Matematica e uso dei calcolatori: uso dei calcolatori

Dario Benedetto

8 marzo 2007

Indice

Capitolo 1

Introduzione

1.1 Programma

Le lezioni si svolgono inm laboratorio e consistono in esercitazioni pratiche. L'esame sarà un'esercitazione pratica individuale. Il programma è diviso in due parti: l'introduzione ai sistemi unix–linux e l'introduzione al programma di statistica R.

– Modalitá non grafica: breve descrizione dell'hardware; cenni su bios e il sistema operativo; sistemi multiutente e autenticazione, privilegi degli utente.

– L'albero delle directory sotto unix, la home directory; comandi per la navigazione (pwd, cd, mkdir, rmdir); i nomi simbolici delle directory (. .. ~); cammino assoluto e relativo. I file e le loro proprietà (permessi, data, dimensioni); comandi per la gestione dei file (ls, cp, mv, rm). Come sono scritti i file: bit, byte, cenni sulle codifice ASCII, ISO8859-15, UTF-8. Lettura dei file (cat, more, less); modifica e creazione di file (editori, emacs). Tipi di file, programmi eseguibili. Directory dei programmi. Compressione e archiviazione dei file (gzip, zip, bzip2, tar), redirezione dell'input e dell'output.

– Modalitá grafica: gestori di fineste. I principali formati dei file e i principali programmi per la loro gestione.

– Introduzione al programma di statistica R. Operazioni elementari, variabili, vettori. Creazione di vettori. Selezione avanzata degli elementi di un vettore, opertori logici fondamentali. Comandi per il disegno. Interazione con il sistema: lettura e scrittura di file di dati, scrittura ed esecuzione di programmi. Principali operazioni statistiche, principali distribuzioni di probabilitá.

1.2 Testi

Queste sono le note del corso, tarate sui calcolatori del laboratorio di calcolo del Dipartimento di Matematica. Non sono "precise" né complete, ma rappresentano gli argomenti delle lezioni.

Il **programma R** e tutta la **documentazione** sono liberamente scaricabili da rete, per piattaforme Linux, Windows e MacOS, al sito

http://www.r-project.org/

(clickare su CRAN (sotto Download nella finestra di sinistra), scegliere un sito mirror (cioè un sito in cui ci sia una copia identica), scegliere il sistema operativo e seguire le istruzioni.

Inoltre, è disponibile nelle librerie il manuale **Statistica con R** di S. Iacus e G. Masarotto, edizioni McGrawHill, che contiene anche il CD con il programma installabile, per Windows e McOs.

Testi consigliati per la parte di introduzione a unix

Nella pagina del centro di calcolo del dipartimento http://www.mat.uniroma1.it/centro-calcolo/ in fondo al Menù a sinistra c'è il link Manuali on line. Nella sezione Unix-Linux ci sono vari documenti. Vi segnalo i seguenti.

–Note del corso di Abilità informatiche: Introduzione a Unix, A. Seghini

Sono le note di un corso abbastanza simile al nostro (a parte la programmazione), e per di piú fatto nel nostro stesso laboratorio (A. Seghini è la responsabile del Centro di Calcolo del Dipartimento); inoltre sono scritte meglio di quelle che state leggendo.

–Appunti di informatica libera (http://a2.pluto.it/)

Opera di dimensioni enormi, con tutto lo scibile linux (ci sono anche le introduzioni ai linguaggi C, pascal, perl, python, java, html, mysql e molti altri). Consiglio i capitoli:

Capitolo 6: conversioni di basi numeriche

Capitoli 11 – 27 (a parte il capitolo 23)

–Corso introduttivo all'utilizzo di Unix, FreakNet MediaLab Catania http://medialab.freaknet.org

È scritto in modo molto divertente. I capitoli che corrispondono al nostro corso sono sostanzialmente 1 – 12. Per comodità riporto gli argomenti dei capitoli:

```
 1 introduzione
 2 tastiera login
 3 paragrafo 2: emacs
 4 ls, cp, rm, mv, cat, more, less
 5 mkdir, cd, pwd, find
 6 mount
 7 < > | * ? ln
 8 gzip gunzip tar date cal diff bc
 9 permessi chmod bit byte
10 processi CTRL-z jobs fg bg kill
11 GPL (generic public licence GNU)
12 man
13 who whoami TCP/IP, talk
14 posta
15 rlogin telnet ftp
16 interfacce grafiche
17 navigazione in rete
18 approfondimento sui permessi
19 editore joe
```

–UNIX: introduzione elementare, M. Liverani

Appunti "seri". I capitoli che corrispondono al corso sono 1 – 4. Riporto l'indice.

```
1 introduzione: il filesystem unix
```

```
2 principali comandi: login, pwd, ls, chmod, mkdir, rmdir, cp, mv, rm, df,
  find, cat, file, wc, who, finger, mail, jobs, fg, bg, ps, cat, du, which,
  grep, date, bc, man
3 editor: vi, emacs
4 interfaccia grafica
5 rete: indirizzi IP, ftp, scp
Appendice A: riassunto dei comandi
Appendice B: riassunto delle sigle
```

–Linux facile, D. Medri

Bella introduzione storica su linux e sul free software. Per il resto: c'è la parte di installazione che in realtà non è una cosa veramente facile.

Capitolo 2

Linux

2.1 Accensione

2.1.1 Dopo l'accensione

Dopo l'accensione, il calcolatore è governato da un programma che si chiama BIOS. Esso controlla e rende operativi i vari elementi dell'hardware, in particolare

- il **processore (CPU)**: l'unità di elaborazione
- la **RAM**: la memoria volatile; è lo spazio di lavoro del processore, ci vengono caricati dati e programmi
- il **disco rigido**: apparato di memorizzazione fisso, cioè l'apparato su cui sono scritti tutti i programmi, tutti i file degli utenti, etc...
- la **tastiera**: apparato di input, che permette di inviare le istruzioni al computer
- lo **schermo**: apparato di output, su cui il calcolatore scrive i messaggi per l'utente
- apparati di **memorizzazione esterna** ad esempio floppy, lettori DVD-CD, masterizzatori, memorie flash, collegati in vari modi: sono apparati di memorizzazione che possono essere attaccati o staccati dal computer (al contrario del disco rigido)
- altri apparati di input: mouse o strumenti per i giochi
- altri apparati di output: stampanti
- apparati di comunicazione: schede di rete, modem

In generale il BIOS controlla tutti gli elementi hardware e i modi di comunicazione tra di loro.

2.1.2 il `boot`: scelta del sistema operativo

Su alcuni calcolatori è disponibile più di un sistema operativo. Il sistema operativo è il programma principale che è in un calcolatore. Esso governa il modo con cui vengono scritti e letti i file su disco, i modi con cui avvengono ad alto livello le comunicazioni tra i vari elementi hardware, il modo con cui il processore compie le sue operazioni etc... Un sistema operativo viene fornito

insieme a molti programmi, che NON fanno parte del sistema operativo, ma sono concepiti per lavorare con quel tale sistema operativo.

In genere due sistemi operativi risiedono, con i loro programmi e con il loro spazio per gli utenti, in due porzioni diverse del disco (dette **partizioni**), che sono anche **formattate** in modo differente (cioè le informazioni sono scritte e indicizzate in modo differente a seconda del sistema operativo).
Nel nostro caso ci sono due scelte `win` che sta per Windows e `linux`, che è il sistema con cui opereremo.

Dunque digitando `linux` dopo `boot:` e premendo INVIO, il calcolatore carica il sistema operativo linux.

Le informazioni che appaiono successivamente sono il riassunto delle operazioni di caricamento del sistema operativo. In particolare viene caricato su RAM ed entra in esecuzione il **kernel** (cioè il sistema operativo vero e proprio), poi vengono riconosciuti gli apparati hardware resi disponibili dal BIOS (processore, dischi, etc...), e vengono attivati i modi di comunicazione con altri elementi hardware e vengono attivati i cosiddetti **servizi**, cioè programmi che offrono servizi all'utente che ha acceso il calcolatore o agli utenti che possono raggiungere il calcolatore via rete.

2.1.3 Utenti ed autenticazione

Alla fine della sequenza di boot, il sistema entra in **modalità grafica**. Appare la **finestra di login** in cui viene richiesto, per accedere al computer, di inserire un **nome di identificazione** (detto spesso **userame** o nome dell'utente). Solo le username registrate nel calcolatore sono valide; questo vuol dire che esiste ed è attivo l' **account** di quell'utente. Successivamente viene richiesta una **password** di identificazione, cioè una parola segreta che dovrebbe conosce solo quell'utente, e senza la quale il login fallisce. La password è uno degli strumenti necessari per la sicurezza.

Un sistema linux è strutturato per essere un sistema **multiutente**. Cioè è previsto che più utenti possano utilizzarlo (anche contemporaneamente), ma tenendo ben distinti gli oggetti e gli spazi che i diversi utenti posso usare. In particolare viene impedito, tra l'altro, che un utente possa accedere agli oggetti di un altro utente. Questa caratteristica si basa sul concetto di **privilegi** che sono associati ad ogni utente, e di **permessi** che sono associati a ciascun oggetto (dati, programmi). In altre parole, per ogni coppia utente-oggetto ci sono operazioni permesse ed operazioni proibite. Qui proibito non significa che non si deve fare, ma che non si può fare.

In generale, un utente normale **non può modificare** la configurazione del sistema (es: l'ora, il nome della macchina, i servizi offerti, l'esistenza degli altri utenti, i loro privilegi), nè può modificare gli oggetti degli altri utenti, o modificare le loro passowrd. L'utente che ha tutti questi privilegi è l'amministratore di sistema, la cui username è "root".

2.1.4 Modalità non grafica

Per uscire dalla modalità grafica, premere contemporaneamente

`CTRL-alt-F1`

(o anche F2, F3 etc...); a questo punto lo schermo diventa nero (abbandona la modalità grafica) e compare una finestra NON grafica per il login. Con la combinazione CTRL-alt-F1, abbiamo

avuto accesso al **primo terminale virtuale**; i terminali virtuali vanno da F1 a F6; F7 è il primo terminale grafico.

L'utente che useremo ha come username `studente`, ed è un utente che non richiede password.

In modalità terminale, potete usare come strumento di input solo la tastiera. Le vostre richieste al calcolatore appariranno su una precisa riga, quella in cui appare il **prompt** (che identifica le righe in cui scrivete i comandi); la riga attiva si riconosce per la presenza del **cursore**, tipicamente una sottolineatura o un rettangolo oscillante.

Le risposte del computer appariranno come caratteri tipografici sullo schermo, come se venissero stampati su un foglio di carta a modulo continuo, in sequenza; alla fine della risposta riapparirà il prompt (nei primi computer in effetti queste comunicazioni con l'utente avvenivano attraverso una stampante a modulo continuo, detta **console**). L'analogo attuale della console è la coppia tastiera-schermo.

2.2 Navigazione nel filesystem

2.2.1 Il filesystem di unix

Ogni cosa sotto linux compare come file, cioè come una sequenza di caratteri scritta da qualche parte nel disco rigido.

I file sono raggruppati in cartelle o **directory**, che sono esse stesse dei file particolari, che in un qualche senso contengono l'indice dei file nella cartella e la descrizione del posto in cui sono (queste informazioni sono invisibili all'utente, l'unica che gli appare è l'elenco dei file presenti). Tutte le cartelle sono raggruppate in un'unica cartella, che dunque contiene tutti i file del calcolatore. Tale cartella si chiama cartella radice e ha un nome particolare: `/` (questo simbolo si legge "slash").

L'organizzazione dei file e delle cartelle è standard, ed è ad albero (ogni cartella è contenuta in una sola altra cartella, e `/` non è contenuta in nessu altra, essa è appunto la radice dell'albero). Quello che segue è una piccola parte dell'albero delle directory presenti in questo sistema linux.

Ad ogni utente è assegnata una directory in cui scrivere i propri file. Questa directory è la **home directory** dell'utente.

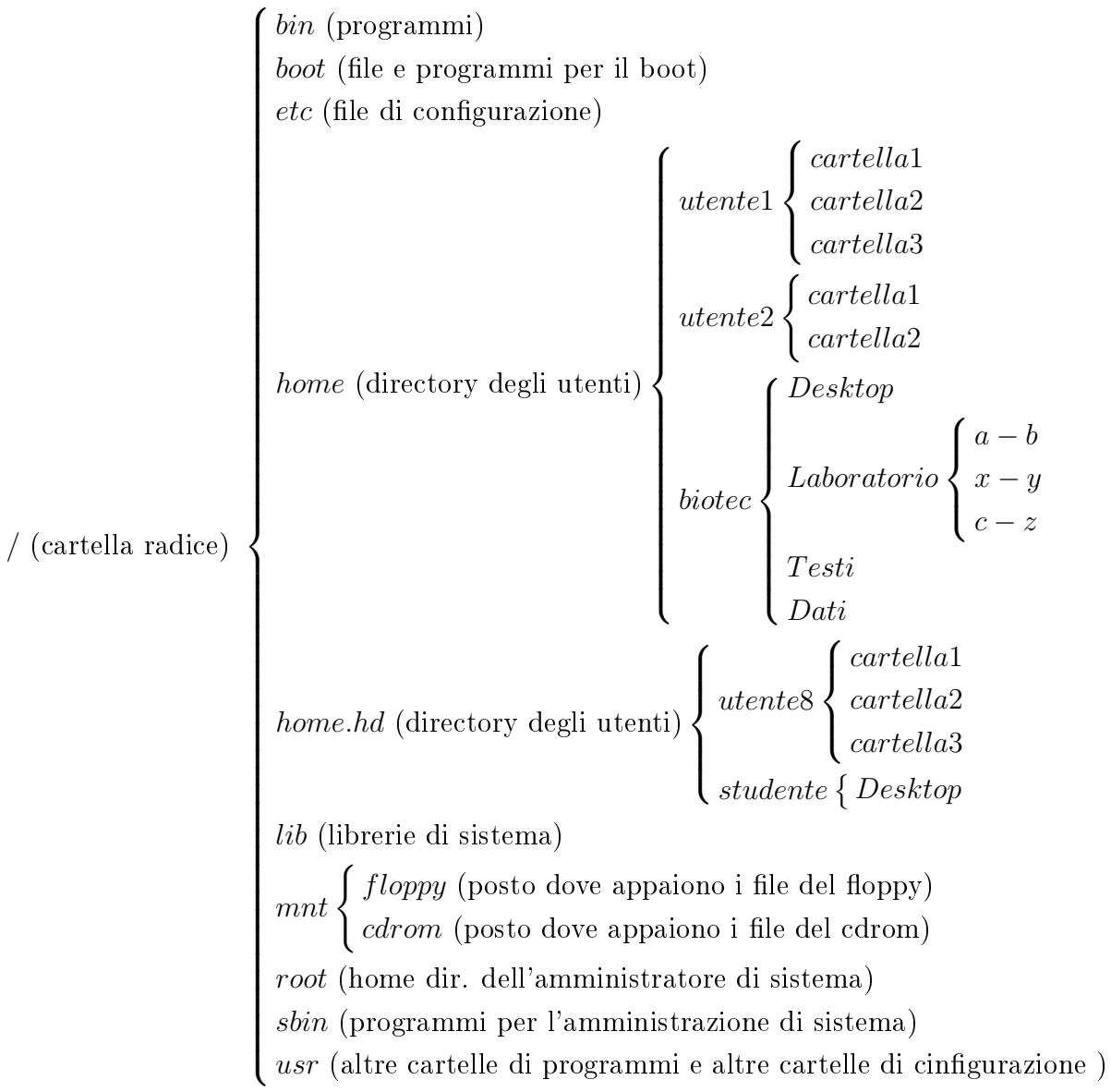
L'utente `studente` ha come home directory la cartella `studente` che è contenuta nella cartella `home.hd` che è contenuta nella cartella `/`. Per indicare il **percorso completo** ("path") che identifica la cartella, si mettono in sequenza tutte le cartelle necessarie, separate dallo slash:

home directory dell'utente `studente` è la cartella `/home.hd/studente`

home directory dell'utente `biotec` è la cartella `/home/biotec`

home directory dell'amministratore di sistema `root` è la cartella `/root`

NB. Nel nostro sistema ci sono due cartelle per gli utenti: `/home` e `/home.hd` ; in un sistema standard c'è solo la cartella `/home`. Qui la situazione è diversa perché la directory `/home.hd` è la directory degli utenti sulla workstation (tra cui appunto `studente`), la directory `/home` è la directory degli utenti sul server, tra cui `biotec`. La directory `/home` è visibile al server e a tutte le postazioni, le directory `/home.hd` delle singole postazioni sono invece visibili solo per le singole postazioni.



2.2.2 Lista dei file e delle directory: il comando `ls`

Completato il processo di login, l'utente si "trova" esattamente nella directory che gli è stata assegnata. Il comando `pwd` mostra il nome della directory in cui siete; la directory in cui siete, che per ora è la home directory, prende il nome di **directory corrente**. Il comando `ls` (abbreviazione di "list") mostra i file e le cartelle della directory in cui siete. Anche il comando `dir` mostra la stessa cosa, ma con una differenza: `ls` usa un accorgimento grafico che permette di dare colori diversi a file di diversi tipi. In particolare:

blu	directory
celeste	link simbolici (detti "collegamenti" in win e "alias" in Mac)
verde	eseguibili (programmi)
rosso	file compressi
violetto	immagini

Un **link simbolico** è un file che punta, indirizza, rimanda ad un altro file, con lo stesso nome, che si trova da qualche altra parte. Usare un link simbolico è un modo, ad esempio, per evitare

di fare molte copie dello stesso file. Supponete che una certa cartella di programmi debba essere contenuta in tutte le carte degli utenti. Invece di copiarla, si puó creare un link simbolico nella cartella di ogni utente, che rimandi alla cartella "vera".

`ls` (senza argomenti)	lista il contentuto della directory corrente
`ls nomedir`	lista il contenuto della directory `nomedir`
`ls nomefile`	lista del file `nomefile1` (lo mostra se c'è, altrimenti dà errore)
`ls nome1 nome2`	lista dei due file o directory
`ls -a`	lista dei **file nascosti**, cioè quelli il cui nome cominica con `.`
`ls -d nomedir`	lista **nomedir** come file, e non lista il suo contenuto
`ls -l`	lista con informazioni

Per la precisione, il comando `ls -l` dà un risultato del tipo

```
drwx------    2 dario     root         4096 gen 22  2003 nsmail
-rw-r--r--    1 dario     root          326 dic 22 21:17 scilab.hist
```

Il primo carattere indica se il file è una directory ("d") o se è un file normale ("-"), o se è un link ("l"), i successivi 9 caratteri indicano i **permessi** per quel file:

r	lettura (read)	file leggibile
w	scrittura (read)	file modificabile
x	esecuzione	file eseguibile

I primi tre permessi si riferiscono al possessore (user) del file, i successivi tre ai membri di un gruppo di utenti (group) (tipicamente il gruppo a cui appartiene lo user), gli ultimi tre si riferiscono a tutti gli altri (other).

Il numero successivo è il numero di link che puntano a quel file (compreso se stesso); i due nomi che seguono sono lo user e il group del file.

Infine compaiono la **dimensione**, misurata in **byte**, del file, la data di ultima modifica, e il nome del file.

2.2.3 Nomi di file, simboli, caratteri speciali

I sistemi Linux-Unix **distinguono** tra lettere maiuscole e minuscole (sono **case sensitive**). I file `abc Abc aBc AbC` sono tutti file diversi.

È buona norma, in un sistema linux, dare nomi di file che contengano solo lettere e numeri, e al massimo i simboli

`. - + _ (‘‘underscore’’)`

non all'inizio del nome. Soprattutto è meglio **evitare gli spazi**. Questo perchè i programmi come `ls` accettano gli argomenti separati dagli spazi. Se lo spazio fa parte del nome il programma funzionerà male. Ad esempio `ls pippo pluto` farà la lista dei due file `pippo` e `pluto`, e non del file `pippo pluto`. Inoltre molti simboli, tra cui $< > | * ? ! \backslash$, le parentesi e le varie virgolette, hanno degli usi particolari quindi è meglio evitarli nei nomi. (Per inciso, si può dare un nome anche con lo spazio e con gli altri simboli sconsigliati, però poi bisogna segnalare ogni volta che si usa un programma che quel particolare simbolo deve essere considerato **letteralmente** e non per il suo valore speciale: `ls pippo\ pluto` fa la lista del file "pippo pluto". Il **backslash** \ serve a dire "lo spazio bianco che lo segue va inteso letteralmente come spazio bianco e non come separatore di due nomi diversi". Tale uso del backslash si applica anche agli altri simboli. Esempio: come si fa a fare la lista del file `pippo\pluto`? Risposta: con il comando `ls pippo\\pluto`.

Uso dell'asterisco *: l'asterisco serve ad indicare parti di nomi arbitrarie:

`ls *txt` lista tutti i file il cui nome finisce con la sequenza "txt";

`ls A*.txt` lista tutti i file il cui nome finisce con la sequenza "txt" e cominciano con "A";

`ls g*a*s*` lista ad esempio i file `gas grasso gradasso` ma non lista `gsa`

`ls *` equivale a `ls` (lista tutti i file il cui nome corrisponde a *, cioè a qualunque sequenza di caratteri);

Di uso meno frequente è il simbolo `?`, che indica `un solo` carattere sconosciuto:

`ls ?.txt` lista ad esempio `a.txt` ma non lista `aa.txt`

Nomi simbolici delle directory

I simboli `.` e `..` indicano delle directory **relativamente** alla directory corrente (non hanno dunque significato assoluto, ma relativo a quale sia la directory corrente):

il nome `.` indica la directory corrente; ad esempio `ls .` coincide con `ls` e `ls *`

il nome `..` indica la directory che contiene la directory corrente; se siete in `/home.hd/studente` il comando `ls ..` coincide con `ls /home.hd` mentre il comando `ls ../studente` coincide con `ls .` e quindi con `ls`.

Il simbolo `~` (tilde) cambia significato a seconda dell'utente. Esso indica l'home directory. Ad esempio se siete in `/home/biotec` ma siete l'utente `studente`, il comando `ls ~` equivale a `ls /home.hd/studente`

2.2.4 Cambio di directory corrente, creazione di directory

Terminato il login, il terminale vi ha messo nella vostra home directory. Per cambiare posto si usa il comando `cd` ("change directory").

`cd /` vi posiziona nella directory radice `/`

`cd /home/biotec/Laboratorio` vi posiziona nella directory `/home/biotec/Laboratorio`

`cd .` vi posiziona nella directory corrente, dunque non vi spostate

`cd ..` vi posiziona nella directory che contiene quella in cui siete; se siete in `/home.hd/studente` vi porta in `/home.hd`

`cd ~` vi riporta nella vostra home directory, dovunque voi siate. Per fare quest'ultima operazione (utile se vi siete persi), in realtà basta il comando `cd` senza nessun argomento.

Per **creare** una directory si usa il comando `mkdir nomedir` ("make directory" nomedir). La directory che create sarà posizionata nella directory corrente. Naturalmente potete creare directory solo dove avete il permesso di farlo, ad esempio nella vostra home directory e nelle sue directory "figlie".

2.2.5 Operazioni sui file

Il comando `cp` serve a copiare ("copy") uno o più file. La **sintassi** del comando è

`cp file1 file2 file3 nomedir`	crea una copia dei file `file1 file2 file3` con lo stesso nom e li mette nella directory `nomedir`
`cp file1 file2`	crea una copia del `file1` e la chiama `file2`

Ad esempio

`cp /home/biotec/Testi/*.txt .` copia tutti i file il cui nome finisce con ".txt", presenti nella cartella /home/biotec/Testi/ nella directory corrente "."

Attenzione: il comando `cp nome1 nome2` fa cose diverse a seconda di cosa sono `nome1 nome2`:

	`nome1` è un file	`nome1` è una dir
`nome2` è un file	crea una copia del file `nome1` con nome `nome2`	dà errore
`nome2` è una dir	copia il file `nome1` dentro la directory `nome2`	dà errore

Per copiare cartelle bisogna dare l'opzione `-r` ("r" sta per "ricorsivo") al comand `cp`. Sia `dir1` la directory da copiare;

se `dir2` è una directory che già esiste `cp -r dir1 dir2` copia la directory `dir1` con tutto il suo contenuto (file e directory) dentro la directory `dir2`

se `dir2` non esiste `cp -r dir1 dir2` crea una copia della directory `dir1` con tutto il suo contenuto (file e directory) e la chiama `dir2`

se `dir2` è un file `cp -r dir1 dir2` dà errore

Il comando `mv` ("move") sposta file e cartelle; si usa anche per cambiare nome a file e cartelle

`mv nome1 nome2`

se `nome2` non esiste, cambia il nome del file o cartella `nome1` in `nome2`

se `nome2` è una cartella che esiste sposta il file o cartella `nome1` nella cartella `nome2`

se `nome2` è un file che esiste, lo cancella e procede come se non esistesse (ci **sovrascrive**)

Il comando `rm nomefile` ("remove") cancella il file `nomefile`. Per cancellare le directory si usa il comando

`rmdir nomedir`, che peró funziona solo se la directory è vuota. Per rimuovere una directory con tutto il suo contenuto si usa il comando più radicale `rm -r nomedir` (rimuove ricorsivamente).

2.2.6 Completamento, richiamo di un comando, scorrimento

Quando digitate comandi sulla tastiera non é direttamente il sistema operativo a rispondervi, ma un particolare programma che prende il nome di **shell** (detto anche interprete di comandi). Come sempre sotto linux ci sono vari possibili programmi di shell, i più usati sono `bash` e `tcsh`, che tra loro hanno lievi differenze. Queste dispense sono pensate per `bash`.
Una delle funzioni più utili nella scrittura dei comandi è il completamento dei comandi. Funziona cosí:

digitate `ls /u` e poi premete il tasto TAB; il comando verrà completato in `ls /usr`

digitate `ls /h` TAB, ottenete `ls /home`; ci sono due directory che iniziano per `/home` e sono la directory `home` stessa e la directory `home.hd`; continuate digitando `/s` TAB; otterrete `/home/studente`

Se la parte che avete digitato non identifica un unico comando o un unico file o directory, sentirete un bip. Per alcune shell (ma purtroppo non sulla nostra), digitando TAB due volte di seguito vi appariranno tutti i possibili completamenti. Ad esempio

`le` TAB TAB vi dà tutti i programmi che iniziano per le

Per richiamare un comando già eseguito, si possono usare le frecce che sono sulla tastiera: la freccia UP (su) scorre i comandi dati all'indietro, la freccia DOWN (giu) li scorre in avanti;

la freccia LEFT muove il cursore verso sinistra lungo il comando, la freccia RIGHT muove il cursore verso destra. Da notare che con le frecce restate sempre sulla riga attiva dello schermo.

Per vedere (e solo vedere) quello che è apparso nelle schermate precedenti, si possono usare i tasti PAGEUP (per andare indietro di una schermata) e PAGEDOWN (per riandare avanti di una schermata). In alcuni sistemi è necessario premere contemporaneamente il tasto SHIFT.

2.2.7 Lettura del contenuto dei file

Esistono molti programmi che vi mostrano il contenuto di un file; quale usare dipende da cosa c'è nel file. Per ora ci occupiamo dei file di testo. Un **file di testo** è (approssimativamente) un file che contiene solo i caratteri tipografici standard: lettere, numeri, interpunzione, simboli usuali (in sostanza quelli che leggete scritti sulla tastiera).

`cat nomefile` è il più sempice programma di lettura (in realtà il suo vero scopo è un altro). Esso riversa il file su schermo. Il problema del suo uso è che se il file è troppo lungo non riuscite a vederne l'inizio.

`more nomefile` è un programma più complesso: si impadronisce di tutto lo schermo e lo usa per visualizzare la parte iniziale del file; per visualizzare le parti successive richiede di digitare INVIO. Quanto tutto il file è stato mostrato il programma termina e vi restituisce il controllo della tastiera e il prompt. Per uscire prima della fine del file digitare `q`

`less nomefile` è un programma ancora più complesso: si impadronisce di tutto lo schermo e lo usa per visualizzare il file. Si può scorrere il file usando i le frecce UP e DOWN. Inoltre il programma si impadronisce anche della tastiera: quello che digitate su tastiera non compare sullo schermo, ma viene reinterpretato dal programma less. Alcuni dei nuovi significati:

q	esce del programma e vi fa tornare su terminale
g	va all'inizio del file
G (SHIFT g)	va alla fine del file
RETURN	avanza di una pagina (come PAGEDOWN)
/	entra in modalità ricerca, quello che scrivere dopo viene cercato nel file
n	cerca la successiva apparizione di quello che avete indicato con /
v	chiama il programma `vi` per modificare il file; NON LO FATE

(se lo fate, per uscire digitate :q, se non funziona ESC e poi :q e se non funziona ancora ESC :q! in casi estremi CTRL-z e poi kill%1)

2.2.8 Scrittura dei file

Il primo modo per scrivere in un file che consideriamo è il frutto di due operazioni distinte: l'uso del comando `echo` e le redirezioni dell'output `>` e `»` . Il comando `echo` è il comando che fa l'eco tra tastiera e schermo, cioè tra input e output. Ad esempio

```
echo questa e' una frase
```

dà come risultato la scritta

```
questa è una frase
```

Per riempire di contenuto un file possiamo utilizzare questo programma e la **redirezione dell'output**. L'apparato di output standard è lo schermo; il simbolo `>` permette di reindirizzare l'output su di un file (che esista o meno). Dunque

```
echo questa e' una frase > pippo
```

scrive la frase "questa e' una frase" dentro il file `pippo` invece che su schermo. Se il file `pippo` non esisteva viene creato, se esisteva viene svuotato e riempito della frase che gli abbiamo inviato. Per aggiungere del testo al file pippo si puó utilizzare la redirezione » che aggiunge l'output alla fine del file. Ad esempio le istruzioni

```
echo parola 1 > pippo
echo parola 2 > pippo
echo parola 3 » pippo
```

hanno come risultato finale che nel file pippo ci sia scritto

```
parola 2
parola 3
```

Un modo molto piú sofisticato di creare un file, o di scriverci dentro, o di modificarlo è quello di utilizzare un **editore** (editor). I programmi di editor sono molti; quello di base per linux, che trovate sempre, è `vi` ma è piuttosto complicato da usare. Quallo che useremo qui è `emacs` , anch'esso non particolarmente ovvio da usare e forse anche troppo potente per i nostri scopi.

Il comando `emacs nomefile` vi permette di **editare** (modificare) il file `nomefile`. Anch'esso prende possesso del terminale; questa volta però potete navigare nel file con tutte e quattro le frecce e potete modificarlo: tutto quello che scrivete apparirà nel punto in cui c'è il cursore (la modalità standard è di inserimento, cioè quello che scrivete si fa spazio in quello che già c'è, che non viene cancellato; questo comportamento si può cambiare con il tasto INS). Per cancellare si può usare DELETE (che cancella ciò che precede il cursore) oppure CANCEL (che cancella ciò che segue).

Lo schermo risulta diviso in due parti, una grande con il file su cui state lavorando, e una formata dall'ultima riga, in cui compare il nome del file. Molti comandi per emacs vengono dati nella forma CTRL-carattere, e CTRL-carattere-carattere. Il comando che avete dato compare nell'ultima linea dello schermo.

CTRL-x-s (save)	salva il file, cioè scrive effettivamente le modifiche fatte
CTRL-x-w (write)	chiede un nome con cui salvare il file
CTRL-x-c (close)	chiude il programma
CTRL-s	chiede una stringa da cercare; ripetendolo si cercano le occorrenze (apparizioni) successive
CTRL-g	annulla le istruzioni date con CTRL, molto utile...

È importante tenere presente che emacs fa una copia del file che volete modificare e la mette in un **buffer**, cioè una porzione di memoria RAM. Quindi voi modificate l'immagine del file che emacs ha nel buffer; per rendere effettiva la modifica dovete "salvare" il file.

Per creare un file è sufficiente editarlo (anche se non esiste): `emacs nomefile` crea un buffer vuoto, che potete riempire, e successivamente salvare con il nome `nomefile`

In genere, quando salvate un file modificato `nomefile`, emacs rinomina la versione precedente chiamandola `nomefile~`.

2.2.9 I programmi

Spesso il nome del file è composto dal nome vero e proprio e da una **estensione**, che è costituita da un punto e da alcune lettere. Sotto Windows questo è il comportamento standard: ogni file ha un'estensione; sotto unix no, un file puó essere chiamato in qualunque modo. Questa estensione indica che tipo di informazione dovrebbe essere contenuta nel file.

estensione	tipo di informazione	alcuni programmi che leggono o modificano tali file
txt	testo	cat, less per leggerli, emacs per modificarli
html	istruzioni in linguaggio html	netscape o explorer per leggerli
pdf	Portable Document Format	gv, xpdf, acroread per vederli
ps, eps	PostScript (per stampanti laser)	gv
gz	file compresso da gzip	gunzip dà la versione decompressa
bzip2	file compresso da bzip2	bunzip2 dà la versione decompressa
zip	file o archivio compresso da zip	unzip dà la versione decomressa
tar	archivio	tar crea e estrae archivi
tgz	archivio compresso	tar o tar e gz
doc, rtf	word e RichTextFormat	kword, abiword, soffice, per leggere e modificare
sh	istruzioni nel linguaggio della shell	eseguito da `sh`
pl	istruzioni in linguaggio perl	eseguito da `perl`
tex	istruzioni per tex, latex	tex e latex per produrre dvi, ps, pdf
c	istruzioni in c	cc gcc compilatori per il c
f	istruzioni in fortran	g77 compilatore per fortran

Tutti i file che contengono istruzioni dirette per il processore, o comandi che il computer può far eseguire ad alcuni altri programmi sono detti **eseguibili**, e appaiono verdi nell'output di `ls`. Al di la del colore, sono eseguibili perché hanno il permesso `x` che è appunto quello di esecuzione.

In modalità grafica i programmi sono spesso elencati nei vari **menu** presenti nelle finestre. In modalità non grafica i programmi vengono chiamati digitando il loro nome, compreso la loro posizione. Ad esempio se `fai1.pl` è un programma eseguibile che si trova nella directory `/home/utente/Prova`, per eseguirlo il comando giusto è

`/home/utente/Prova/fai1.pl` (attraverso il percorso assoluto)

Si può naturalmente usare anche il percorso relativo, ad esempio se siete in `/home/utente/Prova` potete digitare

`./fai1.pl`

Naturalmente, `pwd, ls, cd, cp, rm, mkdir, less, cat, more, man ...` non sono altro che programmi, però per eseguirli non scriviamo il loro percorso. Il fatto è che quando digitate qualche cosa, il sistema (in realtà la ťt shell) cerca il omando che avete digitato dentro alcune particolari directory che contengono programmi. Se lo trova il programma viene eseguito, altrimenti vi viene detto

`-bash: nomecomando: command not found`

cioè: comando non trovato. Tra le directory che contengono i programmi ci sono sempre `/bin /usr/bin /usr/local/bin /usr/X11R6/bin` e, a seconda della configurazione `~/bin .`
La directory `~/bin` è la directory in cui l'utente mette i suoi programmi; la directory `.` è per definizione la directory in cui siete, dunque, come nell'esempio precedente in cui siete in `/home/utente/Prova`, per far girare (runnare) il programma `fai1.pl` basta digitare `fai1.pl`.

L'elenco delle directory in cui la shell cerca i programmi è un fatto di configurazione che può anche essere modificato dall'utente (solo per quello che rigurarda lui, naturalmente). Tale elenco è scritto in una **variabile di ambiente** cioè è memorizzato sotto un nome particolare che prende il nome di PATH. Per visualizzare il contenuto della variabile PATH si digita

`echo $PATH`

Sul mio sistema la risposta è

`/home/dario/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games`

I : servono a separare gli elementi; nella mia configurazione come vedete manca la directory . e questo è buono perché evita di far partire qualche programma per errore.

Esistono molte variabili d'ambiante, ad esempio nella variabile `PS1` è memorizzato il prompt:

```
echo $PS1
```

visualizza, in codice, il prompt. Digitare `man sh` per il significato. Se digitate `PS1=''pippo''` il vostro prompt diventa pippo. Notate che per settare non serve il $, che invece è necessario per visualizzare con `echo`.

2.2.10 Cosa contiene veramente un file: bit, byte e codifica dei caratteri

Tutta l'informazione che è contenuta nei dischi, che viene elaborata dal processore, che viene scambiata da un computer ad un altro, o che viene inviata o ricevuta ad una periferica, è in formato particolare, detto **binario**.

In pratica l'unità indivisibile di informazione è lo stato di un circuito: **aperto** che simboleggiamo 0, e **chiuso** che simboleggiamo con 1. Tale unità di informazione prende il nome di **bit**. Possiamo dunque immaginarlo come un numero che può essere solo 0 o 1.

Con sole due possibilità non possiamo andare molto lontano; possiamo solo dire, ad esempio, NO o SI, o VERO, FALSO (in genere VERO = 1, FALSO = 0). Il trucco è che con molti bit si può dire qualcune cosa; ad esempio si può indicare qualunque numero, attraverso il sistema binario di numerazione.

La notazione decimale dei numeri è **posizionale**; a seconda della posizione della cifra, la cifra stessa assume valori differenti:

$$424 = 4 \cdot 100 + 2 \cdot 10 + 3 \cdot 1 = 4 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0$$

Dunque il valore del primo e dell'ultimo 4 sono diversi: il primo vuol dire 4 centinaia, l'ultimo 4 unità. La numerazione binaria usa lo stesso principio, ma solo per le cifre 0 e 1, e utilizza le potenze di 2 invece che quelle di 10. Ad esempio:

$$\begin{aligned} 10010101 &= 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^1 = \\ &= 128 + 16 + 4 + 1 = 149 \end{aligned}$$

In notazione binaria si può esprimere qualunque numero! Ad esempio i numeri da 0 a 16 sono rappresentabili nel seguente modo

decimale	binario
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Notare che tutte le potenze di due sono del tipo $10\cdots0$, cosí come, in notazione decimale appaiano le potenze di 10. In altre parole: in notazione decimale $10^3 = 1000$; in notazione binaria $8 = 2^3 = 1000$.

L'operazione di somma (e dunque prodotto e divisione) si esegue mediante lo stesso **algoritmo** del caso decimale: ad esempio: $1110 + 100 = 10010$ (provare a metterli in colonna, ricordando che $1 + 1$ dà 0 con riporto di 1...)

Nei file, in genere l'informazione è memorizzata in unità piú grandi del bit. Per la precisione l'unità di informazione registrata in un file è il **byte**, che è una sequenza di 8 bit. Con 8 bit si possono rappresentare i numeri da 00000000 a 11111111, cioè da 0 a 255= 256 – 1 = $2^8 - 1$.

In generale nei file ci sono molte altre cose, rispetto ai numeri da 0 a 255, ad esempio lettere, simboli etc. .

Quello che in realtà accade, è che i caratteri sono **codificati** in byte. Ad esempio, le lettere dell'alfabeto, nelle varianti minuscole e maiuscole, sono 52. Per codificare 52 simboli basterebbero 6 byte (con cui esprimo $2^6 = 64$ oggetti diversi). D'altra parte, oltre alle lettere, vanno codificati le cifre (che sono 10), i segni di interpunzione e i simboli più comuni (* & % etc...).

Una prima codifica prende il nome di **codifica ASCII**, e codifica i simboli con i 7 bit. Con 7 bit si possono indicare $2^7 = 128$ oggetti diversi, le rappresentazioni binarie disponibili vanno da 0000000=0 a 1111111=127. La tabella è tratta dalla pagina `man ascii`

La seguente tabella contiene i 128 caratteri ASCII.

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0'	100	64	40	@
001	1	01	SOH	101	65	41	A
002	2	02	STX	102	66	42	B
003	3	03	ETX	103	67	43	C
004	4	04	EOT	104	68	44	D
005	5	05	ENQ	105	69	45	E
006	6	06	ACK	106	70	46	F
007	7	07	BEL '\a'	107	71	47	G
010	8	08	BS '\b'	110	72	48	H
011	9	09	HT '\t'	111	73	49	I
012	10	0A	LF '\n'	112	74	4A	J
013	11	0B	VT '\v'	113	75	4B	K
014	12	0C	FF '\f'	114	76	4C	L
015	13	0D	CR '\r'	115	77	4D	M
016	14	0E	SO	116	78	4E	N
017	15	0F	SI	117	79	4F	O
020	16	10	DLE	120	80	50	P
021	17	11	DC1	121	81	51	Q
022	18	12	DC2	122	82	52	R
023	19	13	DC3	123	83	53	S
024	20	14	DC4	124	84	54	T
025	21	15	NAK	125	85	55	U
026	22	16	SYN	126	86	56	V
027	23	17	ETB	127	87	57	W
030	24	18	CAN	130	88	58	X
031	25	19	EM	131	89	59	Y
032	26	1A	SUB	132	90	5A	Z
033	27	1B	ESC	133	91	5B	[
034	28	1C	FS	134	92	5C	\ '\\'
035	29	1D	GS	135	93	5D	]
036	30	1E	RS	136	94	5E	^
037	31	1F	US	137	95	5F	_
040	32	20	SPACE	140	96	60	`
041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27	'	147	103	67	g
050	40	28	(	150	104	68	h
051	41	29	)	151	105	69	i

052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	-	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r
063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x
071	57	39	9	171	121	79	y
072	58	3A	:	172	122	7A	z
073	59	3B	;	173	123	7B	{
074	60	3C	<	174	124	7C	|
075	61	3D	=	175	125	7D	}
076	62	3E	>	176	126	7E	~
077	63	3F	?	177	127	7F	DEL

Nella tabella sono rappresentati: nella quarta colonna il simbolo codificato, nella seconda il numero decimale al quale corrisponde la codifica in bit del simbolo, la traduzione in **ottale** (prima colonna) e **esadecimale** (terza colonna) di tale numero. Ad esempio il carattere y è codificato con la sequenza di bit che corrisponde al numero decimale 122 (dunque in 7 bit 1111010, in 8 bit 01111010), la cui traduzione ottale è 172 e la cui traduzione in esadecimale è $7A$.

Il sistema ottale è il sistema di rappresentazione in base 8: usa i simboli da 0 a 8 e le potenze di 8; per rappresentare i numeri da 0 a 256 bastano tre cifre ottali, infatti $8^3 = 256$).

Il sistema esadecimale è il sistema di rappresentazione in base 16: utilizza i 16 simboli 0123456789ABCDEF per i numeri da 0 a 15, poi usa le potenze di 16; per rappresentare i numeri da 0 a 256 bastano due cifre esadecimali, infatti $16^2 = 256$).

Vi faccio notare i seguenti fatti:

- i simboli per i numeri non sono codificati con il loro valore in binario, ad esempio 0 è codificato con il byte che corrisponde al numero 48 (in binario 0010100)
- anche lo spazio vuoto è un carattere ed è codificato dal byte che corrisponde al numero 32
- il carattere di **terminazione di linea**
  - sotto linux è LF "line feed" (10 in decimale)
  - sotto windows è la coppia CR "carriage return" (13 in decimale) e LF
- il carattere che corrisponde al decimale 7 è il beep

Provate ad esempio i comandi
`echo -e ''\043''pippo`, `echo -e ''\040''pippo`,
`echo -e ''\012''pippo`, `echo -e ''\007''pippo`.
Tenete presente che l'ozpione `-e` di echo permette ad echo di interpretare `\xyz` come il carattere di numero ottale `xyz`; inoltre gli ottali 043, 040, 012, 007 sono i decimali 35, 32, 10, 7.

In realtà per rappresentare i caratteri non si usano 7 bit ma 8. Tale unità prende il nome di **byte** (o carattere). La codifica dei simboli usuali è data dalla codifica ASCIII a 7 bit preceduta da 0 (la tabella data sopra dunque continua a funzionare).

Rimangono liberi 128 possibili byte (quelli che in rapp. decimale vanno da 128 a 255). Ci sono codifiche differenti che usano in modo differente questi restanti 128 caratteri. Tutte le lingue occidentali (a parte l'inglese, su cui è modellato il codice ascii) hanno dei caratteri speciali di uso comune: ad esempio in italiano tutte le lettere accentate, in francese la "cedille" , in spagnolo i punti esclamativi e interrogativi al contrario, accenti particolari nelle lingue scandinave. Una codifica internazionale utilizza i restanti 128 caratteri per questi casi. In particolare la codifica iso-8859-15 è la codifica per l'europa occidentale (incluso il simbolo di euro, la iso-8859-1 è senza il simbolo dell'euro). Usare il comando `man iso_8859-15` per vedere come sono codificate le lettere speciali delle lingue occidentali). Qui di seguito riporto parte di questa ulteriore codifica.

```
The following table displays the characters in ISO 8859 Latin-9,  which
are printable and unlisted in the ascii manual page.

Oct   Dec   Hex   Char   Description
--------------------------------------------------------------------
```

```
241   161   A1    ¡    INVERTED EXCLAMATION MARK
242   162   A2    ¢    CENT SIGN
243   163   A3    £    POUND SIGN
244   164   A4    €     EURO SIGN
245   165   A5    ¥    YEN SIGN
344   228   E4    ä    LATIN SMALL LETTER A WITH DIAERESIS
345   229   E5    å    LATIN SMALL LETTER A WITH RING ABOVE
346   230   E6    æ    LATIN SMALL LETTER AE
347   231   E7    ç    LATIN SMALL LETTER C WITH CEDILLA
350   232   E8    è    LATIN SMALL LETTER E WITH GRAVE
351   233   E9    é    LATIN SMALL LETTER E WITH ACUTE
360   240   F0    ð    LATIN SMALL LETTER ETH
361   241   F1    ñ    LATIN SMALL LETTER N WITH TILDE
377   255   FF    ÿ    LATIN SMALL LETTER Y WITH DIAERESIS
```

Altre codifiche iso8859 sono usate per altre lingue:

```
ISO 8859-1    west European languages (Latin-1)
ISO 8859-2    central and east European languages (Latin-2)
ISO 8859-3    southeast European and miscellaneous languages (Latin-3)
ISO 8859-4    Scandinavian/Baltic languages (Latin-4)
ISO 8859-5    Latin/Cyrillic
ISO 8859-6    Latin/Arabic
ISO 8859-7    Latin/Greek
ISO 8859-8    Latin/Hebrew
ISO 8859-9    Latin-1 modification for Turkish (Latin-5)
ISO 8859-10   Lappish/Nordic/Eskimo languages (Latin-6)
ISO 8859-11   Latin/Thai
ISO 8859-13   Baltic Rim languages (Latin-7)
ISO 8859-14   Celtic (Latin-8)
ISO 8859-15   west European languages (Latin-9)
ISO 8859-16   some east European languages (Latin-10)
```

Dunque servono diverse codifiche per diverse lingue. Esistono amche codifiche universali, ad esempio le codifiche UCS2 e UCS4 che usano almeno due byte per codificare ogni carattere; il loro problema è che modificano anche l'insieme base dei caratteri (quelli codificati in ascii, che usa solo un byte). Ampia diffusione ha in questi anni la codifica UTF8, che coincide con ascii per i primi 7 bit, e che usa un numero di bit differente per codificare vari simboli (incluse le lingue dell'estremo oriente, cioè quelle con un grande numero di simboli); tale codifica è un esempio di codifica a **lunghezza variabile**.

In breve.

I file di testo sono quelli che contengono solo caratteri tipografici; purtoppo quali siano i caratteri tipografici dipende dalla lingua...

In generale un file eseguibile come i programmi `ls`, `less`, o un file che trasposta informazioni testuali, ma attraverso un programma che ne curi l'aspetto grafico, (ad esempio un file con

estensione .doc creato da Word su un sistema Windows), è un file in cui la sequenza di byte (o addirittura di bit) non ha nulla a che vedere con il corrispondente carattere. Un tale file è detto **binario**. Provare ad esempio a fare il `less` di file .doc .gz o di un programma tipo ls. A parte qualche singola parola tutto il resto è del tutto illegibile; la traduzione in caratteri delle sequenze di bit o di byte non è utile ai fini della comprensione del contenuto.

D'altra parte, fate il `less` di un file PostScript (estesione .ps). Otterrete un tranquillo file fatto di caratteri leggibilissimi (senza accenti nè altro). Il formato .ps è il formato che viene letto dalle stampanti laser (dette anche PostScript).

2.2.11 Compressione e archiviazione

Un esempio radicale di uso di tutti i 256 caratteri scrivibili con un byte è dato dai file compressi.

Un file di testo usa tipicamente pochi caratteri, (meno di 128), dunque per codificarlo basterebbero 7 bit per carattere (invece di 8). In tal modo la sua dimensione risulterebbe ridotta di 1/8.

Un esempio più profondo: se un file contiene 4000 volte di seguito la lettera "A", invece di usare 6×4000 bit, basterebbe dire "4000 volte A", che impiega, scritto in questo modo, 12×8 bit.

Un programma di compressione ha potenti algoritmi che riducono il numero di bit per scrivere l'informazione da trasmettere o da scrivere su disco, senza **perdere nessuna informazione**. Anche le immagini e i suoni vengono compressi; rispetto all'originale in genere ci si permette di perdere un po' di informazione, pur di ottenere file più piccoli.

Su sistemi linux il programma più usato è `gzip`. Il comando `gzip nomefile` che crea la versione compressa `nomfile.gz`; il programma per ricostruire il file dalla versione complessa è `gunzip nomefile.gz`.

Un ulteriore programma si linux è `bzip2`, con inverso `bunzip2`, che usa algoritmi completamente diversi da gzip, zip, winzip. Comprime meglio (il **rapporto di compressione** è il rapporto tra la lunghezza del file compresso e la lunghezza del file), ma ha lo svantaggio di essere un po' più lento.

In generale, il rapporto di compressione migliora su file grandi. C'è un limite teorico alla comprimibilità di un file, che ha a che fare con l'**entropia**, ed è uno degli argomenti centrali della **teoria dell'informazione**; in realtà la comprimibilità dell'informazione (ovvero la ricerca del numero più piccolo di bit per trasmetterlo integralmente) è stato uno degli argomenti per cui essa è stata creata.

`gzip nomefile` produce il file compresso `nomefile.gz`, cancellando l'originale

`gunzip nomefile.gz` decomprime e ricrea il file originale `nomefile`

Per l'uso immediato, la sintassi della coppia di programmi `bzip2` e `bunzip2` è identica. Per le opzioni (che sono in parte diverse) consultare le pagine di manuale.

Un **archiviatore** è un programma che serve a trasformare un intero albero di directory e file in un unico file. L'uso principale di questi programmi è per fare copie di salvataggio **backup** o per trasferire facilmente (via rete o via mail) intere cartelle conservandone la struttura. Esempio: nella directory di lavoro esiste la cartella `Prova` che contiene vari file e sottodirectory, anch'esse popolate di file e directory. Il comando

```
tar cvf archivio.tar Prova
```

crea il file `archivio.tar` che contiene tutto quello che c'è in `Prova`. Nel formato tar l'archivio è illegibile agli umani. Cancelliamo la directory `Prova`

```
rm -rf Prova
```

ora spacchettiamo l'archivio:

```
tar xvf archivio.tar
```

il risultato di quest'ultima operazione è che la directory `Prova` è stata ricreata con tutto il suo contentuto, esattamente come era (anche nei permessi e nelle date di creazione).

Chiarisco la sintassi di `tar`. Nel comando `tar cvf archivio.tar Prova` le lettere `c v f` sono tre opzioni distinte: `c` vuol dire create, `v` vuol dire verbose, `f` vuol dire file: cioè "crea, dicendomi tutto quello che fai, il file archivio.tar dalla dirctory Prova". Nel comando `tar xvf archivio.tar`, `x` vuol dire extract, dunque il comando è "estrai, dicendomi tutto quello che fai, dal file archivio.tar".

In genere il file di archivio che viene creato è piuttosto voluminoso; ai fini di backup o trasferimento è buona nomra comprimerlo, tipicamente con `gzip`. In rete quasi tutti gli archivi di cose per linux hanno dunque il formato `.tar.gz` o `.tar.bz2`. Un'ulteriore possibilitá è usare per `tar` l'opzione `z` che produce direttamente la versione compressa, o il programma `tgz` che usa `tar` e poi `gzip`; in quest'ultimo caso l'archivio compresso ha estensione `.tgz`.

Il programma standard sotto windows per l'archiviazione (già compressa) è `WinZip`, il cui analogo (compatibile) per linux è `zip` Il comando

```
zip archivio ./Prova
```

crea l'archivio compresso `archivio.zip` con dentro `Prova` e tutti i suoi file. Per decomprimere e spacchettare l'archivio il comando è `unizip nomearchivio`

2.3 Altri aspetti

2.3.1 Redirezione dell'input e dell'output

Questa è una delle funzionalità più caratteristiche di alcune shell unix. Essa permette di spostare l'output di un comando dallo schermo ad un file o ad un programma, e di spostare l'input dalla tastiera ad un file. I simboli che si usano sono `>` che ridireziona l'output, `<` che ridireziona l'input, `|` che manda l'output come input di un programma (si chiama "pipe"), `»` che ridireziona l'output in modalità "append".

Esempi

`ls *.txt > elencofile` manda l'output del comando nel file `elencofile`. Dentro tale file ci sarà il risultato del comando.

`gzip < file1 > file2.gz` usa il file1 come input del programma `gzip`, che lo comprime; il risultato della compressione è messo dentro il `file2.gz` (in genere gzip cancella il file originale; in questo modo esso non viene toccato).

`cat file1 file2 | gzip > nuovofile.gz` fa le cose seguenti: `cat file1 file2` apre in sequenza i fue file e ne riversa il contenuto fuori, il simbolo di "pipe" intercetta questo contenuto e lo manda come input al programma `gzip` che lo comprime, il risultato della compressione è mandato dentro il file `nuovofile.gz`

```
cat file1 > nuovo
cat file2 » nuovo
```

fanno le cose seguenti: il primo `cat` apre il `file1` e ne riversa il contenuto nel file `nuovo`, creandolo se non c'era, e cancellando e ricreandolo se c'era; il secondo cat riversa il contenuto di `file2` in coda al file `nuovo` ("appende"), che dunque conterrà entrambi i file.

`ls /etc | less` manda l'output del comando `ls /etc` al programma `less`. Questo permette di scorrere in su e in giú il lungo elenco di file della directory `/etc` (va peró perso il colore, non gestito dal programma `less`).

2.3.2 Aiuto sui comandi

Ci sono vari modi per ottenere informazioni sull'uso di un comando o programma. A volte funziona il tentativo seguente:

`nomecomando -help` o in qualche caso `nomecomando -h`

Se funziona, si ottiene una descrizione sintetica del comando e della sua sintassi, e una breve descrizione delle opzioni.

I sistemi unix hanno un manuale in linea, che si richiama con il comando `man nomecomando`. Il programma `man` visualizza (usando `less` o `more` a seconda delle configurazioni) le pagine di manuale relative a quel comando. In particolare viene riportato: NAME del programma, SYNOPSIS (sintassi), DESCRIPTION, con l'elenco di tutte le opzioni e il loro utilizzo. Purtroppo solo raramente sono presenti esempi di uso.

Un altro comando utile è `apropos cosa`, che fa l'elenco di tutte le pagine del manuale che riguardano `cosa` (in realtà è una delle tante opzioni di man: `apropos` è equivalente a `man -k`.

Un'altra fonte di informazioni è il programma info, che richiama altre pagine di informazione, piú strutturate. Le pagine sono navigabili attraverso le frecce SU e GIU, il tasto INVIO e altri tasti: digitare `info info` per maggiori informazioni. Quando è presente, la documentazione di `info` è piú estesa rispetto a quella di `man` e soprattutto contiene esempi. Provate a confrontare `man gzip` e `info gzip`.

Altri comandi; per maggiori informazioni usate le pagine di manuale

`head nomefile` visualizza le prime 10 righe del file `nomefile`

`tail nomefile` visualizza le ultime 10 righe del file `nomefile`

`wc nomefile` conta quante linee, parole, byte contiene il file `nomefile`

`grep stringa nomefile` visualizza tutte le righe di `nomefile` in cui c'è la stringa `stringa`

`grep stringa elencofile` visualizza tutte le righe dei file in elenco in cui c'è la stringa `stringa`.

Provate ad esempio

`grep Identifier /etc/X11/XF86Config` e `grep Identifier /etc/X11/XF86Config*`

che differenza c'è? e perché?

`date` stampa la data (serve anche per settare la data ma servono privilegi di root)

`hostname` stampa il nome del computer

`uname -a` stampa tutte le informazioni sul sistema operativo

`who` fa vedere quali utenti sono collegati con il computer

`whoami` stampa il nome dell'utente

`whereis nomeprogramma` cerca nelle directory elencate in $PATH il posto in cui si trova il programma `nomeprogramma`

`whatis nomeprogramma` dice in breve a cosa serve il programma `nomeprogramma` e indica in quale pagina di manuale è descritto

`locate abc` elenca tutti i file del calcolatore il cui nome contiene `abc`

Provate ad esempio `locate F864Con`. La risposta è molto veloce perchè ogni notte il calcolatore rigenera l'elenco di tutti i file, e questo elenco è quello che viene consultato da `locate`. Per far rigenerare l'elenco, il comando è `updatedb`, ma per eseguirlo bisogna avere i privilegi di root. sistema. Se il database non è aggioranto è abbastanza inutile, se non per i file standard del sistema. Ad esempio se avete appena creato `nomefile`, il comando `locate nomefile` non lo troverà perché non è nel database.

`find nomdir -name abc` cerca tutti i file il cui nome contiene `abc` a partire dalla directory `nomedir`. Al contrario di `locate`, il programma `find` scende ricorsivamente in tutte le directory cercando quello che avete chiesto. Non si appoggia dunque ad un database già costruito, e dunque è decisamente più lento.

2.3.3 Processi e loro gestione

Unix è un sistema operativo **multitasking**, che vuol dire la cosa seguente: possono girare contemporaneamente molti programmi, ognuno dei quali usa un po' di ram e un po' di tempo di processore. Un programma che sta girando è chiamato **processo**. I processi dunque richiedono risorse al sistema, sia come memoria disponibile che come tempo della CPU. Le risorse che ogni programma usa sono decise istante per istante dal sistema operativo. Inoltre i processi hanno una struttura ad albero: esiste un primo processo (che è il programma `init`) di cui sono figli tutti i successivi processi. Ad esempio se lanciate da terminale il comando `ls`, quello che accade è che il processo `sh` (la shell che sra girando nel terminale) crea un processo figlio che è appunto `ls`. Quando il comando è stato eseguito il processo termina.
Il comando `ps` elenca i processi lanciati dal terminale in cui siete, il comando `ps x` elenca tutti i processi lanciati da voi (o a nome vostro), il comando `ps ax` elenca tutti i processi attivi lanciati. Provate ad esempio a digitare `cat` e invio su un terminale. In questo caso il programma `cat` è in esecuzione, anche se non fa nulla. Su un altro terminale date il comando `ps ax | less` vedrete che il primo processo (numerato infatti con 1) è `init`; l'ultimo o comunque uno degli ultimi è il processo `cat` che avete lanciato. Per uscire dal programma `cat` digitate sul terminale giusto la sequenze `CTRL-c` (che vuol dire CANCEL).

A volte i programmi si impallano, cioè non si riesce a interrompere la loro esecuzione. Un modo per fermarli è quello di **uccidere il processo**. Ad esempio, lanciate emacs da tastiera, e usate `ps ax` per scoprire che numero di processo gli è stato assegnato, ad esempio 25489. Per ucciderlo digitate

```
kill 25489
```

in casi veri (di programma piantato), potrebbe non funzionare; provate quindi con

```
kill -9 25489
```

Quanto nell'esempio precedente abbiamo lanciato il programma `emacs` da terminale, anche se si è aperta una finestra grafica il controllo del terminale ci è stato tolto. Il motivo è che il processo `sh` ha creato il processo `emacs` come figlio, e, fino a che non termina, continua a prestagli le proprie risorse. In tal caso si dice che il programma gira in **foreground**. Provate invece a lanciare `emacs &`. La `&` dice al processo figlio di girare in **background**, ovvero di usare sue risorse e lasciare libere quelle del processo che lo ha generato. L'elenco dei processi in background si ottiene con

il comando `jobs` che dice quale numero di "job" (cioè lavoro) corrisponde al programma, e in che stato è (Running o Stopped). Un programma, per esempio il numero 1, può essere terminato con

`kill %1` o `kill -9 %1`

o può essere mandato in background con

`bg %1`

o in foreground con

`fg %1`

Cosa fare se avete un processo in foreground e volete spostare in background senza ucciderlo? Prima digitate CTRL-z, che "stoppa" il programma; poi date il comando `jobs` per scoprire quale numero ha, ad esempio 3, poi mandatelo in background con il comando `bg %3`.

2.3.4 Interfaccia grafica

Il sistema grafico di linux si chiama **X window**. Sul sistema grafico girano i cosiddetti desktop-manager, ovvero dei programmi che curano l'aspetto della "scrivania", il comportamento del mouse, i programmi che si possono lanciare, l'aspetto delle finestre. Tali programmi sono dei veri e propri **ambienti grafici** perché governano anche il modo con cui altri programmi possono accedere alla grafica. Naturalmente ce ne sono molti. Quello standard in laboratorio e quello di Knoppix si chiama KDE. Il suo rivale principale si chiama GNOME. Entrambi hanno inglobato vari programmi standard. Ad esempio il visualizzatore di pdf è `xpdf`, la sua reimplementazione sotto GNOME è `gpdf`, sotto kde è `kpdf`.

Infine ci sono i **file manager** che sono i programmi a finestre che permettono di visualizzare in modo "grafico" (ad esempio come icone) il contentuto delle cartelle, e permettono di avviare i programmi, vedere immagini, aprire editori dei file attraverso il click del mouse. Il file manager di GNOME è Nautilus, quello di KDE è Konqueror; entrambi hanno anche alcune funzioni dei browser (possono cioè anche navigare in rete).

2.4 Aspetti avanzati

Questa sezione parla di argomenti più avanzati, necessari se utilizzerete Knoppix. In tal caso, infatti, sarete gli amministratori del vostro sistema ed è importante che sappiate alcune cose.

2.4.1 Device

Uno dei punti più delicati nell'apprendimento di unix è come il sistema gestisce l'hardware, in particolare quello esterno (floppy, cdrom, penne USB).

Sotto unix ogni dispositivo che viene collegato al calcolatore viene associato dal sistema operativo ad un file di tipo particolare (detto file di tipo "device" o dispositivo), che si trova nella directory `/dev`. Se fate `ls /dev` vederete molti file di colori che non sono quelli standard. Meglio ancora digitate `ls /dev/hda /dev/tty1`: della linea dei permessi scoprite che non sono file regolari, ma nemmeno directory e nemmeno link. Insomma questi strani file sono gli strumenti di collegamento tra il sistema e alcuni dispositivi hardware. Ad esempio, in una configurazione standard valgono i seguenti accoppiamenti:

mouse	`/dev/psaux`
primo terminale virtuale	`/dev/tty1`
secondo terminale virtuale	`/dev/tty2`
primo terminale dentro la grafica	`/dev/pts1`
primo disco rigido di tipo IDE	`/dev/hda`
secondo disco rigido di tipo IDE	`/dev/hdb`
prima partizione di hda	`/dev/hda1`
primo disco SCSI	`/dev/sda`
floppy	`/dev/fd0`
lettore cd	`/dev/cdrom`

In genere `/dev/cdrom` è un link simbolico a `/dev/hdc`.

2.4.2 Dischi e partizioni

In particolare, mentre sotto Windows i dischi e i dispositivi di memoria esterni hanno come nomi delle lettere (C per il disco rigido, A per i dischetti, D, E per cdrom e memorie USB), sotto unix i nomi sono più complessi e dipendono dal tipo di unità e da come è collegata.

Ripeto: in una configurazione standard, con un dischо rigido di tipo IDE, il disco è `/dev/hda`, il lettore cdrom è `/dev/hdc`. Un eventuale secondo disco sarebbe tipicamente `/dev/hdb`. Per fare un esempio di situazione non tipica: il mio calcolatore ha dischi SATA che vengono trattati come dischi SCSI, dunque il mio primo disco è `/dev/sda`, mentre il lettore cdrom è `/dev/hda`.

Inoltre, i dischi rigidi spesso sono divisi in `partizioni`, cioè in differenti sezioni che possono avere filesystem di tipo differente. Qui uso la parola filesystem non per indicare ma la gerarchia ad albero delle directory, ma il seguente concetto: "un file system è l'insieme dei tipi di dati astratti necessari per la memorizzazione, l'organizzazione gerarchica, la manipolazione, la navigazione, l'accesso e la lettura dei dati" (Wikipedia) Consultare `man fs` per l'elenco dei filesystem che vengono riconosciuti da linux.

I filesystem tipici delle partizioni per linux sono due `ext3` e `ext2`, il filesystem utilizzato da windows è `vfat`, una sua versione ristretta è quella del DOS: `msdos`. Il tipo di filesystem dei cdrom è iso9660, le memorie USB sono praticamente tutte vfat, un iPod appena comprato ha il filesystem Macintosh hfs+; se lo attaccate ad un calcolatore windows viene tipicamente trasformato in vfat.

Dunque un disco rigido può avere piú partizioni, e anche alcune per un sistema operativo e alcune per un altro. In genere il disco rigido di un sistema linux ha almeno una partizione di tipo `swap`. L'area di questo disco serve al sistema per liberare temporaneamente la RAM se le richieste eccedono le sue dimensioni (sotto windows e Mac si chiamerebbe "memoria virtuale"). In assenza di partizione di `swap` il computer potrebbe piantarsi.

Per rendere accessibile il contenuto di una partizione di un disco rigido il sistema lo attacca da qualche parte all'albero delle directory. Tale operazione prende il nome di `montaggio` della partizione. Dopo il montaggio il contenuto della partizione sarà disponibile come un pezzo qualunque dell'albero delle directory.

Il comando `mount` mostra tutte le partizioni, il loro tipo di filesystem e dove sono montate. Sul server del centro di calcolo si ottiene:

```
/dev/sda1 on / type ext2 (rw)
none on /proc type proc (rw)
/dev/sda5 on /home type ext2 (rw)
/dev/sda6 on /tmp type ext2 (rw)
```

```
/dev/sda9 on /usr type ext2 (rw)
/dev/sda8 on /var type ext2 (rw)
/dev/sdb2 on /usr1 type ext2 (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
```

Tralascio alcuni oggetti particolari, come proc e devpst. Da questo output si scopre che il sistema monta 6 partizioni di uno stesso disco (`/dev/sda`). In particolare tutti i file di `/home` risiedono in effetti nella partizione 5, tutti i file di `/var` nella partizione 8; tutti i file che non sono nelle altre partizioni sono dentro la partizione `/dev/sda1` che è montata sotto `/`. Tutte le partizioni considerate sono di tipo `ext2`.

2.4.3 Montare e smontare

La questione del montaggio potrebbe non interessarci (in fondo il sistemista si occupa dei dischi rigidi), se non fosse che nello stesso modo vengono trattati anche i dispositivi di memorizzazione esterna: il floppy, il cdrom, le memorie USB.

Esempio: ho un dischetto con dei file che voglio trasferire dentro il mio sistema. Inserisco il dischetto. Non accade nulla. Devo dire al calcolatore di montare il dischetto da qualche parte nel sistema. Questo "qualche parte" deve essere una directory vuota. Il comando per effettuare il montaggio è ancora `mount`. Ad esempio spesso esiste la directory vuota `/mnt/floppy`, apposta per montarci sopra il floppy. Il comando da dare è

`mount -t auto /dev/fd0 /mnt/floppy`

L'opzione `-t auto` vuol dire: non so come è formattato, vedi tu se lo riconosci. Come si vede la sintassi è la seguente: monta, cercando di indovinare il tipo, il dispositivo collegato al `/dev/fd0` (che è il floppy) nella directory `/mnt/floppy`. Quando ho finito di lavorare con i file del floppy (che trovo nella directory `/mnt/floppy`) prima di estrarlo devo **smontarlo**. Il comando per smontare è `umount`. Funziona nelle due versioni seguenti:

`umount /dev/fd0` oppure `umount /mnt/floppy`

Abbastanza difficile vero? In realtà è ancora peggio: i comandi scritti sopra li può dare solo l'utente `root`. Ma allora come si fa a montare un floppy? Esiste un file di configurazione che contiene i montaggi standard del sistema. Esso è `/etc/fstab` (fstab=tabella dei filesystem). Quelle che seguono sono tre righe abbastanza standard di `/etc/fstab`:

```
/dev/hda1   /           ext3    defaults              1 1
/dev/cdrom  /mnt/cdrom  iso9660 noauto,user,exec,ro   0 0
/dev/fd0    /mnt/floppy auto    noauto,user           0 0
```

La prima dice al sistema di montare sotto / la partizione `/dev/hda1`, che è di tipo `ext3`. La seconda e la terza riga non vengono in genere eseguite, perchè dentro il floppy e dentro il cdrom in genere non c'è nulla. Però la presenza di `user` nella riga dice che il montaggio può essere effettuato da un utente, e non solo da `root`. In tal caso, il comando che l'utente deve dare è

`mount /dev/cdrom` oppure `mount /mnt/cdrom`

Confrontandolo con quello di prima, si vede che è più facile, ma non permette variazioni: il cdrom verrà montato su `/mnt/cdrom`, il floppy su `/mnt/floppy`.

2.4.4 Troppo difficile?

Niente paura: spesso le cose sono più facili. Intanto a volte c'è il programma `automount`, che controlla l'inserimento di dischetti e CD. Se essi vengono inseriti, vengono anche automaticamen-

te montati come specificato in `/etc/fstab`. Qui il problema è piú che altro lo smontaggio, che va sempre effettuato prima di estrarre il dischetto (il perchè è spiegato nel successivo paragrafo).

Quando usate un'interfaccia grafica lo smontaggio è piuttosto facile: andate su "Risorse del computer", selezionate l'icona del floppy (o del cdrom), clickate con il tasto di destra e, tra le diverse opzioni, dovrebbe apparire "smonta il volume".

Non solo: anche se non c'è il mount automatico, un tentativo di accesso all'icona del floppy o del cdrom in genere effettua il montaggio.

2.5 Perche è necessario smontare

Quando salvate un file su disco rigido o su dischetto, in teoria avete fisicamente scritto dei bit sul disco o sul dischetto. In pratica non è vero: la scrittura su disco o dischetto è una delle operazioni più lente che un calcolatore effettua, dunque il computer aspetta per farla le condizioni adatte (le decide il sistema operativo). Dunque se levate un floppy senza smontarlo, quello che pensavate fosse stato salvato forse non è stato realmente salvato.

Lo stesso discorso si applica se spegnete male il computer (non facendo lo `shutdown` ma premendo il pulsante di spegnimento, o staccando la spina). Il calcolatore, infatti, potrebbe aver già modificato gli indici degli oggetti sui dischi, senza aver fisicamente scritto i file. Quando riavviate ci potrebbe essere una discrepanza tra gli indici e i contenuti, che il calcolatore controlla impiegando un po' di tempo con il programma `fsck`. Se la discrepanza c'è si parla di **filesystem corrotto**. È un danno che può essere riparato (in genere automaticamente) se non è troppo esteso e se non riguarda fondamentali file di sistema. In caso contrario il sistema va reinstallato.

2.5.1 Knoppix

Fatte queste premesse, passo a descrivere la "distribuzione live" detta Knoppix. "Distribuzione live" vuol dire che è un sistema operativo che funziona anche se non viene istallato sul calcolatore. Funziona nel modo seguente:

- Spegnete il calcolatore: Knoppix non è un programma ma un sistema operativo (non è un gene ma un'intero genoma)
- inserite il cd di Knoppix nel lettore cd;
- accendete il calcolatore;
- nella prima schermata in momento/punto c'è scritto come entrare nel SETUP, entrateci;
- tra i menu del SETUP cercate quello che parla di BOOT o device di BOOT, in questo menu c'è scritto su quali dispositivi il calcolatore cerca il sistema operativo (e in quale ordine); in genere sono tre: disco rigido, cdrom, floppy;
- modificate l'ordine in modo che cdrom preceda disco rigido
- salvate le impostazioni del SETUP;
- a questo punto il calcolatore riparte dall'inizio e caricherà il sistema operativo da cdrom.
- vi comparirà una finestra con scritto `boot`; in prima approssimazione premete invio, se non funziona riprovate e leggete le possibili opzioni sotto i tasti F2 e F3.

Knoppix è una distribuzione linux che risiede (compressa) nel cdrom, quindi nulla viene modificato del vostro sistema abituale che risiede nel disco rigido.

Dovendo adattarsi ad un hardware che non conosce, knoppix impiega un po' di tempo a caricarsi. Inoltre tenete presente che gira su cdrom, e la velocità di lettura di un cdrom è molto minore della velocità di lettura del disco rigido. Infine, il sistema vero e proprio viene caricato nella ram, quindi più ram avete e meglio è; se ne avete troppo poca non riuscirete a fare nulla.

Quando finalmente il caricamento è finito vi accorgerete che siete in modalità grafica, sotto ambiente KDE. Cose da notare:

- non vi viene chiesto il login: per definizione siete l'utente `knoppix`
- tra i programmi disponibili c'è il terminale
- esiste l'utente `root`, per avere accesso ai suo privilegi dovete usare la `shell di root` che trovate tre i menu K
- tra i programmi di configurazione c'è quello per il modem o per la linea adsl (`pppoeconf` o qualche cosa del genere); funziona facilmente a meno che non abbiate un "winmodem", in tal caso è praticamente senza speranza
- nella scrivania vedete anche delle icone per le partizioni del disco rigido; per montarle basta clickarci sopra

L'ultimo punto è importante: potete montare il vostro disco rigido, anche se è una partizione windows, e quindi potete leggere e scrivere i vostri abituali file. Non potete invece runnare i programmi, che sono comprensibili solo per il sistema operativo con cui sono stati creati.

Infine, seguendo le istruzioni, è possibile salvare le impostazioni su disco che eventualmente avete modificato (ad esempio la configurazione del modem), o installare sul disco il sistema. Installare linux su disco non è veramente problematico: alla peggio non funziona e ci si può riprovare. Il punto veramente delicato è che per installare anche linux è necessario fare spazio nel disco rigido, **ripartizionando** la partizione windows. Per fare ciò va utilizzato un ripartizionatore sotto windows, tipicamente Disk Druid. Si tratta di ridurre lo spazio di windows per poi poter creare una o più partizioni linux. Si può anche provare con un partizionatore per linux (ad esempio `parted`), che può modificare anche le partizioni windows. Il rischio qui è che perdiate tutti i dati windows. Un modo sicuro di procedere è: salvare su cd o floppy tutte le cose importanti e tenere a portata di mano un disco di istallazione di windows, caso mai il partizionamento vi distrugesse windows stesso.

ATTENZIONE: non vi sto chiedendo di installare linux e nemmeno di runnare Knoppix modificando l'ordine di boot. Non mi ritengo responsabile dei danni morali o materiali che potete fare provandoci. L'uso di Knoppix è a vostra scelta un modo per esercitarvi a casa e non in laboratorio; l'istallazione di linux è una vostra eventuale scelta che in ogni caso non ha nulla a che vedere con il corso. D'altra parte se volete provarci vi assisterò come posso. In caso vogliate installare il sistema leggete cosa dice la guida di Knoppix a tal proposito, o forse tentate di istallare qualche altra distribuzione, ad esempio Ubuntu, Mandriva, Fedora, Suse, Debian (spesso le trovate in edicola insieme a riviste di informatica), leggendo attentamente le istruzioni.

Una pagina con un elenco delle distribuzioni è http://it.wikipedia.org/wiki/Linux

Capitolo 3

R

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs out of the box on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux). It also compiles and runs on Windows 9x/NT/2000 and MacOS.

Il **programma** e tutta la **documentazione** sono liberamente scaricabili da rete, per piattaforme Linux, Windows e MacOS, al sito

http://www.r-project.org/

(clickare su CRAN (sotto Download nella finestra di sinistra), scegliere un sito mirror (cioè un sito in cui ci sia una copia identica), scegliere il sistema operativo e seguire le istruzioni.
Inoltre, è disponibile nelle librerie il manuale **Statistica con R** di S. Iacus e G. Masarotto, edizioni McGrawHill, che contiene anche il CD con il programma installabile, per Windows e McOs.

3.1 Primi comandi

Sotto Linux il programma si lancia digitando `R` in un terminale e dando INVIO. A questo punto il terminale diventa il terminale di `R`, e infatti cambia anche il prompt che diventa `>`.

Il primo uso è quello di una calcolatrice scientifica, con sintassi matematica: `2 + 3*(5/4 - 2/(1+23))` ha le ovvie precedenze di calcolo determinate dalle parentesi. Sono disponibili tutte le principali funzioni, in particolare:

x**y	eleva x alla y
x^y	eleva x alla y
sqrt(x)	radice quadrata di x
exp(x)	esponenziale di x
log(x)	logaritmo naturale di x
log2(x)	logaritmo in base 2 di x
log10(x)	logaritmo in base 10 di x
sin(x)	seno di x
asin(x)	arcoseno di x
etc	etc

3.1.1 help

Per ottenere informazioni sui comandi si può utilizzare l'aiuto in linea:

```
help.search('nome')
```

mostra l'elenco di tutte i comandi che hanno a che fare con `nome`. L'output dell'help è gestito dal programma `less`, dunque si può scorrere il testo con le frecce, si possono cercare parole con \parola INVIO e poi "n" per le successive, e soprattutto, ricordatevi che si esce con "q".

```
help(comando)
```

mostra la pagina di manuale relativa al comando, sempre all'interno di `less`. Un modo più comodo per vedere l'help è di lanciare

```
help.start()
```

Si aprirà una finestra di Netscape con il manuale completo, e le successive chiamate di `help(comando)` manderanno l'output sulla finestra di Netscape.

3.2 Variabili e costanti

`R` permette di assegnare i valori a delle variabili, ad esempio `x = 7` e `y = 8`. Per leggere il valore assegnato alla variabile basta scrivere la variabile e premere INVIO.

```
> x = 7                    assegna 7 a x
> yps = 8                  assegna 8 a ups
> x                        mostra il valore di x
[1] 7                      il numero tra parentesi e' l'indice
> yps                      mostra il valore di yps
[1] 8
> x + yps                  calcola x + yps e mostra il risultato
[1] 15
```

In molti programmi per assegnare un valore ad una variabile non si usa il simbolo di uguale (che viene invece riservato per le "affermazioni"). Anche `R` permette di evitare l'uso del simbolo `=`, che può essere sostituito con i simboli piú espressivi `->` e `<-` :

```
> x <- 7                assegna 7 a x
> 8 -> yps              assegna 8 a ups
> x + yps
[1] 15
```

Esistono nomi proibiti per le variabili, tutti queli che già significano qualche altra cosa, in particolare

```
> pi
[1] 3.141593
```

è il valore di π.

3.3 Vettori

3.3.1 Creazione di vettori

```
> c(2,6,4,3) -> x       crea il vettore (2,6,4,3) e lo assegna a x
> x                     mostra il contenuto di x
[1] 2 6 4 3
> x[1]                  mostra il primo elemento di x
[1] 2
> x[2]                  mostra il secondo elemento di x
[1] 6
> x[3]                  mostra il terzo elemento di x
[1] 4
```

nell'esempio viene creato un vettore con la funzione `c()`, e esso viene assegnato come valore a `x`. Per richiamare ogni singolo elemento del vettore basta scrivere `x[posto]`. Un esempio più sofisticato è

```
> x[c(2,4)]             mostra il secondo ed il quarto elemento di x
[1] 6 3
> x[c(3,2)]             mostra il terzo e il secondo elemento di x
[1] 4 6
```

che mostra il secondo e il quarto elemento del vettore `x`

I vettori possono essere **concatenati**

```
> y <- c(9,8,7)
> c(x,y)                    vettore x seguito da y
[1] 2 6 4 3 9 8 7
> c(y,x)                    vettore y seguito da x
[1] 9 8 7 2 6 4 3
```

Un'altra funzione utile per costruire vettori è `seq()`. Si usa in tre modi:

```
> seq(5)                    interi da 1 a 5
[1] 1 2 3 4 5
> seq(2,5)                  interi da 2 a 5
[1] 2 3 4 5
> 2:5                       equivalente al comando seq(2,5)
[1] 2 3 4 5
> seq(0.3, 1.7, 0.5)        numeri da 0.3 a 1.7 con incrementi di 0.5
[1] 0.3 0.8 1.3
```

Con i vettori si possono fare le operazioni che si fanno con i numeri, ad esempio

```
> x
[1] 2 6 4 3 0
> y
[1] -1  0  2  3  0
> exp(-x)                   esponenziale di -x, elemento per elemento
[1] 0.135335283 0.002478752 0.018315639 0.049787068 1.000000000
> x + y                     somma elemento per elemento
[1] 1 6 6 6 0
> x * y                     prodotto elemento per elemento
[1] -2  0  8  9  0
> x/y                       rapporto
[1]  -2 Inf   2   1 NaN
```

Notare: Inf vuol dire che il risultato è $\pm\infty$, NaN vuol dire che l'operazione non ha significato (NaN = Not A Number).

3.3.2 Selezione avanzata degli elementi

```
> x <- seq(2,8)                   in x vengono messi gli interi da 2 a 8
> x[x<4]                          estrae gli interi minori di 4
[1] 2 3

> x[x < 7 & x >= 4 ]              estrae i valori minori di 7 e maggiori
[1] 4 5 6                         o uguali a 4
```

La & è il simbolo dell'operatore logico AND, mentre | è il simbolo dell'operatore logico OR.

```
> x[x > 7 | x <= 4 ]               estrae i valori maggiori di 7 e i valori
[1] 2 3 4 8                        minori o uguali a 4
```

Più approfonditamente, la disuguaglianza $x < 4$ viene risolta verificando quali valori sono minori di 12 e quali no

```
> x<4
[1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

TRUE (abbr. T) vuol dire vero, cioè il corrispondente valore è minore di 4, FALSE (abbr. F) vuol dire falso, cioè il corrispondente valore non è minore di 4. I valori T o F sono dei tipi di dati particolari, infatti non sono numeri e non sono nemmeno caratteri (anche se lo sembrano): sono i due soli possibili valori **logici** o **boolenai**.

```
> x[x<4]
```

è equivalente a

```
> x[c(T,T,F,F,F,F,F)]
```

che stampa solo i valori in corrispondenza di `T`. Ancora:

```
> x < 7 & x >= 4
```

dà un vettore di valori booleani, con T dove $4 \leq x < 7$, mentre

```
> x > 7 | x <= 4
```

dà un vettore di valori booleani, con T dove $x > 7$ **oppure** $x \leq 4$.

Esempio: costruire il vettore degli interi da 1 a 100 ed estrarre i numeri divisibili per 3 o divisibili per 5

```
> x <- seq(100)
> x[ floor(x/3) == x/3 | floor(x/5) == x/5]
[1]   3   5   6   9  10  12  15  18  20  21  24  25  27  30  33  35  36  39  40
[20] 42  45  48  50  51  54  55  57  60  63  65  66  69  70  72  75  78  80  81
[39]  84  85  87  90  93  95  96  99 100
```

La funzione `floor()` calcola la parte intera (in altri programmi si chiama int(), qui no, purtroppo). Dunque le istruzioni dentro sono rispettivamente: x è divisibile per 3 e x è divisibile per 5 (infatti la parte intera di x/3 è ugule a x/3 solo se x è divisibile per 3).
Attenzione: i simboli $<, >, <=, >=$ si usano, rispettivamente, per testare le affermazioni minore, maggiore, minore o uguale, maggiore o uguale. Per testare l'uguaglianza, non si usa il simbolo $=$ (che si confonderebbe con l'assegnazione), ma il simbolo $==$

Esempio più utile: sia `x` un vettore di misure, e `freq` il vettore delle frequenze assolute:

```
> x <- c(2,9,5,7,3)
> freq <- c(3,4,8,9,3)
```

Estrarre i dati che hanno frequenze maggiori o uguali a 8:

```
> x[ freq >= 8]
[1] 5 7
```

Estrarre le frequenze dei dati compresi tra 3 e 7

```
> freq[ x >=3 & x <= 7]
[1] 8 9 3
```

3.3.3 Assegnazione parziale

Si può modificare ogni elemento di un vettore, semplicemente assegnandogli un nuovo valore:

```
> x <- c(2,9,5,7,3)
> x[4] <- 18
> x
[1]  2  9  5 18  3
```

Naturalmente si può usare la possibilità di estrarre valori anche per modificarli:

```
> x <- c(2,9,5,7,3)
> x[x>6] = 10
> x
[1]  2 10  5 10  3
```

sostituisce 10 ai valori più grandi di 6.
Attenzione: quando R compie operazioni con vettori di lunghezza differente, "allunga" il piú corto fino alla dimensione del piú lungo, ripetendone ciclicamente gli elementi. Dunque il comando di prima è equivalente a

```
> x[x>6] = c(10,10)
```

poiché i valori maggiori di 6 sono esattamente due.
Un altro esempio: sia

```
> x <- c(1,2,3,4,3,2,1)
> y <- c(3,4,5,6,5,4,3)
```

per sostiture ai valori di x uguali a 3 i corrispondenti valori di y si può usare

```
> x[x==3] <- y[x==3]
> x
[1] 1 2 5 4 5 2 1
```

3.4 Comandi per il disegno

Se a e b sono due vettori della stessa lunghezza, i seguenti comandi equivalenti

```
plot(a,b)
plot(x=a,y=b)
plot(y=b,x=a)
```

disegnano i punti di ascissa a e ordinata b, a piccoli cerchi.

```
plot(a,b,pch=2)
```

cambia il simbolo usato per disegnare i punti con l'opzione pch; in particolare pch=20 sono pallini neri. Per disegnare i punti uniti da linee si deve usare

```
plot(a,b,type='l')
```

L'opzione `type='h'` disegna linee verticali per ogni punto. Per modificare il colore, usare l'opzione col=2 (rosso), col=3 (verde), col=4 (blu), etc... L'istruzione plot crea il disegno definendo un rettangolo che ha come estremi in ascissa il massimo e il minimo di a e in ordinata il massimo e il minimo di b. Questo comportamento standard si può cambiare specificando esplicitamente i limiti delle ascisse e delle ordinate, con le opzioni `xlim=c(x1,x2), ylim=c(y1,y2)`, dove x1 x2 sono gli estremi in ascissa e y1 y2 sono gli estremi in ordinata.

Ogni volta che si usa l'istruzione `plot`, il disegno precedente viene cancellato. Spesso però vorremo aggiungere ad un disegno altra cose. Si utilizzano due istruzioni che hanno la stessa sintassi di base del plot

```
points(a,b)      # che disegna i punti
lines(a,b)       # che disegna linee
```

Esistono molte altre opzioni che permettono ulteriori stili di stampa, che gestiscono i titoli etc. Per esse consultare il manuale con l'istruzione `help(plot)`.

3.5 Lettura e scrittura di dati

3.5.1 Lettura

R è un programma per la statistica, e come tale può manipolare grandi quantità di dati. Dunque può leggere e scrivere file di dati. L'istruzione più semplice è

```
scan('nomefile')
```

che crea un vettore con i dati presenti nel file "nomefile" (che deve contenere solo numeri) nell'ordine in cui compaiono. Per poterli utilizzare vanno memorizzati in una variabile

```
scan('nomefile') -> a
```

Attenzione: il programma usa la notazione scientifica standard, che utilizza il "." (punto) per separare i decimali. Molti programmi permetto la "localizzazione", nel senso che si adattano alle convenzioni simboliche dello stato in cui vengono utilizzati; in particolare in Italia spesso si utilizza la "," per separare i decimali. Se il file usa la virgola, per leggerlo sarà necessario dare a `scan` l'opzione dec="," in modo che carichi correttamente i numeri (in cui alla fine apparirà il ".").

Per leggere tabelle di dati su utilizza

```
read.table(''nomefile'') -> a
```

In tal caso, però, la variabile a non conterrà una tabella, ma una struttura di dati più complicata un **data frame** (in altri programmi si chiama anche data matrix o data set; in sostanza è un vettore o una matrice di dati con gli elementi collegati ad un nome: dunque il dato è composto da oggetti matematici e nomi). Ad esempio, se il file contiene

```
100     0    0
 95.1   4.9  0
 84.8  13.8  1.4
 75.2  21.5  3.3
```

l'istruzione precedente dà

```
> a
    V1   V2  V3
1  100    0   0
2 95.1  4.9   0
3 84.8 13.8 1.4
4 75.2 21.5 3.3
```

come si vede, non ci sono solo i dati, ma anche dei nomi "standard" per le colonne di dati ottenute e un numero che indica la riga. I nomi delle colonne potevano essere già contenuti del file, ad esempio

```
 c0   c1    c2
100    0     0
 95.1  4.9   0
 84.8  13.8  1.4
 75.2  21.5  3.3
```

in tal caso l'istruzione corretta di lettura è

```
> read.table('nomefile',header=T) -> a
> a
     c0   c1  c2
1 100.0  0.0 0.0
2  95.1  4.9 0.0
3  84.8 13.8 1.4
4  75.2 21.5 3.3
```

l'opzione header=T (cioè "testata del file" = vero) dice al comando che la prima riga del file contiene i nomi delle colonne. I numeri nella prima colonna sono i numeri che identificano ogni riga (numeri delle righe). Nel file possono essere scritti anche i nomi delle righe:

```
    c0   c1    c2
r1 100    0     0
r2  95.1  4.9   0
r3  84.8  13.8  1.4
r4  75.2  21.5  3.3
```

In tal caso read.table dà:

```
> read.table('nomefile',header=T) -> a
> a
      c0   c1  c2
r1 100.0  0.0 0.0
r2  95.1  4.9 0.0
r3  84.8 13.8 1.4
r4  75.2 21.5 3.3
```

e i nomi delle righe sono effettivamente r1 r2 r3.

Come si possono isolare i singoli vettori colonna dei dati letti? Essi hanno semplicemente i nomi `a$c0 a$c1 a$c2`, infatti `> a$c0` dà

```
[1] 100.0  95.1  84.8  75.2
```

che è esattamente il vettore prima colonna. Un modo per evitare di dover usare la sintassi a$c0 è quello di usare

```
attach(a)
```

che rende accessibili le colonne direttamente con il loro nome:

```
> attach(a)
> c0
[1] 100.0  95.1  84.8  75.2
```

Può essere utile trasformare un data frame in una matrice; l'istruzione è

```
> data.matrix(a) -> b
> b
      c0   c1  c2
r1 100.0  0.0 0.0
r2  95.1  4.9 0.0
r3  84.8 13.8 1.4
r4  75.2 21.5 3.3
```

Sembra identico a prima, ma ora funziona il comando `unname` che elimina i nomi delle righe e delle colonne (su "a" non funzionerebbe)

```
> unname(b) -> ma
> ma
      [,1] [,2] [,3]
[1,] 100.0  0.0  0.0
[2,]  95.1  4.9  0.0
[3,]  84.8 13.8  1.4
[4,]  75.2 21.5  3.3
```

A questo punto le colonne e le righe di `ma` si richiamano semplicemente:

```
> ma[,3]
[1] 0.0 0.0 1.4 3.3
> ma[2,]
[1] 95.1  4.9  0.0
```

In altre parole, mentre `a` è un data frame, `ma` è una matrice, dunque permette operazione che un data frame non consente.

3.5.2 Scrittura

Per scrivere su file un vettore:

```
write(y, file = 'dati',ncolumns=1)
```

crea o riscrive il file dati, scrivendo in una sola colonna gli elementi del vettore y.
Il modo più semplice per scrivere una tabella è creare un data frame e poi scriverlo.

```
> x <- seq(0,1,.1)
> y <- exp(-x)
> da <- data.frame(x,y)      crea un data frame di elementi x e y
> da
     x         y
1  0.0 1.0000000
2  0.1 0.9048374
3  0.2 0.8187308
4  0.3 0.7408182
5  0.4 0.6703200
6  0.5 0.6065307
7  0.6 0.5488116
8  0.7 0.4965853
9  0.8 0.4493290
10 0.9 0.4065697
11 1.0 0.3678794
> write.table(da,file='dati')
```

a questo punto nel file ci sarà la corrispondente tabella con i nomi delle colonne x e y.

3.6 Esecuzione

Spesso, durante complesse sequenze di analisi, è conveniente scrivere in un file separato i comandi che vogliamo far eseguire a R. Questo permette di ripeterli in blocco, senza doverli riscrivere uno ad uno, soprattutto mentre si sperimentano delle modifiche. In altre parole, invece che limitarci a dare i comandi, è conveniente scriverli in sequenza in un file, e poi dire a R di eseguire i comandi scitti in quel file.

Il file deve essere un file di testo. Sotto linux conviene dare ad un file del genere l'estensione .R in modo che emacs lo riconosca come tale e attivi le modalità di colorazione, indentazione e controllo sintattico utili per le istruzioni di R.

Una volta creato il file (che sarà un "programma", ovvero una sequenza di istruzioni che R "interpreta" ed esegue), per eseguirlo basta dare il comando

```
source(''nomefile'')
```

"source" sta per "sorgente" che è il nome tipico dei file di testo che poi, compilati, creano i programmi veri e propri.

Ad esempio, il file `prova.R` contiene le seguenti linee di codice

```
x <- seq(0,2*pi,0.01)          # assegna i valori a x
plot(x,sin(x),type='l')        # diegna il seno
print('valore in 1',quote=F)   # scrive una scritta
print(sin(x)[x==1])            # scrive un valore
```

disegna il grafico di sin(x) tra 0 e 2π, e scrive su terminale il valore del seno per $x = 1$. L'istruzione `print('valore in 1',quote=F)` scrive su terminale la **stringa** "valore in 1", l'opzione `quote=FALSE` elimina le virgolette.
NB. Le virgolette (singole o doppie) distinguono le stringhe, cioè sequenze arbitrarie di caratteri, dalle variabili. Dunque `c('aa','bb')` crea un vettore di stringhe di elementi 'aa' e 'bb'; `c(aa,bb)` concatena, se esistono, i vettori aa e bb.

In un file si possono inserire commenti, sia in righe a parte sia in una riga di comandi, basta farli precedere dal carattere #.

3.7 Prime operazioni statistiche con i vettori

3.7.1 somma

Sia x un vettore di dati

`min(x)`	dà il minimo
`max(x)`	dà il massimo
`sum(x)`	somma tutti gli elementi
`length(x)`	dìl numero degli elementi
`mean(x)`	calcola la media (cioè `sum(x)/length(x)`
`sd(x)`	deviazione standard
`var(x)`	varianza (cioè `sd(x)`^{2})

Tutte le istruzioni precedenti danno come risultato un singolo numero; ci sono istruzioni che danno risultati più complessi:

```
> quantile(x)
```

dà i valori dei quantili: 0% 25% 50% 75% 100%; in altre parole mostra il minimo, il primo quartile, la mediana, il terzo quartile, il massimo. Si possono ottenere anche altri quantili:

```
> quantile(x,c(.2,.6,.8))
```

mostra i quantili 0.2 = 20%, 0.6=60%, 0.8=80%.

```
> sort(x)
```

ridà i valori di x in ordine crescente (statistica ordinata)

```
> cumsum(x)
```

somma cumulata: costruisce un vettore che ha come i-esimo elemento la somma fino all'i-esimo elemento di x.

3.7.2 Un esempio: integrazione numerica

Per integrare numericamente una funzione esistono tecniche piú raffinate della semplice somma delle aree dei rettangoli (è un argomento standard dell'analisi numerica). Noi ci limiteremo appunto alla somma delle aree dei rettangolini.

```
dx <- 0.1                  # e' la lunghezza degli intervallini
x <- seq(0,2*pi,dx)        # punti da 0 a 2 pigreco
plot(x,cos(x),type=''l'')  # disegna il coseno
y <- cumsum(cos(x))*dx     # integrazione numerica
lines(x,y,col=2)           # disegna la funzione integrale
```

3.8 Alcuni plot statistici

La statistica a cinque valori (i quantili 0, 1/4, 1/2, 3/4, 1) viene rappresentata graficamente in un `boxplot`:

```
> boxplot(x)
```

dà un grafico che ha in ordinata i valori di x, e disegna un rettangolo in corrispondenza del primo e del terzo quartile, con in mezzo la linea della mediana. Le due linee separate che compaiono sono, rispettivamente, a distanza 1.5 volte la **distanza interquartile** dalle linee di primo e di terzo quartile. I dati fuori da tale intervallo vengono rappresentati come punti singoli e sono considerati come dati estremi.

```
> hist(x)
```

disegna l'istogramma dei dati in x. Il numero delle classi è scelto da `R`

```
> hist(x,0:9)
```

usa il vettore 0,1,2,3,4,5,6,7,8,9 per identificare gli estremi delle classi.

3.9 Istruzioni per le principali distribuzioni

Ci sono alcune istruzioni che riguardano le principali distribuzioni di probabilità. Per ogni distribuzione esistono 4 funzioni, con una sintassi standard.

Sia "distr" il nome di una particolare distribuzione (vedi l'elenco successivo: ad esempio "distr" può essere norm, binom, exp, ...), le funzioni per "distr" sono

funzione	azione
rdistr(n,parametri)	se n è un numero, genera n numeri a caso con quella distribuzione se n è un vettore, genera un numero a caso per ogni elemento del vettore
ddistr(x,parametri)	dà la densità di probabilità in x
pdistr(x,parametri)	dà la probabilità di estrarre un numero $\leq x$ (è la funzione di distribuzione)
qdistr(x,parametri)	dà il valore di y per cui la probabilità di estrarre un numero $\leq y$ sia x (funzione quantile: è l'inversa della funzione di distribuzione)

I parametri servono a specificare la distribuzione: ad esmpio per quella uniforme i parametri saranno gli estremi dell'intervallo in cui vengono estratti i numeri; per la normale saranno media e deviazione standard, etc. Spesso le funzioni possono essere usate senza specificare i parametri, in tal caso verrà usata una particolare distribuzione: ad esempio nel caso uniforme non mettere i parametri equivale a considerare la distribuzione su $[0, 1]$; nel caso normale equivale a considerare la normale standard $\mu = 0, \sigma = 1$. La seguente tabella riporta i nomi delle distribuzioni, i parametri necessari e se esistono anche i parametri di default (cioè i parametri assunti nel caso non vengano specificati).

distribuzione	pagina di manuale	nome delle funzioni	parametri	parametri di default
uniforme	Uniform	unif	estremi dell'intervallo	0, 1
normale	Normal	norm	μ e σ	$\mu = 0$, $\sigma = 1$
esponenziale	Exponential	exp	tasso = 1/media	1
binomiale	Binomial	binom	numero delle prove e $p(0)$	(no)
poissoniana	Poisson	pois	media λ	(no)
student	TDist	y	gradi di libertà n	(no)
$\mathcal{X}^2$	Chisquare	chisq	gradi di libertà n	(no)
F	FDist	f	gradi di libertà n_1, n_2	(no)

Qualche altra parola sulla distribuzione binomiale:

```
> dbinom(2,4,.2)    probabilita' di evere 2 successi su 4 lanci,
[1] 0.375           se la probabilita' di un successo e' 0.2
```

È equivalente a pensare P(T) = 0.2 e a chiedersi la probabilità di avere 2 T su 4 lanci. Per imitare il singolo lancio, con P(T)=0.2, si usa

```
> rbinom(1,1,.2)
[1] 0
```

che va interpretato così : 1 equivale a T (cioè successo) e 0 equivale a C (insuccesso).
Per generare una sequenza di 7 lanci:

```
> rbinom(1:7,1,.2)
[1] 1 0 0 0 1 0 0
```

3.9.1 Campionamento

La funzione `sample` serve ad estrarre da un vettore una sequenza casuale dei suoi elementi, con ripetizione o senza:

```
> sample(4,3)              estrae 3 elementi tra 1,2,3,4
[1] 2 1 3
> sample(4,3,replace=T)    estrae 3 volte un elemento tra 1,2,3,4
[1] 2 4 4
```

la differenza tra i due esempi è che nel primo caso un elemento estratto non viene "rimesso dentro", nel secondo caso si (`replace=T` vuol dire rimpiazza=VERO).

```
> sample(4,4)             genera una permutazione casuale di 1,2,3,4
[1] 4 2 1 3
> sample(4)               come sopra
[1] 3 4 1 2
> sample(c('A','C','G','T'),10,replace=T)      estrae 10 volte un simbolo
[1] "C" "C" "C" "A" "G" "T" "G" "G" "C" "T"    da ACGT
```

Capitolo 4

Introduzione ai test statistici

4.1 Un esempio introduttivo

Ricordo che la distribuzione binomiale descrive il numero di teste che escono nel lancio di N monete in cui la probabilità di testa sia p.

`dbinom(numero_di_T, N, p)` calcola la probabilità che escano `numero_di_T` teste per il lancio di N monete, con probabilità di testa uguale a p. La "d" di dbinom sta per densità.
`dbinom(2,10,0.5)` dà la probabilità che escano 2 teste per il lancio di 10 monete con $p(T) = p$.
`dbinom(c(2,3,4),10,0.5)` dà la probabilità che escano 2,3,4 teste per il lancio di 10 monete con $p(T) = p$.

Problema.

In un gioco con una moneta si vince se esce testa, si perde se esce croce. Il banco garantisce che è un gioco non truccato, cioè che le probabilità di uscita di T e C sono uguali ad 1/2. Osservo 20 mani di gioco, in cui T esce 6 volte. La frequenza relativa di T è dunque $6/20 = 0.3 = 30\%$, decisamente diversa da quella asserita dal banco.

Il banco mente o no?

La risposta è opinabile: se fossi una persona diffidente tenderei a pensare di si, e mi rifiuterei di giocare. D'altra parte non giurerei che la moneta è truccata.

Un **test statistico** è una procedura che tenta di rispondere alla domanda "quanto sono in accordo i dati sperimentali con l'interpretazione che ne stiamo dando (teoria)"? Dunque nel caso precedente risponde alla domanda:

quanto è credibile che escano 6 teste su 20 lanci per una moneta non truccata?

[In realtà la domanda alla quale vorremmo rispondere è "quanto è credibile la nostra teoria (moneta non truccata) se abbiamo visto 6 teste su 20 lanci"? Il test statistico non può rispondere a questa domanda, ma risponde a quella precedente; all'atto pratico considereremo la risposta del test come un'affermazione sulla teoria]

Il test statistico dà un valore numerico (detto **p-value**, in italiano valore di p del test) alla corrispondenza tra i dati e l'affermazione che la moneta non sia truccata.

L'ipotesi alla quale vogliamo assegnare un valore prende il nome di **ipotesi nulla** e si indica con $\mathbf{H}_0$. Nel caso in esame, la formulazione matematica dell'ipotesi nulla è

$$H_0 : \quad p(T) = 0.5$$

che si legge: l'ipotesi nulla è che la probabilità di testa è 0.5, cioè che la moneta non è truccata

Il modo di ragionare per dare un valore ad H_0 è sostanzialmente il seguente: il dato che osservo è abbastanza distante dal dato teorico (che sarebbe 10 teste su 20 lanci); quanto è probabile osservare un dato cosí "strano" rispetto a quello medio, se davvero valesse $p(T) = 0.5$?

Questa domanda va specificata meglio. Infatti, potrei chiedermi con quale probabilità osservo 6 teste su 20 lanci. Questo risultato è poco probabile, infatti, usando la distribuzione binomiale,

$$P(6) = dbinom(6, 20, 0.5) = 0.03696442$$

D'altra parte, anche la probabilità di osservare un risultato ottimale, cioè esattamente 10 teste su 20, non ha una probabilità molto grande:

$$P(10) = 0.176197$$

Dunque la domanda va formulata meglio: quanto è probabile osservare un dato "brutto" come 6, o più brutto, se davvero valesse $p(T) = 0.5$? In questo caso "più brutto" di 6 vuol dire due cose: che esca un numero di teste minore o uguale a 6, ma anche che esca un numero di teste che sia maggiore o uguale a 14. Infatti, 14 dista dalla media (che è 10) quanto 6. Quindi 14 è "brutto" quanto 6, come 15 è "brutto" quanto 5.

La procedura per dare un valore all'aderenza dei dati all'ipotesi H_0 è quindi quella di calcolare la somma delle probabilità dei casi "brutti" quanto 6:

$$\begin{aligned} \text{p-value} &= P(0) + P(1) + \cdots + P(6) + \ P(14) + P(15) + \cdots + P(20) = \\ &= sum(dbinom(0:6, 20, 0.5)) + sum(dbinom(14:20, 20, 0.5)) = 0.1153183 \end{aligned}$$

Ci resta da valutare il valore ottenuto. Convenzionalmente, nella pratica, si utilizzano i valori di riferimento 0.05, 0.01, 0.001.

Se $p > 0.05$, è abbastanza grande da non smentire l'ipotesi nulla, che dunque viene considerata credibile

Se $0.01 < p < 0.05$ i dati smentiscono **significativamente** l'ipotesi H_0

Se $0.001 < p < 0.01$ i dati smentiscono in modo **molto significativo** l'ipotesi H_0

Se $p < 0.001$ c'è un'evidenza statistica **estremamente significativa** che H_0 sia falsa

Nel nostro esempio p è decisamente grande. Se la moneta non è truccata, in circa il 12% dei casi ci capiterebbe di osservare un dato che ci ha insospettito tanto quanto quello che abbiamo visto. La statistica ci dice che non è il caso di insospettirsi...

Se invece avessimo ossevato 6 test su 30 lanci, avremmo ottenuto $p = 0.001430906$, quindi saremmo stati giustamente sospettosi sull'affermazione del banco.

Nel caso di 6 teste su 40 lanci saremmo stati estremamente sospettosi, infatti $p = 8.364585e-06$

Quello che vi ho descritto è chiamato **test binomiale**, o test binomiale esatto. Per piccoli numeri si può fare a mano. Già in casi importanti, come la verifica delle leggi di Mendel, è necessario l'uso dei calcolatori. In R c'è una funzione che fa il test binomiale per noi, e ha la seguente sintassi

`binom.test(teste osservate,`N`,`p`)`

dove p è la probabilità asserita dall'ipotesi nulla. Nel nostro caso, l'output di binom.test(6,20,0.5) è

```
 Exact binomial test

data:  6 and 20
number of successes = 6, number of trials = 20, p-value = 0.1153
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.1189316 0.5427892
sample estimates:
probability of success
                   0.3
```

Credo sia abbastanza chiaro. L'unico punto da spiegare è l'**intervallo di fiducia (confidenza)** al 95 percento. È l'intervallo dei valori di $p(T)$ che non sono smentiti dai dati osservati (con una soglia di 0.05). In altre parole sono i valori di $p(T)$ compatibili con i dati al 95%

Quelli che seguono i p-value per il test binomiale con $H_0 : p(T) = 1/2$ per vari dati:

teste	lanci	rapporto	p-value
6	20	0.3	0.1153
12	40	0.3	0.01659
18	60	0.3	0.002670
24	80	0.3	0.0004515

Il risultato del test **dipende molto** dal numero delle prove! Osservare una frequenza 0.3 lanciando una moneta non truccata è ragionevole se si sono fatti pochi lanci, è proticamente impossibile se si sono fatti molti lanci.

4.2 Test del χ^2 per date probabilità

Seguendo lo stesso principio dell'esempio precedente, sono stati creati diversi test per verficare delle ipotesi statistiche. Qui faccio solo l'esempio classico delle leggi di Mendel.

Problema: vengono incrociate delle piante eterozigoti rispetto al colore del seme (viola/biaco, con viola dominante). Si osservano 705 semi viola e 224 bianchi. Questi dati sono in accordo con l'ipotesi che viola/bianco siano renotipi segregati con probabilità 3/4 1/4?

In termini statistici l'ipotesi nulla è che gli elementi del vettore `c(705,224)` siano stati estratti con probabilità `c(3/4,1/4)`. Per la verifica dell'ipotesi nulla si può usare la funzione `chisq.test(dati,p=probabilita')`:

```
> chisq.test(c(705,224),p=c(3/4,1/4))

 Chi-squared test for given probabilities
```

```
data:  c(705, 224)
X-squared = 0.3907, df = 1, p-value = 0.5319
```

Essendo il p-value molto grande si conclude che l'ipotesi nulla è daccordo con i dati.

4.3 Test binomiale e χ^2

Il problema di prima può anche essere affrontato con il test binomiale: in fondo ci stiamo chiedendo se da una moneta 3/4 - 1/4 è ragionevole che escano 705 teste su 705+224 lanci:

```
> binom.test(705,705+224,p=3/4)

        Exact binomial test

data:  705 and 705 + 224
number of successes = 705, number of trials = 929, p-value = 0.5446
alternative hypothesis: true probability of success is not equal to 0.75
95 percent confidence interval:
 0.7300398 0.7860693
sample estimates:
probability of success
             0.7588805
```

Il p-value è diverso dal precedente. Il motivo è che rispetto all'ipotesi binomiale, il test del χ^2 effettua diverse approssimazioni, che sono ragionevoli se il numero di dati è molto grande.

Si guardi il p-value per i seguenti (finti) dati sui fenotipi:

viola	bianchi	binom.	χ^2
705	224	0.5446	0.5319
7020	2281	0.2921	0.2893
702012	234450	0.4247	0.4247

Il χ^2 è molto piú usato per vari motivi. Il primo è storico/pratico: il test del χ^2 racchiude tutte le informazioni in due numeri: il cosiddetto valore del χ^2 (`X-squared`), che è calcolabile a mano, e il numero (intero) di "gradi di libertà" (`df`); in corrispondenza di questi due valori i p-value sono stati trascritti in tabelle. Al contrario, il test binomiale richiede i calcolatori e non è sensato trascrivere i p-value su tabella.

Un secondo motivo spiega perché continui ad essere usato anche nell'epoca dei calcolatori: il χ^2 è un test che può esser usato in molti casi, mentre il test binomiale risponde solo alla domanda "quante teste su tot lanci". Ad esempio:

Problema: si osservano le seguenti distribuzioni per i genotipi di un incrocio di piante omozigoti: AA=310, Aa=521, aa=234. La teoria mi dice che tali valori dovrebbero essere in proporzione a 1/4, 1/2, 1/4. La teoria è in accordo con i dati?

Non posso fare un test binomiale, perché si tratta di una distribuzione multinomiale. Posso però fare un χ^2:

```
> chisq.test(c(310,521,234),p=c(1/4,1/2,1/4))
```

```
        Chi-squared test for given probabilities

data:  c(310, 521, 234)
X-squared = 11.3437, df = 2, p-value = 0.003442
```

Dal p-value si deduce che c'è una discrepanza molto significativa tra la teoria e i dati.

Quando può essere usato, il test binomiale ha invece un indubbio vantaggio: ci dà anche i valori dell'intervallo fiduciale, cosa che il χ^2 non può fare.

Problema: state raccogliendo dati da seggi campione per una proiezione, su 12000 voti validi espressi, contate 5880 voti per la coalizione A e 12000–6060=5940 voti per B, che corrispondo rispettivamente alle percentuali 50.5% e 49.5%. Con che grado di sicurezza potete affermare che la colazione A ha vinto?

Il test binomiale dà il seguente risulato:

```
> binom.test(6060,12000,p=1/2)
...
95 percent confidence interval:
 0.4960125 0.5139851
```

(ho dato p=1/2, ma non è importante: non ho ipotesi da fare, voglio solo conoscere l'intervallo di fiducia). Come si vede, al 95% ho una "forchetta" 0.496–0.514, dunque non posso affermare che A ha vonto al 95%.

Arrivano ulteriori dati: su 50000 voti, A ne ha ottenuti 25300 (pari a 50.6 %). Cosa posso affermare?

Il test binomiale dà

```
> binom.test(25300,50000)
95 percent confidence interval:
 0.5016075 0.5103918
```

dunque con una fiducia del 95% posso affermare che A ha vinto le elezioni. Questo livello di fiducia potrebbe essere troppo basso (vuol dire che mi sbaglio in media una volta su 20). Posso chiedere al test binomale l'intervallo ad altri livelli di fiducia:

```
> binom.test(25300,50000,conf.lev=0.99)
99 percent confidence interval:
 0.5002303 0.5117685
```

4.3.1 Test del χ^2 per l'indipendenza

Quella che segue è la tabella in cui sono riportate le frequenze dei semi di piselli rispetto a due coppie di caratteri dominante/recessivo:

	giallo	verde
liscio	315	108
rugoso	101	32

Se i caratteri sono indipendenti, le righe (o le colonne) dovrebbero essere proporzionali (ipotesi nulla), cioè la segregazione dei due caratteri dovrebbe essere indipendente. Il rapporto liscio/rugoso tra i semi gialli è 3.1188, tra i semi verdi è 3.375; tali valori sono piuttosto differenti.

Domanda: sono abbastanza differenti da farci sospettare che la segregazione non sia indipendente (come potrebbe accadrebbe se i geni fossero abbastanza vicini sullo stesso cromosoma).

Il test di indipendenza è ancora il chisq.test. Si procede come segue:

1. si crea una matrice che contiene la tabella `matrice <- matrix(c(315,101,108,32),2,2)` (questa istruzione trasforma un vettore in una matrice, semplicemente mettendo in colonna gli elementi; va specificato il vettore, il numeri di righe, il numero di colonne).

2. si invoca il test: `chisq.test(matrice)`

Il risultato è:

```
> matrice <- matrix(c(315,108,101,32),2,2)
> matrice
     [,1] [,2]
[1,]  315  101
[2,]  108   32
>
>
> chisq.test(matrice)

 Pearson's Chi-squared test with Yates' continuity correction

data:  matrice
X-squared = 0.0513, df = 1, p-value = 0.8208
```

Anche qui la teoria è in accordo con i dati.

4.4 Le istruzioni di R per l'output dei test

L'output di un test è un dato strutturato con vari campi, che possono essere richiamati individualmente. Si consideri il seguente esempio:

```
> risultato <- binom.test(25300,50000,conf.lev=0.99)
> risultato

        Exact binomial test

data:  25300 and 50000
number of successes = 25300, number of trials = 50000, p-value =
0.007388
alternative hypothesis: true probability of success is not equal to 0.5
99 percent confidence interval:
 0.5002303 0.5117685
sample estimates:
probability of success
                 0.506
```

La variabile `risultato` contiene tutto quello che il test afferma. Per richiamare alcuni dei singoli dati del risultato si usa la sintassi seguente:

```
> risultato\$p.value    #e' il p-value
[1] 0.007387835

> risultato\$conf.int    #e' l'intervallo di confidenza
[1] 0.5002303 0.5117685
attr(,"conf.level")
[1] 0.99

> risultato\$conf.int[1]    #primo elemento dell'intervallo di confidenza
[1] 0.5002303

> risultato\$conf.int[2]    #secondo elemento dell'intervallo di confidenza
[1] 0.5117685
```

Per gli altri dati consultare il manuale `help(binom.test)`.

Analogamente, per il testo del χ^2 si usa:

```
> chisq.test(c(12,24,9),p=c(1/4,1/2,1/4))\$p.value  #p-value del test
[1] 0.7408182
```

Maggiori dettagli nel manuale `help(chisq.test)`.

4.5 Correlazione

Fin'ora ci siamo occupati della descrizione statistica di un insieme di dati. Estremamente interessanti ed utili per le applicazioni sono i metodi statistici che riguardano piú insiemi di dati e le loro relazioni.

Per cominciare farò un esempio con dati artificiali. Dare ad R le seguenti istruzioni;

```
set.seed(17); x <- sort(rnorm(100,170,4)); y <- x -110 + 80*runif(x)
```

Fingeremo che i dati siano stati raccolti in un esperimento replicato 100 volte: x rappresenta la concentrazione di un certo composto chimico che si suppone catalizzare una certa reazione, y rappresenta il prodotto di reazione. I dati sono naturalmente ordinati: x[1] e y[1] sono le concentrazioni di catalizzatore e prodotto nel primo esperimento, x[2] e y[2] le analoghe quantità per il secondo esperimento, etc. .

Per i singoli vettori di dati x e y sono definite le seguenti quantità

	media	"varianza"
x:	$< x >$=mean(x)=169.9389	var(x)= 17.90315
y:	$< y >$=mean(y)=98.84576	var(y)=653.7447

I dati sono facilmente rappresentabili in un grafico, in cui ogni dato (coppia di x e y) è un punto del piano:

```
plot(x,y)
```

Il punto di coordinate `(mean(x),mean(y))` è il **baricentro** dei dati (che in effetti è il baricentro fisico dei dati, se ad ogni dato viene assegnata lo stesso peso).

```
points(mean(x),mean(y),pch=20,col=2)
```

Ho scritto "varianza" con le virgolette perché devo spiegare la differenza tra la varianza e la varianza campionaria. La varianza è la media dei quadrati degli scarti, e che scarti sono la differenza tra i dati e il loro valore medio; la radice quadrata della varianza è lo scarto quadratico medio, (o deviazione standard), e misura la dispersione dei dati dalla media. In particolare la varianza è

$$\sigma_x^2 = \sigma_{xx} = \text{varianza dir x} = \frac{1}{N}\sum_{i=1}^{N}(x_i - < x >)^2 = \left\langle (x - < x >)^2 \right\rangle = < x^2 > - < x >^2$$

In realtà quella calcolata da R è la **varianza campionaria**:

$$\texttt{var(x)} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - < x >)^2$$

(anche la deviazione standard `sd(x)` è campionaria, cioè è la radice di `var(x)`). Il motivo per cui si divide per $N-1$ invece che per N è il seguente: se `x` è un **campione** di dati presi da una **popolazione**, per approssimare il valore della varianza di tutta la popolazione è meglio utilizzare la varianza campionaria del campione, che non la varianza del campione.

La **covarianza** misura quanto "variano insieme" le quantità `x` e `y`, ed è la media del prodotto degli scarti di `x` e di `y`:

$$\begin{aligned}\sigma_{xy} = &\text{covarianza di x} = \frac{1}{N}\sum_{i=1}^{N}(x_i - < x >)(y_i - < y >) \\ &= \left\langle (x - < x >)(y - < y >) \right\rangle = < xy > - < x >< y >\end{aligned}$$

Se `x` e `y` fossero indipendenti la covarianza sarebbe nulla. La funzione di R che calcola la covarianza (campionaria) è `cov(x,y)`. Inoltre si definisce il **coefficiente di correlazione** che è il rapporto tra la covarianza e il prodotto delle devizioni standard:

$$\texttt{cor(x)} = \frac{\texttt{cov(x,y)}}{\sqrt{\texttt{var(x)}\,\texttt{var(y)}}} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

(usare le varianze campionarie o non campionarie dà lo stesso valore per il coefficiente di correlazione). Per come è costruito è possibile dimostrare che il coefficiente di correlazione è sempre compreso tra -1 e 1:

$$-1 \leq \texttt{cor(x)} \leq 1.$$

Inoltre:

se `x` e `y` sono indipendenti `cor(x)` è (circa) zero
se `y = ax +b` con $a > 0$, `cor(x) = 1`
se `y = ax +b` con $a < 0$, `cor(x) = -1`

Riassumendo, il coefficiente di correlazione misura quanto i dati sono allineati. Per i dati che abbiamo considerato nell'esempio:

```
> cor(x,y)
[1] 0.3350001
```

che indica una debole correlazione positiva tra le `x` e le `y`; vuol dire che si vede una debole tendenza dei dati ad essere allineati da in basso a destra verso in alto a sinistra (verificare sul grafico)
Dare le seguenti istruzioni:

```
z <- runif(x)*100+54
plot(x,z)
cor(x,z)
[1] -0.07466613
```

In questo caso il coefficiente di correlazione è praticamente zero, dunque `x` e `y` sono scorrelati. Infine, dare le seguenti istruzioni:

```
w <- x - 110 + runif(x) * sqrt(abs((x - 170)))*2
plot(x,w)
cor(x,w)
[1] 0.9521127
```

Il coefficiente di correlazione è molto alto, ed in effetti si vede una tendenza dei dati ad essere allineati.

4.6 Regressione lineare

Se il coefficiente ci correlazione è vicino a 1 o a -1, vuol dire che `x` e `y` sono "abbastanza" allineati. Ma allora possiamo sospettare una relazione (approssimata) tra `x` e `y` di tipo lineare:

$$y = ax + b.$$

Come possiamo scegliere i coefficienti a e b? Il metodo dei "minimi quadrati" sceglie i valori di a e b in modo tale che la retta $y = ax + b$ minimizzi la somma dei quadrati delle differenze in ordinata tra i dati e la retta. Consideriamo ancora le 100 coppie di dati `x` e `y`. Considerata una retta qualunque $ax + b$ l'errore quadratico (in ordinata) che compio rispetto alla retta è:

$$\sum_{i=1}^{100}(ax_i + b - y_i)^2.$$

Trovando il minimo di questa funzione nelle variabili a e b si ottiene:

$$a = \frac{\sigma_{xy}}{\sigma_x^2}$$
$$b = < y > - \frac{\sigma_{xy}}{\sigma_x^2} < x >$$

(Esercizio: calcolare il gradiente in a e b e verificare che il gradiente è nullo per i valori di a e b scritti sopra).

La funzione di R che calcola i coefficienti della retta dei minimi quadrati si chiama `lm` (abbreviazione di "linear modelling"). Ricostruiamo i dati del paragrafo precedente:

```
set.seed(17); x <- rnorm(100,170,4); y <- x -110 + 80*runif(x)
z <- runif(x)*100+54
w <- x - 110 + runif(x) * sqrt(abs((x - 170)))*2
```

Ora vediamo per la coppia `x`, `w` come si costruisce la retta dei minimi quadrati:

```
lm(w~x) # vuol dire: costruisci la retta di w in funzione di x
Call:
lm(formula = w ~ x)

Coefficients:
(Intercept)            x
  -101.9539       0.9627
```

I due numeri che ci interessano sono l'intercetta -101.9539 (ovvero `b`, cioè l'ordinata dell'intersezione con l'asse verticale) e il coefficiente angolare 0.9627. Dare le seguenti istruzioni:

```
plot(x,w)
bea <- lm(w~x)\$coef   # mette i coeff. in vettore, prima b e poi a
b <- bea[1]            # b e' l'intercetta
a <- bea[2]            # a e' il coefficiente angolare
r <- c(min(x),max(x))  # r contiene come ascisse il minimo e il massimo delle x
lines(r,a*r+b,col=4)   # disegna la retta dei minimi quadrati
```

Una questione importante è come decidere se il modello trovato è un buon modello per i nostri dati. Per prima cosa è buona norma analizzare i **residui**. Per definizione i residui sono la differenza tra i dati e la retta: $y_i - (ax_i + b)$. Se la nostra approssimazione è ragionavele, i residui devono apparire come numeri casuali intorno a zero senza regolarità:

```
plot(lm(z~x)\$res)  # lm(z~x)\$res da' il valore dei residui
```

Un ulteriore esempio:

```
q <- sort(runif(100))
p <- q * q
plot(lm(p~q)\$res)
```

In questo caso i resuidi sono tutt'altro che casuali. Infatti la migliore curva che approssima il grafico è una parabola... . ATTENZIONE: il plot dei residui ha senso se le ascisse sono un vettore ordinato, altrimenti un'eventuale buona casualità sarebbe solo frutto dell'ordine casuale in cui sono messi i dati.

Ulteriori informazioni per verificare la bontà della retta dei minimi quadrati si ottiene con l'istruzione `summary` applicata al risultato di `lm`:

```
summary(lm(w~x))
(...)
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -101.95393    5.30848  -19.21   <2e-16 ***
x              0.96268    0.03123   30.83   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.315 on 98 degrees of freedom
Multiple R-Squared: 0.9065,     Adjusted R-squared: 0.9056
F-statistic: 950.3 on 1 and 98 DF,  p-value: < 2.2e-16
```

Ci occuperemo solo delle righe che riguardano i coefficienti: la prima dà il valore (stima) dell'intercetta, la seconda dà il coefficiente angolare. L'ultimo numero sulla riga del coefficiente è il p-value di un test statistico con ipotesi nulla "il coeff. angolare è 0"; tale ipotesi corrisponde a supporre l'indipendenza dei due vettori di dati (infatti il coeff. angolare è proporzionale alla covarianza). Infine il valore `Multiple R-Squared: 0.9065` è tanto piú vicino ad uno quanto più il modello è adeguato ai dati.

In sinstesi, per fare una regressione lineare "ragionevole"

– calcolare il coefficiente di correlazione; se è molto vicino a 0 non ha senso calcolare la retta dei minimi quadrati

– fare il grafico dei residui: se si osservano regolarità vuol dire che il modello lineare non è il migliore

– controllare il valore di R^2: se è basso il modello lineare non è adeguato

– controllare il p-value per il coefficiente angolare: se è grande vuol dire che i dati sono compatibili con l'indipendenza

4.7 Dipendenze non lineari

I dati potrebbero avere una dipendenza di tipo non lineare: ad esempio $y = ax^2$; in tal caso la regressione lineare non darà buoni risultati. D'altra parte,

$$y = ax^2 \to \log y = 2 \log x + \log a$$

Dunque se x e y sono dati postivi, ha senso fare la regressione lineare tra i loro logartimi. Se il procedimento ha successo si ottiene una "legge a potenza".

Un ultimo esempio:

$$y = ae^{bx} \to \log y = bx + \log a$$

in tal caso ha dunque senso tentare la regressione lineare tra il logaritmo di y e x. Se il procedimento ha successo vorrà dire che y dipende esponenzialmente da x.