ABILITÀ INFORMATICHE PER CHIMICI (2003/2004)
Introduzione a UNIX

Dr. Giorgio F. Signorini
Università di Firenze
Dipartimento di Chimica
(stanza 326, tel 055-457-3085, signo@chim.unifi.it)
http://www.chim.unifi.it/~signo/did/unix

15 aprile 2004

Indice

1 Introduzione e obiettivi **4**
1.1 Bibliografia su UNIX . . . 4

2 Sistemi Operativi **4**
2.1 Cos'è un Sistema Operativo . . . 4
2.1.1 S. O. : definizioni ed esempi . . . 5
2.2 Interfacce utente . . . 6
2.2.1 vantaggi dell'interfaccia testuale . . . 7
2.3 Principali caratteristiche di un Sistema Operativo . . . 7
2.3.1 numero di processi contemporanei . . . 7
2.3.2 numero di utenti . . . 7
2.3.3 diritti di proprietà e di uso . . . 8
2.4 Approfondimento: software libero e *open source* . . . 8
2.4.1 definizione di software libero . . . 8
2.4.2 pro e contro il software libero . . . 9
2.5 Organizzazione dei dati su disco: file e directory (cartelle) . . . 9

3 UNIX **11**
3.1 Caratteristiche di UNIX . . . 11
3.2 vantaggi e svantaggi di UNIX su Windows . . . 12
3.3 UNIX come famiglia di sistemi operativi, liberi e non . . . 12
3.3.1 breve storia di UNIX . . . 13
3.4 Linux . . . 13
3.5 GNU . . . 14

4 Uso elementare di UNIX **15**
4.1 premessa: maiuscole e minuscole . . . 15
4.2 sessione di lavoro . . . 15
4.2.1 `login`, `logout` e `exit` su un terminale . . . 15
4.2.2 Sessione grafica con apertura di un terminale . . . 15
4.2.3 sintassi di un comando . . . 15
4.2.4 il comando `passwd` . . . 16
4.2.5 utenti e gruppi; `who` e `finger` . . . 17
4.3 manuali . . . 17
4.4 navigazione nel *filesystem* e gestione elementare di file e directory . . . 18
4.4.1 file e directory; directory corrente e directory genitrice; `ls`, `pwd` e `cd` . . . 18
4.4.2 percorso assoluto e relativo . . . 19
4.4.3 le directory ".", ".." e "~" ; . . . 20
4.4.4 permessi di lettura, scrittura ed esecuzione dei file; `ls -l` . . . 21
4.4.5 nomi di file e directory; file e directory nascosti; `ls -a` . . . 22
4.4.6 lista di una directory diversa dalla directory corrente: `ls` ***`dir`*** . . . 23
4.4.7 comandi per gestire file: `cp, mv, rm` . . . 23
4.4.8 comandi per gestire directory: `mkdir` e `rmdir` . . . 23
4.4.9 vedere il contenuto di un file: `cat, more, less` . . . 24
4.4.10 schema riassuntivo: gestione essenziale di file e directory . . . 25
4.5 editor di testo: `vi`, `pico` e `emacs` . . . 26
4.6 proprietà delle shell . . . 27
4.6.1 reindirizzamento di input e output, pipeline . . . 27
4.6.2 esecuzione di comandi in serie, parallelo e *background*; `ps` e `jobs` . . . 27
4.6.3 interruzione di processi . . . 28
4.6.4 sostituzioni: nomi di file . . . 29
4.6.5 altre sostituzioni . . . 29
4.6.6 proteggere da sostituzione . . . 30
4.6.7 esempi . . . 30
4.6.8 schema riassuntivo delle proprietà della shell . . . 32
4.7 lista di comandi utili . . . 33
4.8 `vi` . . . 34

5 Uso "avanzato" di UNIX **36**
5.1 Regular Expression (espressioni regolari o formali) . . . 36
5.2 awk . . . 37
5.2.1 caratteristiche generali . . . 37
5.2.2 manuali . . . 37
5.2.3 sintassi del comando awk . . . 38
5.2.4 istruzioni di awk . . . 38
5.2.5 Campi . . . 38
5.2.6 Variabili . . . 38
5.2.7 Sequenze (*pattern*) . . . 39
5.2.8 azioni . . . 39
5.2.9 un semplice esempio . . . 39

6 APPLICAZIONI SU UNIX **40**
6.1 gnuplot . . . 40
6.2 grafica molecolare: jmol . . . 41
6.3 esercizi . . . 41

7 Note **41**
7.1 nota: la tastiera . . . 41

1 Introduzione e obiettivi

UNIX è un sistema operativo, cioè un ambiente di lavoro su computer, alternativo a Windows.

È molto utilizzato in campo tecnico-scientifico, e particolarmente adatto per macchine che devono fare da *server* (es. nei siti WWW: vedi `http://leb.net/hzo/ioscount/` oppure `http://econ.csuchico.edu/~harrison/k12linux/`).

A differenza di Windows, UNIX non è un prodotto commerciale; è più simile ad uno standard, o in senso lato ad un linguaggio. Esitono molti "dialetti" UNIX: il più diffuso è 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.

1.1 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 Linux. Un ottimo manuale introduttivo è [Ricar(1999)]; più completo è [Petersen(1998)]. [Attivissimo(2000)] è una guida pratica *on-line* alla migrazione da Windows a 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)].

2 Sistemi Operativi

2.1 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. Più che un programma, è un insieme di programmi di uso generale che

- permette 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
  - ...
- interagisce con
  - utenti
  - programmi applicativi

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. Linux.

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

A sua volta il S.O. è distinto in vari strati

- kernel (nucleo di comandi fondamentali a cui si può rivolgere direttamente un programma, ma di solito non l'utente)
- ...
- interfacce utente: testuali (es shell) e grafiche (GUI)

2.1.1 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.

2.2 Interfacce utente

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) è l'*interfaccia*. 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.

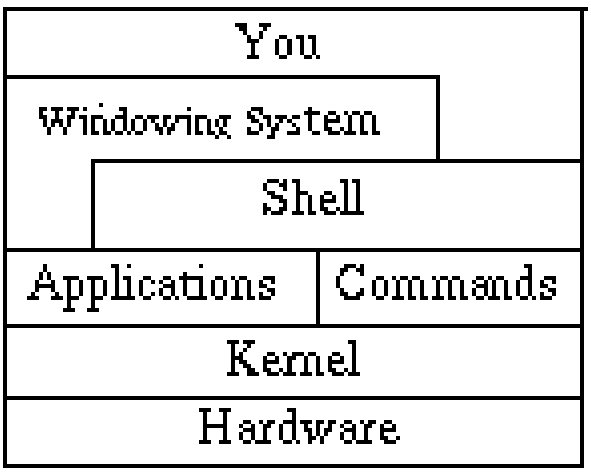
interfaccia grafica (GUI): per compiere operazioni sul sistema si opera, principalmente con il mouse, su elementi grafici come menù, icone, finestre. Quasi sempre l'intefaccia grafica contiene, tra i vari strumenti, anche un terminale. GUI sta per Graphical User Interface (Interfaccia Utente Grafica)

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.

Figura 1: Rappresentazione schematica degli strati di *software* in UNIX. Notare che il sistema operativo propriamente detto consiste nel kernel, nei comandi/applicazioni e nella shell. L'interfaccia grafica ed i programmi utente (che stanno concettualmente accanto alle applicazioni di sistema) sono installati "sopra" il sistema operativo.



2.2.1 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

2.3 Principali caratteristiche di un Sistema Operativo

2.3.1 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.

2.3.2 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).

2.3.3 diritti di proprietà e di uso

Come tutto il software, anche il sistema operativo può essere libero o proprietario. Ci sono molti modi di concepire questa distinzione, ma il più utile e accreditato è il seguente:

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 una casa.

2.4 Approfondimento: software libero e *open source*

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.

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*).

2.4.1 definizione di software libero

Secondo la definizione più diffusa, 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 una discussione 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. In particolare:

- Notare la differenza tra software libero e:

  software di dominio pubblico = 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 tempo limitato

- 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

2.4.2 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.

Dibattito:

- 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 che 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, UNIX) 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 anche la pubblica amministrazione comincia a raccomandare l'uso dell'*Open Source* e dei formati aperti [Ministero dell'Innovazione(2002)]

2.5 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 2: 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: `a0100131...`

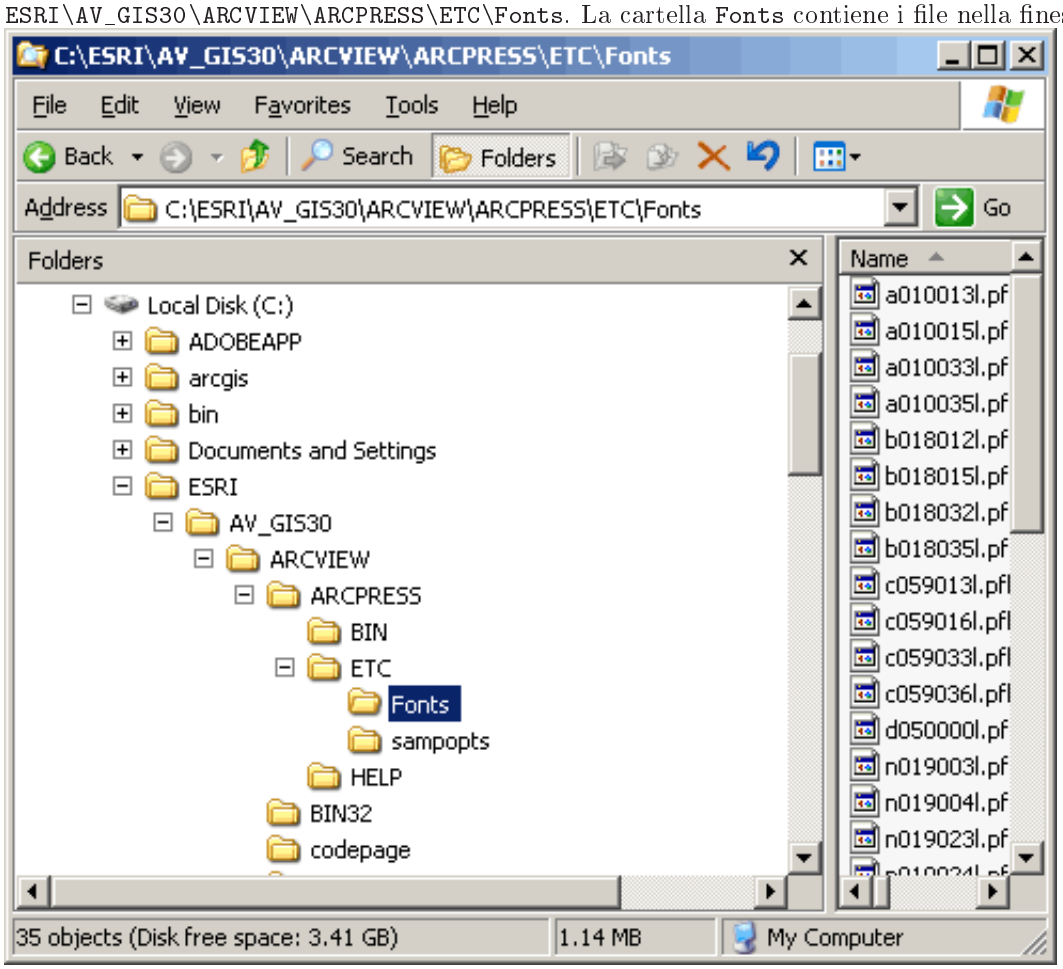
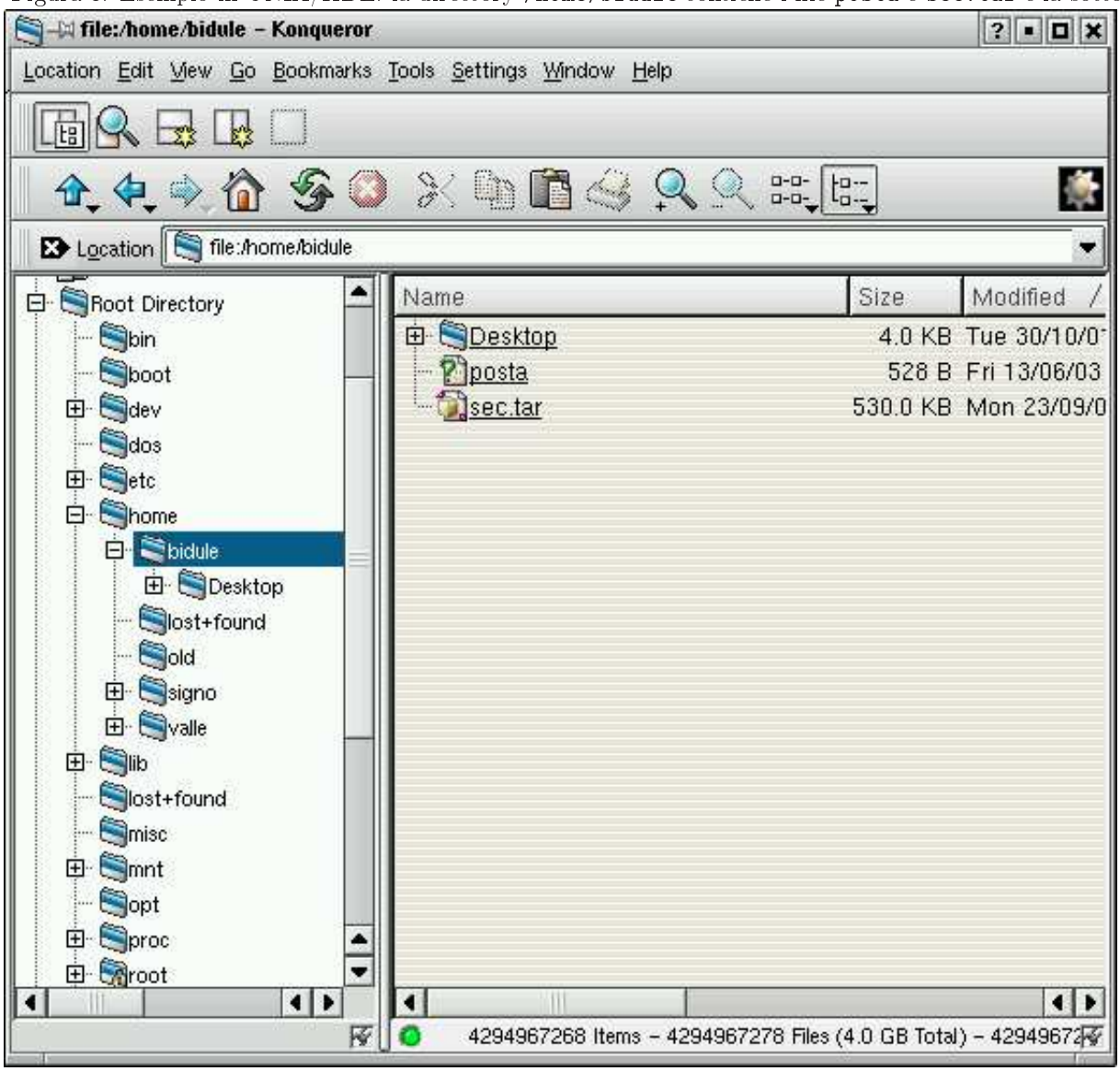


Figura 3: Esempio in UNIX/KDE: la directory `/home/bidule` contiene i file `posta` e `sec.tar` e la sottodirectory `Desktop`



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 UNIX

3.1 Caratteristiche di UNIX

Caratteristiche di UNIX rispetto ad altri Sistemi Operativi:

- multi-utente
- multi-processo
- ci sono tante versioni, molte libere
- UNIX *è* solo l'insieme di

  - kernel
  - comandi e applicazioni di sistema (es. `ls`, `awk`, `vi`)
  - shell

  cioè ciò che si usa da un'interfaccia testuale, o terminale. Però praticamente sempre contiene anche

  - GUI (X-windows e W.M/Desktop)
  - connettività in rete (TCP/IP)
  - funzionalità di server (HTTP, posta elettronica, FTP, ...)

- "filosofia" = attrezzi (efficienti), combinabili

3.2 vantaggi e svantaggi di UNIX su Windows

cfr [Attivissimo(2000), capitolo 3]

- vantaggi

  versatilità: funziona su diversi hardware, non solo su PC

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

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

  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"; 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

3.3 UNIX come famiglia di sistemi operativi, liberi e non

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

- UNIX commerciali (o della famiglia *System V):* (generalmente contengono almeno una parte dell'originale sorgente AT&T, e la relativa licenza)
- UNIX (più o meno) liberi (famiglia *BSD*): FreeBSD, NetBSD, ... ; quelli 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. Esiste uno standard "puro": POSIX. Praticamente, un sottoinsieme di UNIX®

   POSIX (Portable Operating System Interface):
   A set of IEEE standards designed to provide application portability between Unix variants. IEEE 1003.1 defines a Unix-like operating system interface, IEEE 1003.2 defines the shell and utilities and IEEE 1003.4 defines real-time extensions.

3. Altro standard: *Single UNIX Specification*, prodotto da Open Group. È derivato pesantemente da POSIX, ma attualmente (2001) è più diffuso, principalmente perché i documenti sono più facili da ottenere. (!)

3.3.1 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.

3.4 Linux

Linux è un Sistema Operativo della famiglia UNIX con le seguenti caratteristiche:

- libero (GPL)
- POSIX

"distribuzione" Linux: Linux kernel + comandi, shell + TCP/IP + X11 + Desktop Manager + ...

Sono a pagamento: (eventualmente) qualche "abbellimento" software non essenziale, la commercializzazione, la guida stampata, i CD.
Esempi di distribuzioni Linux:

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

Nelle esercitazioni useremo RedHat 7.2.

3.5 GNU

GNU è un insieme di strumenti di software libero (comandi, shell, applicazioni) da usare "sopra" un kernel UNIX: GNU/Hurd (Sistema Operativo composto dal kernel Hurd e da GNU), GNU/Linux

È un "progetto" creato dalla Free Software Foundation (FSF)

> 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).

4 Uso elementare di UNIX

NOTA: i comandi che appaiono sottolineati, come `less`*, non sono UNIX, ma estensioni GNU (o altro) che comunque si trovano nella maggior parte delle distribuzioni di 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

Molti sistemi UNIX si presentano alla partenza con una maschera grafica in cui oltre a immettere nome utente e password si può scegliere il tipo di interfaccia utente (window manager, desktop, o interfaccia testuale) da far partire. Se si sceglie un'interfaccia grafica spesso tra le altre cose viene aperto automaticamente un terminale; altrimenti lo si può aprire usando il menù principale.

Dal terminale si esce con `exit`. Per uscire dalla sessione 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 grafica di login. Il sistema è ancora accesso, in attesa di nuovi login. Per spengere il sistema si sceglie l'opzione Shutdown.

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:

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

```
date
date -u
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 manuali

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:

`man` *`comando`*	guida esauriente su *`comando`* (qualche pagina)
`whatis` *`comando`*	definizione di *`comando`* (una riga)
`apropos` *`xxx`*	lista di comandi che hanno a che vedere con l'argomento *xxx*
`info` *`menu`*	manuali specializzati, in forma ipertestuale

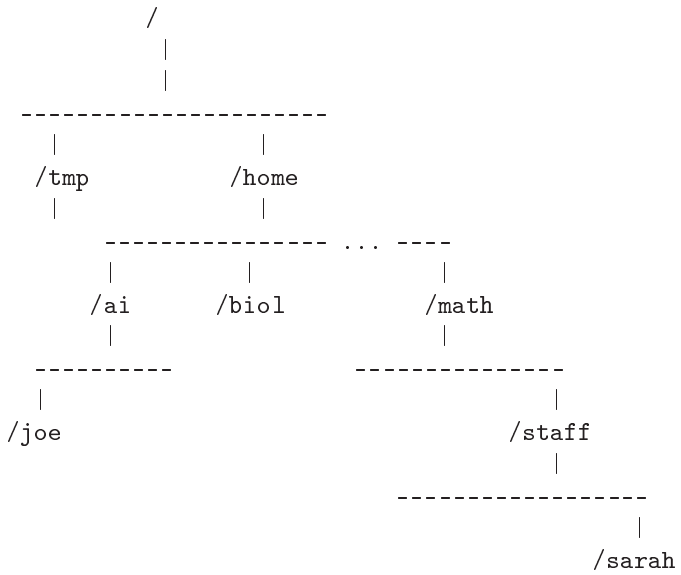
Es:

```
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 genitrice; `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 "genitrice" (=sopra la corrente, cioè all'indietro 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`

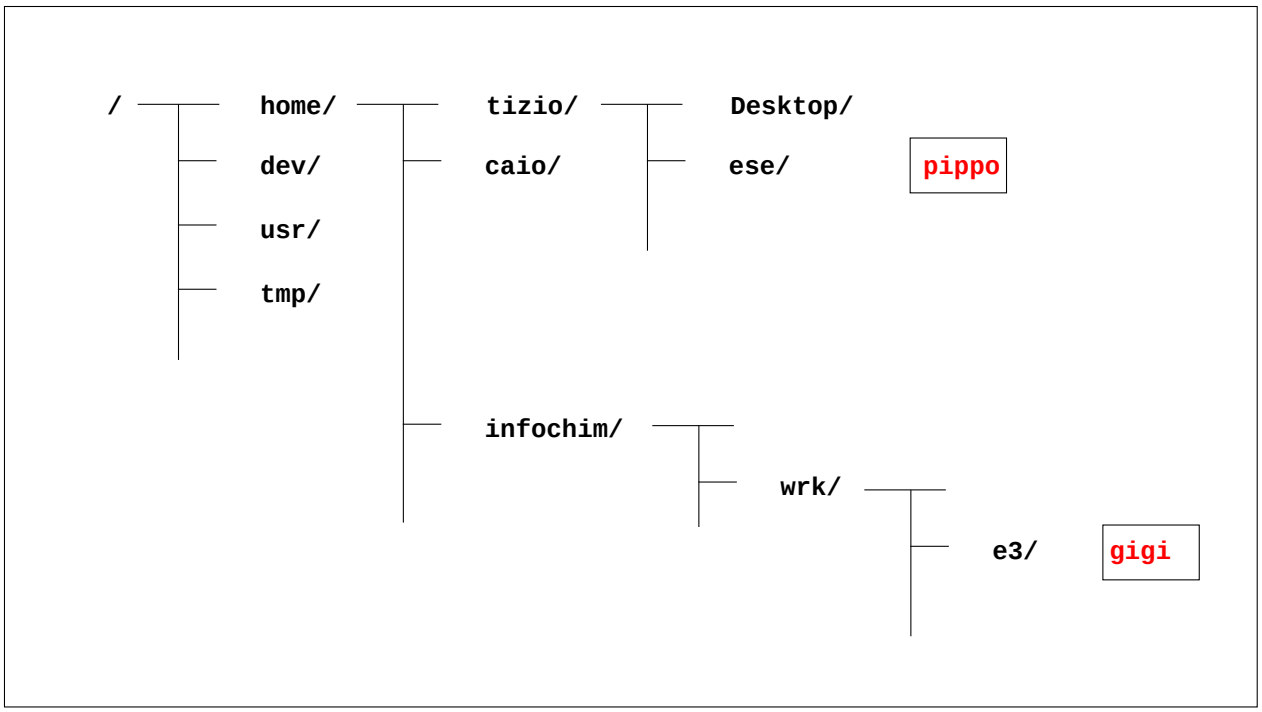


Figura 4: esempio di percorsi

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 "genitrice" (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)

Notare il comportamento di `cp` e `mv`:

- 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*, ...	

4.4.8 comandi 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

4.4.9 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
  ```

4.4.10 schema riassuntivo: gestione essenziale di file e directory

I parametri tra [] sono opzionali.

`.`	la directory corrente
`..`	la directory "genitrice" (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 ...***
`cat` ***file***	visualizza ***file***
`more` ***file***	visualizza ***file***, una schermata alla volta
`less` ***file***	visualizza ***file***, con possibilità di scorrere indietro
`cp` ***file1 file2***	copia ***file1*** in ***file2***
`cp` ***file1 dir***	copia ***file1*** in ***dir/file1***
`cp` ***file1 file2 ... dir***	copia ***file1, file2, ...*** in ***dir/file1, dir/file2, ...***
`mv` ***file1 file2***	rinomina ***file1*** in ***file2***
`mv` ***file1 dir***	rinomina ***file1*** in ***dir/file1*** (lo *muove* in ***dir***)
`mv` ***file1 file2 ... dir***	muove ***file1, file2, ...*** in ***dir***
`rm` ***file1 file2 ...***	rimuove (cancella) ***file1 file2 ...***
`mkdir` ***dir***	crea la directory ***dir***
`rmdir` ***dir***	cancella la directory *vuota* ***dir***

4.5 editor di testo: `vi`, `pico` e `emacs`

editor di testo: sono programmi per editare file di testo non formattati; ad es. programmi, semplici liste, etc

I documenti formattati in genere contengono istruzioni di formattazione (es: testo in grassetto, allineamento, tabelle), codificate dal programma che li ha creati (es. MS Word) e possono essere modificati solo dallo stesso programma. Eccezione: documenti in HTML, le istruzioni di formattazione sono scritte in chiaro.

- `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. Vantaggio:
  - fa parte del S. O. , 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:

`pico` *file*

4.6 proprietà delle shell

queste sono proprietà per lo più comuni a tutte le shell

4.6.1 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 così:

standard input	tastiera
standard output	video

Si possono associare input e output a file normali 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:

```
date                          stampa la data e l'ora
date > a.000                  mette il risultato nel file a.000
wc -w < a.000                 conta le parole nel file a.000
```

Gli ultimi due comandi possono essere combinati in una *pipeline*, senza passare attraverso il file `a.000`:

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

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

```
comando1 | comando2 | ...  | comando99
```

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

```
comando1 < file1 | comando2 | ...  | comando99 > file99
```

4.6.2 esecuzione di comandi in serie, parallelo e *background*; `ps` e `jobs`

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:

esecuzione in serie: per eseguire due comandi in successione (o serie):

comando1 ; *comando2*

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

Esempio:

```
$ date; ls -l; date
mar dic  9 14:29:03 CET 2003
totale 924
-rw-r--r--    1 infochim chimica       160  4 dic  2002 elenco.txt
drwxr-xr-x    2 infochim chimica      4096 18 dic  2002 pdb
-rw-r--r--    1 infochim chimica    163840 10 dic  2002 pdbs.tar
drwxr-xr-x    2 infochim chimica      4096 11 dic  2002 sost
-rw-r--r--    1 infochim chimica    254889 10 dic  2002 telefonico.txt
mar dic  9 14:29:03 CET 2003
$
```

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*

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
- 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
```

4.6.3 interruzione di processi

Ctrl-C arresta definitivamente il comando che si sta eseguendo

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

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

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

`kill` *numero-processo*

oppure

`kill` *%numero-job*

4.6.4 sostituzioni: nomi di file

Nei nomi di file 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: differenza da 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
   ```

4.6.5 altre sostituzioni

Oltre a quella sui nomi di file, 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*
`* ? []`	nomi di file	`mv capitolo*.txt ../libro`
`~`	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"`

4.6.6 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
```

4.6.7 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`

4.6.8 schema riassuntivo delle 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)
~	directory di partenza (*home directory*) di questo utente
~*xxx*	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 \

4.7 lista di comandi utili

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

guide

```
man                 - formatta e mostra le pagine di guida in linea
apropos             - ricerca stringhe nel database di whatis
whatis              - ricerca il database whatis per una parola completa
```

file, directory, dischi

```
ls                  - visualizza i contenuti delle directory
rm                  - rimuove file o directory
mv                  - rinomina file
cp                  - copia file e directory
cd                  - change working directory
pwd                 - stampa il nome della directory di lavoro corrente
mkdir               - crea directory
rmdir               - rimuove directory vuote
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

```
cat                 - concatena file e li stampa sullo standard output
more                - file perusal filter for crt viewing
less                - l'opposto di more
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

```
lpr                 - stampa off line
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
```

4.8 `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

5 Uso "avanzato" di UNIX

5.1 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:

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`".

5.2 awk

5.2.1 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*

5.2.2 manuali

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

5.2.3 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)

5.2.4 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

5.2.5 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.

5.2.6 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)

5.2.7 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

5.2.8 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()`

5.2.9 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
```

6 APPLICAZIONI SU UNIX

6.1 gnuplot

- È un programma che serve per diagrammare
  - file di dati
  - oppure funzioni
- Si lancia con '`gnuplot`'
- Il comando principale è '`plot`'

per 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 [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

per diagrammare una funzione:

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

- si possono definire costanti e funzioni
- Per uscire: `quit`
- Per guida: `help`

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.)
```

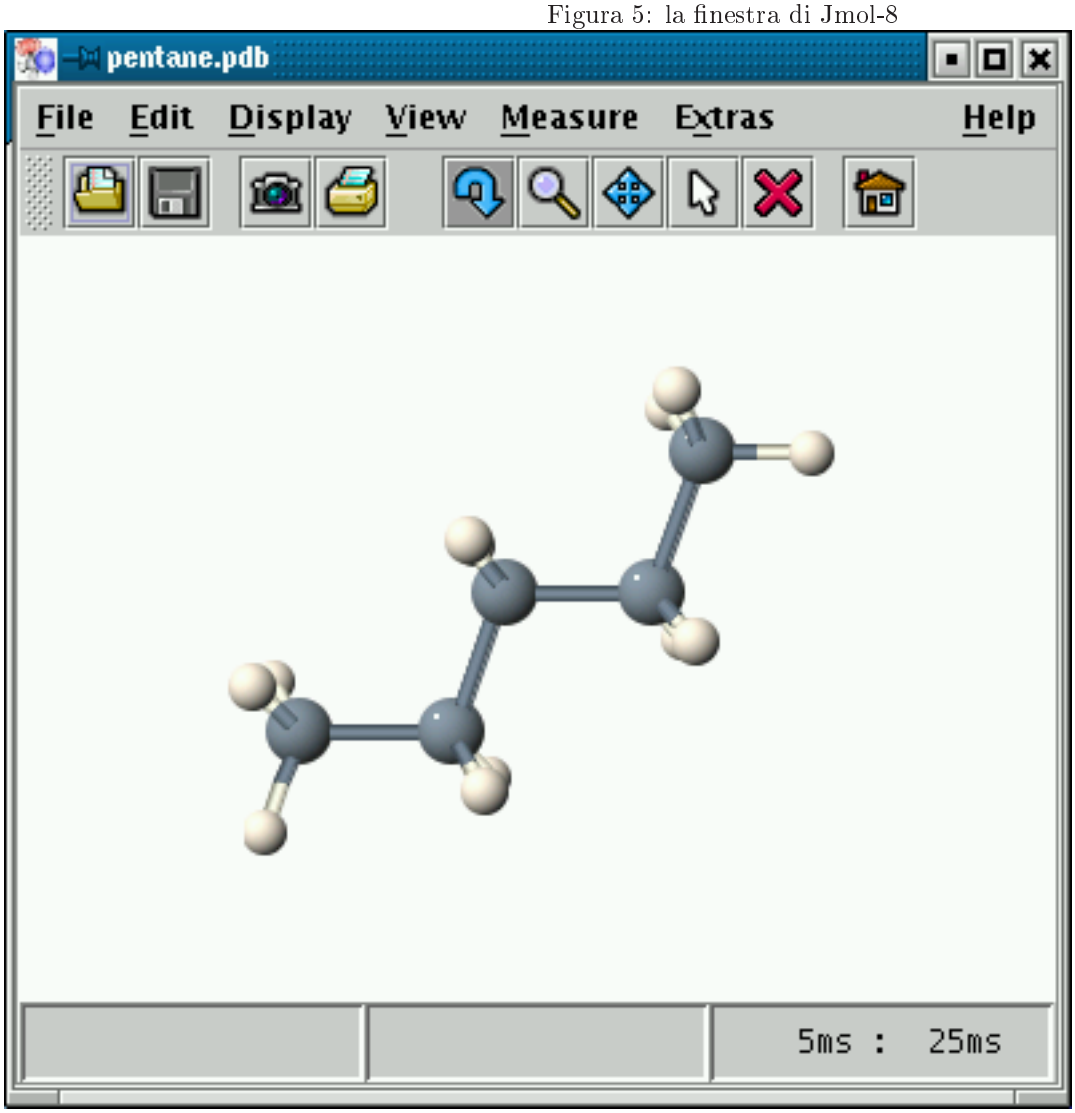
6.2 grafica molecolare: jmol

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 5: la finestra di Jmol-8



6.3 esercizi

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 ì
{	AltGr 7
}	AltGr =
‘	AltGr ’

Riferimenti bibliografici

[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.

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

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

[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.

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

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

[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_commi%ssione_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_sourc%e.shtml`.