall'editor senza prima salvare il file.

La maggior parte degli utenti non usano `ed` come editor principale. Gli editor più moderni sono a tutto schermo, ed hanno insiemi di comandi più flessibili. `ed` è una buona introduzione a `vi` e spiega da dove provengono i suoi comandi.

A.3 Breve manuale di Vi

Lo scopo di questo manuale è di cominciare ad usare l'editor `vi`; non assume nessuna esperienza di `vi`, quindi vi saranno spiegati i dieci comandi più usati, che sono sufficienti per fare la maggior parte delle modifiche ai file; potrete allargare il vostro vocabolario di `vi` quando vi serve. È preferibile che abbiate una macchina su cui fare pratica, a mano a mano che andate avanti.

A.3.1 Avviare vi

Per avviare vi, digitate semplicemente le lettere `vi` seguite dal nome del file che volete creare. Vedrete uno schermo con una colonna di tilde (~) sul lato sinistro. `vi` è ora in modalità comandi: qualsiasi cosa che inserirete verrà interpretata come un comando. I due comandi di input base sono:

```
i     inserisce del testo a sinistra del cursore
a     aggiunge del testo a destra del cursore
```

Dato che siete all'inizio di un file vuoto, non importa quale dei due usate. Digitatene uno, e poi inserite il testo seguente (una poesia di Augustus DeMorgan trovate in *The Unix Programming Environment* di B.W. Kernighan e R. Pike):

```
Great fleas have little fleas<Invio>
  upon their backs to bite 'em,<Invio>
And little fleas have lesser fleas<Invio>
  and so ad infinitum.<Invio>
And the great fleas themselves, in turn,<Invio>
  have greater fleas to go on;<Invio>
While these again have greater still,<Invio>
  and greater still, and so on.<Invio>
<Esc>
```

Notate che bisogna premere il tasto Esc per finire l'inserzione e tornare in modalità comandi.

A.3.2 Comandi di spostamento del cursore

`h`	`sposta il cursore uno spazio a sinistra`
`j`	`sposta il cursore uno spazio verso il basso`
`k`	`sposta il cursore uno spazio verso l'alto`
`l`	`sposta il cursore uno spazio a destra`

Questi comandi possono essere ripetuti tenendo premuto il tasto: provate a spostarvi in giro per il testo. Se provate a fare uno spostamento impossibile, come premere la lettera **k** quando il cursore è sulla prima linea, lo schermo lampeggerà, o il terminale farà un suono. Non vi preoccupate, non morde, e non fa male al file.

A.3.3 Cancellare del testo

`x`	`cancella il carattere sul cursore`
`dd`	`cancella una linea`

Spostate il cursore sulla seconda linea e posizionatelo in modo che sia sotto l'apostrofo nella parola *'em*. Premete la lettera x, e l'apostrofo scomparirà. Ora premete la lettera i per tornare in modalità inserimento e digitate le lettere **th**. Premete Esc quando avete finito.

A.3.4 Salvare i file

```
:w    salva (scrive su disco)
:q    esce
```

Assicuratevi di essere in modalità comandi premendo il tasto Esc. Ora digitate **:wq** e salverete il vostro lavoro scrivendolo su un file sul disco.

Il comando per uscire da vi è q. Se volete combinare il salvataggio e l'uscita, digitate semplicemente **:wq**. C'è anche un'abbreviazione conveniente per **:wq**—**ZZ**. Dato che la maggior parte del lavoro di programmazione consiste nel far girare un programma, trovare un problema, richiamare il programma in un editor, fare dei piccoli cambiamenti, e poi uscire dall'editor per far girare di nuovo il programma, **ZZ** sarà un comando che userete spesso. (In realtà, **ZZ** non è un sinonimo esatto di **:wq** — se non avete fatto nessuna modifica al file dall'ultimo salvataggio, **ZZ** uscirà senza salvare, mentre **:wq** salverà lo stesso.)

Se avete incasinato le cose senza speranza, e volete solo ricominciare da capo, potete digitare **:q!** (ricordatevi prima di premere il tasto Esc). Se non mettete il punto esclamativo, vi non vi farà uscire senza salvare.

A.3.5 E poi?

I dieci comandi che avete appena imparato dovrebbero esservi sufficienti per lavorare; comunque, avete solo grattato la superficie dell'editor vi: ci sono comandi per copiare e spostare materiale da un posto all'altro in un file, per spostare materiale da un file ad un altro, per modificare l'editor secondo i vostri gusti personali, eccetera. In tutto esistono circa 150 comandi.

A.4 Manuale avanzato di vi

Il vantaggio e la potenza di `vi` sono nella possibilità di usarlo con successo conoscendo solo un piccolo sottoinsieme dei comandi. La maggior parte degli utenti di `vi` si sentono un po' strani all'inizio, ma dopo poco tempo sentono il bisogno di conoscerlo meglio.

Il manuale che segue assume che l'utente abbia finito il manuale breve (qui sopra) e si senta a proprio agio con `vi`; mostrerà alcune delle caratteristiche più potenti di `ex/vi`, dalla copia del testo alle definizioni delle macro; c'è una sezione su `ex` e le sue impostazioni, che aiuta a personalizzare l'editor. Questo manuale descrive i comandi, piuttosto che portarvi a conoscerli insieme per insieme. È preferibile che spendiate del tempo a provarli su un testo di esempio, che possiate distruggere.

Questo manuale non descrive tutti i comandi di `vi` anche se sono coperti tutti quelli usati più comunemente. Anche se sceglieste di usarlo come editor alternativo, si spera che apprezziate `vi` e quello che offre a chi lo sceglie.

A.4.1 Spostarsi nel testo

La funzione più semplice di un editor è di spostare il cursore nel testo. Ecco alcuni altri comandi di spostamento:

`h`	sposta il cursore uno spazio a destra
`j`	sposta il cursore una linea in basso
`k`	sposta il cursore una linea in alto
`l`	sposta il cursore una linea a destra

Alcune implementazioni permettono anche di spostare il cursore con i tasti freccia.

`w`	sposta all'inizio della parola seguente
`e`	sposta alla fine della parola seguente
`E`	sposta alla fine della parola seguente prima di uno spazio
`b`	sposta all'inizio della parola precedente
`0`	sposta all'inizio della linea
`^`	sposta alla prima parola della linea corrente
`$`	sposta alla fine della linea
`<CR>`	sposta all'inizio della linea seguente
`-`	sposta all'inizio della linea precedente
`G`	sposta alla fine del file
`1G`	sposta all'inizio del file
`nG`	sposta alla linea numero n
`<Cntl> G`	mostra il numero di linea corrente
`%`	alla parentesi corrispondente
`H`	alla prima linea della schermata
`M`	alla linea centrale della schermata
`L`	all'ultima linea della schermata
`n|`	sposta il cursore alla colonna n

Lo schermo scorrerà automaticamente quando il cursore raggiunge o la cima o il fondo della schermata. Ci sono dei comandi alternativi che possono controllare lo scorrimento dello schermo:

```
<Cntl> f   scorre una schermata avanti
<Cntl> b   scorre una schermata indietro
<Cntl> d   scorre mezza schermata avanti
<Cntl> u   scorre mezza schermata indietro
```

I comandi riportati qui sopra controllano lo spostamento del cursore. Alcuni di questi comandi accettano un modificatore nella forma di un numero che precede il comando, che di solito ripeterà il comando stesso quel numero di volte.

Per spostare il cursore un numero di posizioni a sinistra:

```
nl         sposta il cursore n posizioni a sinistra
```

Se voleste inserire n spazi prima di una parte di testo, potreste usare il modificatore di comando per il comando di inserimento: digitate il numero di volte che volete ripetere l'inserimento, poi premete Esc.

```
ni         inserisce del testo e lo ripete n volte
```

I comandi che riguardano le linee usano il modificatore per riferirsi al numero di linee: un buon esempio è il comando G.

```
1G        sposta il cursore alla prima linea
```

`vi` ha un gran numero di comandi che possono essere usati per spostare il cursore all'interno del file, da spostamenti di un carattere singolo a spostamenti diretti del cursore su una linea. `vi` può anche posizionare il cursore ad una linea scelta da linea di comando.

```
vi +10 mio.tex
```

Questo comando apre il file *mio.tex* e posiziona il cursore a 10 linee dall'inizio del file.

Provate alcuni dei comandi di questa sezione. Pochissime persone se li ricordano tutti insieme; la maggior parte ne usano solo un piccolo sottoinsieme.

Bene. Ora vi potete spostare nel testo: come fate a modificarlo?

A.4.2 Modificare il testo

Lo scopo è di cambiare il contenuto del file, e `vi` offre un vastissimo insieme di comandi per aiutare a farlo.

Questa sezione focalizzerà l'attenzione su come aggiungere testo, modificare quello esistente, e cancellarlo. Alla fine di questa sezione avrete le conoscenze necessarie per creare qualsiasi file di testo. Le sezioni restanti trattano di comandi più comodi da digitare.

Quando si inserisce del testo, si possono inserire delle linee multiple usando il tasto Invio. Se bisogna correggere un errore di battitura e siete sulla linea in questione in modalità inserimento, potete usare il tasto backspace per spostare il cursore nel testo. Le diverse versioni di `vi` si comportano in maniera diversa; alcune spostano semplicemente il cursore indietro e il testo può essere ancora visualizzato ed accettato, altre rimuoveranno il testo a mano a mano che tornate indietro, alcuni cloni permettono addirittura di spostare il cursore dalla modalità inserimento con i tasti freccia; non è il comportamento standard di `vi`. Se il testo è visibile e usate il tasto Esc quando siete sulla linea in cui avete cancellato il testo con backspace, il testo dopo il cursore viene cancellato. Usate la vostra versione dell'editor per abituarvici.

`a`	`Aggiunge del testo dopo la posizione corrente del cursore`
`A`	`Aggiunge del testo alla fine della linea`
`i`	`Inserisce del testo a sinistra del cursore`
`I`	`Inserisce del testo a sinistra del primo carattere non vuoto della linea corrente`
`o`	`Apre una nuova linea ed inserisce il testo sotto la linea corrente`
`O`	`Apre una nuova linea ed inserisce il testo sopra la linea corrente`

`vi` ha un piccolo insieme di comandi di cancellazione che possono essere migliorati con l'uso dei modificatori.

`x`	`Cancella un carattere da sotto il cursore`
`dw`	`Cancella dalla posizione corrente alla fine della parola`
`dd`	`Cancella la linea corrente`
`D`	`Cancella dalla posizione corrente alla fine della linea`

I modificatori possono essere usati per aggiungere potere ai comandi. Gli esempi qui sotto sono solo alcune delle possibilità fornite.

`nx`	`Cancella n caratteri da sotto il cursore`
`ndd`	`Cancella n linee`
`dnw`	`Cancella n parole (equivalente a ndw)`
`dG`	`Cancella dalla posizione corrente alla fine del file`
`d1G`	`Cancella dalla posizione corrente all'inizio del file`
`d$`	`Cancella dalla posizione corrente alla fine della linea (Lo stesso di D)`
`dn$`	`Cancella dalla linea corrente alla fine dell'n-esima linea`

L'elenco di comandi qui sopra mostra che l'operazione di cancellazione è molto potente. Questo diventa evidente quando viene applicata in combinazione con i comandi di spostamento del cursore. Un comando da annotare è `D`, dato che ignora le direttive dei modificatori.

In determinate occasioni potreste voler disfare i cambiamenti fatti. I seguenti comandi riportano il testo alla situazione precedente ai cambiamenti:

`u`	`Annulla l'ultimo comando`
`U`	`Annulla tutti i cambiamenti fatti sulla linea corrente`
`:e!`	`Riporta allo stato dell'ultimo salvataggio`

`vi` non vi permette solo di annullare i cambiamenti, ma annulla gli annullamenti. Usando il comando `5dd` cancellate 5 linee, poi riportatele con il comando `u`. Le modifiche possono essere rieseguite con un altro comando `u`.

`vi` offre dei comandi che permettono di fare delle modifiche al testo senza dover prima cancellare e poi ridigitare la nuova versione:

`rc`	`Sostituisce il carattere sotto il cursore con c Sposta il cursore a destra se viene ripetuto con un modificatore, come in 2rc)`
`R`	`Sovrascrive il testo con il nuovo testo`
`cw`	`Modifica la parola corrente`
`c$`	`Modifica il testo dalla posizione corrente alla fine della linea`
`cnw`	`Modifica le prossime n parole (lo stesso di ncw).`
`cn$`	`Modifica fino alla fine dell'n-esima linea`
`C`	`Modifica fino alla fine della linea (lo stesso di c$)`
`cc`	`Modifica la linea corrente`
`s`	`Sostituisce il testo che digitate al posto del carattere corrente`

`ns`	`Sostituisce il testo che digitate al posto dei seguenti n caratteri`

Dai comandi di modifica che permettono di dare una stringa di caratteri si esce con il tasto Esc.

Il comando cw parte dalla posizione corrente nella parola alla fine della parola stessa; quando si usa un comando di modifica che specifica una distanza verrà applicato il cambiamento: `vi` metterà un **$** come ultimo carattere. Il nuovo testo può essere più lungo o più breve del testo originale.

A.4.3 Copiare e spostare delle sezioni di testo

Spostare del testo coinvolge un buon numero di comandi, che vengono combinati per raggiungere il risultato finale. Questa sezione introdurrà dei buffer con e senza nome insieme ai comandi che tagliano ed incollano il testo.

Copiare del testo implica tre passi principali:

1. **Yanking** (copia) del testo in un buffer.
2. **Moving** (sposta) del cursore alla posizione destinata
3. **Pasting** (incolla) del testo nel buffer di modifica.

Per copiare del testo nel buffer senzanome usate il comando y.

`yy`	`Sposta una copia della linea corrente al buffer senzanome.`
`Y`	`Sposta una copia della linea corrente al buffer senzanome.`
`nyy`	`Sposta le n linee successive nel buffer senzanome.`
`nY`	`Sposta le n linee successive nel buffer senzanome.`
`yw`	`Sposta una parola nel buffer senzanome.`
`ynw`	`Sposta n parole nel buffer senzanome.`
`nyw`	`Sposta n parole nel buffer senzanome.`
`y$`	`Sposta la posizione corrente alla fine della linea.`

Il buffer senzanome è un buffer temporaneo che viene corrotto facilmente da altri comandi comuni. In determinate occasioni il testo può essere necessario per un lungo periodo di tempo; in questo caso viene usato un buffer con nome. `vi` ha 26 buffer con nome, che si identificano con le lettere dell'alfabeto. Per distinguere tra un comando e il nome di un buffer, `vi` usa il carattere ". Quando si usa un buffer con nome con la lettera minuscola, il contenuto viene sostituito, mentre se si usa la lettera maiuscola, il contenuto viene aggiunto a quello già presente nel buffer.

`"ayy`	`Sposta la linea corrente nel buffer a.`
`"aY`	`Sposta la linea corrente nel buffer a.`
`"byw`	`Sposta la parola corrente nel buffer b.`
`"Byw`	`Aggiungi la parola corrente al contenuto del buffer b.`
`"by3w`	`Sposta le prossime 3 parole nel buffer b.`

Usate il comando `p` per incollare il contenuto del buffer tagliato nel buffer di modifica.

`p`	`Incolla dal buffer senzanome a DESTRA del cursore`
`P`	`Incolla dal buffer senzanome a SINISTRA del cursore`
`nP`	`Incolla n copie del buffer senzanome a SINISTRA del cursore`
`"ap`	`Incolla dal buffer a a DESTRA del cursore`
`"b3P`	`Incolla 3 copie del buffer b a SINISTRA del cursore`

Quando state usando `vi` dall'interno di un xterm, avete una o più opzioni per copiare del testo. Selezionate la sezione di testo che volete copiare trascinando con il mouse sopra il testo. Tenere premuto il pulsante sinistro del mouse e trascinare dall'inizio alla fine del testo invertirà il colore del testo stesso; questo piazza automaticamente il testo in un buffer riservato dal server X. Per incollare il testo premete il pulsante centrale del mouse. Ricordatevi di mettere `vi` in modalità inserimento, dato che l'input potrebbe essere interpretato come comandi e il risultato è ignoto. Usando la stessa tecnica potete copiare una parola singola cliccando due volte sulla parola con il pulsante sinistro del mouse. Solo la parola verrà copiata. Per incollare la procedura è la stessa di prima. Il contenuto del buffer cambierà solamente quando viene evidenziata una nuova area.

Per spostare del testo bisogna fare tre passi:

1. **Cancellare** il testo spostandolo in un buffer con o senza nome.
2. **Spostare** il cursore al punto in cui si vuole spostare il testo.
3. **Incollare** il buffer.

Il processo è lo stesso che per copiare, con il cambiamento nel primo passo, in cui si deve cancellare il testo. Quando si usa il comando `dd` la linea viene cancellata e messa all'interno del buffer senzanome. Potete incollarne il contenuto come quando avete copiato il testo nella posizione desiderata.

`"add`	`Cancella la linea e la mette nel buffer a.`
`"a4dd`	`Cancella 4 linee e le mette nel buffer a.`
`dw`	`Cancella una parola e la mette nel buffer senzanome.`

Vedere la sezione sulla modifica del testo per altri esempi su come cancellare del testo.

Quando il sistema crasha il contenuto dei buffer con e senza nome viene perso, ma il contenuto del buffer di modifica può essere recuperato (vedere "Comandi utili").

A.4.4 Cercare e sostituire del testo

`vi` ha molti comandi di ricerca; potete cercare dei singoli caratteri, e anche espressioni regolari.

I due comandi principali di ricerca di caratteri sono `f` e `t`.

`fc`	`Trova il carattere c successivo. Si sposta a DESTRA verso il prossimo.`
`Fc`	`Trova il carattere c successivo. Si sposta a SINISTRA verso il precedente.`
`tc`	`Si sposta a DESTRA al carattere prima del prossimo c.`
`Tc`	`Si sposta a SINISTRA al carattere seguente il c precedente. (In alcuni cloni e' lo stesso di Fc)`
`;`	`Ripete gli ultimi comandi f,F,t,T`
`,`	`Lo stesso di ; ma inverte la direzione del comando originale.`

Se non trovate il carattere che stavate cercando, `vi` suonerà o vi darà un qualche altro segnale.

`vi` vi permette di cercare una stringa nel buffer di modifica:

`/str`	`Cerca a destra ed in basso il successivo evento di str.`
`?str`	`Cerca a sinistra ed in alto il successivo evento di str.`
`n`	`Ripete l'ultimo comando / o ?`
`N`	`Ripete l'ultimo comando / o ? in direzione inversa`

Quando si usano i comandi `/` o `?` viene liberata una linea in fondo allo schermo, dove si inserisce la stringa di ricerca seguita da `Invio`.

La stringa nel comando `/` o `?` può essere un'espressione regolare, cioè una descrizione di un insieme di caratteri, che viene costruita usando del testo intermezzato con dei caratteri speciali: `. * [] ^$`.

`.`	`Fa corrispondere un qualsiasi singolo carattere tranne la newline.`

`\`	Considera i caratteri speciali come normali.
`*`	Fa corrispondere 0 o piu' eventi del carattere precedente.
`[]`	Fa corrispondere esattamente uno dei caratteri tra parentesi.
`^`	La corrispondenza del carattere seguente deve essere all'inizio della linea.
`$`	Fa corrispondere i caratteri precedenti alla fine della linea.
`[^]`	Fa corrispondere tutti i caratteri non inclusi dopo il carattere di negazione.
`[-]`	Fa corrispondere un insieme di caratteri.

L'unico modo per capire bene le espressioni regolari è usarle. Qui sotto ci sono una serie di esempi:

`c.pe`	Fa corrispondere cope, cape, caper ecc
`c\.pe`	Fa corrispondere c.pe, c.per ecc
`sto*p`	Fa corrispondere stp, stop, stoop ecc
`car.*n`	Fa corrispondere carton, cartoon, carmen ecc
`xyz.*`	Fa corrispondere xyz alla fine della linea
`^The`	Fa corrispondere qualsiasi linea che cominci con The
`atime$`	Fa corrispondere qualsiasi linea che finisca con atime
`^Only$`	Fa corrispondere qualsiasi linea che contenga solo la parola Only
`b[aou]rn`	Fa corrispondere barn, born, burn.
`Ver[D-F]`	Fa corrispondere VerD, VerE, VerF.
`Ver[^1-9]`	Fa corrispondere Ver seguito da un carattere che non sia un numero
`the[ir][re]`	Fa corrispondere their,therr, there, theie.
`[A-Za-z][A-Za-z]*`	Fa corrispondere qualsiasi parola.

`vi` usa la modalità comandi di `ex` per fare le operazioni di ricerca e di sostituzione. Tutti i comandi che cominciano con un due punti sono richieste in modalità `ex`.

I comandi di ricerca e sostituzione permettono di usare le espressioni regolari su un intervallo di linee e sostituire la stringa corrispondente. L'utente può chiedere conferma prima di attuare la sostituzione. Vale la pena di riguardare come `ed` rappresenta i numeri di linea.

`:<inizio>,<fine>s/<trova>/<sostituisci>/g`	Comando generale

`:1,$s/the/The/g`	`Cerca nell'intero file e sostituisci the con The.`
`:%s/the/The/g`	`% sta per l'intero file (come sopra)`
`:.,5s/^.*//g`	`Cancella il contenuto dalla linea corrente alla linea 5.`
`:%s/the/The/gc`	`Sostituisci the con The ma chiedi conferma prima di fare la sostituizione.`
`:%s/^....//g`	`Cancella i primi quattro caratteri di ciascuna linea.`

Il comando di ricerca è molto potente se usato in combinazione con le stringhe di ricerca con le espressioni regolari. Se non si mette la direttiva $\boxed{\text{g}}$ il cambiamento viene fatto solo sulla prima corrispondenza di ciascuna linea.

Talvolta potreste voler usare la stringa di ricerca originale nel risultato delle sostituzioni. Si può digitare di nuovo il comando sulla linea, ma `vi` permette di inserire nella stringa di sostituzione alcuni caratteri speciali:

`:1,5s/help/&ing/g`	`sostituisce help con helping nelle prime 5 linee`
`:%s/ */&&/g`	`Raddoppia il numero degli spazi tra le parole.`

Usare le stringhe di corrispondenza complete ha i suoi limiti, quindi `vi` usa le parentesi di escape $\boxed{(}$ e $\boxed{)}$ per scegliere l'intervallo di sostituzione. La sostituzione può essere costruita usando il tasto con l'escape $\boxed{1}$, che identifica l'intervallo nell'ordine della definizione.

`:s/^\(.*\):.*/\1/g`	`Cancella tutto cio' che e' prima dei due punti, compresi i due punti stessi.`
`:s/\(.*\):\(.*\)/\2:\1/g`	`Inverte le parole ai due lati dei due punti.`

Molto probabilmente rileggerete queste ultime perle. `vi` offre dei comandi molto potenti, che moltissimi editor moderni non hanno; il prezzo da pagare è anche l'argomento più usato contro `vi`: i comandi possono essere difficili da imparare e da leggere. Anche se la maggior parte delle cose possono apparire un po' strane all'inizio, con un po' di esercizio alla volta l'insieme di comandi di `vi` diventerà per voi una seconda lingua.

Appendice B

The GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact

all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

    a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
    b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
    c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

    The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

    If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License.

Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE

PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> *one line to give the program's name and an idea of what it does.*
> Copyright © 19*yy name of author*
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139,

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright © 19*yy name of author*
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> *signature of Ty Coon*, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendice C

The GNU Library General Public License

GNU LIBRARY GENERAL PUBLIC LICENSE
Version 2, June 1991

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

Terms and Conditions for Copying, Distribution and Modification

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".

   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

   The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

   "Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. The modified work must itself be a software library.

   b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

   d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

   (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the

Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a “work that uses the library”. The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a “work that uses the Library” uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a “work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer’s own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable ”work that uses the Library”, as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

   b. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

   b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

    If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

    It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

    This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation

excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING

> OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> *one line to give the library's name and a brief idea of what it does.*
> Copyright (C) *year name of author*
>
> This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.
>
> You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.
>
> *signature of Ty Coon*, 1 April 1990
> Ty Coon, President of Vice

That's all there is to it!

Bibliografia

[1] Almesberger, Werner. *LILO: Generic Boot Loader for Linux.* Disponibile in forma elettronica: `tsx-11.mit.edu`. 3 Luglio 1993.

[2] Bach, Maurice J. *The Design of the UNIX Operating System.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc. 1986.

[3] Lamport, Leslie. *LaTeX: A Document Preparation System.* Reading, Massachusetts: Addison-Wesley Publishing Company. 1986.

[4] Stallman, Richard M. *GNU Emacs Manual*, ottava edizione. Cambridge, Massachusetts: Free Software Foundation. 1993.

Indice analitico