ABILITÀ INFORMATICHE PER CHIMICI (2002/2003)
Introduzione a UNIX

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

18 dicembre 2002

1 Organizzazione del modulo

	LU	MA	ME	GI	VE
8:30-10:00				**2, 3**	
10:30-12:30			**lezione**		
14:00-15:30				**3, 1**	
15:30-17:00				**1, 2**	

lezione	**Esercitazione**
4/12/2002	5/12
11/12/2002	12/12

Indice

1 Organizzazione del modulo **1**

2 Argomenti generali **4**
2.1 Introduzione e obiettivi . . . 4
2.2 Sistemi Operativi . . . 4
2.2.1 S. O. : grafici esplicativi, definizione, esempi . . . 4
2.3 Interfacce utente . . . 6
2.4 Caratteristiche di UNIX . . . 6
2.5 vantaggi e svantaggi di UNIX su Windows . . . 6
2.6 sw libero . . . 7
2.6.1 definizione di software libero . . . 7
2.7 UNIX come famiglia di sistemi operativi, liberi e non . . . 7
2.7.1 breve storia di UNIX . . . 8
2.8 Linux . . . 9
2.9 GNU . . . 9

3 UNIX: nozioni fondamentali **10**
3.1 navigazione nel *filesystem* e gestione elementare di file e directory . . . 10
3.1.1 `login`, `logout` e `exit`; sintassi di un comando; il comando `passwd` . . . 10
3.1.2 utenti e gruppi; `who` e `finger` . . . 11
3.1.3 file e directory; directory corrente e directory genitrice;`ls`, `pwd` e `cd` . . . 12
3.1.4 percorso assoluto e relativo . . . 13
3.1.5 le directory ".", ".." e "~" ; . . . 14
3.1.6 lista di una directory diversa dalla directory corrente: `ls` ***dir*** . . . 15
3.1.7 nomi di file e directory; file e directory nascosti; `ls -a` . . . 15
3.1.8 permessi di lettura, scrittura ed esecuzione dei file; `ls -l` . . . 16
3.1.9 comandi per gestire file: `cp, mv, rm` . . . 16
3.1.10 comandi per gestire directory: `mkdir` e `rmdir` . . . 17
3.1.11 vedere il contenuto di un file: `cat, more, less` . . . 17
3.1.12 cambiare il contenuto di un file: gli editor di testo `vi` e `emacs` . . . 17
3.1.13 manuali . . . 17
3.2 proprietà delle shell . . . 18
3.2.1 reindirizzamento di input e output, pipeline . . . 18
3.2.2 esecuzione di comandi in serie, parallelo e *background*; `ps` e `jobs` . . . 18
3.2.3 interruzione di processi . . . 19
3.2.4 sostituzioni: nomi di file . . . 19
3.2.5 altre sostituzioni . . . 20
3.2.6 proteggere da sostituzione . . . 20
3.2.7 esempi . . . 21
3.2.8 schema riassuntivo delle proprietà della shell . . . 22
3.3 lista di comandi utili . . . 23
3.4 vi . . . 24

4 UNIX: Approfondimenti **26**
4.1 Regular Expression (espressioni regolari o formali) . . . 26
4.2 awk . . . 27
4.2.1 caratteristiche generali . . . 27
4.2.2 manuali . . . 27
4.2.3 sintassi del comando awk . . . 27
4.2.4 istruzioni di awk . . . 27
4.2.5 Campi . . . 28
4.2.6 Variabili . . . 28
4.2.7 Sequenze (*pattern*) . . . 28
4.2.8 azioni . . . 28
4.2.9 un semplice esempio . . . 29

5 APPLICAZIONI SU UNIX **29**
5.1 gnuplot 29
5.2 grafica molecolare: jmol (openscience) 30
5.3 esercizi 30

6 Note **30**
6.1 nota: la tastiera 30

2 Argomenti generali

2.1 Introduzione e obiettivi

UNIX è un ambiente di lavoro alternativo a Windows. Serve per compiere le principali operazioni su un computer e eseguire programmi.

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; e più simile ad una lingua. 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.

2.2 Sistemi Operativi

Il S.O. è il s/w di basso livello e di uso generale che

- permette di gestire tutte le risorse del calcolatore:
  - dischi e spazio disco
  - CPU e esecuzione programmi
  - dispositivi di ingresso/uscita
  - periferiche
  - ...
- interagisce con
  - utenti
  - programmi applicativi

Notare bene la differenza tra SO e programmi applicativi. Ad es: OpenOffice.org è un programma applicativo che funziona sia su Windows che su Linux.

A sua volta il SO è distinto in vari strati

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

2.2.1 S. O. : grafici esplicativi, definizione, esempi

[Free OnLine Dic Of Computing]:
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.

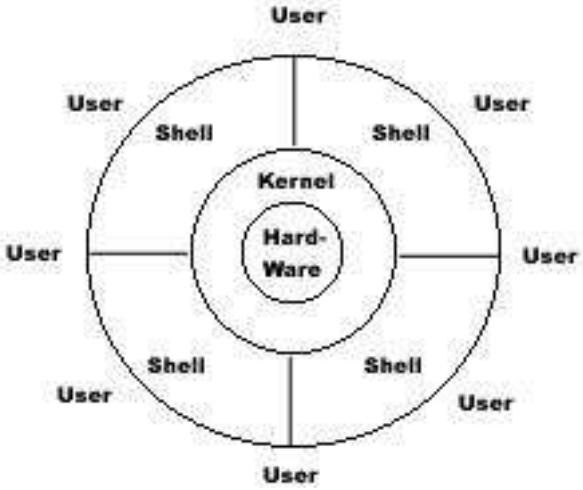
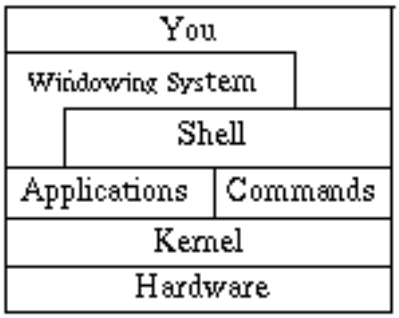


Figura 1: vari strati di *software* tra utenti e *hardware* in un tipico ambiente UNIX

Figura 2: un'altra rappresentazione degli strati di software in UNIX



2.3 Interfacce utente

Servono per permettere all'utente di svolgere operazioni comuni, ad es., spostare un file da una directory all'altra o lanciare un programma. In UNIX si hanno

- interfaccia testuale (sempre):
  - si ha a disposizione una riga dove si possono scrivere comandi.
  - 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 SO vero e proprio; in uno stesso UNIX si possono usare diverse shell con caratteristiche leggermente diverse tra loro
  - sullo schermo compaiono eventuali risultati (output) del comando in formato testo
- interfaccia grafica (componente *non essenziale*):
  - X-windows (solo gestione della parte formale: cornici, bottoni, menu)
  - Scrivania (menu avvio, barre, strumenti tipo file manager)
  - contiene un terminale dove si può lavorare con interfaccia testuale; dal terminale si possono lanciare comandi che aprono nuove finestre

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. Noi lavoreremo con quest'ultima (il *terminale*), all' interno di quella grafica.

- vantaggi dell'interfaccia testuale
  - si hanno a disposizione molti più comandi di quanti non possano contenerne i menu
  - si possono combinare più comandi in una "frase" .

  Ad es:

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

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

2.4 Caratteristiche di UNIX

Caratteristiche di UNIX rispetto ad altri Sistemi Operativi:

- multi-utente
- multi-processo
- "filosofia" = attrezzi (efficienti), combinabili

Insieme alle componenti fondamentali del sistema operativo, in UNIX si trovano sempre

- connettività in rete (TCP/IP)
- ambiente grafico (X-window system)

2.5 vantaggi e svantaggi di UNIX su Windows

cfr `http://www.attivissimo.net/w2l1/03.htm`

- vantaggi

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

  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

  minore offerta di programmi: non adatto se dovete assolutamente usare un programma in particolare

  aggiornamento hardware: è difficile far funzionare un componente hardware di recente produzione

2.6 sw libero

In genere, si intende s/w libero = senza diritti d'autore (si può copiare legalmente)
Dibattito:

- alcuni pensano: siccome gli autori non possono guadagnarci, il s/w libero è di bassa qualità
- secondo altri: se gli strumenti intellettuali non sono coperti da diritto d'autore (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

Di fatto:

- 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 s/w (Mozilla, OpenOffice.Org)

Aspetto fondamentale: per favorire lo sviluppo, non è tanto importante che il s/w sia gratuito, quanto che sia modificabile (codice sorgente aperto)

2.6.1 definizione di software libero

(Basato su [AIL] e http://www.gnu.org/philosophy/free-sw.html)
Secondo la definizione più diffusa, è "libero" il s/w che permette a chiunque di:

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

Altre definizioni:

s/w di dominio pubblico = senza copyright né licenza (qualcuno può "appropriarsene")

copyleft = licenza che garantisce diritti di s/w libero, ma impone di mantenerlo libero (controesempio: X11, è distribuito con licenza libera, ma non impedisce restrizioni- alcune versioni sono a pagamento e sono le uniche a funzionare su un certo h/w)

GNU Public License (GPL) = particolare licenza con *copyleft* che permette anche l'uso commerciale del s/w

s/w semi-libero = s/w libero, ma senza permesso di trarne profitto

freeware = s/w gratuito, spesso senza sorgente

shareware = s/w gratuito per tempo limitato

2.7 UNIX come famiglia di sistemi operativi, liberi e non

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

- UNIX commerciali (generalmente contengono almeno una parte dell'originale sorgente AT&T, e la relativa licenza)
- UNIX (più o meno) liberi: FreeBSD, NetBSD, ... (quelli con GPL sono: Linux, Hurd)

Note:

1. UNIX è un marchio registrato

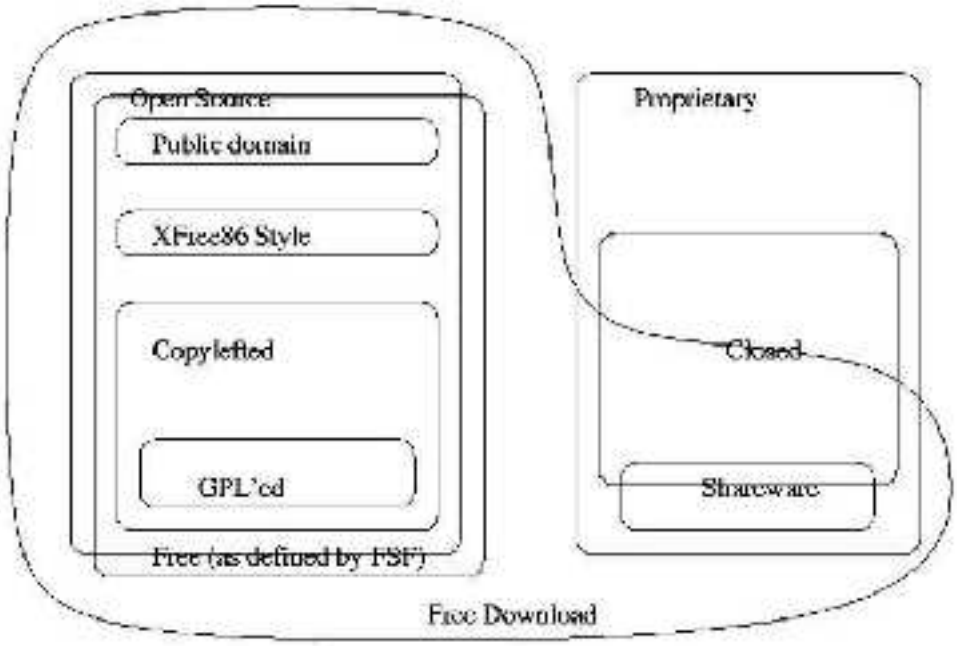
   "UNIX® is a registered trademark of The Open Group""

   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.

Figura 3: vari tipi di software libero



2.7.1 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.

> [AIL]
> 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.

2.8 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" s/w non essenziale, la commercializzazione, la guida stampata, i CD.

Esempi di distribuzioni Linux:

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

Nelle esercitazioni useremo RedHat 7.2.

2.9 GNU

GNU è un insieme di strumenti di s/w 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 [AIL]:
>
> 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 UNIX: nozioni fondamentali

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

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

3.1.1 `login`, `logout` e `exit`; sintassi di un comando; il comando `passwd`

- maiuscole e minuscole sono diverse!
- la shell: invito ai comandi (*prompt*)
- 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 cominiciano 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
  grep Rossi elenco.txt
  grep -i Rossi elenco.txt
  tar --help
  ```

- un comando importante: `passwd` (oppure `yppasswd`)
- `logout` e `exit`

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

3.1.2 utenti e gruppi; `who` e `finger`

- 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

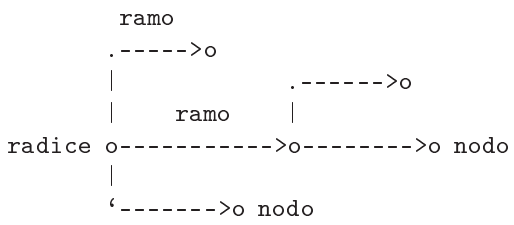
Esempi:

- `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

3.1.3 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



  ogni nodo può essere terminale (file) o ramificato (directory);
  in altre parole, 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 ..
  ```

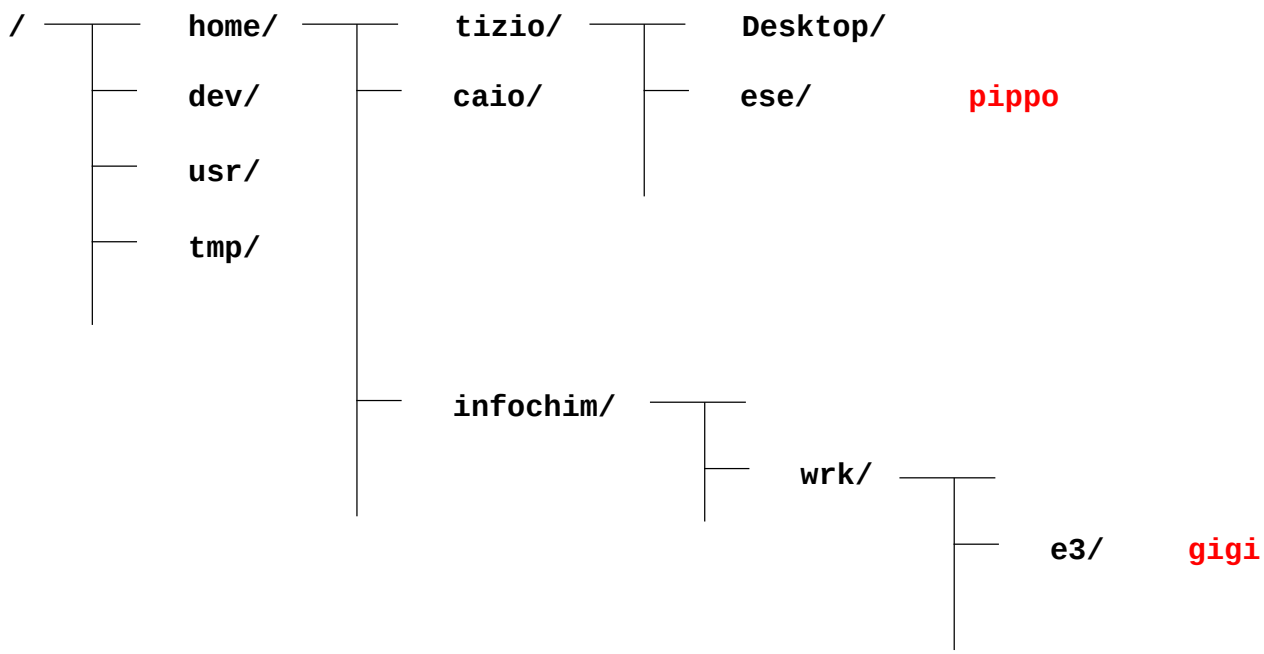


Figura 4: esempio di percorsi

3.1.4 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
   ```

3.1.5 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`

Esempi:

directory corrente	*nome del file*
(qualunque)	`/home/tizio/ese/pippo`
`/home/tizio/ese/`	`pippo`
`/home/tizio/`	`ese/pippo`
`/home/tizio/ese/`	`./pippo`
`/home/caio`	`../tizio/ese/pippo`
(qualunque; utente= ***tizio****)*	`~/ese/pippo`
(qualunque; utente= ***caio****)*	`~tizio/ese/pippo`

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.

3.1.6 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 .
   ```

3.1.7 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, `/~!@#$%^&*([{}])<>'` etc. Caratteri permessi: `ABC...Z, abc...z` "_", "-" "."
  - 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
- il carattere iniziale ".": directory e file "nascosti";
  si possono vedere con l'opzione `-a` di `ls`:

  ```
  ls -a
  ```

3.1.8 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.

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
```

3.1.9 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*
`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, ...`

3.1.10 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

3.1.11 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
  ```

3.1.12 cambiare il contenuto di un file: gli editor di testo `vi` e `emacs`

- editor di testo: sono adatti per editare file di testo non formattati, come programmi, semplici liste, etc; documenti formattati in genere contengono istruzioni di formattazione codificate (es. MS Word) e possono essere modificati solo dal programma che li ha creati o da uno compatibile. Controesempio: documenti in HTML, le istruzioni di formattazione sono scritte in chiaro

- `vi` è un editor di testo rudimentale, scomodo da usare. Vantaggio:

  - fa parte del S. O. , lo trovate sempre
  - può essere usato in condizioni di ristrettezza di risorse (es. collegamento remoto)

- `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:

  ```
  emacs file
  ```

3.1.13 manuali

`man` *`comando`*	guida esauriente su comando (qualche pagina)
`whatis` *`comando`*	definizione (una riga)
`apropos` *`xxx`*	lista di comandi che hanno a che vedere con l'argomento xxx

Es:

```
man passwd; whatis ls; apropos calendar
```

3.2 proprietà delle shell

queste sono proprietà per lo più comuni a tutte le shell

3.2.1 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
```

3.2.2 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*

esecuzione in parallelo: per eseguire due comandi in contemporanea (o parallelo):

comando1 & *comando2*

Il sistema esegue *comando1* e senza aspettare che sia terminato esegue *comando*2

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`

  ```
  $ cat gigafile | sort &
  [2] 10334
  $ jobs
  [1]  + Running                       emacs
  [2]  - Running                       cat gigafile | sort &
  ```

  Notare che ad ogni comando del job 2 (`cat` 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 cat
  10334 pts/0    00:00:00 sort
  10335 pts/0    00:00:00 ps
  ```

3.2.3 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 in *background* si usa il comando

`kill` *numero-processo*

oppure

`kill` *%numero-job*

3.2.4 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
   ```

3.2.5 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)	`!!`
‘‘	risultato del comando	`echo "Ci sono ‘who | wc -l‘ utenti collegati"`

3.2.6 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
```

3.2.7 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

```
uncompress -c a.pdb.Z               decomprime il file su standard output
uncompress -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

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

4. 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)
```

5. Statistiche sugli utenti del sistema

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

3.2.8 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 \

3.3 lista di comandi utili

guide

```
man     - format and display the on-line manual pages
apropos - search the whatis database for strings
whatis  - search the whatis database for complete words
```

file, directory, dischi

```
ls      - list directory contents
rm      - remove files or directories
mv      - move (rename) files
cp      - copy files and directories
cd      - Change working directory
pwd     - print name of current/working directory
mkdir   - make directories
rmdir   - remove empty directories
chmod   - change file access permissions
chown   - change file owner and group
find    - search for files in a directory hierarchy
df      - report filesystem disk space usage
du      - estimate file space usage
```

manipolazione di file

```
cat     - concatenate files and print on the standard output
more    - file perusal filter for crt viewing
less    - opposite of more
sort    - sort lines of text files
head    - output the first part of files
tail    - output the last part of files
grep    - print lines matching a pattern
wc      - print the number of bytes, words, and lines in files
diff    - find differences between two files
```

manipolazione avanzata di file

```
sed             - a Stream EDitor
gawk [awk]      - pattern scanning and processing language
```

stampa e simile

```
lpr     - off line print
lpq     - spool queue examination program
echo    - display a line of text
```

archiviazione e compressione

```
tar     - The GNU version of the tar archiving utility
compress [uncompress] - compress and expand data
gzip    - compress or expand files
```

utenti, processi

```
passwd   - update a user's authentication tokens(s)
yppasswd - change your password in the NIS database
finger   - user information lookup program
who      - show who is logged on
w        - Show who is logged on and what they are doing
ps       - report process status
kill     - terminate a process
```

3.4 vi

(ottimo manuale: http://www.geek-girl.com/Unixhelp/vi/index.html [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 UNIX: Approfondimenti

4.1 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`".

4.2 awk

4.2.1 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*

4.2.2 manuali

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

4.2.3 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)

4.2.4 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

4.2.5 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.

4.2.6 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)

4.2.7 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

4.2.8 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()`

4.2.9 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
```

5 APPLICAZIONI SU UNIX

5.1 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.)
```

5.2 grafica molecolare: jmol (openscience)

5.3 esercizi

calcolo proprietà di una molecola da geometria:

- dipolo
- centro di massa, momento di inerzia

6 Note

6.1 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 ’

Riferimenti bibliografici

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

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

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

[AIL] Appunti di Informatica Libera:.http://www.pluto.linux.it/ildp/AppuntiLinux/a2.html

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

[UHU] Unix Help for Users: http://www.geek-girl.com/Unixhelp/

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