ABILITÀ INFORMATICHE PER CHIMICI
Introduzione a UNIX
(2006/2007)[1]

Dr. Giorgio F. Signorini
Università di Firenze
Dipartimento di Chimica
signo@chim.unifi.it
http://www.chim.unifi.it/~signo

14 novembre 2011

[1]This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 2.5 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/2.5/it/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Indice

1 Introduzione **5**
1.1 generalità e obiettivi . . . 5
1.2 caratteristiche del software . . . 5
1.3 caratteristiche di UNIX . . . 6
1.4 vantaggi e svantaggi di UNIX su Windows . . . 6
1.5 Bibliografia su UNIX . . . 7

2 Approfondimenti: Software libero e *open source* **8**
2.1 diritti di proprietà e di uso . . . 8
2.1.1 gratuità . . . 8
2.2 definizione di software libero . . . 8
2.2.1 Particolari aspetti del software libero: . . . 8
2.3 altre forme di software non-proprietario . . . 9
2.4 pro e contro il software libero . . . 9

3 Approfondimenti: Sistemi Operativi **10**
3.1 Cos'è un Sistema Operativo . . . 10
3.1.1 S. O. : definizioni ed esempi . . . 10
3.2 Architettura a livelli del software . . . 11
3.3 Interfacce utente . . . 14
3.3.1 vantaggi dell'interfaccia testuale . . . 15
3.4 Principali caratteristiche di un Sistema Operativo . . . 15
3.4.1 numero di processi contemporanei . . . 15
3.4.2 numero di utenti . . . 15
3.5 Organizzazione dei dati su disco: file e directory (cartelle) . . . 15
3.6 UNIX come famiglia di sistemi operativi, liberi e non . . . 17
3.6.1 breve storia di UNIX . . . 18
3.7 GNU . . . 18
3.8 GNU/Linux . . . 19

4 Uso elementare di UNIX **20**
4.1 premessa: maiuscole e minuscole . . . 20
4.2 sessione di lavoro . . . 20
4.2.1 `login`, `logout` e `exit` su un terminale . . . 20
4.2.2 Sessione grafica con apertura di un terminale . . . 20
4.2.3 sintassi di un comando . . . 21
4.2.4 il comando `passwd` . . . 21
4.2.5 utenti e gruppi; `who` e `finger` . . . 22
4.3 guide . . . 22
4.4 navigazione nel *filesystem* e gestione elementare di file e directory . . . 23
4.4.1 file e directory; directory corrente e directory superiore; `ls`, `pwd` e `cd` . . . 23
4.4.2 percorso assoluto e relativo . . . 24
4.4.3 le directory ".", ".." e "~" ; . . . 25
4.4.4 permessi di lettura, scrittura ed esecuzione dei file; `ls -l` . . . 26
4.4.5 nomi di file e directory; file e directory nascosti; `ls -a` . . . 27
4.4.6 lista di una directory diversa dalla directory corrente: `ls` *`dir`* . . . 28
4.4.7 comandi per gestire file: `cp, mv, rm` . . . 28
4.4.8 uso di `cp` e `mv` per copiare e rinominare directory . . . 29
4.4.9 comandi speciali per gestire directory: `mkdir` e `rmdir` . . . 29
4.4.10 vedere il contenuto di un file: `cat, more, less` . . . 29
4.5 editor di testo: `vi`, `nano (pico)` e `emacs` . . . 31
4.5.1 differenza tra *word processor* e *editor di testo* . . . 31
4.5.2 `vi` . . . 31
4.5.3 `nano (pico)` . . . 32

4.6 stampa . . . 32
4.6.1 code di stampa . . . 32
4.6.2 stampa da un'applicazione . . . 33
4.6.3 stampa da terminale . . . 33
4.6.4 stampa di file codificati: PostScript, filtri e driver, convertitori . . . 33
4.6.5 stampa da applicazioni . . . 33
4.6.6 schema riassuntivo della procedura di stampa . . . 33
4.7 proprietà delle shell . . . 35
4.7.1 sostituzioni . . . 35
4.7.1.1 nomi di file . . . 35
4.7.1.2 altre sostituzioni . . . 36
4.7.1.3 proteggere da sostituzione . . . 36
4.7.2 reindirizzamento di input e output, pipeline . . . 36
4.7.3 esecuzione di comandi in serie, parallelo e *background* . . . 37
4.7.3.1 esecuzione in serie . . . 37
4.7.3.2 esecuzione in parallelo: . . . 37
4.7.3.3 esecuzione in *background* . . . 38
4.7.4 processi e loro interruzione . . . 38
4.7.4.1 processi e job . . . 38
4.7.4.2 interruzione di processi e job . . . 38
4.7.5 esempi . . . 39
4.8 altri comandi utili . . . 41

5 Uso "avanzato" di UNIX **42**
5.1 `emacs` . . . 42
5.1.1 aree dello schermo: . . . 42
5.1.2 menu, scorciatoie e comandi "complessi" . . . 42
5.1.3 help: Apropos, Key, Whereis (locate) . . . 42
5.1.4 Interruzione: `C-g` . . . 43
5.1.5 Editare un file . . . 43
5.1.5.1 Aggiornare il contenuto di un buffer . . . 43
5.1.6 "Editare" una directory (`dired`) . . . 43
5.2 Regular Expression (espressioni regolari o formali) . . . 43
5.3 awk . . . 44
5.3.1 caratteristiche generali . . . 44
5.3.2 manuali . . . 44
5.3.3 sintassi del comando awk . . . 45
5.3.4 istruzioni di awk . . . 45
5.3.5 Campi . . . 45
5.3.6 Variabili . . . 45
5.3.7 Sequenze (*pattern*) . . . 45
5.3.8 azioni . . . 46
5.3.9 un semplice esempio . . . 46

6 APPLICAZIONI SU UNIX **47**
6.1 `gnuplot` . . . 47
6.1.1 uso . . . 47
6.1.2 diagrammare un file di dati esistente . . . 47
6.1.3 diagrammare una funzione . . . 47
6.1.4 Esempio: il pH di una soluzione per aggiunta di base forte. . . . 47
6.1.5 ottimizzazione (fit) di una funzione per la riproduzione di un insieme di dati . . . 47
6.1.6 registrare una sessione `gnuplot` . . . 48
6.1.7 registrare e stampare un grafico . . . 48
6.2 grafica molecolare: jmol . . . 49
6.3 esercizi . . . 49

7 Note **50**
7.1 nota: la tastiera . . . 50

Elenco delle tabelle

4.1 **Sessione, sintassi, utenti, guide** 30
4.2 **Gestione essenziale di file e directory.** 30
4.3 **Comandi di stampa** 34
4.4 **Proprietà della shell** 40

Elenco delle figure

3.1 hardware e software 12
3.2 sistema operativo e applicazioni 12
3.3 kernel, comandi, interfaccia utente 13
3.4 interfacce utente 13
3.5 varie componenti del sistema operativo 14
3.6 Esempio di struttura a cartelle in MS Windows. 16
3.7 Esempio di struttura a cartelle in UNIX/KDE 17

4.1 esempio di percorsi 24

6.1 la finestra di Jmol-8 49

Capitolo 1

Introduzione

1.1 generalità e obiettivi

UNIX è un sistema operativo, cioè un ambiente di lavoro su computer, che su un PC svolge un ruolo paragonabile a Windows della Microsoft.

È molto utilizzato in campo tecnico-scientifico, e particolarmente adatto per un uso esperto; ad esempio, per macchine che devono fare da *server* in WWW (vedi new.netcraft.com `http://news.netcraft.com/archives/web_server_survey.html`).

A differenza di Windows, UNIX non è un prodotto commerciale. Lo si può considerare come uno standard, o in senso lato un linguaggio. Esitono molti "dialetti" UNIX: il più diffuso è GNU/Linux, che funziona sui PC.

Obiettivo di questo modulo è mettere in grado gli studenti di lavorare con UNIX in modo elementare: operare su un terminale ASCII dando semplici comandi per gestire file, creare ed editare file, elaborare file di dati, ed usare semplici applicazioni; e confrontare il modo di operare su terminale con l'uso di un'interfaccia grafica con *desktop*.

Non ci si propone di insegnare l'installazione e la gestione di un sistema UNIX.

1.2 caratteristiche del software

Per illustrare le caratteristiche che rendono il software un prodotto diverso da altri, partiamo dal confronto tra due oggetti di uso comune:

- il personal computer (PC)
- il televisore

Apparentemente, hanno molto in comune. Tra le altre cose, entrambi sono costituiti da aspetti materiali (la meccanica e l'elettronica o *hardware*) e da una "logica" (*software*), cioè uno schema di funzionamento impartito all'hardware che permette all'utente di dare istruzioni.

Il PC ha però una caratteristica fondamentale: hardware (h/w) e software (s/w) non sono collegati in modo inscindibile. È possibile sostituire il s/w mantenendo lo stesso h/w, o trasferire il s/w su un h/w diverso; cosa che in genere non è possibile fare su un televisore.

Questa *modularità* di h/w e s/w nel PC ha alcuni vantaggi

- permette la specializzazione dei produttori in un campo particolare, e la formazione e la collaborazione di *competenze diverse*
- favorisce la concorrenza, lasciando all'utilizzatore la *scelta* della combinazione di prodotti che fa per lui

Con l'evolversi di un prodotto, anche la sua modularità tende ad estendersi. Come nell'h/w ci si specializza nei vari componenti (schede, periferiche, etc), anche il s/w si è via via specializzato attraverso la divisione in elementi o livelli che hanno funzioni diverse:

- Sistema Operativo
- Applicazioni

Il Sistema Operativo è il s/w base, che permette di fare un po' tutto e sul quale si possono innestare programmi a funzione specifica, o applicazioni. A sua volta i S.O. si è via via articolato in vari componenti: kernel, comandi, interfacce utente, ...

Perché l'organizzazione in moduli indipendenti dia i migliori frutti è necessario che il punto di contatto o *interfaccia* tra i vari livelli abbia caratteristiche note e pubbliche: sia cioè uno *standard*. La possibilità di utilizzare diverse apparecchiature nell'impianto elettrico di casa è legata all'esistenza di prese di corrente sempre uguali.

Si è visto che un secondo elemento che risulta utile è la possibilità di *conoscere il funzionamento* del s/w, ad esempio del Sistema Operativo. In questo modo si possono produrre applicazioni che non solo possono funzionare (grazie all'intefaccia standard) ma anche sfruttano al meglio le caratteristiche del S.O. È noto che sul sistema Windows della Microsoft le applicazioni Microsoft funzionano meglio di quelle concorrenti, e ciò grazie ad una proficua collaborazione tra gli sviluppatori

delle applicazioni e quelli del S.O. Questa possibilità è aperta a tutti quando il funzionamento è noto, o, come si dice, il codice origine è in chiaro (*open source*)

Il massimo dell'efficienza lo si raggiunge quando l'utente o sviluppatore hanno del s/w una *disponibilità piena*, cioè quando il s/w è "*libero*". Con questo non si intende *gratuito*. Si intende che una volta acquistato se ne acquisiscono pieni diritti. Illustriamo questo concetto con un confronto tra i diritti che si hanno quando si compra un'automobile e quando si compra un software "*proprietario*" come Microsoft Windows:

l'acquirente ha diritto di	automobile	Windows
usarlo	SI	SI
studiarne il funzionamento	SI	NO
modificarlo	SI	NO
cederlo, anche modificato	SI	NO

L'esperienza sembra dimostrare che rendere il s/w libero e a codice in chiaro è remunerativo anche dal punto di vista del produttore: ad esempio, di alcuni prodotti gira una versione libera che permette al produttore della versione commerciale di mettere a punto un prodotto migliore:

- Mozilla per Nescape
- OpenOffice.org per Staroffice

In ambiti scolastici si è notato che l'uso di s/w libero permette di:

- risparmiare, specialmente sulla gestione
- eliminare la dipendenza da un unico fornitore di s/w
- sfruttare le potenzialità didattiche

1.3 caratteristiche di UNIX

Concentriamoci adesso su quella componente fondamenteale del s/w che è il Sistema Operativo. Col nome di UNIX si indica un'intera famiglia di sistemi operativi, liberi (nel senso indicato sopra) e non.

Nel ramo degli UNIX liberi di gran lunga il più popolare è GNU/Linux. Notare che il termine "Linux" indica soltanto la parte più interna del S.O., il kernel, mentre il resto, cioè quello che è direttamente usato dall'utente, è GNU, sviluppato da una comunità di utenti sotto la Free Software Foundation.

Un sistema GNU/Linux è fatto di numerose componenti, confezionate insieme in una "distribuzione". Acquistando (o scaricando gratuitamente) una distribuzione si ottiene tutto il sistema operativo con i programmi per installarlo.

1.4 vantaggi e svantaggi di UNIX su Windows

Abbiamo visto quali vantaggi comporti l'uso di un sistema operativo libero quale GNU/Linux. Consideriamone altri, e anche quali possono essere gli svantaggi nei confronti di un sistema "proprietario" come Microsoft Windows (cfr [Attivissimo(2000), capitolo 3])

- vantaggi

  versatilità: funziona su diversi hardware, non solo su PC

  modulabilità: kernel, shell, X-windows e Desktop sono elementi indipendenti e possono essere combinati a piacimento.

  potenza: permette di fare più cose (adatto a fare da "server")

  programmi: c'è abbondanza di software gratis

  stabilità, sicurezza, trasparenza: si blocca meno; maggiore resistenza ai virus; migliore protezione (permessi di scrittura ed esecuzione); controllo totale (codice sorgente "aperto")

  prezzo: molte versioni di UNIX sono "libere"; GNU/Linux si può copiare *legalmente*.

- svantaggi

  sforzo iniziale: ci vuole un po' di tempo per impararlo

  programmi: non sempre sono allo stesso livello degli equivalenti su MS-Windows

  aggiornamento hardware: è difficile far funzionare un componente hardware di recente produzione (in compenso, è facile far funzionare UNIX su un hardware datato)

1.5 Bibliografia su UNIX

Su UNIX, un libro classico di riferimento è [Kernighan and Pike(1989)]. In rete si trovano innumerevoli libretti e corsi, tra i quali mi paiono utili: [BIU(), 4LW(), UHU()].

Chi segue questo corso, avrà a che fare essenzialmente con GNU/Linux. Un ottimo manuale introduttivo è [Ricar(1999)], ora forse un po' superato; potete cercare in libreria analoghi manualetti di 100-200 pagine, come [Georges Piriou(2004)], preferendo quelli che si occupano più dell'uso che dell'installazione e gestione del sistema. Una guida completa è [Petersen(1998)]. [Attivissimo(2000)] è una guida pratica *on-line* alla migrazione da Windows a GNU/Linux.

Un libro *on-line* particolarmente esauriente, dedicato oltre che a UNIX a tutto il software libero è [Giacomini()]; esiste anche una versione ridotta, a stampa: [Giacomini(2001)].

Capitolo 2

Approfondimenti: Software libero e *open source*

2.1 diritti di proprietà e di uso

La distinzione tra software "libero" e "proprietario" è legata ai diritti di uso e di proprietà:

s/w proprietario: chi lo produce ne mantiene tutti i *diritti di proprietà*, e ne concede all'utente finale solo *l'uso* sotto alcune condizioni (licenza) e generalmente dietro pagamento.

s/w libero: il produttore concede al destinatario (eventualmente dietro pagamento) *la proprietà* di fatto del software, compreso quindi il diritto di copiarlo, modificarlo, combinarlo con altri prodotti, distribuirlo, ed anche rivenderlo.

Si può vedere questa differenza come la differenza tra l'affitto e l'acquisto di un bene.

2.1.1 gratuità

Notare che secondo la definizione appena data la differenza tra software libero e proprietario non è tanto nel pagamento, quanto nei diritti concessi al destinatario.

- perché un software sia libero *non è affatto necessario* che esso sia *gratuito*, basta che se ne possa disporre *liberamente*
- il software libero *non* comporta la rinuncia ai diritti dell'autore, il quale può pretendere di essere citato e può imporre alcune restrizioni sulla modificabilità.

2.2 definizione di software libero

Per poter disporre completamente di un software è essenziale conoscerne il funzionamento, senza lati nascosti; bisogna quindi che il testo del programma, o codice originario, sia leggibile (*open source*). In questo senso i due termini "libero" e "open source" sono spesso usati come sinonimi. La trasparenza del codice, tuttavia, è un requisito necessario ma non sufficiente per poter definire "libero" un software.

Secondo la definizione più accreditata, quella della Free Software Foundation[Fre(a), Fre(b)], è "libero" il software che permette a chiunque di:

1. usarlo;
2. studiare come funziona (accesso al codice);
3. ridistribuirlo;
4. modificarlo e distribuirlo modificato

Per quanto questa definizione possa sembrare "rivoluzionaria" a molti difensori del software proprietario, essa non differisce molto da quello che qualunque industria di automobili (o di lavatrici, o di telefonini) è abituata a concedere agli acquirenti dei propri prodotti!

Per una discussione di questo argomento, vedere il testo di D. Giacomini [Giacomini()] ed anche il documento della Commissione Governativa ([Commissione Governativa sull'Open Source(2003)]), che contiene anche utili definizioni di termini.

2.2.1 Particolari aspetti del software libero:

copyleft ("permesso d'autore"): licenza che garantisce diritti di software libero, ma impone di *mantenerlo* libero (controesempio: X11, è distribuito con licenza libera, che però non impedisce restrizioni- alcune versioni sono a pagamento e sono le uniche a funzionare su un certo hardware)

GNU Public License (GPL): particolare licenza con *copyleft* che permette anche l'uso commerciale del software

software semi-libero: software libero, ma senza permesso di trarne profitto

2.3 altre forme di software non-proprietario

Ci sono software che pur essendo gratuiti non sono liberi nel senso definito sopra. Tra gli altri ricordiamo:

software di dominio pubblico: è software senza copyright né licenza; non è di nessuno e quindi qualcuno può "appropriarsene" trasformandolo in software non libero

freeware: software gratuito, spesso senza sorgente

shareware: software gratuito per un tempo limitato

2.4 pro e contro il software libero

Anche se non è necessariamente gratuito, il software libero si può copiare legalmente, e quindi di fatto è meno remunerativo per gli autori.

A questo propostito esiste un dibattito tra due visioni diverse:

- alcuni pensano: siccome gli autori non possono guadagnare in proporzione a tutti quelli che lo usano, il software libero è di bassa qualità
- secondo altri: se gli strumenti intellettuali sono distribuiti liberamente (es. ricerca scientifica, definizioni di standard, etc.) il loro sviluppo è più rapido ed efficiente; a differenza della proprietà materiale, la "proprietà intellettuale" può essere remunerativa anche se non è esclusiva (cfr. ad esempio [Boldrin and Levine(2002), Bessen(2003)]).

La differenza può essere vista così: i primi considerano il software un *prodotto finale*, mentre i secondi lo considerano uno *strumento*.

È comunque un dato di fatto che:

- i prodotti di maggior successo negli ultimi anni (TCP/IP, WWW, GNU/Linux) sono privi di copyright
- la rinuncia alla segretezza dei codici sorgente è una strategia sempre più adottata per lo sviluppo del software (Mozilla, OpenOffice.Org)

Notare inoltre che in Italia anche la pubblica amministrazione raccomanda l'uso dell'*Open Source* e dei formati aperti [Ministero dell'Innovazione(2002)]

Secondo alcuni sostenitori [Schoolforge-UK(2006)], l'uso del s/w libero nelle scuole consente:

- risparmio sui costi totali
- eliminare dipendenza da un unico fornitore di sw
- maggiore funzione didattica

Capitolo 3

Approfondimenti: Sistemi Operativi

3.1 Cos'è un Sistema Operativo

Il Sistema Operativo è il software *essenziale* di un calcolatore, cioè quel software senza cui il calcolatore non può essere usato. È un insieme di funzioni di base che permettono di gestire tutte le risorse (hardware) del calcolatore:

- dispositivi principali di ingresso/uscita (tastiera, mouse, schermo)
- unità centrale (CPU)
- memoria
- (esecuzione programmi)
- dischi
- altre periferiche
- ...

Questo insieme di funzioni di base fornisce un'interfaccia verso l'hardware

- comune a tutti i programmi applicativi (ad es: il sistema di stampa)
- direttamente utilizzabile dagli utenti (ad es: la gestione di file e directory)

3.1.1 S. O. : definizioni ed esempi

Riportiamo due definizioni di Sistema Operativo.

[da Wikipedia -[Wikipedia(), voce Sistema Operativo]]
In informatica, un sistema operativo (SO) é il software di sistema responsabile del diretto controllo e gestione dell'hardware che costituisce un computer e delle operazioni di base.
Informalmente, il termine é spesso usato per indicare il software che viene fornito con il computer prima che sia stata installata qualunque applicazione.
Il sistema operativo provvede che le altre applicazioni possano usare la memoria, i dispositivi di input/output e l'accesso al file system. Se molteplici applicazioni sono in esecuzione, il sistema operativo le gestisce in modo che tutte abbiano un tempo sufficiente di accesso al processore.
Esempi di Sistemi Operativi

- Unix
- Linux
- Windows
- MacOS
- Solaris

Classificazione e Terminologia
Un sistema operativo é contettualmente diviso in due componenti, la shell e il kernel. Come il nome implica, la Shell (guscio, conchiglia) è un involucro esterno per il kernel che a sua volta dialoga direttamente con l'hardware.
Hardware <-> Kernel <-> Shell <-> Applicazioni
In alcuni sistemi operativi la shell e il kernel sono entità completamente separate, permettendo di avere diverse combinazioni di shell e kernel (es. Unix), in altri la loro separazione è solo concettuale (es. Windows).
Le ideologie di progettazione del kernel includono il kernel monolitico, il microkernel e l'exokernel. Tra i sistemi commerciali, come Unix e Windows, l'approccio monolitico è predominante, con alcune eccezioni (es. QNX). Il microkernel è più popolare nei sistemi operativi di ricerca. Molti sistemi operativi di tipo embedded usano exokernel ad-hoc.

[Free OnLine Dic Of Computing [FOLDOC(), voce Operating System]]:

> **operating system:**
> The low-level software which handles the interface to peripheral hardware, schedules tasks, allocates storage, and presents a default interface to the user when no application program is running.
>
> The OS may be split into a kernel which is always present and various system programs which use facilities provided by the kernel to perform higher-level house-keeping tasks, often acting as servers in a client-server relationship.
>
> Some would include a graphical user interface and window system as part of the OS, others would not. The operating system loader, BIOS, or other firmware required at boot time or when installing the operating system would generally not be considered part of the operating system, though this distinction is unclear in the case of a rommable operating system such as RISC OS.
>
> The facilities an operating system provides and its general design philosophy exert an extremely strong influence on programming style and on the technical cultures that grow up around the machines on which it runs.
>
> Example operating systems include 386BSD, AIX, AOS, Amoeba, Angel, Artemis microkernel, BeOS,
>
> Brazil, COS, CP/M, CTSS, Chorus, DACNOS, DOSEXEC 2, GCOS, GEORGE 3, GEOS, ITS, KAOS, Linux,
>
> LynxOS, MPV, MS-DOS, MVS, Mach, Macintosh operating system, MINIX, Multics, Multipop-68, Novell NetWare,
>
> OS-9, OS/2, Pick, Plan 9, QNX, RISC OS, STING, System V, System/360, TOPS-10, TOPS-20, TRUSIX, TWENEX,
>
> TYMCOM-X, Thoth, Unix, VM/CMS, VMS, VRTX, VSTa, VxWorks, WAITS, Windows 3.1, Windows 95,
>
> Windows 98, Windows NT.

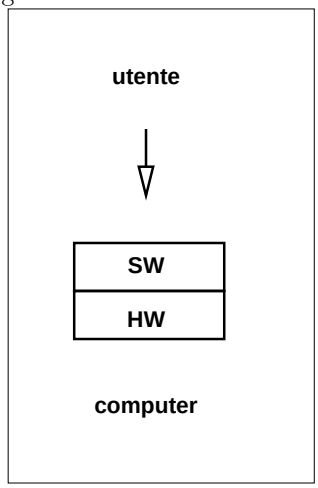
3.2 Architettura a livelli del software

Il ruolo e la composizione del S.O. sono meglio compresi in una rappresentazione grafica a livelli. Nel grafico, in alto sta l'utente, in basso il computer. Ogni livello scambia dati solo con quello superiore e quello inferiore.

1. Tutti conoscono la distinzione tra *hardware* (elettronica) e *software* (logica). Il punto fondamentale è che in un computer

(ad es. un PC), a differenza di altre apparecchiature (come un telefonino o un elettrodomestico) *hardware* e *software* sono separati: su uno stesso *hardware* (HW) si possono installare differenti *software* (SW). Questo è molto utile.

Figura 3.1: hardware e software

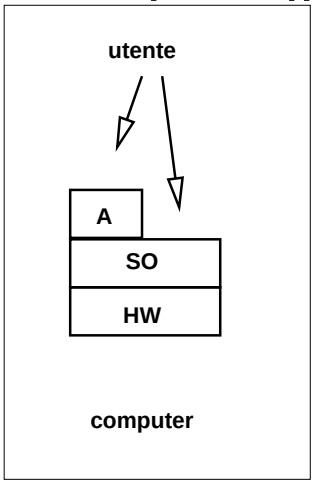


2. A sua volta, il SW è diviso in due parti: il Sistema Operativo (SO) e le Applicazioni (A) o programmi. Il SO è essenziale e serve a comandare tutto il computer, ma *sopra* il SO si possono installare tutte le A che si vuole. In quest'approssimazione, l'utente interagisce: o con il SO direttamente, o con un'A.

   Notare bene la differenza tra S.O. e programmi applicativi. Ad es: OpenOffice.org è un programma applicativo che funziona sia sul S.O. Windows che sul S.O. GNU/Linux.

   Altro esempio: si può usare un calcolatore senza MS-Office, ma non senza MS-Windows (se questo è il S.O. installato)

Figura 3.2: sistema operativo e applicazioni

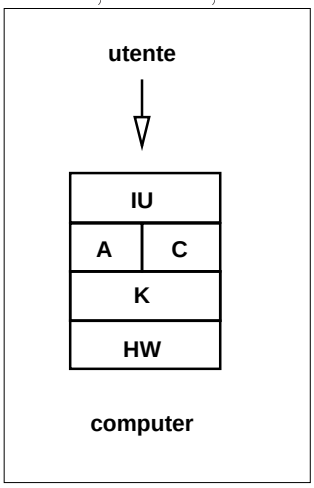


3. Una rappresentazione migliore distingue ulteriormente le componenti del SO:

   - il Kernel (K) o nucleo (comandi fondamentali a cui si può rivolgere direttamente un program ma, ma di solito non l'utente)
   - i comandi (C) disponibili all'utente (ad esempio per cancellare un file)
   - un'interfaccia utente (IU) che riceve istruzioni dalla tastiera o dal mouse e le passa ai C o alle A (che a loro volta dialogano con il K, che a sua volta dialoga con l'HW)

   Notare che le applicazioni A si inseriscono "tra" kernel e interfaccia utente, che quindi devono essere in grado di dialogare con tutte le A in modo unificato, l'una (K) dal lato hardware, l'altra (IU) dal lato utente.

Figura 3.3: kernel, comandi, interfaccia utente

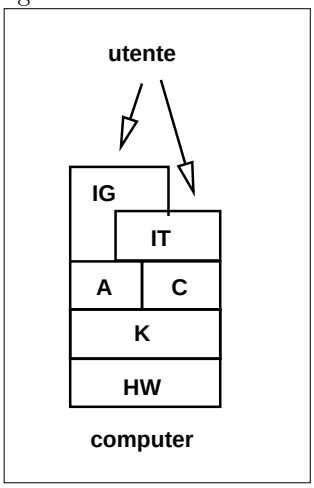


4. Infine, l'IU è di due tipi:

   - Interfaccia Grafica (IG), basata su finestre (Windowing System)
   - Interfaccia Testuale (IT) in cui l'input e l'output sono sequenze di caratteri

   L'utente si può rivolgere o all'IG o all'IT; ma può rivolgersi all'IT anche attraverso l'IG: cioè può aprire entro quest'ultima una finestra con un'interfaccia testuale o terminale

Figura 3.4: interfacce utente

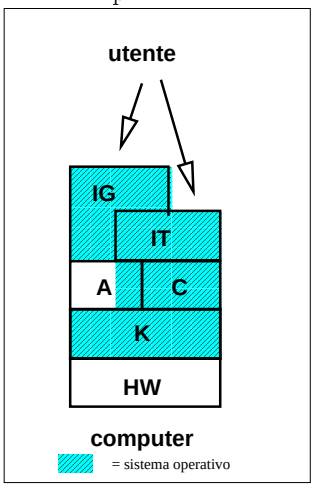


L'importanza della struttura a livelli sta nell'indipendenza reciproca dei vari livelli. In alcuni sistemi operativi, come Windows, K+C+IG+IT appaiono come un unico blocco; invece in altri, come UNIX, ciascuna di queste componenti può essere sostituita lasciando invariate le altre. Per esempio su uno stesso K possono essere installati diversi sistemi di C; un utente può scegliere tra diversi tipi di IT (in particolare, di shell: vedi sotto); può usare alternativamente diverse interfacce grafiche (in particolare, diversi Desktop Manager: vedi sotto).

Tutte queste componenti possono essere realizzate indipendentemente, da soggetti diversi.

Anche se ciò che lo caratterizza è il kernel, spesso si usa il termine "sistema operativo" per riferirirsi ad un insieme di kernel, comandi, applicazioni di sistema e interfacce utente:

Figura 3.5: varie componenti del sistema operativo



In questo schema, l'unica parte di software che *non* fa parte del sistema operativo sono le applicazioni non di sistema. Anche se queste occupano nella figura uno spazio piccolo (la parte bianca del blocco A), possono essere una parte rilevante (p.es. in termini di spazio disco) del s/w installato sul calcolatore, e comunque dell'uso che l'utente fa del calcolatore.

3.3 Interfacce utente

L'interfaccia utente è il mezzo attraverso cui l'utente interagisce con il computer (ad es., per spostare un file da una directory all'altra o lanciare un programma). Ce ne sono due tipi:

interfaccia testuale (terminale): ad ogni azione corrisponde una frase (comando) che l'utente scrive sulla tastiera e che compare in un'area dello schermo simile al foglio di una macchina da scrivere (terminale); eventuali risposte del computer sotto forma di testo sono stampate sullo stesso terminale.

interfaccia grafica (GUI): per compiere operazioni sul sistema si opera, principalmente con il mouse, su elementi grafici come menù, icone, finestre. Quasi sempre l'intefaccia grafica contiene, tra i vari strumenti, anche un terminale. GUI sta per Graphical User Interface (Interfaccia Utente Grafica)

Ad esempio:

- nei sistemi MicroSoft
  - l'interfaccia grafica è Windows
  - quella testuale è il DOS. Quest'ultimo si può lanciare alla partenza in alternativa a Windows, oppure da dentro Windows come terminale ("Prompt DOS" o "Prompt dei comandi")
- In UNIX si hanno
  - interfaccia testuale (sempre):
    - la *riga di comando* è interpretata da una *shell*: piccolo ambiente di programmazione (variabili, controllo di flusso etc.), che prepara il comando da passare al S.O. vero e proprio
    - in uno stesso UNIX si possono usare diverse shell con caratteristiche leggermente diverse tra loro
  - interfaccia grafica (*non fa parte di UNIX*, ma c'è quasi sempre). È composta di:
    - X-windows (programma di sistema per gestire la grafica pura a finestre, basato su un meccanismo client-server)
    - uno "Window Manager": un programma dell'utente che gestisce gli accessori delle finestre (cornici, bottoni, menu) e in generale tutto lo schermo, con un menù base per l'avvio delle principali applicazioni grafiche. Es: mwm, fvwm, WindowMaker.
    - una forma più evoluta dello W.M. è la Scrivania (Desktop) che contiene un pacchetto di applicazioni grafiche per eseguire praticamente tutte le operazioni di sistema (file manager, pannello di controllo, etc.), senza aprire un terminale, in modo simile a MS-Windows. Contiene comunque un terminale, un menù di avvio, una barra di applicazioni ed icone, etc. Es. GNOME, KDE, ...

Le varie interfacce grafiche sono simili tra loro, e in sostanza simili a Windows o al sistema MacIntosh, e come tali abbastanza intuitive; ciò che caratterizza UNIX rispetto agli altri S.O. è invece l'interfaccia testuale, assai potente. Noi lavoreremo principalmente con quest'ultima (il *terminale*), all' interno di quella grafica.

3.3.1 vantaggi dell'interfaccia testuale

È evidente che l'interfaccia grafica è molto più intuitiva e pratica per eseguire i compiti più semplici. Ma quella testuale ha dei vantaggi.

Si può dire: l'interfaccia testuale sta a quella grafica come il linguaggio parlato (o quello matematico) sta a quello figurato o alla mimica.

- Con i caratteri presenti sulla tastiera si può scrivere un numero praticamente infinito di comandi; con il sistema "a menù" dell'interfaccia grafica diventa difficile gestirne più di qualche decina
- Un comando sotto forma di frase può avere una sintassi complessa (azione, oggetto/i dell'azione, opzioni, ...) pur essendo facile da scrivere; per raggiungere la stessa complessità un'interfaccia grafica deve usare strumenti più scomodi e più difficilmente adattabili, come menù e finestre di dialogo.

  Ad es in Windows:

  - *copia fileA da directoryB a directoryC*, essendo una funzione già programmata nel S.O., è facile da eseguire; ma
  - *copia il contenuto di tanti file A, B, C, ... in un unico file Z usando il formato x*, non essendo pre-inserito nel S.O., richiede l'apertura di un'applicazione specifica (se c'è) e la selezione di azioni, oggetti e opzioni con il mouse

- combinando più comandi con degli opportuni nessi si possono formare comandi molto elaborati, così come nel linguaggio parlato varie frasi si combinano in un periodo (o come si possono legare con operatori espressioni matematiche semplici per formarne di più complesse) .

  Ad es:

  ```
  grep Error /var/log/httpd/access.log | wc -l
  ```

  esamina il file di statistiche del server HTTP (`/var/log/httpd/access.log`) selezionando le righe che contengono la parola "`Error`" (primo comando, fino alla "|") e poi le conta (secondo comando).

  Al confronto, è difficile legare tra loro operazioni compiute graficamente

- si può richiamare facilmente un comando già dato per eseguirlo di nuovo (o per verificare ciò che si è fatto)
- si possono riunire più comandi in un programma (o *script*)
- l'interfaccia testuale richiede poche risorse, ed è quindi adatta per collegamenti lenti

3.4 Principali caratteristiche di un Sistema Operativo

3.4.1 numero di processi contemporanei

I sistemi operativi più rudimentali (es. MS-DOS) possono eseguire un solo processo alla volta.

Gli elaboratori centrali (*mainframe*) hanno invece sempre avuto la necessità di fare funzionare più processi contemporaneamente (ad. esempio, un programma per gestire la posta elettronica e molti accessi contemporanei ad un database).

I sistemi operativi che si trovano sui PC attuali sono tutti, in misura maggiore o minore, multi-processo (*multi-tasking*): per esempio, si può correggere un documento con un elaboratore di testi mentre si attende la visualizzazione di una pagina WWW.

3.4.2 numero di utenti

Si possono definire, sullo stesso sistema, più profili di accesso o utenti. Ciascun utente ha un suo identificativo (*account*) e quasi sempre una *password*.

Se il sistema è multi-processo e ci sono più terminali, gli utenti possono lavorare contemporaneamente.

Ogni utente può avere un suo ambiente di lavoro e delle sue prerogative. Tipicamente si distingue tra utenti di basso livello e utenti di alto livello, dove solo questi ultimi possono eseguire funzioni di gestione del sistema (es: aggiornare il software, aggiungere periferiche, etc).

3.5 Organizzazione dei dati su disco: file e directory (cartelle)

I dati su supporto fisso (disco fisso, dischetti, CD, etc.) sono organizzati in *file* e *directory o cartelle (folder)*.

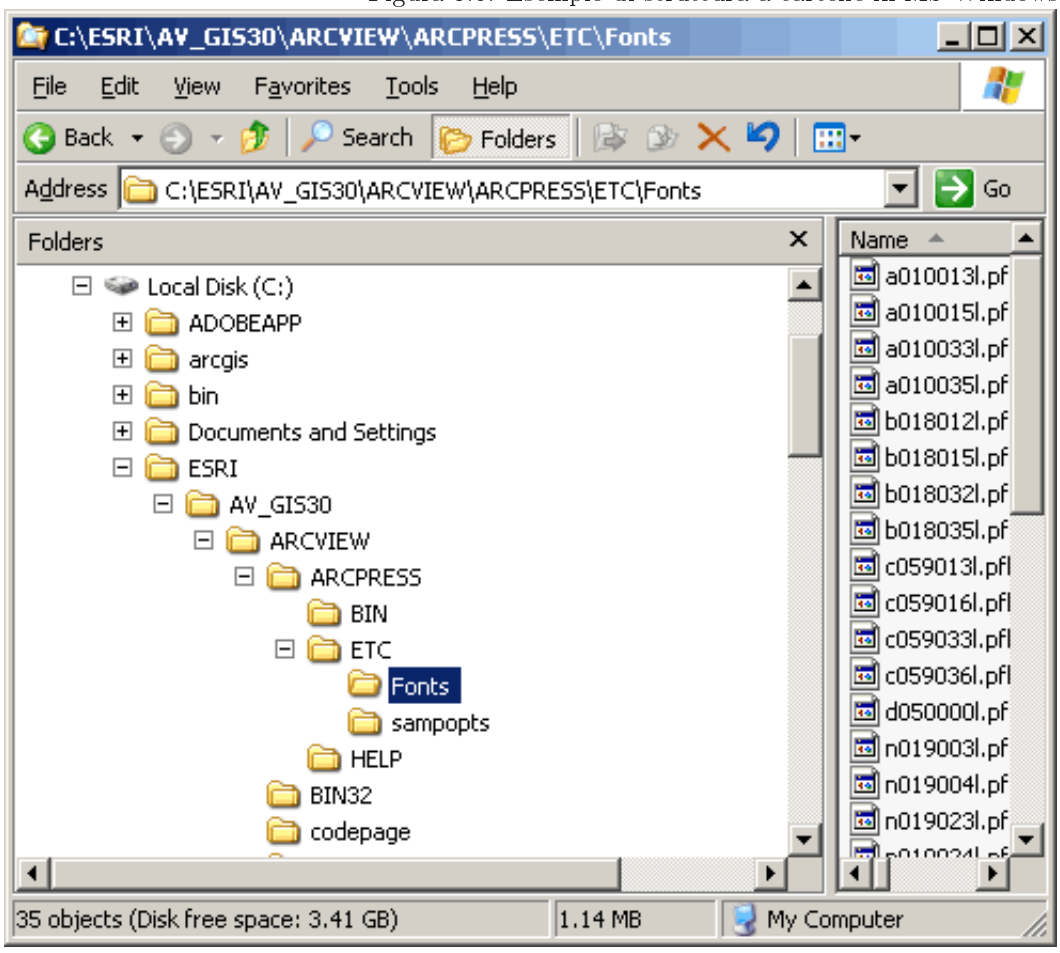
Un *file* è un insieme di dati visti dal sistema operativo come un'unità a sè stante. Ad esempio, un file può essere un documento di testo, un insieme omogeneo di dati, un programma. Il S.O. è in grado di creare e cancellare un file o metterlo a disposizione di un'applicazione. Un file in generale ha

- un nome
- una data

- altri attributi

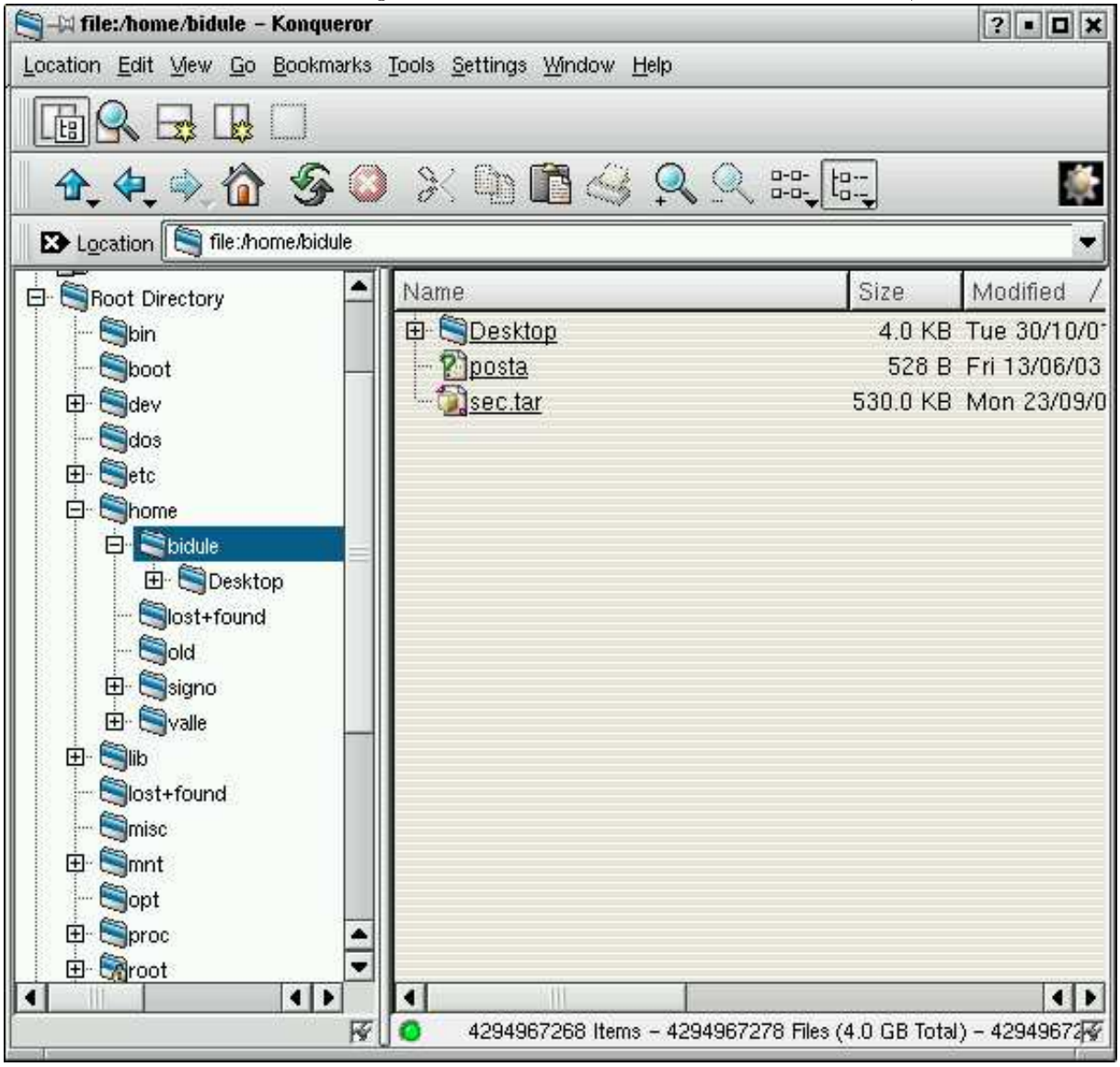
Una *directory o cartella (folder)* è un contenitore di file. In genere, oltre ai file una directory può contenere anche *sotto-directory;* in questo modo i dati sono organizzati in un sistema gerarchico di directory e sottodirectory.

Figura 3.6: Esempio di struttura a cartelle in MS Windows.



Il disco locale contiene `ESRI\AV_GIS30\ARCVIEW\ARCPRESS\ETC\Fonts`. La cartella `Fonts` contiene i file nella finestra di destra: `a010013l...`

Figura 3.7: Esempio di struttura a cartelle in UNIX/KDE



La directory `/home/bidule` contiene i file `posta` e `sec.tar` e la sottodirectory `Desktop`

le stesse informazioni in UNIX si possono ottenere con un semplice comando da terminale:

```
# ls -l /home/bidule
total 544
drwxr-xr-x    3 bidule   bidule       4096 Oct 30  2001 Desktop
-rw-------    1 bidule   bidule        528 Jun 13 15:30 posta
-rw-rw-r--    1 bidule   bidule     542720 Sep 23  2002 sec.tar
#
```

3.6 UNIX come famiglia di sistemi operativi, liberi e non

Ci sono vari "dialetti" UNIX, catalogabili in due gruppi:

- ***famiglia "System V"***: UNIX commerciali (generalmente contengono almeno una parte dell'originale sorgente AT&T, e la relativa licenza)
- ***famiglia "Berkeley" o "BSD"***: UNIX (più o meno) liberi: FreeBSD, NetBSD, ... ; i kernel con GPL sono: Linux, Hurd.

Note:

1. UNIX è un marchio registrato

   "UNIX® is a registered trademark of The Open Group"

   però di fatto si è trasformato in uno standard; è proprietà di un'organizzazione più interessata agli standard industriali che alla proprietà intellettuale. Ci sono test passati i quali un S.O. può dirsi legalmente UNIX.

2. Esistono standard veri e propri:

   (a) *POSIX* (Portable Operating System Interface): Una collezione di standard IEEE (1003.x) scritti per permettere la portabilità di applicazioni tra diverse varianti di UNIX

   (b) *Single UNIX Specification* (SUS), prodotto da Open Group. È derivato pesantemente da POSIX, ma si è diffuso più di quest'ultimo, principalmente perché i documenti sono più facili da ottenere. (!)

   (c) *SUS Version 3 = IEEE Std 1003.1-2001*: ultima versione, comune, dei due precedenti. Comprende quattro "volumi"

      i. definizioni e concetti generali (XBD)
      ii. funzioni e subtroutine di sistema (es. `gethostbyaddr`) (XSH)
      iii. shell e utility (es. `cp`) (XCU)
      iv. altro (XRAT)

Notare che questi standard (a differenza della Linux Standard Base, vedi avanti) non contengono alcuni elementi che vengono spesso percepiti come essenziali: alcune applicazioni come Emacs, e l'interfaccia grafica (X11 più Desktop)

Ci sono poi standard che si riferiscono ad aspetti particolari del sistema operativo, tra cui:

(a) Filesystem Hierarchy Standard (FHS): sull'organizzazione del filesystem

(b) Executable and Linkable Format (ELF): sul formato dei file eseguibili

3.6.1 breve storia di UNIX

1969 Thompson (Bell Laboratories, NJ, USA)
Principali famiglie di discendenti (anni '80-'90):

SystemV	(AT&T)	commerciale, solida
BSD	(Università Berkeley)	non-commerciale, dinamica

BSD deriva da versioni date alle Università per scopi di studio, compreso il codice sorgente. Dal 1993 c'è versione (4.4BSD Lite) ripulita dal codice proprietario AT&T.

> Da [Giacomini()]:
> I primi utenti di UNIX sono state le università, a cui in particolare questo sistema operativo veniva fornito a costo contenuto, ma senza alcun tipo di supporto tecnico, né alcuna garanzia. Proprio questa assenza di sostegno da parte della casa che lo aveva prodotto, stimolava la cooperazione tra gli utenti competenti, in pratica tra le università.
> Il maggior fermento intorno a UNIX si concentrò presso l'università della California a Berkeley, dove a partire dal 1978 si cominciò a distribuire una variante di questo sistema operativo: BSD (Berkeley Software Distribution).
> Per difendere il software prodotto in questo modo, nacque una licenza d'uso che rimane il progenitore della filosofia del software libero: la licenza BSD.
> Per molto tempo, la variante BSD di UNIX rimase relegata all'ambito universitario o a quello di aziende che avevano acquistato i diritti per utilizzare il codice sorgente dello UNIX originale. Ciò fino a quando si decise di ripulire lo Unix BSD dal codice proprietario.
> Il risultato iniziale fu 386BSD, che venne rilasciato nel 1992 con la versione 0.1. [...] si svilupparono altri progetti indipendenti per ottenere, finalmente, un sistema BSD libero. Il primo di questi fu nominato NetBSD, al quale si aggiunse subito dopo FreeBSD; più tardi, apparve anche OpenBSD.
> [...]
> Allo stato attuale, le tre varianti *BSD sono tutte riconducibili a BSD 4.4-Lite, dove le differenze più importanti riguardano le piattaforme hardware in cui possono essere installate e l'origine della distribuzione. Infatti, il punto di forza della variante OpenBSD, sta nel fatto di essere realizzata in Canada, da dove possono essere distribuiti anche componenti per la comunicazione crittografica.

Negli anni '90 dal punto di vista del linguaggio le due famiglie (BSD e SysV) si sono avvicinate.

3.7 GNU

GNU è un sistema operativo tipo UNIX, completamente "libero", costruito dalla Free Software Foundation utilizzando:

- s/w libero sviluppato da altri (sul quale è apposta la licenza GPL) , e
- software libero sviluppato espressamente da FSF.

Le componenti più caratterizzanti del sistema sono quelle più vicine all'utente (comandi, shell, applicazioni); il kernel (che si chiama Hurd) è stato sviluppato per ultimo (ca. 2001) e non è considerato altrettanto valido di altri kernel liberi, soprattutto di Linux. La combinazione del kernel Linux con il resto del sistema GNU è il sistema operativo tipo UNIX più diffuso sui PC, generalmente noto col semplice nome di "Linux". Il corretto nome è invece GNU/Linux.

Tra le più importanti componenti del sistema GNU ci sono il compilatore (gcc), la libreria C (glibc), l'editore Emacs e il Desktop GNOME.

In senso più generale, si definisce "software GNU" tutto il software coperto da licenza GPL, che può essere installato separatamente su sistemi operativi non-GNU.

> Da [Giacomini()]:
>
> Nel 1985, Richard Stallman ha fondato la FSF, Free Software Foundation, con lo scopo preciso di creare e diffondere la filosofia del «software libero». Libertà intesa come la possibilità data agli utenti di distribuire e modificare il software a seconda delle proprie esigenze e di poter distribuire anche le modifiche fatte.
>
> Queste idee filosofiche si tradussero in pratica nella redazione di un contratto di licenza d'uso, la General Public License, studiato appositamente per proteggere il software libero in modo che non potesse essere accaparrato da chi poi avrebbe potuto impedirne la diffusione libera. Per questo motivo, oggi, il copyright di software protetto in questo modo, viene definito copyleft.
>
> Il software libero richiede delle basi, prima di tutto il sistema operativo. In questo senso, l'obiettivo pratico che si prefiggeva Richard Stallman era quello di realizzare, con l'aiuto di volontari, un sistema operativo completo.
>
> Nacque così il progetto GNU (Gnu's Not Unix), con il quale, dopo la realizzazione di un compilatore C, si volevano costruire una serie di programmi di servizio necessari nel momento in cui il cuore del sistema fosse stato completo.
>
> Il progetto GNU diede vita così a una grande quantità di software utilizzabile sulla maggior parte delle piattaforme Unix, indirizzando implicitamente il software libero nella direzione dei sistemi di questo tipo.
>
> Nel 1990 inizia lo sviluppo del kernel Hurd e intorno al 2000 inizia la distribuzione del sistema GNU/Hurd (sistema GNU basato su kernel Hurd).

3.8 GNU/Linux

Il Sistema Operativo della famiglia UNIX più diffuso sui PC è GNU/Linux, spesso chiamato semplicemente -ma erroneamente- "Linux" (Linux è invece solo il kernel, v. sopra).

Il sistema GNU/Linux è "libero" in quanto distribuito con licenza GPL. Si può copiare legalmente: ad esempio, è possibile scaricarlo gratuitamente da WWW.

Esistono numerose versioni di GNU/Linux (più di 300 nel 2006), che si differenziano per la versione del kernel e per la scelta del software GNU da associargli. Una "distribuzione Linux" è l'unione di una versione GNU/Linux e (eventualmente) altro sofware non-GNU, ma generalmente libero, confezionata in modo compatto e con strumenti per l'installazione e la manutenzione.

Una distribuzione GNU/Linux in generale rispetta gli standard di tipo POSIX con vaste estensioni. Comprende anche software non incluso negli standard POSIX, tra cui soprattutto:

- la suite di comunicazione in rete: TCP/IP
- l'interfaccia grafica: X11 + Desktop Manager + ...

In effetti per i sistemi GNU/Linux esiste uno standard di fatto più esteso del POSIX: la Linux Standard Base.

Esempi di distribuzioni GNU/Linux:

- Debian
- Caldera
- RedHat
- Slackware
- SuSE
- ...

Capitolo 4

Uso elementare di UNIX

In questa sezione e nella successiva si illustra l'uso di UNIX da interfaccia testuale, o terminale. L'uso dell'interfaccia grafica è molto intuitivo, e in compenso è molto meno uniforme tra versioni diverse del sistema operativo; perciò lo si illustra soltanto in laboratorio.

NOTA: i comandi che appaiono sottolineati, come less, non sono UNIX, ma estensioni GNU, o altre che comunque si trovano nella maggior parte delle distribuzioni di GNU/Linux

4.1 premessa: maiuscole e minuscole

In UNIX, maiuscole e minuscole sono lettere diverse! Ad esempio il file `Mail` è un'altra cosa dal file `mail`, oppure le password

- `15SET03`
- `15Set03`
- `15set03`

sono tutte diverse tra loro.

4.2 sessione di lavoro

4.2.1 `login`, `logout` e `exit` su un terminale

Per lavorare su UNIX bisogna aprire una *sessione*, cioè qualificarsi come un certo utente riconosciuto dal sistema con certe sue caratteristiche. Alla fine della sessione si scollega l'utente, ma il sistema in generale resta acceso in attesa di nuove sessioni.

Come prima cosa, UNIX chiede di immettere il nome utente e la relativa password. Ad esempio, un terminale può presentare questa scritta:

```
AIX Version 4
(C) Copyright by IBM and others 1982,1994
login:
```

Una volta entrati, ci viene presentato un invito ai comandi (*prompt*). La nostra riga di comando viene interpretata dalla *shell*.

Alla fine del lavoro, si termina la sessione su terminale dando `exit` o `logout`[1]

4.2.2 Sessione grafica con apertura di un terminale

Quasi tutti i sistemi UNIX con cui si ha a che fare attualmente gestiscono una sessione grafica a finestre, all'interno della quale si può aprire una finestra a interfaccia testuale o terminale.

Da una sessione non-grafica si può far partire la sessione grafica con il comando `startx` o `xinit`. Si viene immessi in un Desktop con varie applicazioni; talvolta un terminale si apre automaticamente; altrimenti lo si può aprire usando il menù principale (che tipicamente si lancia con un bottone in basso a sinistra dello schermo).

Dal terminale si esce con `exit`. Per uscire dalla sessione grafica si usa un'opzione del menù principale del desktop, oppure si premono i tasti `Ctrl-Alt-Backspace` contemporaneamente.

In questo modo si ritorna alla maschera di login. Il sistema è ancora accesso, in attesa di nuovi login. Per spengere il sistema si premono i tasti `Ctrl-Alt-Delete` contemporaneamente.

Molto spesso la stessa maschera di login è grafica. In questo caso si può scegliere tra varie sessioni grafiche (che differiscono per il Desktop, etc.), nonché una sessione non-grafica (failsafe) da far partire. Dalla maschera di login grafica si può spengere il sistema scegliendo l'opzione `Shutdown.`

[1] `exit` è il comando generico di uscita dalla shell (funziona sempre); `logout` fa uscire solo da una shell di login.

4.2.3 sintassi di un comando

- una riga di comando è ciò che si digita al prompt, fino al tasto <invio> (o fino al primo "`;`")
  - la riga di comando è costituita di parole separate da uno o più spazi
  - la sintassi di un comando è sempre

    `comando [opzioni] [argomento1 argomento2 ...]`

  `comando:` la prima parola

  `opzioni:` (o *switch*) parole che cominciano con il carattere "`-`" e stanno tra comando e argomenti (cambiano il comportamento del comando)

  `argomenti:` tutte le parole dopo il comando e le eventuali opzioni

  Es:

```
date
date -u
banner ciao

banner -w80 ciao

grep Rossi elenco.txt
grep -i Rossi elenco.txt
tar --help
```

4.2.4 il comando `passwd`

Un comando importante è quello che permette di cambiare la propria password. In generale è

```
passwd
```

nel nostro caso invece è

```
yppasswd
```

Si deve immettere la vecchia password, poi la nuova due volte. Usare solo lettere, numeri e caratteri che si trovano su tutte le tastiere.

4.2.5 utenti e gruppi; `who` e `finger`

- ricordare: si accede sempre come un utente
- utenti hanno accesso differenziato a diverse risorse
- utenti sono riuniti in gruppi; anche i gruppi hanno accesso differenziato a risorse
- l'utente *root* (=radice) ha tutti gli accessi: serve per gestione sistema (non lo trattiamo qui)
- un utente ha:
  - nome
  - password
  - gruppo di appartenenza
  - (nome reale)
  - directory di partenza (*home directory*); in generale, per l'utente `tizio`, è

    `/home/tizio`

  - shell di partenza

Esempi:

- `who`
- `finger, finger` *`utente`*

  `finger` fa vedere gli utenti collegati, in un modo simile a `who`; con `finger` *`utente`* si ottengono informazioni su qualunque utente, anche se non collegato

4.3 guide

Per avere informazioni sul funzionamento di un comando, spesso si può usare l'opzione `-h` o `--help` del comando stesso.
Es:

```
gzip -h
tar --help
```

Altrimenti si possono usare altri comandi:

`whatis` *`comando`*	definizione di *`comando`* (una riga)
`man` *`comando`*	guida esauriente su *`comando`* (qualche pagina)
`apropos` *`parola`*	elenco di comandi che hanno *parola* nella loro definizione
`info` *`menu`*	manuali specializzati, in forma ipertestuale

Es:

```
man passwd
whatis ls
apropos calendar
info emacs
```

4.4 navigazione nel *filesystem* e gestione elementare di file e directory

4.4.1 file e directory; directory corrente e directory superiore; `ls`, `pwd` e `cd`

- il file è l'unità fondamentale di informazione dal punto di vista del sistema operativo:
  può essere un documento, un insieme di dati in forma binaria, un programma
- i file sono organizzati in un sistema a directory ramificato che parte da una radice; una directory può contenere
  - file
  - (sotto-)directory
- in ogni momento ci si trova su una directory (la directory *corrente*): per sapere quale, dare

  ```
  pwd
  ```

  per conoscere il contenuto della directory corrente, dare

  ```
  ls
  ```

- per cambiare directory, dare

  ```
  cd nuovadirectory
  ```

  in particolare, per andare alla directory *superiore* o "genitrice" (cioè, quella che contiene la directory corrente e che si raggiunge dalla directory corrente risalendo verso la radice):

  ```
  cd ..
  ```

4.4.2 percorso assoluto e relativo

In UNIX ogni oggetto (file o directory) può essere sempre identificato in tre modi:

1. se si trova sulla **directory corrente**, con il semplice nome (es. `pippo` o `gigi`)
2. se **non** si trova sulla directory corrente, facendo precedere il nome dal **percorso** da compiere (cioè la sequenza di directory da attraversare) per arrivare all'oggetto. Il percorso (*path*) può essere specificato in modo
   (a) **relativo:** cioè a partire dalla directory corrente. In questo caso, il nome **non** comincia mai con la barra (/).
   es: se ci si trova sulla directory `/home/tizio/`, il primo file si chiama
   `ese/pippo`
   (b) **assoluto:** a partire dalla radice (/). In questo caso, il nome comincia sempre con la barra.
   es: il percorso assoluto per i due file evidenziati è, rispettivamente:
   `/home/tizio/ese/pippo`
   `/home/infochim/wrk/e3/gigi`

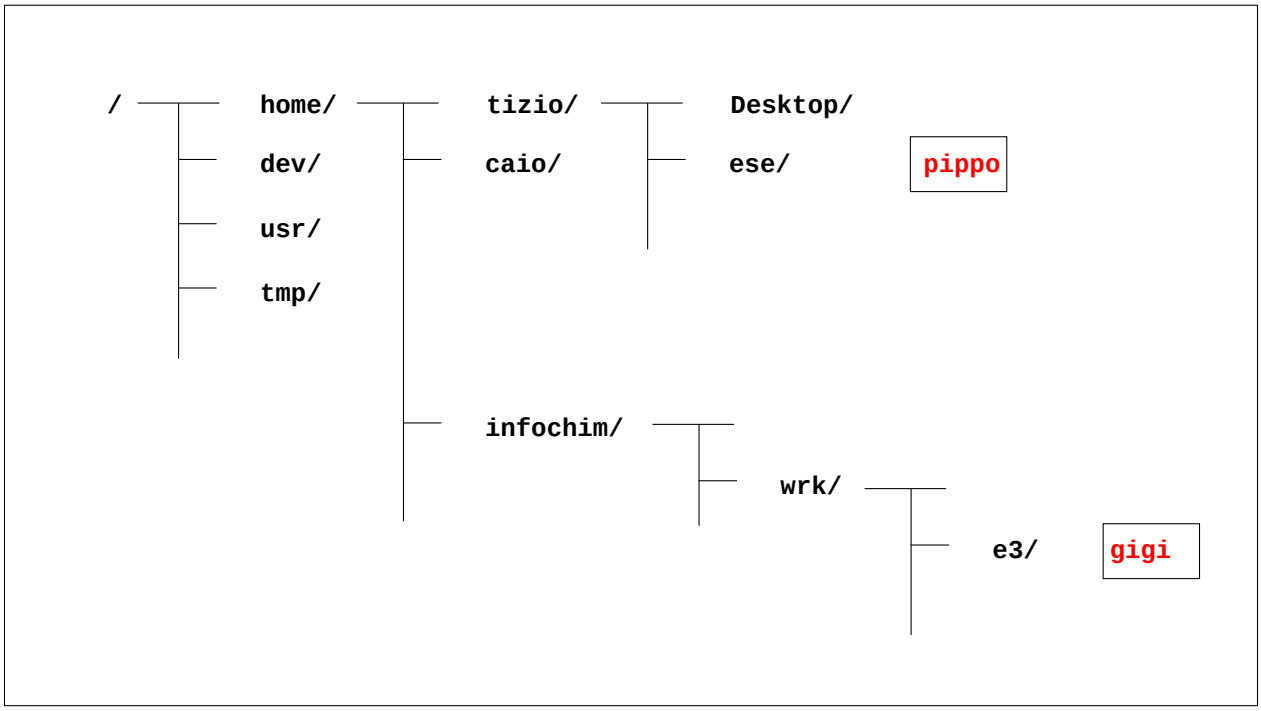


Figura 4.1: esempio di percorsi

Esempio: percorsi del file `pippo` (cfr. la figura precedente)

se la directory corrente è:	*il nome del file è:*
(qualunque)	/home/tizio/ese/pippo
/home/tizio/ese/	pippo
/home/tizio/	ese/pippo
/home/tizio/ese/	./pippo
/home/caio	../tizio/ese/pippo

se l'utente è:	*il nome del file è:*
tizio	~/ese/pippo
caio	~tizio/ese/pippo

4.4.3 le directory ".", ".." e "~" ;

Ci sono dei nomi speciali di directory:

`..`	la directory "superiore" (l'abbiamo già vista nel comando `cd`)
`.`	la directory corrente
`~`	la mia home directory
`~tizio`	la home directory dell'utente `tizio`

Notare che

- `.` `..` `~` sono directory diverse a seconda di dove ci si trova e di che utente siamo.
- riferirsi a un file come *`./pippo`* o come *`pippo`* è identicamente la stessa cosa.

4.4.4 permessi di lettura, scrittura ed esecuzione dei file; `ls -l`

- ogni file appartiene ad un utente e a un gruppo
- permessi:

  ogni file può avere il permesso di

  - lettura (`r`)
  - scrittura (`w`)
  - esecuzione (`x`)

  e questi permessi possono essere in relazione a

  - utente
  - gruppo
  - tutti gli altri

  si esprimono con una stringa di nove caratteri:

Permesso	utente			gruppo			altri		
sì	r	w	x	r	w	x	r	w	x
no	-	-	-	-	-	-	-	-	-

Utente e gruppo di appartenenza, e permessi dei file, si possono visualizzare con il comando

```
ls -l
```

Es:

```
-rw-r--r-- 1 infochim chimica 6661 Nov 12 15:30 Mailbox
```

significa: leggibile e scrivibile, ma non eseguibile, dall'utente (`infochim`); solo leggibile dagli appartenti al gruppo (`chimica`); solo leggibile da tutti gli altri

Notare che la stringa dei permessi contiene 10 caratteri: il primo indica (fra l'altro) se si tratta di una directory, nel qual caso compare una `d`

Le altre informazioni che vengono stampate con questo comando (le ultime colonne prima del nome del file) sono la data e l'ora di ultima modifica del file.

4.4.5 nomi di file e directory; file e directory nascosti; `ls -a`

- nomi dei file e directory: non c'è alcuna limitazione vera e propria, ma sono assolutamente da evitare caratteri che hanno significato speciale:
  - spazi
  - `/~!@#$%^&*([{}])<>'`

  Caratteri permessi: `ABC...Z abc...z 0123...9 _ - .`
- notare che in Windows si fa largo uso di spazi nel nome di un file; in UNIX gli spazi separano parole, cioè argomenti diversi di un comando
- file o directory il cui nome comincia con "." sono "nascosti"; cioè, non sono visualizzati con il comando `ls`
  - si possono vedere con l'opzione `-a` di `ls`:

    ```
    ls -a
    ```

  N.B. si possono combinare le opzioni `-l` e `-a`, in uno qualunque dei seguenti modi

  `ls -l -a`

  `ls -a -l`

  `ls -la`

  `ls -al`

4.4.6 lista di una directory diversa dalla directory corrente: `ls` *dir*

- Il comando

  `ls` *dir*

  dà la lista dei file contenuti nella directory *dir*.

  Note

  1. La stessa cosa si può ottenere portandosi sulla directory e dando il comando `ls`:

     ```
     cd dir
     ls
     ```

  2. Un comando equivalente a `ls` è

     ```
     ls .
     ```

- Il comando

  `ls` *file1 file2 ...*

dà la lista dei file *file1*, *file2*, etc. Se non si dà l'opzione `-l`, sullo schermo compaiono (banalmente) soltanto i nomi dei file, se esistono. Con l'opzione `-l` si visualizzano anche tutte le informazioni. È una forma comoda per fare la lista dei file i cui nomi corrispondono ad un'espressione, ad es:

```
ls -l ~/doc/*.txt
```

dà la lista di tutti i file nella sottodirectory `doc` della nostra home directory i cui nomi finiscono in `.txt`.

4.4.7 comandi per gestire file: `cp`, `mv`, `rm`

`rm` rimuove (cancella)

`cp` copia

`mv` sposta (rinomina)

Il comando `rm` ha uno o più argomenti, il (o i) file da cancellare:

`rm` *file1*

`rm` *file1 file2 ...*

I comandi `cp` e `mv` hanno due o più argomenti.

- la sintassi è:

  comando origine ... destinazione

- si comportano in modo diverso a seconda che *destinazione* sia un file o una directory. Ad esempio:

riga di comando	*spiegazione*	*note*
`cp` *file1 file2*	copia *file1* in *file2*	se *file2* esiste già, ci riscrive sopra
`cp` *file1 dir*	copia *file1* in *dir/file1*	se *dir* non esiste o non è una directory, copia *file1* **nel file** *dir*
`cp` *file1 file2 ... dir*	copia *file1*, *file2*, ... in *dir/file1*, *dir/file2*, ...	

- la differenza tra `cp` e `mv` è che il secondo, dopo avere creato la copia, distrugge l'origine o le origini

4.4.8 uso di `cp` e `mv` per copiare e rinominare directory

Il primo argomento dei comandi `cp` e `mv`, l'origine, può anche essere una directory. In questo caso

- `cp` deve essere seguito dall'opzione `-R`
- ci deve essere un secondo argomento, che indica la directory di destinazione
- il comportamento è diverso a seconda che la directory di destinazione esista già o no:
  - se esiste già, la directory origine (con tutto il suo contenuto) viene copiata/spostata entro la directory destinazione, come sua sottodirectory
  - se non esiste, la directory origine viene copiata/rinominata col nome specificato dalla destinazione

riga di comando	*spiegazione*	*note*
`cp -R` ***dir1 dir2***	crea una copia di ***dir1*** col nome di ***dir2***	***dir2*** non esiste
`cp -R` ***dir1 dir***	crea una copia di ***dir1*** in ***dir*** (creando ***dir/dir1***)	***dir*** esiste già
`mv` ***dir1 dir2***	rinomina ***dir1*** come ***dir2***	***dir2*** non esiste
`mv` ***dir1 dir***	sposta ***dir1*** in ***dir*** (creando ***dir/dir1)***	***dir*** esiste già

4.4.9 comandi speciali per gestire directory: `mkdir` e `rmdir`

- per creare una directory:

  `mkdir` ***directory***

- per rimuovere una directory:

  `rmdir` ***directory***

  per poterla rimuovere, la directory deve essere vuota

4.4.10 vedere il contenuto di un file: `cat, more, less`

- per visualizzare sullo schermo il contenuto di un file:

  ```
  cat file
  ```

- lo stesso, ma una schermata per volta:

  ```
  more file
  ```

- lo stesso, con la possibilità di scorrere anche all'indietro:

  ```
  less file
  ```

Tabella 4.1: **Sessione, sintassi, utenti, guide**

sintassi generale di un comando:

comando [opzioni] [argomento1 argomento2 ...]	le opzioni cominciano sempre con una o due " -"

comandi per sessione, utenti, manuali:

comando	*note*
`exit`	per uscire dalla shell e dall'interfaccia testuale
`passwd` o `yppasswd`	per cambiare la propria password
`finger`	per vedere chi è collegato
`finger` *xxx*	informazioni sull'utente *xxx* (anche se non è collegato)
comando `-h`	(solo in certi casi) visualizza la sintassi e una guida breve di *comando*
comando `--help`	(solo in certi casi) visualizza la sintassi e una guida breve di *comando*
`whatis` *comando*	visualizza la definizione di *comando*
`man` *comando*	guida esauriente su *comando* (digitare `q` per uscire)
`apropos` *parola ...*	elenco di comandi che hanno *parola* nella loro definizione
`info` *argomento*	guida ipertestuale su *argomento* (digitare `q` per uscire)
`info`	sommario ipertestuale dei vari argomenti visibili con `info`

Tabella 4.2: **Gestione essenziale di file e directory.**
I parametri tra [] sono facoltativi

`.`	la directory corrente
`..`	la directory "superiore" (che contiene la directory corrente)
`~`	la directory di partenza (*home directory*) di questo utente
`~`*xxx*	la home directory dell'utente *xxx*
`pwd`	visualizza la directory corrente
`cd` *dir*	cambia la directory corrente a *dir*
`ls`	visualizza la lista della directory corrente
`ls -a`	visualizza la lista della dir. corr., compresi i file nascosti
`ls -l`	visualizza la lista della dir. corr. in formato lungo
`ls [`*opzioni*`]` *dir*	visualizza la lista della directory *dir*
`ls [`*opzioni*`]` *file1 file2 ...*	visualizza la lista dei file *file1 file2 ...*
`cp` *file1 file2*	copia *file1* in *file2*
`cp` *file1 dir*	copia *file1* nella directory *dir (esistente)*, con lo stesso nome; ovvero, copia `file1` in `dir/file1`
`cp` *file1 file2 ... dir*	copia *file1, file2, ...* nella directory *dir (esistente)*
`cp -R` *dir1 dir2*	copia la directory *dir1* in *dir2 (non esistente)*
`cp -R` *dir1 dir*	copia *dir1* nella directory *dir (esistente)*, con lo stesso nome; ovvero, copia `dir1` in `dir/dir1`
`mv` *file1 file2*	rinomina *file1* in *file2*
`mv` *file1 dir*	sposta *file1* nella directory *dir (esistente)*, con lo stesso nome; ovvero, rinomina `file1` in `dir/file1`
`mv` *file1 file2 ... dir*	sposta *file1, file2, ...* nella directory *dir (esistente)*
`mv` *dir1 dir2*	rinomina la directory *dir1* in *dir2 (non esistente)*
`mv` *dir1 dir*	sposta *dir1* nella directory *dir (esistente)*, con lo stesso nome; ovvero, rinomina `dir1` in `dir/dir1`
`rm` *file1 file2 ...*	rimuove (cancella) *file1 file2 ...*
`mkdir` *dir*	crea la directory *dir*
`rmdir` *dir*	cancella la directory *vuota dir*
`cat` *file*	visualizza *file*
`more` *file*	visualizza *file*, una schermata alla volta
`less` *file*	visualizza *file*, con possibilità di scorrere indietro

4.5 editor di testo: vi, nano (pico) e emacs

4.5.1 differenza tra *word processor* e *editor di testo*

Un file, oltre alle lettere, ai numeri e agli altri caratteri usati per rappresentare un testo, può contenere caratteri o sequenze "di controllo" interpretati dai programmi che lo leggono. Alcuni programmi evoluti di creazione di documenti (*word processor*) usano queste sequenze per dare un formato al testo (ad es. grassetto, allineato, disposto in una tabella), secondo una codifica standard (es. HTML) o specifica del programma (MS-Word).

Un *editor di testo*, invece, come i comandi per la visualizzazione di file (cat, more, less), mostra invece tutti i caratteri contenuti in un file, compresi quelli di controllo. Gli editor di testo sono utili quando si vuole lavorare sul contenuto "puro" di un file, o quando si lavora su file non formattati: ad esempio, dati numerici, tabulati di programmi, etc. Un esempio in MS Windows è NotePad. In ambiente GNU/UNIX ci sono `vi`, `nano` (o `pico`) e `emacs`.

- `vi` è un editor di testo rudimentale, che funziona su qualunque terminale (testuale). È un po' scomodo da usare perché ha una sintassi di comando rigida e poco o nessun aiuto in linea. Vantaggi:
  - fa parte del Sistema Operativo (POSIX), lo trovate sempre
  - può essere usato in condizioni di ristrettezza di risorse (es. in un collegamento remoto, senza tasti `Ctrl` o `Alt`)
- `nano` (GNU) o `pico` (Washington University) sono altrettanto ridotti (funzionano su terminale), ma più semplici da usare: c'è manuale in linea, si usano molte sequenze di controllo, tipo `Ctrl-X`.
- `emacs` è un programma GNU; è il miglior editor di testo disponibile. In ambiente grafico apre una finestra a sè con barra dei menu e funzionamento abbastanza intuitivo.

Per editare un file si dà il comando seguito dal nome del file, ad es.:

`pico` *file*

Se dal prompt UNIX si dà solo il comando (senza nome di file), il programma parte con un'area vuota, spesso detta *buffer*, sulla quale si può caricare il contenuto di un file esistente, o nella quale si può scrivere e poi salvare su un file nuovo. Se si dà il nome di un file e il file esiste, se ne carica nel buffer il contenuto per editarlo; se il file non esiste, si crea un file nuovo con quel nome collegato al buffer vuoto.

Nei prossimi paragrafi si riportano degli schemi di apprendimento di `vi` e `nano`; l'illustrazione di `emacs` è rimandata al capitolo successivo

4.5.2 vi

(ottimo manuale: "Using the `vi` *editor" in [UHU])*

concetti base

- modalità di operazione:
  - modalità comando
  - modalità inserimento
- il registro (buffer)

avviamento e apertura di file

- un file esistente
- un file nuovo
- senza argomenti

salvare e uscire

`:w [nomefile]`	salvare
`:q`	uscire
`:wq`	salvare e uscire
`:q!`	uscire senza salvare

inserimento di testo

`a`	inserire testo dopo il cursore
`i`	inserire testo prima del cursore
`o`	aggiungere una riga sotto
`O`	aggiungere una riga sopra

muoversi

`l`	cursore a destra
`h`	cursore a sinistra
`k`	cursore su
`j`	cursore giù

cancellare

`x`	carattere
`dh`	carattere precedente
`dd`	intera riga
`d$`	fino a fine riga
`p`	annulla ultima cancellazione

cercare e sostituire

`/`	cercare una serie di caratteri
`?`	cercare una serie di caratteri, all'indietro
`n`	cercare di nuovo in avanti
`N`	cercare di nuovo indietro
`:s/abc/xyz`	sostituire `abc` con `xyz` su una riga
`:g/abc/s//xyz/gc`	sostituire `abc` con `xyz` in tutto il documento, in modalità interattiva

4.5.3 `nano` (`pico`)

Lanciando `nano` da terminale, si viene immessi nell'ambiente di lavoro dell'editor. Si può tornare al terminale solo uscendo dall'editor.

1. aree dello schermo:
   (a) testata: nome programma e file
   (b) area di lavoro
   (c) barra pro-memoria di comandi
2. caricare un file: ^R, salvare ^O, uscire ^X

4.6 stampa

4.6.1 code di stampa

Il sistema di stampa più diffuso in UNIX è LPD/LPR: nel sistema operativo sono definite diverse *code di stampa*
Una coda di stampa è l'insieme di

- una stampante su una connessione
- varie opzioni di stampa (es. PostScript, fronte/retro, etc.)

4.6.2 stampa da un'applicazione

Se il file è caricato da un'applicazione, questa ha il suo menù di stampa, in cui in generale si può scegliere la coda di stampa a cui inviare il file

4.6.3 stampa da terminale

Da terminale (IT), il comando generico per stampare un file (senza "aprirlo" con un'applicazione) è

```
lpr -Pcoda file
```

se non si specifica la coda, il sistema stampa sulla coda di default.

Questo è il comando adatto per stampare un file che si trova già nel formato adatto alla coda di stampa: ad esempio un file solo-testo se la coda di stampa è in grado di stampare file di quel tipo, o un file in formato PostScript se la coda di stampa è PostScript.

- N.B. Nel desktop (IG) non esiste un comando generico di stampa. Ad esempio, il File Manager Konqueror (a differenza di quello di Microsoft Windows) non permette di stampare un file senza aprirlo. Cliccando sopra il nome del file si apre l'applicazione associata, e la stampa si fa da dentro l'applicazione

4.6.4 stampa di file codificati: PostScript, filtri e driver, convertitori

Tutte (o quasi) le stampanti sono capaci di stampare un file carattere per carattere. Questo è quello che si vuole per un file solo-testo.

I file che rappresentano un oggetto di qualità grafica (testo formattato, figure) sono codificati in un formato compreso dalle applicazioni (HTML, JPEG, MS-Word, ...) e devono essere tradotti in un formato compreso dalla stampante.

Lo standard "di fatto" per stampe di qualità grafica in UNIX è il PostScript.

Per stampare un file PostScript:

- se la stampante è PostScript, basta inviare il file così com'è alla relativa coda; ad es.:

  ```
  lpr -Pps file.ps
  ```

- se la stampante non comprende il linguaggio PostScript, il file deve essere tradotto nel formato della stampante da appositi "filtri" (es GhostScript) e "driver"; questo in generale viene fatto all'installazione del sistema creando una coda di stampa PostScript, che dall'utente viene vista come una stampante PostScript a tutti gli effetti

Se il file non è PostScript, per stamparlo su una coda PostScript lo si deve prima convertire in PostScript con un opportuno comando. Ad es, per stampare il file `locandina.pdf`, in formato PDF, lo si deve prima convertire in PostScript e poi inviare ad una coda di stampa Postscript:

```
pdf2ps locandina.pdf locandina.ps
lpr -Pps locandina.ps
```

Ci sono due comandi GNU di conversione a PostScript: enscript e a2ps.

- enscript converte un file solo-testo a PostScript
- a2ps converte un file di qualunque formato a PostScript

Entrambi, se non si specifica un file di output, inviano il file convertito direttamente alla coda di stampa. Quindi possono essere usati come comandi di stampa al posto di lpr per stampare file di qualunque formato su una coda di stampa PostScript.

4.6.5 stampa da applicazioni

Le applicazioni, attraverso il menù di stampa, spesso preparano un file temporaneo PostScript che inviano ad una coda di stampa PostScript. Così funziona ad esempio Mozilla.

4.6.6 schema riassuntivo della procedura di stampa

La procedura di stampa è quindi:

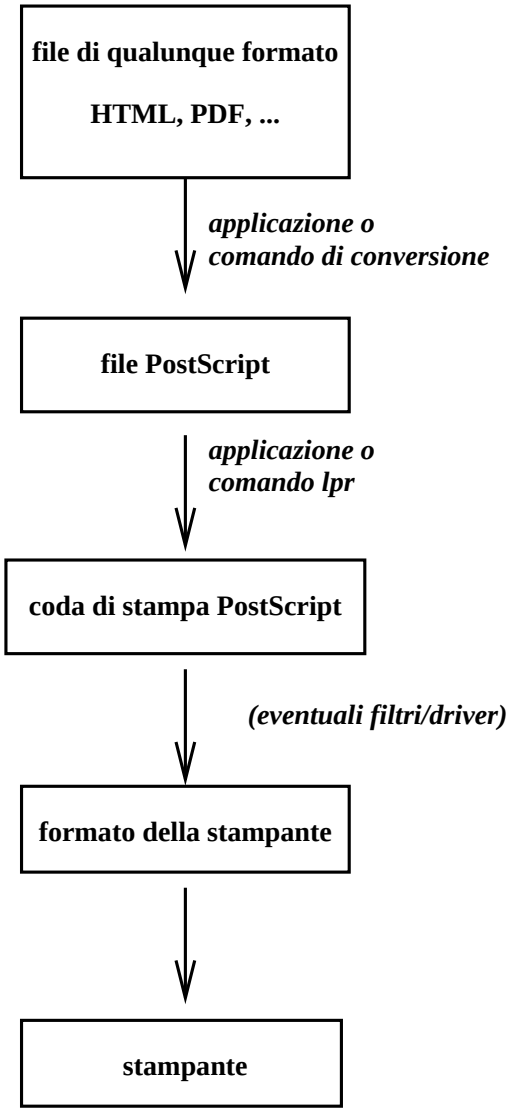


Tabella 4.3: **Comandi di stampa**

`lpr` *[-Pcoda] file*	stampa un file solo-testo su una coda solo-testo, o un file PostScript su una coda PostScript
`enscript` *[-Pcoda] file*	stampa un file solo-testo su una coda PostScript
`a2ps` *[-Pcoda] file*	stampa qualunque file su una coda PostScript

4.7 proprietà delle shell

Queste sono proprietà per lo più comuni a tutte le shell

4.7.1 sostituzioni

4.7.1.1 nomi di file

Quando ci si riferisce a file (o directory) si possono usare caratteri "jolly":

carattere jolly	*significato*
`*`	un qualunque numero di caratteri (anche nessuno)
`?`	un solo carattere
`[abc]`	un solo carattere tra quelli citati tra parentesi

Esempi:

`*.c`	tutti i file che finiscono per `.c`
`/etc/*`	tutti i file della directory `/etc`
`capitolo?.txt`	`capitolo1.txt capitolo2.txt ...` (ma non `capitolo23.txt`)
`[ab]*`	tutti file che cominciano per `a` o per `b`

La shell verifica quali file corrispondono all'espressione indicata e li sostituisce *pedissequamente* nella riga di comando, che poi esegue

N.B. : questo comportamento è differente dal DOS!

Esempio: supponiamo che la directory corrente contenga i file

```
lettera.doc
memo.txt
riassunto.doc
CV.doc
```

1. Il comando

   ```
   cp *.doc ~/backup-dir
   ```

   si traduce nel comando

   ```
   cp lettera.doc riassunto.doc CV.doc ~/backup-dir
   ```

   che copia i tre file nella directory `~/backup-dir`

2. Attenzione! Il comando

   ```
   cp lettera.doc *.txt
   ```

   diventa

   ```
   cp lettera.doc memo.txt
   ```

   che copia il primo file nel secondo, **distruggendolo!**

   NOTA: in DOS, il comando

   ```
   copy lettera.doc *.txt
   ```

   significa invece

   ```
   copy lettera.doc lettera.txt
   ```

4.7.1.2 altre sostituzioni

Oltre a quella dei caratteri jolly, la shell esegue una serie di sostituzioni sulla riga di comando. Ogni sostituzione è distinta da un carattere speciale.

caratteri	*sostituzione eseguita*	*esempio di comando*
`~`	directory di partenza (home directory) degli utenti	`ls ~tizio/tmp`
`$`	variabili	`echo Terminale definito come $TERM`
`!`	cronologia (ultimi comandi dati)	`!!`
‘comando‘	output del comando *comando*	`echo "Ci sono ‘who | wc -l‘ utenti collegati"`

Consideriamo in dettaglio soltanto il primo tipo di sostituzione. Un carattere “tilde” (`~`) all'inizio di un nome di file o directory, eventualmente seguito da una barra più altri caratteri, viene sostituito dal nome della *home directory* dell'utente attivo.

Se `~`, all'inizio di un nome di file o directory, è seguito immediatamente, senza barre interposte, dal nome di un utente, viene sostituito il nome di quella della directory home di quell'utente. Es:

`~`	la home directory dell'utente attivo
`~/xxx`	il file xxx sulla home directory dell'utente attivo
`~xxx`	la home directory dell'utente `xxx`
`~xxx/yyy`	il file `yyy` sulla home directory dell'utente `xxx`

4.7.1.3 proteggere da sostituzione

Quando si vuole usare un carattere speciale senza operare sostituzioni, lo si può proteggere:

\ protegge carattere successivo

’’ protegge tutto il testo contenuto tra `’ e ’`

"" protegge una frase, tranne `$ ‘` e \ (che vengono interpretati)

Es:

```
$ cd /
$ echo come va?
come var
$ echo come va\?
come va?
$ echo 'questo carattere si chiama tilde: ~'
questo carattere si chiama tilde: ~
$ echo questo carattere si chiama tilde: ~
questo carattere si chiama tilde:  /home/infochim
```

4.7.2 reindirizzamento di input e output, pipeline

In UNIX, i dati prodotti da un comando sono detti *standard output* e quelli forniti al comando sono detti *standard input*. Questi flussi di dati in generale sono associati a due dispositivi, tastiera e video:

standard input	tastiera
standard output	video

Si possono associare input e output a file con la seguente sintassi

comando > *file*	esegui *comando* e dirigi output su *file*
comando < *file*	esegui *comando* prendendo input da *file*
comando1 | *comando2*	l'output di *comando1* diventa input di *comando2*
comando >> *file*	aggiungi output di *comando* in coda a *file*

Esempio di redirezionamento dell'output:

```
ls                          stampa la lista della directory corrente
ls > a.000                  mette il risultato nel file a.000
```

Esempio di redirezionamento dell'input. Il comando bc è una calcolatrice che accetta input da tastiera, e dà il risultato nello standard output :

```
$ bc
6.1*2                          si digita un'operazione
12.2                           risposta del sistema
...                            ulteriore input
<ctrl-D>                       per uscire dal comando bc
$
```

Possiamo scrivere l'input per bc su un file, bc.dat; ed utilizzare questo come input del comando bc:

```
$ echo "6.1*2" > bc.dat
$ bc < bc.dat
12.2                                        risposta del sistema
$
```

Altro esempio di reindirizzamento dell'input: il comando wc (con l'opzione -w) conta le parole contenute in un file

```
wc -w < a.000                    conta le parole nel file a.000
```

I comandi ls e wc -w possono essere combinati in una *pipeline*, senza passare attraverso il file a.000:

```
ls | wc -w                       conta le parole nell'output di ls
```

Nota: una *pipeline* può essere costituita da più di due comandi in fila:

```
comando1 | comando2 | ...
```

e si può ridirigere l'input del primo comando e l'output dell'ultimo:

```
comando1 < file1 | comando2 | ...  > file
```

4.7.3 esecuzione di comandi in serie, parallelo e *background*

Quando si dà un comando dal prompt, non è possibile dare ulteriori comandi finché questo non è terminato. Dal prompt però si possono però dare due o più comandi per volta:

4.7.3.1 esecuzione in serie

Per eseguire due comandi in successione (o serie):

comando1 ; *comando2*	*Il sistema esegue* comando1, *aspetta che sia terminato, poi esegue* comando2

L'esecuzione in serie può essere applicata ripetutamente.

Esempio:

```
$ date; sleep -1; date
mar dic  9 14:29:03 CET 2003
mar dic  9 14:29:04 CET 2003
$
```

4.7.3.2 esecuzione in parallelo:

Per eseguire due comandi in contemporanea (o parallelo):

comando1 & *comando2*	*Il sistema esegue* comando1 *e senza aspettare che sia terminato esegue* comando2

Anche l'esecuzione in parallelo può essere applicata ripetutamente.

Esempio:

```
$ date & sleep -1 & date
...

mar dic  9 14:29:03 CET 2003
mar dic  9 14:29:03 CET 2003
...

$
```

4.7.3.3 esecuzione in *background*

Per eseguire un comando in sottofondo (*background*):

comando &

Il sistema esegue *comando* in sottofondo e presenta il prompt per nuovi comandi.

- L'esecuzione in parallelo può essere vista come un caso particolare di quest'ultimo, in cui *comando1* viene eseguito in *background* e *comando2* viene eseguito in superficie

4.7.4 processi e loro interruzione

4.7.4.1 processi e job

Ogni comando che viene eseguito sulla macchina viene detto ***processo*** e ad esso è assegnato un numero d'ordine unico.
Per vedere tutti i processi, dare il comando `ps`. Esempio:

```
$ ps
PID TTY          TIME CMD
17686 pts/3    00:00:00 tcsh
18829 pts/3    00:00:00 ps
```

Una successione di uno o più processi lanciati con un unico comando (ad esempio, una *pipeline*) viene detta ***job***. In particolare, se il job è eseguito in *background* ad esso è assegnato un numero d'ordine, che è unico per una sessione di shell (p. es. un terminale). Il numero del job viene scritto tra parentesi quadre sul terminale al momento che esso è lanciato. Es:

```
$ emacs &

    [1] 10301
```

Il secondo numero che viene scritto è il numero di processo dell'ultimo processo del job.

Per vedere su terminale la lista di tutti i job, dare il comando `jobs`

```
$ find /usr -mtime -7 | sort &
[2] 10334
$ jobs
[1]- Running       emacs
[2]+ Running       find /usr -mtime -7 | sort &
```

Il comando `find /usr -mtime -7` cerca in `/usr` e nelle sue sottodirectory tutti i file modificati negli ultimi 7 giorni; l'output viene poi filtrato da `sort`, cioè i file sono ordinati in ordine alfabetico. L'esplorazione del disco in genere richiede un po' di tempo, perciò è utile mandare questa pipeline in background.

Notare che ad ogni comando del *job* 2 (`find` e `sort`) viene assegnato un distinto numero di *processo*:

```
$ ps
10284 pts/0 00:00:00 bash
10301 pts/0 00:00:00 emacs
10333 pts/0 00:00:00 find
10334 pts/0 00:00:00 sort
10335 pts/0 00:00:00 ps
```

4.7.4.2 interruzione di processi e job

Ctrl-C cancella (definitivamente) il comando che si sta eseguendo su questo terminale (in superficie)

Ctrl-Z sospende temporaneamente il comando che si sta eseguendo e rende il prompt; per ripristinarlo:

- `fg` (lo ripristina in superficie o *foreground*)
- `bg` (lo ripristina in sottofondo o *background*)

Per cancellare definitivamente un processo o un job in *background* si usa il comando

`kill` *%numero-job*

oppure

`kill` *numero-processo*

Per sospendere un job che si sta eseguendo in sottofondo, lo si deve prima riportare in superficie:

```
$ jobs
[1]- Running       emacs
[2]+ Running       find /usr -mtime -7 | sort &
$ fg %2
find /usr -mtime -7 | sort &
$ ^Z
[2]+ Stopped       find /usr -mtime -7 | sort &
$ bg %2
[2]+ find /usr -mtime -7 | sort &
```

4.7.5 esempi

1. esaminare l'output di un comando una schermata alla volta

`last`	*fa vedere tutte le ultime sessioni su questo computer*
`last > a.0`	*scrive il risultato su* `a.0`
`less < a.0`	*legge il risultato una schermata alla volta*
`last | less`	*combina i due comandi in uno solo*

2. selezionare nomi in un elenco telefonico

`grep Acciai telefonico.txt`	*seleziona righe che contengono* `Acciai`
`grep 'Acciai ' telefonico.txt`	*elimina* `Acciaioli`
`grep 'Acciai ' telefonico.txt > a.0`	*mette il risultato su* `a.0`
`sort -k2 a.0`	*ordina le righe di* `a.0` *secondo il nome (seconda parola)*

   Lo stesso risultato si ottiene con l'unico comando

   ```
   grep 'Acciai ' telefonico.txt | sort -k2
   ```

3. contare gli atomi di idrogeno in un file PDB compresso

`zcat -c a.pdb.Z`	*decomprime il file su standard output*
`zcat -c a.pdb.Z > a.pdb`	*decomprime il file su* `a.pdb`
`grep ' H ' a.pdb > idrogeni`	*seleziona le righe che contengono atomi di H, e le mette in un nuovo file*
`wc -l idrogeni`	*conta le righe di questo nuovo file*

   Lo stesso risultato si ottiene con l'unico comando

   ```
   zcat -c a.pdb.Z | grep ' H ' | wc -l
   ```

4. Uso di variabili

`odir=OpenOffice.org1.1.0`	*inizializza la variabile* `odir`
	- NB: la sintassi in altre shell è diversa!
`cd $odir`	*equivale a* `cd OpenOffice.org1.1.0`

5. Statistiche sugli utenti del sistema

`ypcat passwd > a.pw`	*visualizza tutti gli utenti definiti sul sistema*
`wc -l a.pw`	*conta le righe di* `a.pw`
`echo ci sono 'ypcat passwd | wc -l' utenti`	*scrive quanti sono gli utenti definiti*
`grep :2000:  a.pw`	*seleziona solo gli utenti di chimica (gruppo 2000)*
`sort -t:  -k3 a.pw`	*ordina gli utenti secondo il numero utente (terzo campo)*

6. Ricerca di file

`find ~ -mtime +30`	*visualizza tutti i miei file non modificati negli ultimi 30 giorni*
`mv 'find ~ -mtime +30' ~/oldfiles/`	*li sposta nella directory* `oldfiles`

Tabella 4.4: **Proprietà della shell**

input e output

comando > *file*	dirigi output di *comando* su *file*
comando < *file*	prendi input di *comando* da *file*
comando1 | *comando2*	l'output di *comando1* diventa input di *comando2* (*pipeline*)
comando >> *file*	aggiungi output di *comando* in coda a *file*

processi

comando1 ; *comando2*	esegui *comando1* , aspetta che sia terminato, poi esegui *comando2*
comando1 & *comando2*	esegui *comando1* , e senza aspettare che sia terminato esegui *comando2*
comando &	esegui *comando* in *background*
Ctrl-C	arresta definitivamente il processo in corso e rende il *prompt*
Ctrl-Z	sospende il processo in corso
`fg`	ripristina il processo sospeso, in *foreground*
`bg`	ripristina il processo sospeso, in *background*
`jobs`	visualizza lista dei job
`ps`	visualizza lista dei processi
`kill` *%j*	arresta definitivamente il job *j*
`kill` *p*	arresta definitivamente il processo *p*

sostituzioni

espressione	*sostituzione eseguita*
*	nomi file: zero o più caratteri (tranne un . iniziale)
?	nomi file: un solo carattere (tranne un . iniziale)
[abc]	nomi file: un solo carattere tra quelli citati tra parentesi (idem)
~ *(da solo)*	directory di partenza (*home directory*) di questo utente
~/ *(all'inizio di un nome di file o dir)*	directory di partenza (*home directory*) di questo utente
~xxx *(da solo)*	home directory dell'utente *xxx*
~xxx/ *(all'inizio di un nome di file o dir)*	home directory dell'utente *xxx*
$*var*	valore della variabile *var*
!!	ultimo comando dato
‘*comando*‘	risultato del comando *comando*
\	protegge da sostituzione il carattere successivo
’’	protegge tutto il testo contenuto tra ’ e ’
""	protegge il testo tra "", tranne $ ‘ e \

4.8 altri comandi utili

Questa lista riporta l'output di `whatis` *`comando`*, in italiano se disponibile

file, directory, dischi

```
chmod               - cambia i permessi di accesso di un file
chown               - cambia l'utente e il gruppo proprietari dei file
find                - cerca i file in una gerarchia di directory
df                  - visualizza l'ammontare di spazio libero su disco
du                  - visualizza la quantità  usata di spazio su disco
```

manipolazione di file

```
head                - output the first part of files
tail                - output the last part of files
grep                - print lines matching a pattern
sort                - sort lines of text files
wc                  - stampa il numero di byte, parole e righe nei file
diff                - trova differenze tra due file
```

manipolazione avanzata di file

```
sed             - a Stream EDitor
gawk [awk]      - linguaggio di ricerca ed elaborazione di configurazioni di testo
```

stampa e simile

```
lpq                 - programma di controllo della coda di stampa
echo                - mostra una riga di testo
```

archiviazione e compressione

```
tar                 - La versione GNU del programma di utilità  tar
compress            - comprime ed espande dati (versione 4.1)
gzip                - comprime e decomprime i file
```

utenti, processi

```
passwd              - cambia la password
yppasswd            - change your password in the NIS database
finger              - user information lookup program
who                 - mostra chi è loggato
w                   - Mostra chi è loggato e cosa stanno facendo
ps                  - riporta lo stato dei processi
kill                - termina un processo
```

varie

```
bc (1)              - An arbitrary precision calculator language
```

Capitolo 5

Uso "avanzato" di UNIX

5.1 emacs

Emacs è un editor di testo molto potente. Al suo interno ospita anche applicazioni (posta elettronica, la shell UNIX, etc.) nelle quali si possono utilizzare molte delle funzionalità dell'editor (ad es. nella shell si può copiare un comando già dato e correggerlo prima di inviarlo).

Emacs può lavorare entro il terminale, o lanciare una finestra propria (in ambiente X-windows). Xemacs è una variazione di Emacs che sfrutta meglio la grafica.

La via migliore per imparare a usare emacs è leggere l'esercitazione guidata o tutorial (Da dentro emacs, battere: `<Alt>-x help-with-tutorial`).

5.1.1 aree dello schermo:

1. barra menu

   contiene i classici menu "File","Edit", etc. con i principali comandi

2. [barra degli strumenti]

   (nella nostra configurazione è disattivata)

3. area di lavoro: il buffer

   è l'area in cui si può immettere testo, detta *buffer*; emacs può lavorare su più di un buffer, ciascuno dei quali è spesso (ma non sempre) associato ad un file

4. riga di stato (Mode Line)

   dà informazioni sul buffer attivo

5. minibuffer

   è una riga usata da emacs per dare messaggi, e dall'utente per inserire comandi

6. barra verticale

   serve per scorrere sul buffer con il mouse ma soprattutto per segnalare visivamente quale porzione del buffer è visualizzata

5.1.2 menu, scorciatoie e comandi "complessi"

Tutte le azioni che si fanno in emacs, ad esempio aprire un file, sono associate univocamente ad un comando cosiddetto "complesso" che si può eseguire digitando `<Alt>-x` seguito dal nome del comando, ad esempio `find-file`. I comandi più comuni però sono associati a scorciatoie che si possono immettere da tastiera (ad esempio `<Ctrl>-x <Ctrl>-f`) e/o ad una voce di menu (ad es. `File/Open`).

Qui di seguito, seguendo la convenzione adottata dentro emacs stesso, abbrevieremo `<Ctrl>-` con `C-` e `<Alt>-` con `M-`

5.1.3 help: Apropos, Key, Whereis (locate)

Si può conoscere il rapporto tra comandi, scorciatoie e menù con i seguenti comandi di Help:

- `where-is` (`C-h w`) seguito dal nome di un comando fornisce le scorciatoie o le voci di menù che eseguono quel comnando
- `describe-key` (`C-h k`) seguito da una sequenza di caratteri fornisce il comando di cui quella sequenza è una scorciatoia; funziona anche selezionando una voce di menù con il mouse.

Inoltre, accanto alle voci di menù è riportata la scorciatoia corrispondente, se esiste.

Con il comando `apropos-command` seguito da una parola chiave si ottiene una lista di comandi che sono in relazione con quella parola chiave.

5.1.4 Interruzione: C-g

battendo C-g durante l'esecuzione di un comando, lo si cancella.

5.1.5 Editare un file

Lo si fa con il comando

M-x find-file NOMEFILE <ENTER>

La scorciatoia ufficiale per M-x find-file è C-x C-f.

Dando questo comando, nel minibuffer emacs offre di aprire un file nella directory corrente.

5.1.5.1 Aggiornare il contenuto di un buffer

Se un file viene modificato da fuori emacs (da un programma o da un altro utente) mentre il buffer ad esso collegato è aperto in emacs, il contenuto del buffer non si aggiorna automaticamente con il nuovo contenuto. Per aggiornarlo, occorre caricare nuovamente il file con il comando consueto (M-x find-file NOMEFILE <ENTER>); se il contenuto è cambiato da quando lo si è caricato la prima volta, emacs lo segnala e chiede conferma che lo si voglia ricaricare (buttando via eventuali modifiche fatte da allora). Una scorciatoia comoda è:

```
C-x C-f <down-arrow> <ENTER>
```

con la freccia-giu (<down-arrow>) si inserisce nel minibuffer al posto del nome del file da caricare quello attuale.

5.1.6 "Editare" una directory (dired)

Se al momento di aprire un file si dà invece il nome di una directory (senza la barra finale[1]), nel buffer viene caricata la lista (lunga) di quella directory, con possibilità di aprire un file direttamente dando invio sulla riga corrispondente

5.2 Regular Expression (espressioni regolari o formali)

- Sono espressioni che descrivono sequenze di caratteri. Vengono usate in funzioni e comandi (come grep o awk) che effettuano una ricerca di sequenze di caratteri all'interno di righe o stringhe.
- Si dice che c'è corrispondenza (*match*) tra una RE e una sequenza di caratteri contenuta in una stringa (e v.v.)
- La forma più semplice di una RE è una pura sequenza di caratteri, che corrisponde soltanto alla sequenza stessa:

Es: La RE /ranc/ corrisponde alla sequenza "ranc" contenuta sia nella stringa "Francesco" che nella stringa "melarancia"

- Con caratteri speciali si possono costruire RE che corrispondono a più di una sequenza di caratteri. Ad es., il punto (.) rappresenta un qualsiasi carattere.

Es: /or.o/ corrisponde sia a "orto" che a "orco", che a ...

Una RE è composta da:

elemento:	*corrisponde a:*
c	il carattere (non-speciale) c
\c	il carattere speciale c, preso letteralmente. Caratteri speciali: .*\[]^$-'''
.	qualsiasi carattere
[abc...]	uno qualsiasi dei caratteri abc...
[v-z]	= [vwxyz]
[^abc...]	qualsiasi carattere eccetto abc...
[^v-z]	= [^vwxyz]
^	inizio stringa
$	fine stringa

che si possono combinare nel modo seguente:

r1r2	l'espressione *r1* seguita dall'espressione *r2*
*r**	l'espressione *r* ripetuta zero o più volte
r+	l'espressione *r* ripetuta una o più volte

Esempi:

[1] anche con barra finale, a partire da emacs 22

RE	*è contenuta in (in neretto sottolineato la corrispondenza)*	*non è contenuta in*
/ranc/	"Francesco", "melarancia"	
/or.o/	"orco", "orto"	"oro"
/si do/	"famosi dottori"	"famosi dottori"
/[Ff]ranc/	"Francesco", "francesi", "Francia"	"melarancia"
/Franc.*a/	"Francesca", "Francia"	"Francesco", "francesca"
/[Ff]rancesc[oa]/	"Francesco", "Francesca", "francesco", "francesca", "Francescato", "Pierfrancesco"	"Francia"
/tel\.com/	"www.alcatel.com"	"telecom"
/^car.*o/	"carota", "carico", "cartone"	"cara", "Icaro"
/car.*o$/	"Icaro"	"carota"
/bana[^n]e/	"banale"	"banane"
/ta.*a/	"tanto va la gatta al lardo"	

In generale, una RE corrisponde alla stringa più lunga possibile.
Es:
nella stringa "tanto va la gatta al lardo", la RE /ta.*a/ corrisponde alla sottostringa "tanto va la gatta al la".

5.3 awk

5.3.1 caratteristiche generali

- È un comando che serve per elaborare file (di testo o numerici).

Es:

```
awk '{print $1}' /etc/hosts
```

- Di fatto, è un linguaggio di programmazione *interpretato*, con sua sintassi e comandi. Si possono scrivere programmi in *awk*.
- C'è uno standard POSIX di linguaggio *awk*.
- Su Linux RedHat trovate la versione GNU di *awk* che si chiama anche *gawk*

5.3.2 manuali

- *http://www.gnu.org/manual/gawk/index.html*
- `man awk`

5.3.3 sintassi del comando awk

Da `man awk`:

```
SINTASSI
      awk [ opzioni ] testo-del-programma file ...
      awk [ opzioni ] -f file-di-programma file ...
```

Cioè:

- si possono dare istruzioni
  - O sulla riga di comando (poche istruzioni, cfr. esempio precedente)
  - O in un programma già scritto, con l'opzione -f
- awk è sempre dipendente da un file. (C'è comunque un "trucco" per eseguire un programma senza file di input)

5.3.4 istruzioni di awk

- processa una riga (record) dell'input alla volta
- su ciascuna riga esegue tutte le istruzioni, nell'ordine

Le istruzioni sono sempre del tipo

```
pattern {action}
```

che significa: se la riga contiene una stringa corrispondente a `pattern`, viene eseguita l'istruzione `action`.
Esempio:

```
awk '/polosci/ {print $1,$2}' /etc/hosts
```

Se `pattern` è assente, l'azione `action` viene sempre eseguita; se l'`action` è assente, si stampa l'intera riga

5.3.5 Campi

Ogni volta che awk legge un record lo spezza in campi, usando il valore della variabile FS (espressione regolare) come separatore di campo (default=sequenza di spazi o un tabulatore o un accapo). Se FS è un singolo carattere, i campi sono separati da quel carattere.
Es: il file */etc/passwd* ha il formato `user:password:userid:groupid:name:home:shell`

```
xavier:x:1224:1000:Francesco:/home/xavier:/bin/bash
```

Ogni campo nel record in ingresso può essere individuato dalla sua posizione: `$1`, `$2`, e così via. `$0` è l'intero record.

5.3.6 Variabili

Si possono definire ed utilizzare delle variabili:

```
r=25.
b="ciao"
crf=r*2*3.14
```

il riferimento alle variabili è fatto in modo diretto, senza il carattere "$" (differenza da shell!)
I loro valori sono numeri in virgola mobile, o stringhe, o entrambe le cose, a seconda di come sono usati.
AWK dispone di vettori monodimensionali (i vettori multidimensionali possono essere simulati):

```
x[i]=15
```

Al lancio del programma sono impostate parecchie variabili predefinite (es: FS)

5.3.7 Sequenze (*pattern*)

- espressioni regolari riportate tra / e /
- oppure: espressioni relazionali
  cioè quelle che usano (tra gli altri) gli operatori
  `==` `!=` `<` `>` `<=` `>=`
- combinazioni di pattern, ad esempio con operatori logici && ||
- BEGIN e END

5.3.8 azioni

Le istruzioni possono essere assegnamenti o istruzioni condizionali o iterative o di I/O come si trovano nella maggior parte dei linguaggi.

Gli operatori, le strutture di controllo e le istruzioni di input/output ricalcano le corrispondenti del linguaggio C

Operatori `+ - * / ^` etc.

Istruzioni di controllo

`if (`*condizione*`)` *istruzione* `[ else` *istruzione* `]`
`exit`

Istruzioni di I/O

`print [`*expr-list*`]`
`next`

Funzioni predefinite

- numeriche: `cos()`, `exp()`, ...
- di stringa; es `length()`

5.3.9 un semplice esempio

Nel file *~infochim/wrk/esercitaz* sono riportati i punteggi ottenuti nelle prime due esercitazioni dai vari studenti del corso. Il file contiene nell'ordine: nome account, turno, punteggio eserc.1, punteggio eserc. 2:

```
acciai       D 2 5
allegra      A 2 3
allegri      A 0 0
arturoni     A 2 5
...
```

La media dei punteggi della prima esercitazione si ottiene con il seguente programma awk:

```
                {s=s+$3;n=n+1}
END             {print "media= ",s/n}
```

Possiamo scrivere questo programma nel file *media.awk* e eseguire il comando

```
awk -f media.awk esercitaz
```

Sull'output viene riportata la risposta:

```
media= 1.54082
```

Se ora vogliamo il punteggio medio di chi appartiene al turno A, possiamo farlo introducendo un pattern nella prima istruzione:

```
/A/             {s=s+$3;n=n+1}
```

Questo sistema funziona solo se siamo sicuri che l'account è sempre in minuscolo, e che quindi la stringa `"A"` può essere presente solo nel campo del turno.

Un sistema più corretto e sicuro è richiedere che la `A` si trovi nel secondo campo. In questo caso il pattern deve essere non un'espressione regolare, ma un'espressione relazionale:

```
$2=="A"         {s=s+$3;n=n+1}
```

Se invece vogliamo stampare in fondo a ogni riga la somma dei due punteggi:

```
                {print $0, $3+$4}
```

Questa istruzione invece stampa solo le righe relative a studenti che hanno preso almeno 4 nella seconda esercitazione:

```
$4 > 3
```

Capitolo 6

APPLICAZIONI SU UNIX

6.1 gnuplot

- È un programma che serve per diagrammare
  - file di dati
  - funzioni analitiche
  - entrambi contemporaneamente

6.1.1 uso

- Si lancia con '`gnuplot`'
- Il comando principale è '`plot`'
- Per uscire: `quit`
- Per guida: `help`

6.1.2 diagrammare un file di dati esistente

`plot 'a.0'`	usa la prima colonna come x, la seconda come y
`plot 'a.0' using 1:3`	" " " " , la terza come y
`plot 'a.0' using 1:($2/$3)`	diagramma il rapporto di seconda e terza colonna in funzione della prima
`plot [1:10] 'a.0'`	nell'intervallo [x=1, x=10]
`plot [1:10][-100:100] 'a.0'`	nell'intervallo [x=1, x=10] e [y=-100, y=100]
`plot 'a.0' with lines`	formato del diagramma ('`help plot with`' per elenco di stili)

6.1.3 diagrammare una funzione

`plot sin(x)`	100 punti nell'intervallo [x=-10,x=10]

Si possono definire costanti e funzioni:

```
gnuplot> beta=.28
gnuplot> p(x)=exp(-beta*x)
gnuplot> plot [0:10] p(x)
```

6.1.4 Esempio: il pH di una soluzione per aggiunta di base forte.

Per diagrammare i punti sperimentali contenuti nel file `~infochim/wrk/titol` e la curva teorica $pH(x) = 14 + \log_{10}(x)$ tra $x = 0.01$ e $x = 1$:

```
cd ~infochim/wrk
gnuplot
gnuplot> plot [0.01:1] 'titol', 14.+log(x)/log(10.)
```

6.1.5 ottimizzazione (fit) di una funzione per la riproduzione di un insieme di dati

```
gnuplot> p(x)=1./q*exp(-beta*x)
gnuplot> fit p(x) 'distrib.dat' u 0:1 via q,beta
gnuplot> plot 'distrib.dat' u 0:1, p(x)
```

6.1.6 registrare una sessione `gnuplot`

Il comando `save` registra sul file tutti i comandi dati durante la sessione

```
gnuplot> save 'session.gnu'
```

NB: non usare `plot` per salvare un grafico su file; per far questo si usano i comandi `set term` e `set output` (cfr. sotto)

Per ricaricare i comandi salvati con `save`:

```
gnuplot> load 'session.gnu'
```

6.1.7 registrare e stampare un grafico

Per registrare un grafico su file si deve

1. definire il formato del file:

   ```
   gnuplot> set term postscript
   ```

2. definire il nome del file

   ```
   gnuplot> set output 'figure.ps'
   ```

3. dare un nuovo comando di stampa

   ```
   gnuplot> plot [0:10] p(x)
   ```

Per riottenere il grafico su schermo (X-window):

```
gnuplot> set term x11
gnuplot> set output
gnuplot> replot
```

Per stampare un grafico, conviene prima salvarlo su file e poi stampare questo file dalla shell. Si possono dare comandi di shell da dentro gnuplot facendoli precedere da un punto esclamativo:

```
gnuplot> !lpr -Pps figure.ps
```

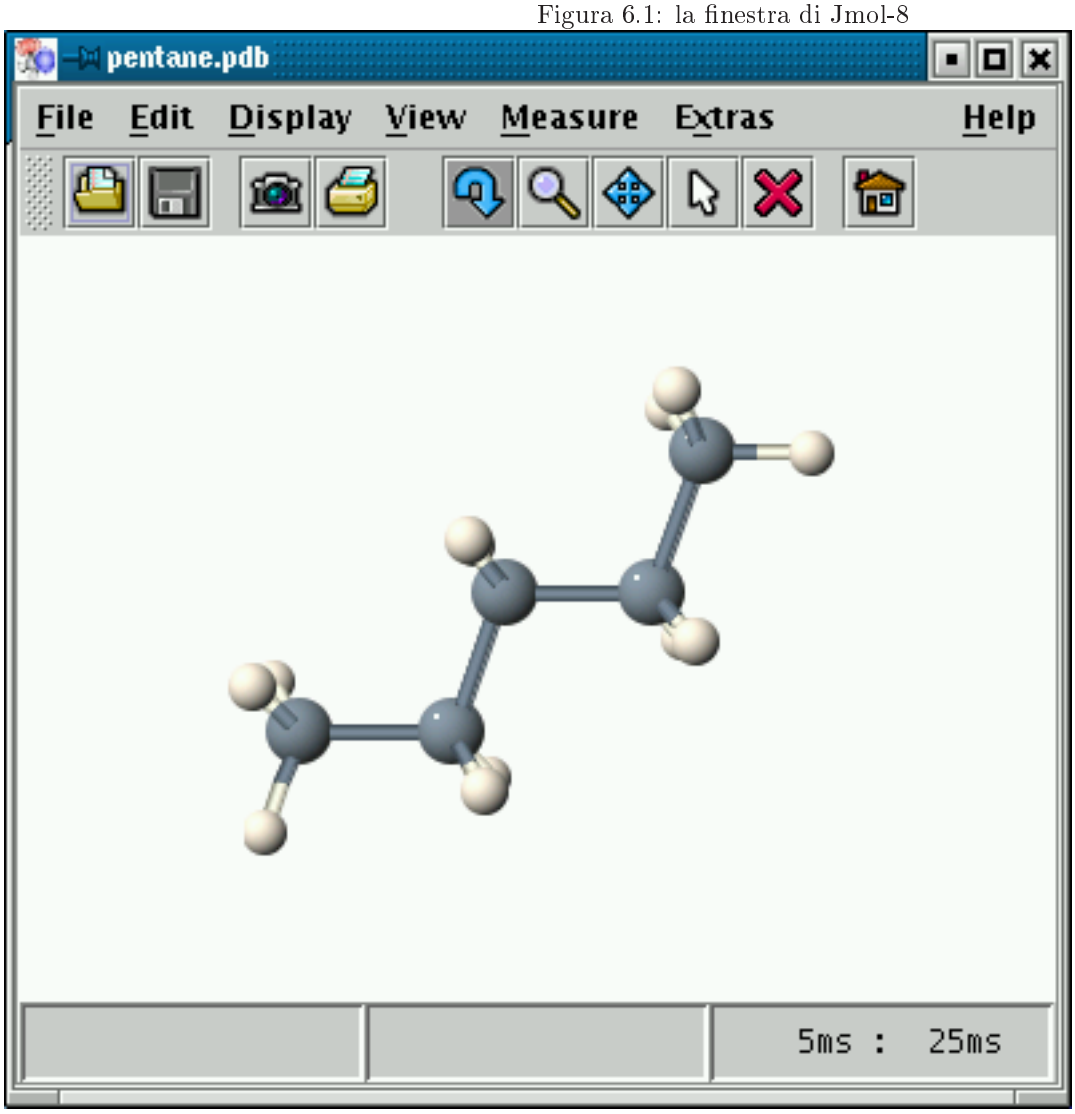
6.2 grafica molecolare: jmol

Jmol è un programma *"libero"* per la semplice visualizzazione di molecole, cristalli ed altri aggregati atomici. L'input è un file di coordinate atomiche in formato XYZ, PDB, etc.

Si lancia con `jmol [file-dati]`.

Si apre una finestra grafica con la figura della molecola. Attraverso i menu è possibile caricare nuovi file, muovere la molecola, ingrandirla, misurare angoli e distanze, etc.

Figura 6.1: la finestra di Jmol-8



6.3 esercizi

Capitolo 7

Note

7.1 nota: la tastiera

UNIX fa uso di alcuni caratteri che non sono presenti sulle tastiere italiane. Sui calcolatori dell'aula didattica (aula 40) questi caratteri si ottengono con le seguenti combinazioni:

Per scrivere:	*digitare:*
~	AltGr í
{	AltGr 7
}	AltGr =
‘	AltGr ’

Bibliografia

[Attivissimo(2000)] Paolo Attivissimo. *Da Windows a Linux*. Apogeo, Milano, 2000. ISBN 88-7303-735-6. URL http://attivissimo.homelinux.net/w2l1/.

[Kernighan and Pike(1989)] B. W. Kernighan and R. Pike. *UNIX*. Zanichelli, 1989. ISBN 88-08-04376-2.

[BIU()] A Brief Introduction to Unix. URL http://staff.washington.edu/~corey/unix-intro+man.html.

[4LW()] Unix is a Four-Letter Word. URL http://www.caspur.it/guidautenti/unix/4ltrwrd/.

[UHU()] Unix Help for Users. URL http://unixhelp.ed.ac.uk/.

[Ricar(1999)] M. A. Ricar. *Linux flash*. Apogeo, 1999. ISBN 88-7303-506-X.

[Georges Piriou(2004)] Marco Tripolini Georges Piriou. *Linux pocket*. Apogeo, 2004. ISBN 88-50321988.

[Petersen(1998)] R. Petersen. *Linux: the Complete Reference*. Sec - Osborne McGraw-Hill, 1998. ISBN 0-07-882461-3.

[Giacomini()] D. Giacomini. *Appunti di Informatica Libera*. URL http://www.pluto.linux.it/ildp/AppuntiLinux/a2.html.

[Giacomini(2001)] D. Giacomini. *Appunti di Informatica Libera Economici*. Systems Comunicazioni srl <http://www.systems.it>, 2001. ISBN 88-86422-43-1.

[Fre(a)] The Free Software Definition. a. URL http://www.fsf.org/philosophy/free-sw.html.

[Fre(b)] Cos'e' il Software Libero. b. URL http://www.fsf.org/philosophy/free-sw.it.html.

[Commissione Governativa sull'Open Source(2003)] Commissione Governativa sull'Open Source. Indagine conoscitiva sul software a codice sorgente aperto nella Pubblica Amministrazione. 2003. URL http://www.innovazione.gov.it/ita/comunicati/open_source/indagine_commissione_os.pdf.

[Boldrin and Levine(2002)] Michele Boldrin and David K. Levine. The Case Against Intellectual Property. *American Economic Review*, 92:209–212, 2002. URL http://levine.sscnet.ucla.edu/papers/intellectual.htm.

[Bessen(2003)] James Bessen. The Case Against Intellectual Property, by M. Boldrin and D. K. Levine. *Technological Innovation and Intellectual Property*, 2003. URL http://www.researchoninnovation.org/tiip/archive/2003_2_a.htm.

[Ministero dell'Innovazione(2002)] Ministero dell'Innovazione. L'Open Source nella pubblica amministrazione. 2002. URL http://www.innovazione.gov.it/ita/egovernment/infrastrutture/open_source.shtml.

[Schoolforge-UK(2006)] Schoolforge-UK. Benefits of FLOSS. 2006. URL http://www.schoolforge.org.uk/index.php/Benefits_

[Wikipedia()] Wikipedia. *Wikipedia*. URL http://it.wikipedia.org/wiki.

[FOLDOC()] FOLDOC. *Free OnLine Dictionary Of Computing*. URL http://foldoc.doc.ic.ac.uk/foldoc.