

Università degli Studi di Roma "La Sapienza"



Corso di laurea in Matematica
a.a. 2009–2010

Note per il corso

Abilità Informatiche: Introduzione a Unix

versione 1.3 (*Febbraio 2008*)

Alessandra Seghini

Note per il corso "Abilità Informatiche: Introduzione a Unix"

di Alessandra Seghini



Queste note sono pubblicate sotto una Licenza Creative Commons.
<http://creativecommons.org/licenses/by-nc-sa/2.0/>

PREMESSA

Questi appunti forniscono una traccia degli argomenti svolti nella corso di "Abilità Informatiche: Introduzione a Unix". Vengono introdotti alcuni concetti di base per l'utilizzo del sistema operativo UNIX e vengono presentati alcuni fra i comandi più utilizzati. Per ciascuno dei comandi vengono illustrate solo alcune delle opzioni disponibili. Per una descrizione dettagliata dei comandi e per un approfondimento dei concetti introdotti, si rimanda al manuale in linea ed alla bibliografia riportata sulla [pagina web del corso](#).

INDICE

Prima di cominciare

Cenni storici

Accesso al sistema e primi passi

 Login e Logout

 Cambio password

 Iniziamo a guardarci intorno

Formato generale di una riga di comando ed utilizzo delle man pages

La shell e le variabili d'ambiente

Interfaccia uomo-macchina e sistema X Window

Gestione di file e directory

 L'albero delle directory

 La home directory (~)

 La directory corrente e la directory di livello superiore (. e ..)

 Percorsi assoluti e percorsi relativi

 Contenuto di una directory, tipi di file, ricerca di un file (ls, file, find)

 Uso di caratteri jolly * e ? e del tasto [TAB]

 Creazione, cancellazione e cambio di directory (mkdir, rmdir, cd)

 Copia, spostamento/rinomina, e cancellazione di file (cp, mv, rm)

 Visualizzazione del contenuto di un file (more, cat, tail, head)

 Altri comandi per analizzare il contenuto di un file (diff, grep, sort, wc)

 Link simbolici (ln)

 Modifica dei permessi per l'accesso a file e directory (chmod)

Rindirizzamento dell'output e concatenazione di comandi

Come localizzare un comando

Editing di file di testo

 Cenni sull'utilizzo dell'editor vi

 Cenni sull'utilizzo dell'editor emacs

 Cenni sull'utilizzo dell'editor pico

Stampa del contenuto di un file

Controllo dello spazio disco ed archiviazione di file e directory

 Controllo dello spazio disco occupato (du, quota, df)

 Archiviare e comprimere file e directory (gzip, compress, zip, tar)

Gestione dei processi

Utilizzo di floppy disk, pendrive usb e cdrom

Editare, compilare ed eseguire un programma

 Esempio di un semplice programma C

 Esempio di un semplice programma fortran

Scrivere un file LaTeX

 Esempio di un semplice documento LaTeX

Semplici esempi di shell script

Cenni sull'utilizzo di ssh ed sftp

PRIMA DI COMINCIARE

L'insieme delle componenti fisiche di un computer costituisce l'**hardware**, mentre l'insieme dei programmi caricati su un computer costituisce il **software**.

Le componenti fisiche standard di un personal computer sono:

- la **CPU** (*Central Processing Unit*) o processore, ovvero l'unità centrale di elaborazione, che si occupa dell'esecuzione delle istruzioni;
- la **RAM** (*Random Access Memory*), memoria principale del computer, utilizzata dal processore in fase di elaborazione; è una memoria volatile, il cui contenuto viene perso allo spegnimento del computer;
- la **memoria cache**, una piccola memoria ad accesso più veloce rispetto alla normale RAM, utilizzata dal processore per memorizzare i dati e le istruzioni utilizzate più frequentemente;
- una memoria non volatile (ROM, EPROM, flash, etc) sulla quale sono memorizzate le prime istruzioni eseguite dal computer all'accensione. Questo codice costituisce il BIOS (*Basic Input–Output System*) del computer ed ha il compito di garantire la comunicazione a basso livello con le periferiche e di caricare il **sistema operativo** all'avvio;
- l'**hard disk**, ovvero la memoria di massa, utilizzata per conservare dati e programmi in maniera permanente. L'hard disk può essere collegato al computer tramite diversi tipi di interfaccia. Tra le più diffuse IDE (*Integrated Device Electronic*), SCSI (*Small Computer System Interface*), SATA (*Serial Advanced Technology Attachment*). Le unità IDE sono generalmente indicate con la sigla *hd*, le unità SCSI e SATA con *sd*;
- le unità di I/O (*Input/Output*) standard, ovvero tastiera+mouse (input) e monitor (output);
- la **scheda video**, la **scheda audio**, la scheda per il collegamento alla **rete** (ethernet, wireless, modem, etc);
- il **floppy disk drive** e l'unità per la lettura/masterizzazione di **CDROM/DVD**;
- alcune **porte** (parallele, seriali, USB – *Universal Serial Bus* –, etc.) per il collegamento di periferiche esterne come la stampante, lo scanner, etc.

Fra i programmi, un posto particolare spetta al *sistema operativo*.

Il **Sistema operativo** è il software di base che gira su un computer. Esso svolge le due funzioni fondamentali che rendono possibile l'interazione uomo–macchina: si occupa della gestione delle risorse hardware e fornisce un'interfaccia, più o meno amichevole (*friendly*), che consente all'utente di comunicare con il sistema. Più precisamente, il sistema operativo:

- permette di controllare e di utilizzare l'hardware del computer (dischi, memoria, tastiera, mouse, stampante, etc.);
- si occupa delle operazioni di *task scheduling*, ovvero controlla i vari processi (*task*) che operano sul computer, assegnando loro l'accesso alle risorse disponibili (CPU, memoria, dischi, etc.) e permettendo l'interazione fra i processi;
- consente di utilizzare software applicativo.

Il sistema operativo è formato da due componenti che svolgono funzioni complementari: il **kernel** (*nucleo*) e la **shell** (*guscio*). Il kernel fornisce le funzioni di basso livello, essenziali per il sistema e direttamente legate all'hardware. In particolare si occupa della gestione della memoria, delle risorse e delle periferiche del sistema, assegnandole di volta in volta ai processi in esecuzione. La shell interpreta e manda in esecuzione

i comandi inviati al sistema e fornisce l'interfaccia fra l'utente ed il sistema operativo. Il sistema operativo è poi completato dai **device driver**, ovvero da tutti quei programmi che integrano il kernel, gestendo la comunicazione con le varie componenti hardware (dischi, scheda di rete, scheda video, stampanti, etc.).

Un **programma applicativo** è invece un pacchetto software che consente di effettuare particolari operazioni, non collegate alla gestione del sistema. Sono applicativi, ad esempio, i programmi di videoscrittura, i giochi, i programmi per la gestione della contabilità di un ufficio o per la gestione di un magazzino, i compilatori, pacchetti come *Matlab* e *Mathematica*, etc. Un software applicativo viene generalmente distribuito in diverse versioni che possono essere installate su diverse piattaforme hardware e su diversi sistemi operativi.

CENNI STORICI

UNIX nasce nel 1971 dalle ceneri del progetto Multics (Multiplexed Information and Computing Service) avviato nel 1965 dai Bell Labs, divisione dell' AT&T, in collaborazione con la General Electric ed il MIT (Massachusetts Institute of Technology). Dopo la chiusura di Multics, il progetto venne portato avanti, nonostante notevoli problemi di budget, grazie soprattutto agli sforzi di **Ken Thompson**, **Dennis Ritchie** e **Brian Kernighan**.

Inizialmente UNIX era scritto in linguaggio assembler per architettura PDP: la prima versione nacque su un vecchio e piccolo calcolatore DEC PDP-7 utilizzato nella prima fase di sviluppo e di sperimentazione del progetto. Nel 1970, grazie all'entusiasmo dei primi utenti, venne acquistato un calcolatore più potente e venne realizzata la migrazione su un DEC PDP-11.

Nel 1973, Ritchie e Thompson riscrissero in C⁽¹⁾ il Kernel (nucleo) UNIX, che fu così il primo software di sistema scritto in un linguaggio diverso dal codice assembler.

Dopo la presentazione al *Symposium for Operating System Principles*, l'interesse per il nuovo sistema operativo iniziò a diffondersi. La distribuzione di UNIX sotto forma di codice sorgente permetteva agli utilizzatori di personalizzare il sistema operativo in base alle proprie esigenze. Tale possibilità di personalizzazione ed il basso costo attrasse soprattutto le Università, che iniziarono a modificare il codice UNIX per utilizzarlo su computer diversi. Questo portò, alla fine degli anni '70, a due diversi standard: Unix System V della AT&T e BSD (*Berkeley Software Distribution*) dell'Università di Berkeley California.

Nel 1991 **Linus Torvalds**, allora studente dell'Università di Helsinki, iniziò come passatempo, la scrittura di una estensione del sistema operativo Minix, realizzato a scopo didattico dal prof. Tanenbaum. L'idea era quella di scrivere una versione di Unix per PC, ispirata a Minix, e sulla quale tutti potessero apportare modifiche al codice sorgente.

Nel 1994 venne rilasciata la prima versione definitiva (versione 1.0) del kernel di **Linux** e nacquero RedHat, Debian e SUSE, che sono ancora oggi tra le distribuzioni più diffuse. Linux viene distribuito come software **open source**, nel rispetto della General Public License (GPL) del movimento GNU Open Source. Il suo sviluppo procede senza soste, eventuali errori vengono scoperti e corretti velocemente grazie a quella che è ormai diventata la più grande comunità di software libero mai esistita. Fra i meriti di Linux c'è anche quello di aver fortemente contribuito alla diffusione dell'idea di OpenSource, lanciata nel 1984 da **Richard Stallman**, ricercatore del MIT AI Lab, fondatore della FSF (Free Software Foundation) e del progetto GNU (acronimo ricorsivo di "GNU's Not Unix"). Numerose aziende storicamente impegnate sul fronte Unix proprietario hanno deciso di supportare Linux offrendo soluzioni basate su di esso ed oggi un'importante caratteristica di questo sistema operativo è la portabilità, cioè la compatibilità con le diverse famiglie di processori.

(1) il linguaggio C fu rilasciato da Kernighan e Ritchie negli anni fra il 1973 ed il 1978.

ACCESSO AL SISTEMA E PRIMI PASSI

UNIX è un sistema operativo **multiutente**, ovvero consente a più utenti di accedere contemporaneamente al sistema e di codividerne le risorse, e **multitasking**, ovvero consente di eseguire più processi contemporaneamente.

Ogni utente è identificato da uno **username** (pubblico) e da una **password** (segreta), che devono essere digitate per effettuare il *login*. Esiste un utente speciale, l'amministratore di sistema, identificato dallo username **root**.

Gli utenti sono divisi in **gruppi**. L'organizzazione in gruppi permette di definire delle regole per l'accesso alle risorse disponibili (*policy*) condivise da tutti gli utenti dello stesso gruppo.

Ogni utente ha una propria directory personale, la **home directory**.

Le informazioni relative alle utenze definite su un sistema, sono contenute in un file molto importante, aggiornato e mantenuto dall'amministratore di sistema, il file `/etc/passwd`.

Un utente può entrare su un sistema UNIX direttamente dalla **console**, ovvero dall'unità di I/O (monitor, tastiera, mouse) connessa direttamente al sistema e normalmente utilizzata per le operazioni di gestione, manutenzione, aggiornamento. È possibile collegarsi anche da **terminale**. Per terminale, si intende un'unità fisica indipendente collegata al sistema mediante linea seriale o via rete. Ad esempio, se un PC ed un server UNIX sono collegati sulla stessa LAN (*Local Area Network*), sarà possibile collegarsi al server dal PC utilizzando un programma per accesso remoto (*telnet, ssh, etc.*). Attivato il collegamento, il PC diventerà un terminale del server UNIX.

Di seguito vengono presentati i primi comandi che consentono di entrare ed uscire dal sistema, cambiare la propria password ed "iniziare a guardarsi intorno" (chi sono? a che gruppo appartengo? quali altri utenti sono collegati al sistema? cosa stanno facendo? etc.).

Login e Logout

```
Red Hat Linux release 7.3
Kernel 2.4.18-10custom on an i686

archimede login: studente
password: *****
Last login: Fri Jan 21 08:51:00 2005 on tty2
archimede%
archimede%exit
```

La stringa con la quale il sistema risponde e si dichiara pronto ad accettare comandi, si chiama **prompt dei comandi**. Il prompt è spesso costituito da un solo carattere (`$`, `#`, `%`, `>`) ed è personalizzabile (nel nostro esempio il prompt è `archimede%`).

UNIX è un sistema **case-sensitive**, ovvero distingue fra lettere maiuscole e minuscole, quindi *studente* è diverso da *Studente*. Generalmente i comandi UNIX sono scritti in minuscolo.

Cambio password

```
archimede%passwd
Changing password for user studente.
Changing password for studente
(current) UNIX password: *****
New password: *****
Retype new password: *****
passwd: all authentication tokens updated successfully.
```

Iniziamo a guardarci intorno

```
archimede%whoami
studente
archimede%groups
users
archimede%groups root
root : root bin daemon sys adm disk wheel
archimede%id
uid=700(studente) gid=100(users) gruppi=100(users)
archimede%pwd
/home/studente
archimede%date
Ven gen 21 09:10:19 CET 2005
archimede%who
alex pts/0 Jan 17 08:01 (giove.mat.uniroma1.it)
studente tty2 Jan 21 09:10
```

```

archimede#w
 9:10am up 4 days, 1:23, 2 user, load average: 0.59, 0.15, 0.05
USER  TTY      FROM             LOGIN@   IDLE   JCPU   PCPU   WHAT
alex   pts/0    giove.mat.unirom Mon 8am   1      1:06   1:06   rsh venire
studente tty2     archimede        9:10am   0.00s  0.16s  0.03s  w
archimede#finger alex
Login name: alex                In real life: Alex Bianchi
Directory: /home/alex           Shell: /bin/tcsh
On since Jan 17 08:01:10 on pts/0 from giove.mat.uniroma1.it
22 minutes Idle Time
Mail last read Fri Jan 21 00:17:08 2005
No Plan.
archimede#history
 1  9:10  passwd
 2  9:10  whoami
 3  9:10  groups
 4  9:10  groups root
 5  9:10  id
 6  9:10  pwd
 7  9:10  date
 8  9:10  who
 9  9:10  w
10  9:11  finger
11  9:11  history
archimede#clear

```

Riepilogo dei comandi presentati

```

clear      – pulisce lo schermo del terminale
date       – visualizza la data e l'ora corrente
exit       – chiude l'esecuzione della shell
finger     – mostra informazioni sugli utenti
groups     – visualizza il gruppo di appartenenza
history    – visualizza e gestisce la command history list
id         – visualizza l'UID (User IDentifier) ed il GID (Group IDentifier) dell'utente
logout     – chiude l'esecuzione della shell
passwd     – modifica la login password
pwd        – visualizza il nome della directory corrente
w          – visualizza l'elenco degli utenti collegati al sistema e cosa stanno facendo
who        – visualizza l'elenco degli utenti collegati al sistema
whoami     – visualizza lo username dell'utente

```

Per maggiori dettagli sui comandi presentati utilizzare il manuale in linea.

FORMATO GENERALE DI UNA RIGA DI COMANDO ED UTILIZZO DELLE MAN PAGES

Per utilizzare un comando è necessario conoscerne la **sintassi**, ovvero bisogna sapere esattamente come il comando deve essere digitato, quali sono le eventuali opzioni per modificare o estendere le sue funzionalità, quali sono gli eventuali argomenti.

Il formato generale di una riga di comando è il seguente:

```
comando [-opzioni] [argomenti]
```

Riprendiamo ad esempio il comando `who` utilizzato precedentemente:

```

archimede#who
alex   pts/0    Jan 17 08:01    (giove.mat.uniroma1.it)
studente tty2     Jan 21 09:10

```

Digitato senza nessuna opzione e senza nessun argomento, `who` mostra l'elenco degli utenti che stanno utilizzando il sistema, riportando nome di login, linea del terminale, ora di login, nome host remoto.

Utilizzando l'opzione `-H`, si ottiene una riga di intestazione per le colonne:

```
archimede%who -H
NAME      LINE      TIME      FROM
alex      pts/0     Jan 17 08:01 (giove.mat.uniroma1.it)
studente  tty2     Jan 21 09:10
```

Esistono altre opzioni utilizzabili con il comando `who`. Per avere l'elenco di tutte le opzioni disponibili si deve utilizzare il manuale in linea, richiamabile con il comando **man**. Imparare ad utilizzare il manuale in linea è un prerequisito indispensabile per poter utilizzare in maniera avanzata i comandi disponibili sul sistema. Tramite il comando `man` è possibile avere informazioni sia sui comandi di sistema, sia su molti dei programmi applicativi disponibili, che generalmente sono forniti di documentazione compatibile con `man`.

Il manuale in linea è organizzato in sezioni (comandi di sistema e applicativi, funzioni di sistema, etc.) Ogni sezione è composta da diversi documenti, le *man pages*. Ogni *man page* si riferisce ad una particolare voce contenuta nel manuale in linea.

La sintassi del comando `man` è molto semplice. Digitando `man comando`

si otterrà la prima schermata della pagina di manuale relativa al comando specificato. Per scorrere all'interno del documento utilizzare i tasti:

[INVIO]	– avanza di una riga
barra spaziatrice	– avanza di una schermata
b	– torna indietro di una schermata
q	– esce

Una pagina di manuale è generalmente composta da diverse parti. In particolare:

Name	– nome del comando con breve descrizione.
Synopsis	– sintassi del comando.
Description	– descrizione dettagliata.
Options	– descrizione e significato delle opzioni supportate.
Files	– nomi dei files che il comando utilizza e/o modifica.
See also	– rimando ad altri comandi e ad altre pagine di manuale correlate.
Bugs	– eventuali errori noti del comando.

Una opzione particolarmente utile del comando `man` è l'opzione `-k`.

Inviando il comando: `man -k parola-chiave`

si ottiene l'elenco delle pagine di manuale che riguardano la parola-chiave specificata, come nel seguente esempio:

```
archimede%man -k mathematica
GNU TeXmacs [texmacs](1) - a WYSIWYG mathematical text editor
math (1) - Mathematica kernel
mathematica (1) - Mathematica system for X
motifps (1) - display utility for Mathematica under the X Window System
psrender (1) - render Mathematica PostScript output
```

Lo stesso risultato si ottiene con il comando: `apropos parola-chiave`

Il comando: `whatis comando`

restituisce solo il nome ed una breve descrizione del comando specificato.

Riepilogo dei comandi presentati

<code>apropos</code>	– ricerca una stringa nel <i>whatis database</i> (stesso risultato con " <code>man -k</code> ")
<code>man</code>	– visualizza la pagina del manuale in linea relativa al comando specificato
<code>whatis</code>	– visualizza una breve descrizione del comando specificato

Per maggiori dettagli sui comandi presentati utilizzare il manuale in linea.

LA SHELL E LE VARIABILI D'AMBIENTE

La shell è il programma che interpreta i comandi inviati al sistema, fornendo l'interfaccia fra l'utente ed il sistema operativo. A differenza di altri sistemi operativi, UNIX e Linux permettono all'utente di scegliere fra diverse shell, a seconda del proprio gusto e delle proprie esigenze. Le shell più note sono:

Bourne shell	<i>sh</i>
Bourne again shell	<i>bash</i>
C shell	<i>cs</i>
Teach C shell	<i>tcsh</i> (C shell avanzata)
Korn shell	<i>ksh</i>

In particolare la *bash* è la shell normalmente utilizzata in ambiente GNU/Linux.

Ogni utente ha una sua shell di *default*, che parte automaticamente al login. Questa shell è definita dall'amministratore di sistema al momento della configurazione dell' *account* per l'utente e, come tutte le altre informazioni relative alle utenze, è memorizzata nel file `/etc/passwd`.

La shell, oltre a fornire il tramite fra l'utente ed il sistema operativo è anche un linguaggio di programmazione e, come tutti i linguaggi di programmazione, dispone di variabili, di operatori e di strutture logiche. Un programma scritto utilizzando i comandi della shell viene chiamato **shell script**. Questi script possono essere comodi per eseguire sequenze di comandi ripetitivi o per effettuare operazioni complesse. Più avanti in queste note, vengono riportati due semplici [esempi](#).

Fra le variabili utilizzate dalla shell, particolarmente importanti sono le **variabili d'ambiente**. Queste variabili determinano molte caratteristiche della sessione di lavoro e possono essere personalizzate dall'utente. Vengono inizializzate dagli **script di login**, programmi eseguiti automaticamente quando un utente si collega al sistema. Le variabili d'ambiente hanno nomi standard, scritti interamente in maiuscolo, come ad esempio:

HOME	nome della home directory
LOGNAME	login name
SHELL	nome della shell attiva

Seguono alcuni comandi che permettono di operare sulle variabili d'ambiente:

- **printenv**
utilizzato senza argomenti, il comando `printenv` riporta l'elenco delle variabili d'ambiente con i relativi valori.
Utilizzato nella forma
`printenv NOME-VAR`
riporta il valore della variabile specificata
- **setenv NOME-VAR valore**
permette di modificare il valore di una variabile o di definirne una nuova.
Esempio: `setenv PRINTER mia_stampante`
- **unsetenv NOME-VAR**
cancella una variabile d'ambiente.
- **echo \$NOME-VAR**
visualizza il valore di una variabile d'ambiente.
Esempio: `echo $PRINTER`

Riepilogo dei comandi presentati

<code>echo \$NOME-VAR</code>	– visualizza il valore della variabile d'ambiente NOME-VAR
<code>printenv</code>	– visualizza il valore delle variabili d'ambiente
<code>setenv</code>	– imposta il valore di una variabile d'ambiente
<code>unsetenv</code>	– cancella una variabile d'ambiente

Per maggiori dettagli sui comandi presentati utilizzare il manuale in linea.

INTERFACCIA UOMO-MACCHINA E SISTEMA X WINDOW

Le shell presentate nel paragrafo precedente sono di tipo **CLI** (*Command Line Interface*), ovvero sono *interfacce a linea di comando* che consentono all'utente di dialogare con il sistema esclusivamente in modalità alfanumerica.

L'evoluzione grafica delle shell a caratteri è costituita dalle **GUI** (*Graphical User Interface*), oggi in assoluto le più diffuse. Le GUI sono molto *amichevoli* e consentono anche ad utenti decisamente inesperti di utilizzare il computer. La comunicazione fra l'utente ed il sistema operativo avviene tramite un sistema di finestre, icone e menù che consente di effettuare operazioni complesse con un semplice click del mouse.

Nella scelta di un'interfaccia grafica è però anche necessario considerare che generalmente, quanto più un'interfaccia è sofisticata, accattivante, semplice ed intuitiva nell'utilizzo, tanto maggiori saranno le risorse (memoria, spazio disco, etc.) ad essa necessarie. D'altra parte le GUI presentano sicuramente numerosi vantaggi, prima fra tutti, la possibilità di sfruttare in maniera immediata le potenzialità del multitasking.

Il sistema **X Window** è l'interfaccia grafica normalmente utilizzata sui sistemi UNIX/Linux.

L'ambiente grafico X window è costituito essenzialmente da 3 componenti:

- l' **X server** è il processo di "basso livello" che comunica con l'hardware del computer, in particolare con la scheda grafica, e che si occupa della gestione del display;
- il **window manager** è il gestore delle finestre. Controlla l'aspetto e le funzionalità delle finestre che appaiono a video (la barra del titolo, i bottoni per chiudere, ingrandire, etc.) e del desktop (le icone, i vari menù disponibili, etc.);
- i **client X** sono tutti i programmi che lavorano in ambiente grafico e che l'utente può utilizzare. Ad esempio un browser come Mozilla, un visualizzatore di file PDF come Acroread, altri programmi con interfaccia grafica come Matlab e Mathematica, etc.

Esistono molti window manager ed ogni utente può scegliere fra quelli configurati sul sistema in base ai propri gusti ed alle proprie esigenze. Fra i più diffusi citiamo:

<code>fvwm</code>	F? Virtual Window Manager (<i>Rob Nation, l'autore originale di fvwm, non ricorda per cosa stava la F iniziale, quindi ci sono molte versioni: Fabulous, Fast, Free, etc.</i>)
<code>gnome</code>	GNU Object Model Environment
<code>kde</code>	K Desktop Environment
<code>mlvwm</code>	Macintosh Like Virtual Window Manager
<code>olvwm</code>	OpenLook Virtual Window Manager
<code>twm</code>	Tab Window Manager
<code>xfce</code>	Free desktop environment for unix (<i>"the name Xfce originally stood for XForms Common Environment ... currently the acronym doesn't stand for anything"</i>)

Generalmente in fase di login, l'utente può scegliere quale window manager utilizzare tramite i tasti funzione <F1>, <F2>, etc. Ad esempio nel nostro laboratorio, se dopo aver digitato username e password si preme il

tasto [Enter] o il tasto <F1>, si avvierà mlvwm, se invece si preme il tasto <F2> si avvierà kde. I tasti <F3>, <F4> ed <F5> permettono rispettivamente di attivare sessioni con xfce, twm e ion.

Lavorando in ambiente X window sono disponibili molte utility, richiamabili utilizzando le apposite voci dei menù messi a disposizione dal window manager, o direttamente da linea di comando. Ecco alcuni esempi:

- **xterm -fn 10x20 -geometry 80x24+100-50 -sb**

il comando `xterm` apre una finestra di emulazione terminale, all'interno della quale viene lanciata una shell. In questa finestra sarà possibile lavorare in modalità alfanumerica, avendo però a disposizione anche tutte le funzionalità dell'ambiente grafico.

Il comando `xterm` dispone di molte opzioni. Vediamo il significato di quelle usate nell'esempio:

<code>-fn 10x20</code>	specifica le dimensioni dei font da utilizzare. In questo caso font 10x20;
<code>-geometry 80x24+100-50</code>	specifica le dimensioni e la posizione della finestra. In questo caso verrà aperta una finestra con 80 colonne e 24 righe, a 100 punti di distanza dal margine sinistro ed a 50 punti dal margine inferiore;
<code>-sb</code>	abilita la <i>scroll bar</i> ;

La finestra può essere richiusa utilizzando gli opportuni bottoni messi a disposizione dal window manager, o digitando il comando `"exit"`.

Notiamo che la shell dalla quale è stato lanciato il comando `xterm` resta "congelata" fino a quando la nuova finestra non viene chiusa. Questo perchè la shell ha attivato un processo "figlio" e non può proseguire fino a quando quest'ultimo non sia stato richiuso. Sfruttando il multitasking, è possibile sganciare il processo "figlio" (l'esecuzione del comando `xterm`) dal processo "padre" (la shell dalla quale il comando è stato lanciato), in modo tale che entrambi i processi possano procedere indipendentemente. Per fare ciò basta aggiungere il carattere "&" in fondo alla riga di comando, che diventa

```
xterm -fn 10x20 -geometry 80x24+100-50 -sb &
```

- **xsetroot -solid SteelBlue4**

il comando `xsetroot` permette di personalizzare lo sfondo. In particolare, l'opzione `"-solid"` serve a definire il colore dello sfondo che nell'esempio è impostato ad azzurro (SteelBlue4).

- **xclock -d &**

apre un orologio. Con l'opzione `"-d"`, l'ora è mostrata in forma digitale.

Il carattere "&" in fondo alla riga di comando, serve a sganciare il processo che esegue il comando `xclock` dalla shell che lo ha lanciato.

- **xcalc &**

apre una calcolatrice.

Il carattere "&" in fondo alla riga di comando, serve a sganciare il processo che esegue il comando `xcalc` dalla shell che lo ha lanciato.

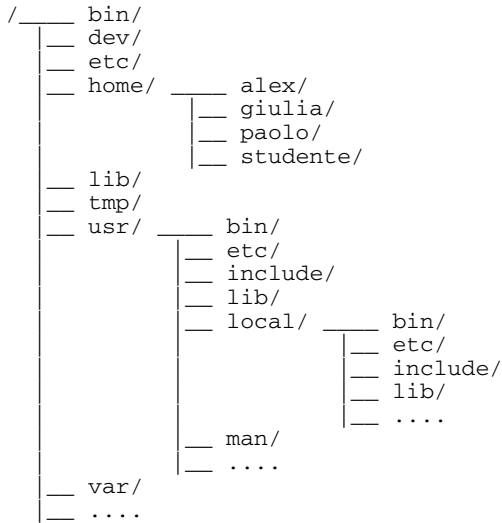
Per avere l'elenco delle utility disponibili sotto X window, si può consultare il manuale in linea alla voce "X" (`man x`). Sul manuale in linea è inoltre possibile trovare l'elenco ed il significato di tutte le opzioni disponibili per i vari comandi.

GESTIONE DI FILE E DIRECTORY

L'albero delle directory

Su un sistema UNIX i **file** sono suddivisi in **directory** (cartelle). Le directory sono organizzate gerarchicamente in una struttura ad albero che prende il nome di **filesystem**. La directory radice è indicata con il simbolo / e prende il nome di *root directory* (da non confondere con l'utente *root*, amministratore del sistema).

Un esempio di filesystem è il seguente:



/bin contiene file binari (eseguibili);

/dev contiene i *device driver*, ovvero i file che controllano i vari dispositivi hardware disponibili sul sistema;

/etc contiene file di configurazione del sistema;

/home contiene le home directory degli utenti;

/lib contiene le librerie condivise da più applicazioni;

/tmp è utilizzata per i file temporanei;

/usr contiene la maggior parte dei file di sistema, ed i pacchetti applicativi;

/var contiene file di log ed aree di *spool* (mailbox degli utenti, spool delle stampanti).

La home directory (~)

Al login ogni utente si trova posizionato sulla propria *home directory*. Nel filesystem dell'esempio precedente, la home directory dell'utente giulia è `/home/giulia/`. È possibile riferirsi alla home directory di un utente con `~username`, ad esempio `~giulia` può essere utilizzato per indicare la directory `/home/giulia/`. Ogni utente può riferirsi alla propria home directory semplicemente con il carattere `~`, senza specificare lo username.

La directory corrente e la directory di livello superiore (. e ..)

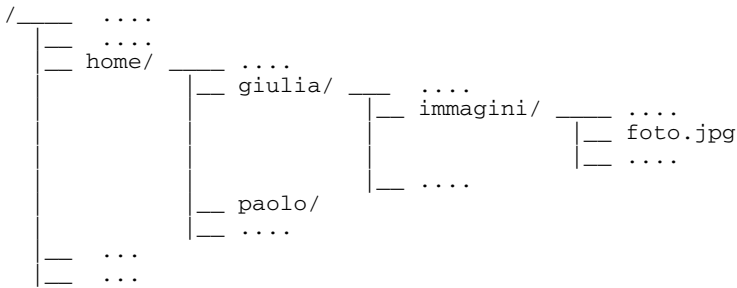
La *directory corrente* di lavoro è quella sulla quale si è posizionati. È possibile conoscere il nome della directory corrente, utilizzando il comando `pwd`. Subito dopo il login, la directory corrente di un utente è la sua home directory.

La directory corrente è indicata con il simbolo `."`

La directory "genitore" della directory corrente è indicata con il simbolo `.."`

Percorsi assoluti e percorsi relativi

Per individuare un file sul sistema, è necessario conoscere il suo *pathname*, costituito dal nome del file, preceduto dal nome della directory che lo contiene.



Ad esempio, il nome completo del file `foto.jpg` contenuto nella directory `/home/giulia/immagini/` è `/home/giulia/immagini/foto.jpg`. Abbiamo così specificato il *percorso assoluto*, che permette di raggiungere il file `foto.jpg` a partire dalla radice (`/`) del filesystem e che, pertanto, è valido qualunque sia la directory corrente.

È possibile individuare un file anche specificando il *percorso relativo*, rispetto alla directory sulla quale ci si trova posizionati, ovvero rispetto alla directory corrente. Se ad esempio siamo sulla directory `/home/giulia/`, possiamo riferirci al file `/home/giulia/immagini/foto.jpg` semplicemente indicando `immagini/foto.jpg` (senza il carattere `/` iniziale). In questo caso, il percorso per raggiungere il file `foto.jpg` non parte dalla radice del filesystem, ma dalla directory corrente e, pertanto, è valido solo se la directory corrente è `/home/giulia/`.

Nello specificare il *pathname* di un file, si possono utilizzare i caratteri `.`, `..` e `~` per indicare rispettivamente la directory corrente, la directory di livello superiore rispetto alla directory corrente e la home directory di un utente.

Ad esempio, un percorso assoluto per individuare il nostro solito file `foto.jpg`, è anche `~giulia/immagini/foto.jpg`.

Se la directory corrente è la home directory dell'utente `paolo` `/home/paolo/` (o `~paolo/`), il percorso relativo per individuare il file `foto.jpg` sarà `../giulia/immagini/foto.jpg`.

Contenuto di una directory, tipi di file, ricerca di un file (comandi `ls`, `file`, `find`)

• `ls [nome-dir]`

riporta l'elenco dei file contenuti nella directory `nome-dir`. Se non è specificato il nome di una directory, riporta l'elenco dei file contenuti nella directory corrente.

```
archimede%ls
Mail          bookmarks.html  help          help-alex     lq
```

Qualche opzione del comando `ls` (è possibile combinare insieme più opzioni):

`ls -a [nome-dir]`

riporta l'elenco di tutti (*all*) i file contenuti nella directory `nome-dir`, compresi i file *nascosti* il cui nome inizia con il carattere `.`. Nell'elenco sono compresi anche i file `.` e `..`, che indicano rispettivamente, la directory corrente e la directory "genitore" della directory corrente.

```
archimede%ls -a
.          .Mathematica  .history      bookmarks.html  help-alex
..         .cshrc        Mail          help            lq
```

`ls -l [nome-dir]`

riporta l'elenco dei file contenuti nella directory `nome-dir`, in formato esteso (*long*).

```
archimede%ls -la
total 84
drwxr-xr-x  5 giulia  users      512 Jan 26 08:52 .
drwxr-xr-x 31 root    root      2560 Jan 26 09:11 ..
drwxr-xr-x  8 giulia  users      512 Jun  7 2004 .Mathematica
-rw-r--r--  1 giulia  users     1487 Nov 29 08:52 .cshrc
```

```

-rw----- 1 giulia users      2607 Jan 25 15:23 .history
drwx----- 2 giulia users        512 Apr 13 2000 Mail
-rw----- 1 giulia users    282244 Jul 17 2002 bookmarks.html
drwxr-xr-x 2 giulia users        512 Jan 12 2004 help
lrwxrwxrwx 1 giulia users         15 Jan 26 08:52 help-alex -> /home/alex/help
-rwxr-xr-x 1 giulia users        280 Jun 15 2004 lq

```

Osserviamo in dettaglio una riga dell'output del comando `ls -la`

```
drwxr-xr-x 8 giulia users      512 Jun 7 2004 .Mathematica
```

In prima colonna troviamo la stringa `drwxr-xr-x`. Il primo carattere, nell'esempio "d", indica il tipo di file.

I tipi di file più comuni nelle directory degli utenti sono:

- d directory

- l link (puntatore)

- file ordinario

I nove caratteri successivi indicano le autorizzazioni all'accesso, nell'esempio:

r	w	x	r	-	x	r	-	x
autorizzazione per il proprietario			autorizzazione per il gruppo			autorizzazione per gli altri		

I diritti di accesso sono:

- r read (accesso aperto in lettura)

- w write (accesso aperto in scrittura)

- x execute (accesso aperto in esecuzione)

- permesso negato

È possibile modificare i diritti di accesso su un file utilizzando il comando `chmod`, descritto più avanti in queste note.

Il numero in seconda colonna indica il numero di file collegati al file. Se il file è un directory, indica il numero di sottodirectory contenute (ce ne sono sempre almeno due, la `.` e la `..`). Le due colonne successive contengono rispettivamente il nome dell'utente e del gruppo proprietario del file. In quinta colonna è riportata la dimensione in byte. Seguono la data e l'ora di ultima modifica ed infine il nome del file.

`ls -F [nome-dir]`

dopo il nome di ciascun file, è visualizzato un carattere che indica il tipo di file.

```
archimede%ls -F
Mail/          bookmarks.html  help/          help-alex@    lq*
```

Il carattere "/" indica una directory, il carattere "@" indica un link, il carattere "*" indica un file eseguibile.

`ls -R [nome-dir]`

riporta ricorsivamente l'elenco dei file contenuti nella directory `nome-dir` e nelle sue sottodirectory.

`ls -t [nome-dir]`

riporta l'elenco file contenuti nella directory `nome-dir` in ordine rispetto alla data e ora di ultima modifica.

- **file nome-file**

riporta il tipo di file.

Riprendiamo i file contenuti nell'esempio precedente:

```
archimede%file Mail/
Mail/:          directory
archimede%file bookmarks.html
bookmarks.html: ascii text
archimede%file help-alex
help-alex:      symbolic link to /home/alex/help
```

- **find nome-dir -name nome-file**

cerca il file `nome-file` a partire dalla directory `nome-dir` scendendo ricorsivamente nelle sottodirectory. Può essere utile per cercare un file di cui si conosce il nome, ma di cui non si ricorda il *path* completo. Supponiamo ad esempio di voler trovare il file `uso-matlab` che si trova in una delle sottodirectory della nostra home directory. Con il comando:

```
archimede%find . -name uso_matlab -print
./help/uso_matlab
```

chiediamo al sistema di cercare il file `uso-matlab` a partire dalla directory ".", ovvero dalla directory corrente. Il sistema ci risponde, indicando che il file cercato è contenuto nella directory `help`, sottodirectory della directory corrente.

Uso dei caratteri jolly * e ? e del tasto [TAB]

- Il carattere `*` utilizzato nel nome di un file, sostituisce un qualunque gruppo di caratteri.

Se ad esempio, nella directory corrente sono contenuti i file:

```
archive articolo.tex index.html lettera.txt mail memo1.txt memo2.txt
```

con il comando `"ls *.txt"`, otterremo il seguente output:

```
lettera.txt memo1.txt memo2.txt
```

- Il carattere `?`, utilizzato nel nome di un file, sostituisce un singolo carattere.

Nell'esempio precedente, inviando il comando `"ls memo?.txt"` otterremo:

```
memo1.txt memo2.txt
```

- Nell'invio di un comando, molte shell permettono di utilizzare il tasto [TAB] per completare il nome di un file, quando questo è univocamente individuato dai caratteri digitati.

Supponiamo di essere ancora posizionati nella directory dell'esempio precedente e di voler inviare il comando `"ls -l archive"`.

Potremo digitare `"ls -l arc"` e poi premere il tasto [TAB]. Il nome del file sarà completato automaticamente.

Creazione, cancellazione e cambio di directory (comandi mkdir, rmdir, cd)

- **mkdir nome-dir**

crea una nuova directory

- **rmdir nome-dir**

rimuove la directory specificata (deve essere vuota)

- **cd [nome-dir]**

cambia directory, spostandosi nella directory indicata. Se `nome-dir` non è specificato, si sposta nella home directory dell'utente

Copia, spostamento/rinomina, e cancellazione di file (comandi cp, mv, rm)

- **cp file1 file2**

copia il file `file1` sul file `file2`. Se `file2` non esiste, crea un nuovo file, altrimenti lo sovrascrive.

Nell'indicare `file1` e `file2` è necessario specificare, ove necessario, il *pathname* completo.

Vediamo qualche esempio:

```
cp lettera1.txt lettera2.txt
```

copia il file `lettera1.txt`, contenuto nella directory corrente, sul file `lettera2.txt`, nella directory corrente;

```
cp ~paolo/lettera1.txt lettera2.txt
```

copia il file `lettera1.txt`, contenuto nella home directory di `paolo`, sul file `lettera2.txt`, nella directory corrente;

```
cp ~paolo/lettera1.txt ./lettera2.txt
```

come sopra;

<code>cp ~paolo/lettera1.txt .</code>	copia il file <code>lettera1.txt</code> dalla home directory di paolo, alla directory corrente, mantenendo lo stesso nome <code>lettera1.txt</code> ;
<code>cp /tmp/foto.jpg immagini/foto.jpg</code>	copia il file <code>foto.jpg</code> dalla directory <code>/tmp</code> alla directory <code>immagini</code> , sottodirectory della directory corrente;
<code>cp /tmp/foto.jpg immagini/</code>	come sopra.

Qualche opzione del comando `cp` (è possibile combinare insieme più opzioni):

`cp -i file1 file2`
entra in modalità interattiva e chiede conferma prima di sovrascrivere `file2`, se già esiste.

`cp -r dir1 dir2`
copia ricorsivamente la directory `dir1`, con tutto il suo contenuto, sulla directory `dir2`. Se la directory `dir2` già esiste, la directory `dir1` viene copiata dentro la directory `dir2`.

• **mv file1 file2**

sposta il file `file1` sul file `file2`. Se `file2` non esiste, crea un nuovo file, altrimenti lo sovrascrive. Questo comando può essere utilizzato per cambiare il nome di un file o per spostarlo in un'altra directory.

Nell'indicare `file1` e `file2` è necessario specificare, ove necessario, il *pathname* completo.

Come per il comando `cp`, è possibile utilizzare l'opzione `-i` per entrare in modalità interattiva ed evitare la sovrascrittura del file di destinazione senza richiesta di conferma.

Vediamo in particolare il caso in cui `file1` e `file2` sono i nome di due directory:

`mv dir1 dir2`

se la directory `dir2` non esiste, la directory `dir1` viene rinominata/spostata sulla directory `dir2`. Se la directory `dir2` esiste, la directory `dir1` viene spostata dentro la directory `dir2`.

• **rm file1 [file2 file3 ...]**

elimina il/i file indicati.

Qualche opzione del comando `rm`:

`rm -i file1 [file2 file3 ...]`

entra in modalità interattiva e chiede conferma prima di cancellare un file (su alcuni sistemi è la modalità di lavoro standard del comando `rm`);

`rm -r dir1`

rimuove ricorsivamente la directory `dir1` e tutto il suo contenuto, comprese le sottodirectory.

Visualizzazione del contenuto di un file (comandi `more`, `cat`, `tail`, `head`)

• **more nome-file**

visualizza sul terminale il contenuto di un file di testo, mostrando una schermata alla volta. Per scorrere all'interno del documento utilizzare i tasti:

[INVIO]	– avanza di una riga
barra spaziatrice	– avanza di una schermata
b	– torna indietro di una schermata
q	– esce

• **cat file1 [file2 file3 ...]**

concatena file e li invia su *standard output* (normalmente sullo schermo).

• **tail nome-file**

visualizza la parte terminale di un file di testo, normalmente le ultime 10 righe. Utilizzando l'opzione `-n`, è possibile specificare il numero di righe da visualizzare. Ad esempio, il comando

`"tail -20 lettera.txt"` visualizzerà le ultime 20 righe del file `lettera.txt`.

- **head nome-file**

visualizza la parte iniziale di un file di testo, normalmente le prime 10 righe. Utilizzando l'opzione `-n`, è possibile specificare il numero di righe da visualizzare. Ad esempio, il comando `"head -20 lettera.txt"` visualizzerà le prime 20 righe del file `lettera.txt`.

Altri comandi per analizzare il contenuto di un file (comandi diff, grep, sort, wc)

- **diff file1 file2**

confronta il contenuto di due file e riporta le differenze.

- **grep stringa nome-file**

visualizza le righe del file `nome-file` che contengono la stringa specificata. Ad esempio il comando `"grep giulia /etc/passwd"` visualizzerà tutte le righe del file `/etc/passwd` contenenti il nome `giulia`. Utilizzato con l'opzione `-i` (*ignore*), il comando `grep` non distingue fra lettere maiuscole e minuscole.

- **sort file1 [file2 ...]**

ordina alfabeticamente le linee dei file di testo `file1`, `file2`, ... e riporta il risultato sullo standard output (normalmente sullo schermo).

L'opzione `-r` inverte l'ordine

L'opzione `-o` permette di specificare il nome di un file da utilizzare come output,

esempio: `sort file1 -o file2`.

- **wc nome-file**

riporta il numero di righe, parole e caratteri contenuti nel file `nome-file`.

Con l'opzione `-c` stampa solo il numero di caratteri.

Con l'opzione `-w` stampa solo il numero di parole.

Con l'opzione `-l` stampa solo il numero di linee.

Link simbolici (comando ln)

- **ln -s file1 puntatore1**

crea il puntatore `puntatore1` verso il file `file1`. Consente di riferirsi ad uno stesso file/directory, utilizzando un nome diverso e/o seguendo un percorso diverso. Può essere utile per accedere a file "lontani", con un *pathname* molto lungo, digitando un nome più breve. Ad esempio:

```
archimede%ln -s /usr/local/mathematica/Configuration/Licensing/mathpass mio-mathpa
archimede%ls -l mio-mathpa
lrwxrwxrwx 1 studente users          55 Jan 27 09:58 mio-mathpa ->
/usr/local/mathematica/Configuration/Licensing/mathpass
```

In questo modo sarà possibile riferirsi al file `/usr/local/mathematica/Configuration/Licensing/mathpass`, utilizzando il nome `mio-mathpa`.

Modifica dei permessi per l'accesso a file e directory (comando chmod)

- **chmod permessi nome-file**

Il comando `chmod` permette di modificare le autorizzazioni per l'accesso ad un file.

Abbiamo già visto come si legge la *maschera dei permessi*. Supponiamo ad esempio che nella directory corrente sia contenuto il file `mio_programma`. Con il comando `"ls -l mio_programma"` possiamo vedere quali sono le autorizzazioni per l'accesso al file:

```
archimede%ls -l mio_programma
-rwxr-xr-- 1 giulia users          9 Jan 28 09:50 mio_programma
```

L'utente proprietario del file (*giulia*) ha accesso al file in lettura, scrittura, ed esecuzione (rwx).

Il gruppo proprietario del file (*users*) ha accesso al file in lettura ed esecuzione (r-x).

Gli altri hanno accesso in sola lettura (r-).

Con il comando `chmod` è possibile aggiungere (+) o togliere (-) l'accesso in lettura (r), scrittura (w), esecuzione (x) all'utente proprietario (u), al gruppo proprietario (g), agli altri *others* (o) o a tutti *all* (a).

Vediamo alcuni esempi:

<code>chmod u-w mio_programma</code>	toglie l'accesso in scrittura all'utente proprietario.
<code>chmod go-wx mio_programma</code>	toglie l'accesso in scrittura ed in esecuzione al gruppo ed agli altri.
<code>chmod ugo+rx mio_programma</code>	apre l'accesso in lettura ed in esecuzione a tutti (utente, gruppo e altri).
<code>chmod a+rx mio_programma</code>	come sopra.

Solo l'utente proprietario del file, o l'utente `root` (amministratore del sistema), possono modificare i permessi sul file.

Riepilogo dei comandi presentati

<code>cat</code>	– concatena e visualizza file
<code>cd</code>	– cambia la directory corrente
<code>chmod</code>	– modifica i permessi di accesso ad un file
<code>cp</code>	– copia file e directory
<code>diff</code>	– trova differenze tra due file
<code>file</code>	– riporta il tipo di file
<code>find</code>	– cerca file in una gerarchia di directory
<code>grep</code>	– seleziona le righe di un file che contengono una stringa specificata
<code>head</code>	– visualizza la prima parte di un file
<code>ln</code>	– crea un collegamento tra file
<code>ls</code>	– visualizza il contenuto di una directory
<code>mkdir</code>	– crea una directory
<code>more</code>	– visualizza il contenuto di un file
<code>mv</code>	– sposta/rinomina file e directory
<code>rm</code>	– cancella file o directory
<code>rmdir</code>	– cancella una directory (vuota)
<code>sort</code>	– ordina le linee di un file di testo
<code>tail</code>	– visualizza la parte finale di un file
<code>wc</code>	– riporta il numero di caratteri, parole e righe di un file

Per maggiori dettagli sui comandi presentati utilizzare il manuale in linea.

RINDIRIZZAMENTO DELL'OUTPUT E CONCATENAZIONE DI COMANDI

È possibile inviare l'output di un comando su un file, utilizzando gli operatori di reindirizzamento `>` e `>>`.
È possibile utilizzare l'output di un comando come input di un nuovo comando, utilizzando l'operatore di concatenazione `|` (*pipe*).

- **comando > nome-file**

ridirige su un **file nuovo** l'output di un comando. Se il file già esiste, viene sovrascritto.

Ad esempio:

```
ls -l /tmp > lista      crea sulla directory corrente il file lista, contenente l'elenco dei file
                        contenuti nella directory /tmp. Se nella directory corrente è già presente il
                        file lista, viene sovrascritto.

cat file1 file2 > file3 crea sulla directory corrente il file file3, concatenando il contenuto di
                        file1 ed il contenuto di file2. Se nella directory corrente è già presente il
                        file file3, viene sovrascritto.

sort file1 file2 > file3 crea sulla directory corrente il file file3, contenente le righe dei file file1 e
                        file2 ordinate alfabeticamente.
```

- **comando >> nome-file**

ridirige su un **file già esistente** l'output di un comando, aggiungendo le nuove informazioni alla fine del file.

Ad esempio:

```
cat file2 >> file1     aggiunge il contenuto del file file2, in fondo al file file1.

tail -5 file2 >> file1 aggiunge le ultime 5 righe del file file2, in fondo al file file1.
```

- **comando1 | comando2**

esegue i comandi `comando1` e `comando2`, utilizzando l'output di `comando1` come input per `comando2`. Il `comando1` è *filtrato* dal `comando2`.

Ad esempio:

```
ls -l /tmp | more      visualizza l'elenco dei file contenuti nella directory /tmp, mostrando una
                        schermata alla volta.

ls -l /etc | grep passwd visualizza l'elenco dei file contenuti nella directory /etc, i cui nomi
                        contengono la stringa "passwd".

ls -a | wc -w          riporta il numero di file contenuti nella directory corrente.
```

COME LOCALIZZARE UN COMANDO

Quando viene digitato un comando, il sistema cerca l'eseguibile corrispondente in una delle directory contenute nella variabile d'ambiente PATH. È possibile visualizzare il contenuto della variabile PATH, utilizzando il comando `echo` o il comando `printenv`:

```
archimede%echo $PATH                (oppure  printenv PATH)
/usr/local/bin:/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin:.:
/usr/local/java/jdk/jre/bin:/usr/local/java/jdk/bin:/usr/X11R6/bin
```

Per sapere in quale directory sia contenuto un comando, possiamo utilizzare l'istruzione:

which comando

`which` cerca il comando specificato nelle directory contenute nella variabile PATH e restituisce il nome del comando, completo del percorso. Cerchiamo ad esempio il nome completo del comando `who`:

```
archimede%which who
/usr/bin/who
```

Se inviamo un comando e questo non viene trovato, il sistema restituisce un messaggio di errore, come nel seguente esempio:

```
archimede%adduser
adduser: Command not found.
```

In questo caso il comando potrebbe non essere disponibile sul sistema, oppure potrebbe trovarsi in una directory non compresa nella variabile PATH. Possiamo allora tentare di localizzarlo, utilizzando l'istruzione:

whereis comando

`whereis` cerca il sorgente, l'eseguibile e le pagine di manuale relative al comando specificato in una lista di directory standard.

```
archimede%whereis adduser
adduser: /usr/sbin/adduser /usr/share/man/man8/adduser.8.gz
```

Riepilogo dei comandi presentati

- `whereis` – cerca il sorgente, l'eseguibile e le pagine di manuale di un comando
- `which` – visualizza il path completo di un comando

Per maggiori dettagli sui comandi presentati utilizzare il manuale in linea.

EDITING DI FILE DI TESTO

Un **editor di testo** è un programma per creare e modificare file ASCII (*American Standard Code for Information Interchange*), come ad esempio il sorgente di un programma C o fortran, o una lettera ad un amico. Un editor di testo non dispone di comandi "avanzati" di formattazione, per cui non sarà possibile inserire caratteri in grassetto o in corsivo, aumentare o ridurre la dimensione dei caratteri, etc. Il documento sul quale l'editor lavora è chiamato *buffer*.

Gli editor più utilizzati in ambiente UNIX sono **vi** ed **emacs**. Un altro editor, meno potente, ma molto semplice da utilizzare è **pico**.

Cenni sull'utilizzo dell'editor vi

Per editare un file con vi, inviare il comando:

```
vi nome-file
```

Il file *nome-file* sarà caricato da vi nella finestra di *editing*.

L'editor vi ha due diverse modalità operative: la *modalità comando*, che consente di inserire comandi per la gestione dell'intero file, e la *modalità inserimento*, che consente di inserire testo. All'avvio di vi, ci troveremo automaticamente in modalità comando.

Per passare dalla modalità comando alla modalità inserimento utilizzare il tasto "i", per inserire a sinistra del cursore, o il tasto "a" per inserire a destra del cursore.

Per tornare in modalità comando utilizzare il tasto [Esc].

Per uscire da vi, bisogna trovarsi in modalità comando e digitare i seguenti caratteri:

:q! per uscire senza salvare le modifiche;

:wq per uscire salvando le modifiche.

In modalità inserimento è possibile inserire caratteri nel punto sul quale ci si trova posizionati. Per muoversi nel testo, per cancellare, per sostituire parti del testo, ci si deve posizionare in modalità comando ed utilizzare le opportune sequenze di caratteri. Seguono alcuni dei comandi più utili:

j per spostarsi in basso;

k per spostarsi in alto;

l per spostarsi a destra;

h per spostarsi a sinistra;

i per inserire testo a sinistra del cursore ([Esc] per terminare l'inserimento);

a per inserire testo a destra del cursore ([Esc] per terminare l'inserimento);

r *carattere* per sostituire un carattere;

x per cancellare un carattere;

dd per cancellare una linea;

:s/*vecchio*/*nuovo* per sostituire, nella linea corrente, la prima ricorrenza di *vecchio* con *nuovo*;

:s/*vecchio*/*nuovo*/g per sostituire, nella linea corrente, tutte le ricorrenze di *vecchio* con *nuovo*;

:1,\$ s/*vecchio*/*nuovo*/g per sostituire, nell'intero testo, tutte le ricorrenze di *vecchio* con *nuovo*.

vi è un editor potente che dispone di moltissimi altri comandi.

Per maggiori informazioni si rimanda ai manuali specifici ed alla documentazione in linea sul sistema.

Cenni sull'utilizzo dell'editor emacs

Per editare un file con emacs, inviare il comando:

```
emacs nome-file
```

Il file *nome-file* sarà caricato nella finestra di *editing*.

Per uscire da emacs, selezionare "Exit Emacs" alla voce "Files" del menù, oppure digitare la sequenza di tasti <CTRL>x <CTRL>c. Se sono state apportate modifiche al file aperto, emacs chiederà la conferma, prima di salvare il file.

Per salvare le modifiche, senza uscire dall'editor, selezionare "Save Buffer" alla voce "Files" del menù, oppure digitare la sequenza di tasti <CTRL>x <CTRL>s.

emacs è certamente più facile da usare rispetto a vi. Tutto funziona in maniera molto semplice. Il testo viene inserito automaticamente nel punto in cui si trova posizionato il cursore. È possibile spostarsi nel testo utilizzando sia il mouse sia la tastiera. Sul menù sono disponibili la maggior parte dei comandi tipici dei programmi di editing (copy, cut, paste, search, etc.)

Per maggiori informazioni si rimanda all'help in linea, richiamabile dal menù di emacs alla voce "Help".

Cenni sull'utilizzo dell'editor pico

Per editare un file con pico, inviare il comando:

```
pico nome-file
```

Il file *nome-file* sarà caricato nella finestra di *editing*.

Pico è uno degli editor più facili da usare ed è utilizzato dal programma per la gestione della posta elettronica pine, come editor per la scrittura dei mail.

Utilizzando pico, è possibile spostarsi nel testo utilizzando la tastiera (non il mouse) ed inserire nuovo testo nel punto in cui è posizionato il cursore.

Per uscire da pico digitare contemporaneamente i tasti <CTRL>x, (comando "Exit"). Se sono state apportate modifiche al file aperto, pico chiederà la conferma prima di salvare il file.

Per salvare le modifiche, senza uscire dall'editor, digitare contemporaneamente i tasti <CTRL>o, (comando "Write Out"). Verrà richiesto la conferma del nome da dare al file.

Gli altri comandi disponibili si ottengono sempre digitando il tasto <CTRL> insieme ad un altro carattere e sono indicati nella parte bassa della finestra di editing. Alcuni fra i più utili sono:

- <CTRL>x per uscire (*Exit*);
- <CTRL>o per salvare (*WriteOut*);
- <CTRL>r per importare un file nel buffer di editing (*Read File*);
- <CTRL>w per cercare una stringa (*Where is*);
- <CTRL>k per tagliare una riga o la parte di testo selezionata (*Cut Text*);
- <CTRL>u per incollare il testo tagliato (*UnCut Text*);
- <CTRL>^ per impostare/rimuovere un marcatore (*Mark Set*). Utile soprattutto per segnare l'inizio di una sezione di testo da tagliare. Una volta impostato il marcatore, si può selezionare il testo muovendosi con le frecce o con i tasti <CTRL>v (*Next Page*) e <CTRL>y (*Previous Page*).

Per maggiori informazioni e per avere l'elenco completo dei comandi si rimanda alla documentazione in linea sul sistema ed all'help richiamabile da pico con il comando <CTRL>g (*Get Help*).

STAMPA DEL CONTENUTO DI UN FILE

Esistono vari modi per stampare il contenuto di un file.

La maggior parte delle applicazioni installate sul sistema, se opportunamente configurate, consentono di inviare file in stampa, utilizzando la voce "Print" dal "menù a tendina" (*pop menù*) dell'applicazione stessa.

È anche possibile inviare file in stampa direttamente dalla *shell*, utilizzando gli appositi comandi UNIX.

Prima di vedere questi comandi, sono necessarie alcune premesse.

- Le operazioni di stampa sono operazioni tipicamente "*site dependent*", ovvero dipendono dal modello e dalla configurazione delle stampanti disponibili; dalle scelte organizzative per l'accesso ai servizi di stampa; dai *filtri* di stampa installati.
- I *filtri di stampa* sono programmi che intercettano ed elaborano il file prima che questo arrivi alla stampante. Un filtro può essere utilizzato, ad esempio, per inserire automaticamente intestazioni, o per convertire il formato del file da stampare, in un formato accettato dalla stampante che si desidera utilizzare. È importante sapere, infatti, che una stampante non è in grado di stampare un file, qualunque sia il suo formato. Una stampante PostScript (ps), ad esempio, accetta solo file in formato ps ed è quindi necessario convertire il file dal formato originario in formato ps, prima di poterlo stampare. Su alcuni sistemi i filtri vengono richiamati automaticamente all'invio del comando di stampa e la conversione avviene in maniera del tutto trasparente per l'utente. Su altri sistemi, diversamente organizzati, è necessario che l'utente converta il file nel formato accettato dalla stampante prima di poterlo stampare.
- Generalmente su un sistema UNIX è possibile indirizzare più di una stampante. Le stampanti possono essere collegate direttamente al sistema, ad esempio su una porta parallela o su una porta USB, o possono essere accessibili via rete. Ogni sistema ha una stampante predefinita, la *stampante di default*, che viene indirizzata automaticamente quando l'utente non indica il nome di una stampante specifica. Un utente può modificare la scelta della stampante di default, utilizzando la variabile d'ambiente PRINTER. Per definire e visualizzare il valore di una variabile d'ambiente, si possono utilizzare rispettivamente i comandi `setenv` ed `echo 0 printenv`:

```
archimede%setenv PRINTER mia_stampante
archimede%echo $PRINTER
mia_stampante
```

- **lpr nome-file**

invia il file `nome-file` alla stampante di default, ovvero alla stampante predefinita sul sistema o alla stampante definita nella variabile d'ambiente PRINTER. Se il file da stampare è un file in formato testo, come ad esempio il sorgente di un programma C o fortran, potrà essere necessario trasformare il file in formato *postscript*, prima di mandarlo in stampa. Ad esempio, per stampare il file `mioprogram.cpp`, si potrà utilizzare la sequenza di comandi:

```
a2ps mioprogram.cpp -o mioprogram.ps
lpr mioprogram.ps
```

Il comando `a2ps` trasforma il file testo ricevuto in input in formato *postscript*. Se non vengono specificate opzioni, i fogli del file di output saranno orientati in senso orizzontale e su ogni foglio saranno stampate due pagine. Se si desidera che il file di output abbia i fogli orientati in senso verticale e che su ogni foglio venga stampata una sola pagina, si dovrà utilizzare il comando:

```
a2ps mioprogram.cpp -o mioprogram.ps --chars-per-line=80 --columns=1 --portrait
```

Prima di inviare in stampa il file `mioprogram.ps`, è possibile visualizzarlo a schermo utilizzando il comando:

```
gv mioprogram.ps &
```

Vediamo ora qualche opzione del comando `lpr`:

```
lpr -P nome-stampante nome-file
```

l'opzione `-P` permette di specificare il nome della stampante.

Ad esempio `lpr -Pcolor_jw lettera.txt` invia il file `lettera.txt` alla stampante `color_jw`.

lpr -h nome-file

l'opzione `-h` sopprime la stampa della pagina d'intestazione. Spesso le stampanti sono già configurate per non stampare la pagina d'intestazione.

lpr -s nome-file

l'opzione `-s`, anzichè copiare il file nella directory di *spool* della stampante, utilizza un link simbolico. Questo permette di stampare file di dimensioni maggiori di quelle consentite dalla directory di *spool*. Ovviamente il file non deve essere rimosso o modificato fino a quando la stampa non è ultimata.

- **lpq**

visualizza l'elenco dei lavori (*job*) sulla coda della stampante di default.

L'opzione `"-P nome-stampante"` permette di indicare una stampante specifica.

- **lprm numero_job**

rimuove un job dalla coda della stampante di default.

L'opzione `"-P nome-stampante"` permette di indicare una stampante specifica.

Vediamo un esempio, in cui vengono utilizzati i comandi `a2ps`, `lpr`, `lpq`, `lprm`.

```
archimede%lpr -Pmia_stampante lettera.txt
archimede%a2ps esempio.cpp -o esempio.ps
archimede%lpr -Pmia_stampante esempio.ps
archimede%lpq -Pmia_stampante
lp is ready and printing
Rank  Owner      Job  Files              Total Size
active giulia   341  lettera.txt         11 bytes
1st   giulia   342  esempio.ps        184418 bytes
archimede%lprm -Pmia_stampante 341
341 dequeued
archimede%lpq -Pmia_stampante
lp is ready and printing
Rank  Owner      Job  Files              Total Size
active giulia   342  esempio.ps        184418 bytes
```

Riepilogo dei comandi presentati

<code>lpr</code>	– stampa un file
<code>lpq</code>	– mostra il contenuto di una coda di stampa
<code>lprm</code>	– rimuove un lavoro da una coda di stampa

Per maggiori dettagli sui comandi presentati utilizzare il manuale in linea.

CONTROLLO DELLO SPAZIO DISCO ED ARCHIVIAZIONE DI FILE E DIRECTORY

Su un sistema *multitasking* e *multiutente*, le risorse sono condivise ed è quindi molto importante utilizzarle in maniera oculata, evitando inutili sprechi di spazio disco, di tempo di CPU, di utilizzo dei servizi di stampa, etc.

In questo paragrafo vengono descritti alcuni comandi che consentono all'utente di controllare lo spazio disco occupato e di archiviare e comprimere file e directory.

Controllo dello spazio disco occupato (comandi `du`, `quota`, `df`)

Il sistema operativo UNIX dispone di comandi specifici, che consentono ad un utente di individuare velocemente le directory ed i file che occupano una notevole quantità di spazio disco (comando `du`), di controllare i limiti di quota disco definiti dall'amministratore del sistema (comando `quota`) e lo spazio disco disponibile (comando `df`).

• `du -k`

riporta l'ammontare di spazio su disco usato dalla directory corrente e dalle sue sottodirectory.

Utilizzando l'opzione `-k` lo spazio disco è misurato in Kbyte, ovvero in blocchi da 1024 byte. Su alcuni sistemi, senza l'opzione `-k`, lo spazio disco è misurato in blocchi da 512 byte.

Con l'opzione `-h` lo spazio disco è riportato in kilobyte (1 KB = 1024 byte), in megabyte (1 MB = 1024 KB) o in gigabyte (1 GB = 1024 MB) e risulta più facilmente leggibile.

È possibile specificare come argomento il nome di un file `"du -k nome-file"` o il nome di una directory `"du -k nome-dir"`. In quest'ultimo caso, il comando `du` scende ricorsivamente nelle sottodirectory della directory `nome-dir` e riporta il totale dello spazio occupato da tutte le sottodirectory e da tutti i file in essa contenuti.

Vediamo un esempio in cui si utilizza il comando `du`:

```
archimede%ls -la
total 84
drwxr-xr-x  4 giulia  users      512 Feb  1 12:20 .
drwxr-xr-x 31 root    root      2560 Feb  1 09:49 ..
-rw-r--r--  1 giulia  users     1487 Nov 29 08:52 .cshrc
-rw-----  1 giulia  users     2607 Jan 25 15:23 .history
drwx-----  2 giulia  users      512 Apr 13 2000 Mail
-rw-----  1 giulia  users    28244 Jul 17 2002 bookmarks.html
-rw-r--r--  1 giulia  users      8 Feb  1 10:20 esempio.f90
drwxr-xr-x  2 giulia  users      512 Jan 31 12:16 help
lrwxrwxrwx  1 giulia  users      15 Jan 26 08:52 help-alex -> /home/alex/help
-rwxr-xr-x  1 giulia  users      280 Jun 15 2004 lq
archimede%du -k bookmarks.html
28      bookmarks.html
archimede%du -k
1      ./Mail
22     ./help
60     .
```

• `quota -v`

riporta lo spazio disco occupato dall'utente ed i suoi limiti di quota.

Lo spazio disco è misurato in Kbyte (1 Kbyte = 1024 byte).

```
archimede%quota -v
Disk quotas for giulia (uid 174):
Filesystem      usage  quota  limit  timeleft  files  quota  limit  timeleft
/home           46973 150000 160000          795   2500   2600
```

Nell'esempio, l'utente *giulia* sta occupando circa 46 Mbyte (1 Mbyte = 1024 Kbyte) di spazio disco sul filesystem `/home`. L'amministratore del sistema ha definito un *soft limit*, pari a circa 150 Mbyte, ed una *hard limit*, pari a circa 160 Mbyte, per la quota disco di *giulia*. *giulia* ha 795 file. Per il numero di file, il *soft limit* assegnatole è pari a 2500, l'*hard limit* è pari a 2600.

Ogni utente può superare il proprio *soft limit* per un breve periodo, generalmente per 7 giorni. Se il *soft limit* è stato superato, nel campo *timeleft* viene indicato il numero di giorni per i quali è ancora consentito lo "sforamento" di quota.

Se sul sistema non sono definiti limiti di quota disco, si otterrà un messaggio del tipo:

```
archimede%quota -v
Disk quotas for user giulia (uid 174): none
```

- **df -k**

visualizza l'elenco dei filesystem montati sul sistema, lo spazio utilizzato e lo spazio disponibile su ciascuno di essi.

Utilizzando l'opzione `-k` lo spazio disco è misurato in Kbyte (1 Kbyte = 1024 byte). Su alcuni sistemi, senza l'opzione `-k`, lo spazio disco è misurato in blocchi da 512 byte.

Con l'opzione `-h` lo spazio disco è riportato in kilobyte (1 KB = 1024 byte), in megabyte (1 MB = 1024 KB) o in gigabyte (1 GB = 1024 MB) e risulta più facilmente leggibile.

Un *filesystem* è una *gerarchia* di directory, ovvero una struttura ad albero nella quale sono organizzate in maniera gerarchica directory e sottodirectory. La directory radice di un filesystem UNIX, prende il nome di `root` ed è indicata con il simbolo `/`. Sotto il `root filesystem` sono generalmente "montati" altri filesystem, che corrispondono a:

- partizioni del disco, o dei dischi, attaccati direttamente sul computer;

- directory contenute sul disco di un computer remoto, accessibile via rete (*network filesystem*);

- altre unità di memorizzazione disponibili sul sistema (cdrom, floppy disk, pendrive usb, etc.).

Nel filesystem sono quindi accorpate ed organizzate tutte le unità di memorizzazione disponibili sul sistema. Ciascuna unità di memorizzazione è "agganciata" sotto una directory, che prende il nome di *mount point*. Su tale directory sono disponibili i file e le directory contenuti nell'unità di memorizzazione.

```
archimede%df -k
Filesystem                1k-blocks      Used Available Use% Mounted on
/dev/sda1                  101089         69477    26393    73% /
/dev/sda5                  1517920        539620   901192    38% /home
/dev/sda6                   497829          500    471627     1% /tmp
/dev/sda9                  5629284        3727264  1616068    70% /usr
/dev/sda8                   497829          84709   387418    18% /var
/dev/sdb2                   8301712        4141628  3738380    53% /usr1
/dev/hdd1                  39516244       13302340 24206584    36% /mnt/hdd
marte:/W3                  26835134       2202063  24364720     9% /W3
/dev/fd0                    1423           1254      170    89% /mnt/floppy
```

Nell'esempio il sistema monta i seguenti filesystem:

- `/`, `/home`, `/tmp`, `/usr`, `/var`, partizioni dello SCSI disk a (*sda*);

- `/usr1` partizione dello SCSI disk b (*sdb*);

- `/mnt/hdd` partizione del hard disk IDE d (*hdd*);

- `/W3` *network filesystem*, corrispondente alla directory `/W3` dell'host remoto *marte*.

- `/mnt/floppy` corrispondente al floppy disk (*fd0*).

Archiviare e comprimere file e directory (comandi `gzip`, `compress`, `zip`, `tar`)

Su un sistema UNIX sono generalmente disponibili varie utility per archiviare e comprimere file e directory. I comandi di compressione più utilizzati sono `gzip`, `compress` e `zip`. Le utility di archiviazione più diffuse sono `zip` e `tar`. È importante conoscere ed utilizzare questi comandi, per evitare inutili sprechi di spazio disco.

Seguono alcune indicazioni sulle modalità di utilizzo di queste utility, per informazioni più dettagliate si rimanda al manuale in linea.

- **gzip nome-file**

- **gunzip nome-file**

comprime e decomprime file.

Il file compresso è rimpiazzato da un file con lo stesso nome e con estensione `.gz`, mantiene gli stessi diritti di accesso e la stessa data di modifica.

L'ammontare della compressione ottenuta dipende dalle caratteristiche del file di input. Tipicamente, testi come codici sorgenti e file di testo sono ridotti del 60-70%. Nel seguente esempio il file compresso occupa uno spazio pari a circa il 29% rispetto al file originale:

```
archimede%ls -l tesi*
-rwxr-xr-x  1 giulia  users      83199 Apr  6 14:32 tesi.txt
archimede%gzip tesi.txt
archimede%ls -l tesi*
-rwxr-xr-x  1 giulia  users      24105 Apr  6 14:32 tesi.txt.gz
```

- **compress nome-file**
- **uncompress nome-file**

comprime e decomprime file.

Il file compresso è rimpiazzato da un file con lo stesso nome e con estensione `.z`, mantiene gli stessi diritti di accesso e la stessa data di modifica.

L'ammontare della compressione ottenuta dipende dalle caratteristiche del file di input. Tipicamente, testi come codici sorgenti e file di testo sono ridotti del 50-60%. Nel seguente esempio il file compresso occupa uno spazio pari a circa il 43% rispetto al file originale:

```
archimede%ls -l tesi*
-rwxr-xr-x  1 giulia  users      84422 Apr  6 14:56 tesi.txt
venere%compress tesi.txt
archimede%ls -l tesi*
-rwxr-xr-x  1 giulia  users      36487 Apr  6 14:56 tesi.txt.Z
```

- **zip nome-zip-file file1 file2 file3 ...**
- **unzip nome-zip-file**

utility per la compressione e l'archiviazione di file, compatibile con `PKZIP` utilizzata sui sistemi MS-DOS/Windows.

Il programma `zip` salva uno o più file compressi in un unico file con estensione `.zip`. Un'intera directory può essere "impacchettata" in un archivio `.zip` con il solo comando

```
zip -r nome-zip-file nome-dir
```

L'ammontare della compressione ottenuta dipende dalle caratteristiche dei file di input. Tipicamente, per file di testo il rapporto di compressione varia da 2:1 a 3:1.

Vediamo un esempio in cui si utilizza il comando `zip` per comprimere ed archiviare il seguente albero di directory:

```
miadir_____ tesi.pdf
      |_____ tesi.ps
      |_____ immagini _____ logo.gif
      |_____ _____ mozart.gif
```

```
archimede%zip -r miadir.zip miadir
adding: miadir/ (stored 0%)
adding: miadir/images/ (stored 0%)
adding: miadir/images/logo.gif (deflated 1%)
adding: miadir/images/mozart.gif (deflated 3%)
adding: miadir/tesi.pdf (deflated 16%)
adding: miadir/tesi.ps (deflated 78%)
archimede%du -k miadir          | il comando du è utilizzato per
28      miadir/images          | controllare il rapporto di compressione
64      miadir
archimede%du -k miadir.zip
32      miadir.zip
```

A questo punto, si potrà cancellare l'intera directory `miadir`, utilizzando il comando `"rm -r miadir"`. La directory potrà essere ripristinata con il comando `"unzip miadir"`.

- **tar**

l'utility `tar` (acronimo per *Tape ARchive*) permette di impacchettare in un unico file, chiamato *tarfile*, molti file diversi, conservando tutte le informazioni (proprietario, permessi, data) e la struttura delle directory. Può essere utilizzato per archiviare l'intero contenuto di un albero di directory in un unico file.

Modalità di utilizzo:

- per archiviare una directory in un *tarfile* compresso: `tar czvf nome-tar-file nome-dir`
- per vedere l'elenco dei file contenuti in un *tarfile* compresso: `tar tzvf nome-tar-file`
- per estrarre e ripristinare i file archiviati: `tar xzvf nome-tar-file`

Le chiavi `c`, `z`, `v`, `f`, `t`, `x` hanno il seguente significato:

- `c` create
- `z` filtra l'archivio con `gzip`
- `v` verbose
- `f` file
- `t` type
- `x` extract

Un *tarfile* compresso ha generalmente estensione `.tgz`. Un *tarfile* non compresso (generato senza l'opzione `z`) ha generalmente estensione `.tar`.

Su alcuni sistemi l'opzione `z` non è disponibile. In questo caso il file archivio generato con l'utility `tar` può essere compresso utilizzando, ad esempio, il comando `gzip`.

Segue un esempio in cui si utilizzano i comandi `tar` e `gzip` per archiviare e comprimere la directory `miadir` dell'esempio precedente. Nell'esempio si utilizza il comando `du` per controllare il rapporto di compressione.

```
archimede%tar cvf miadir.tar miadir
miadir/
miadir/images/
miadir/images/logo.gif
miadir/images/mozart.gif
miadir/tesi.pdf
miadir/tesi.ps
archimede%gzip miadir.tar
archimede%du -k miadir          | il comando du è utilizzato per
28      miadir/images          | controllare il rapporto di compressione
64      miadir
archimede%du -k miadir.tar.gz
32      miadir.tar.gz
```

A questo punto, si potrà cancellare l'intera directory `miadir`, utilizzando il comando `"rm -r miadir"`. La directory potrà essere ripristinata con i comandi:

```
archimede%gunzip miadir.tar.gz
archimede%tar xvf miadir.tar
miadir/
miadir/images/
miadir/images/logo.gif
miadir/images/mozart.gif
miadir/tesi.pdf
miadir/tesi.ps
```

Riepilogo dei comandi presentati

<code>compress</code>	– comprime, decomprime file
<code>df</code>	– riporta la quantità di spazio usato e disponibile sui filesystem
<code>du</code>	– riporta lo spazio disco utilizzato da directory o file
<code>gzip</code>	– comprime, decomprime file
<code>quota</code>	– riporta lo spazio disco utilizzato ed i limiti di quota
<code>tar</code>	– utility di archiviazione file e directory
<code>zip</code>	– archivia e comprime file

Per maggiori dettagli sui comandi presentati utilizzare il manuale in linea.

GESTIONE DEI PROCESSI

Su un sistema UNIX lavorano contemporaneamente più processi (*multitasking*), che condividono la CPU, utilizzandola a turno per brevi periodi (*timesharing*). Lo *scheduling* della CPU, ovvero l'attività che organizza e garantisce l'accesso alla CPU ed alle risorse del sistema, è uno dei compiti più importanti del sistema operativo ed è fondamentale per un efficiente funzionamento del computer.

Ogni processo è identificato sul sistema dal suo PID (*Process Identifier*).

Per ogni utente collegato al sistema, è sempre attivo almeno un processo, la sua shell di login.

Ogni comando attiva un nuovo processo, che sarà "figlio" del processo dal quale è stato generato. Il processo "padre" resta "congelato" fino a quando il processo "figlio" non è terminato. Sfruttando il multitasking, è possibile sganciare il processo "figlio" (il comando in esecuzione) dal processo "padre" (la shell dalla quale il comando è stato lanciato), in modo tale che entrambi i processi possano procedere indipendentemente. Per fare ciò, basta aggiungere il carattere **&** in fondo alla riga di comando (ad esempio "emacs &"). In questo modo il nuovo processo viene attivato in *background*, restituendo immediatamente il controllo della shell e consentendo l'invio di altri comandi.

Seguono alcuni comandi che permettono di monitorare i processi:

- **ps**

mostra un'istantanea dei processi correnti. Per ogni processo viene indicato:

PID process identifier
TTY numero del terminale
TIME tempo di cpu usato dal processo
CMD comando

Lanciato senza opzioni, mostra solo i processi dell'utente che sono stati lanciati dal terminale (TTY) corrente.

```
archimede%ps
      PID TTY          TIME CMD
 23666 pts/3    00:00:00 tcsh
 29286 pts/3    00:00:00 ps
```

Il comando ps ha molte opzioni. Le opzioni differiscono fra UNIX System V e UNIX BSD. Alcune opzioni servono per controllare il formato dell'output del comando ps, mentre altre permettono di selezionare alcuni processi specifici nella lista di output. Alcune opzioni devono essere precedute dal carattere -, mentre altre opzioni non lo vogliono. Si consiglia di consultare l'help in linea per avere l'elenco completo delle opzioni disponibili.

Alcune fra le opzioni più utili, sono le seguenti:

-e (*every*) riporta la lista di tutti i processi
-A (*all*) stessa funzione di -e
-f (*full listing*) riporta l'output in formato *full listing*
u (*user-oriented*) riporta l'output in formato *user-oriented*

In particolare, per avere l'elenco di tutti i processi in formato *user-oriented* utilizzare il comando:

```
ps -e u
```

per avere l'elenco di tutti i processi in formato *full listing* utilizzare il comando:

```
ps -ef
```

per avere l'elenco di tutti i processi lanciati da un determinato utente, utilizzare i comandi:

```
ps -e u | grep username    0    ps -ef | grep username
```

Utilizzando le opzione u (*user-oriented*) e -f (*full listing*), per ogni processo nella lista di output, vengono riportati, oltre ai campi PID, TTY, TIME e CMD, anche altre informazioni, come si vede nei seguenti esempi:

```
archimede%ps u
```

```

USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
giulia   10060  0.0  0.0  3332 1804 pts/0    Ss   08:02   0:00 -tcsh
giulia   11939  0.0  0.0  2584  884 pts/0    R+   09:19   0:00 ps u

```

```

archimede%ps -f
  UID      PID    PPID  C STIME TTY          TIME CMD
  giulia   10060  10055  0 08:02 pts/0      00:00:00 -tcsh
  giulia   11929  10060  0 09:18 pts/0      00:00:00 ps -f

```

I campi visualizzati con le opzioni `u` e `-f` hanno il seguente significato:

C	Numero dell'ultimo processore utilizzato
CMD	comando
COMMAND	comando
%CPU	percentuale cputime/realtime (sempre <=100%)
%MEM	percentuale di memoria utilizzata
PID	Process IDentifier
PPID	Parent Process IDentifier
RSS	dimensione insieme residente
START	start time
STAT	stato del processo
STIME	starting time del processo
TIME	tempo di cpu usato dal processo
TTY	terminale di controllo
UID	User Id dell'utente proprietario del processo
USER	nome dell'utente proprietario del processo
VSZ	dimensione totale della VM in byte

Lo stato del processo è descritto da una stringa che può avere fino a 3 caratteri.

Il primo campo, sempre presente, può assumere uno dei seguenti valori:

- S Sleeping
- R Running
- Z Zombie
- T sTopped
- D sleep non interrompibile

Il secondo ed il terzo campo servono rispettivamente ad indicare se il processo sta utilizzando l'area di *swap* e se sta girando con una priorità più bassa della priorità di default: il secondo campo è *W* se il processo è *sW*appato; il terzo campo è *N* se il processo ha un valore *nice* positivo.

La funzione *nice* serve a diminuire la priorità di un processo. Valori *nice* positivi comportano maggiore "cortesia" ovvero una diminuzione della priorità. Ad esempio, se si lancia l'editor `emacs` digitando il comando `"nice +2 emacs &"` il processo di editing avrà un valore *nice* pari a 2 e girerà ad una priorità più bassa.

- **top**

fornisce in tempo reale un *report* circa l'attività della CPU e mostra la lista dei task che ne stanno facendo un uso più intenso. Dispone di un'interfaccia interattiva, i comandi principali che si possono utilizzare sono:

- `h (help)` mostra l'help in linea
- `k (kill)` chiede il PID di un processo e lo interrompe
- `q (quit)` esce
- `u (user)` chiede uno username e mostra solo i processi dell'utente selezionato

Lanciato senza opzioni il comando `top` fornisce alcune informazioni generali sullo stato del sistema e di seguito, per ogni processo compreso nella lista di output, indica:

PID	Process IDentifier
USER	utente proprietario processo
PR	priorità del processo
NI	valore nice (un valore nice negativo indica una priorità più alta, mentre un valore nice positivo indica una priorità più bassa)
VIRT	dimensione della memoria utilizzata per codice, dati e <i>shared libraries</i> ; comprende sia la memoria fisica sia la memoria swap
RES	dimensione della memoria fisica utilizzata (esclusa la swap)
SHR	dimensione della memoria condivisa con altri processi

S stato del processo
%CPU utilizzo della CPU (in percentuale sul totale disponibile)
%MEM utilizzo della memoria fisica (in percentuale sul totale disponibile)
TIME+ tempo totale di cpu usato dal processo
COMMAND comando

- **kill PID**

interrompe il processo identificato dal *Process Identifier* PID.

Il PID di un processo può essere ricavato utilizzando il comando `ps`.

Ogni utente può terminare solo i propri processi. Il superutente *root* può interrompere qualsiasi processo.

L'opzione "-9" invia un segnale di *terminazione incondizionata*: `kill -9 PID`.

Segue un esempio in cui si utilizzano i comandi `ps` e `kill`:

```
archimede%ps -e u | grep giulia
giulia  7474  0.0  0.0  3128 1528 pts/0    Ss   17:01   0:00 -tcsh
giulia  7915  0.0  0.0  4052 1788 pts/0    S+   17:25   0:00 ssh giulia@luna
giulia  7938  0.0  0.0  3116 1520 pts/1    Ss   17:26   0:00 -tcsh
giulia  7955  0.0  0.0  2564  844 pts/1    R+   17:26   0:00 ps -e u
giulia  7956  0.0  0.0  1912  588 pts/1    S+   17:26   0:00 grep giulia
archimede%kill -9 7915
                ( interrompe il collegamento remoto su luna )
```

Riepilogo dei comandi presentati

`kill` – termina un processo
`ps` – riporta lo stato dei processi
`top` – mostra i processi che utilizzano il top della CPU

Per maggiori dettagli sui comandi presentati utilizzare il manuale in linea.

UTILIZZO DI FLOPPY DISK, PENDRIVE USB E CDROM

Un sistema UNIX vede il floppy disk, la pendrive usb ed il cdrom come directory del suo filesystem. Per poter accedere al contenuto di ciascuna di queste periferiche è prima necessario "agganciarle" al filesystem "sotto" una determinata directory. Questa directory rappresenta il "punto di aggancio" e si chiama *mount point*. I *mount point* corrispondenti alle diverse periferiche sono definiti nel file `/etc/fstab` e generalmente hanno nomi standard, come ad esempio `/mnt/floppy` per il floppy disk, `/mnt/usb` per la pendrive usb ed `/mnt/cdrom` per il cdrom.

Per agganciare il contenuto del floppy, della penna usb o del cdrom al filesystem è generalmente necessario richiederlo ogni volta esplicitamente al sistema con il comando `mount`.

Ad esempio, per montare il floppy disk, si dovrà digitare:

```
archimede%mount /mnt/floppy
```

Per smontarlo:

```
archimede%umount /mnt/floppy
```

Comandi analoghi dovranno essere inviati per montare/smontare la penna usb o il cdrom.

Su alcuni sistemi il floppy, la penna usb ed il cdrom sono configurati in *automount*. In questo caso non è necessario utilizzare i comandi `mount/umount`, poichè le periferiche vengono montate ed agganciate automaticamente al filesystem alla prima richiesta di accesso. Analogamente, le unità vengono smontate automaticamente trascorsi alcuni secondi dall'ultimo accesso.

Una volta agganciato il contenuto della periferica al filesystem, sarà possibile lavorare sulle directory `/mnt/floppy`, `/mnt/usb` o `/mnt/cdrom` come su una qualunque altra directory. Vediamo un esempio di utilizzo del floppy disk.

```
archimede%mount /mnt/floppy
archimede%ls /mnt/floppy
mio_programma.c  mioprogramma.f90  lettera.txt
archimede%cp /mnt/floppy/lettera.txt /tmp
archimede%umount /mnt/floppy
```

Attenzione: per poter smontare il floppy disk, non dobbiamo essere posizionati sulla directory `/mnt/floppy`, altrimenti il comando fallirà e riceveremo un messaggio di errore del tipo

```
umount: /mnt/floppy: device busy
```

Su alcuni sistemi, solo l'amministratore del sistema (l'utente *root*) può utilizzare il comando `mount`. In questo caso sarà opportuno consultare gli help in linea relativi alla configurazione locale o chiedere direttamente all'amministratore per sapere come procedere.

Riepilogo dei comandi presentati

<code>mount</code>	– monta un filesystem
<code>umount</code>	– smonta un filesystem

Per maggiori dettagli sui comandi presentati utilizzare il manuale in linea.

EDITARE, COMPILARE ED ESEGUIRE UN PROGRAMMA

Scopo di questo paragrafo, non è quello di illustrare la sintassi dei linguaggi c o fortran, ma semplicemente di indicare la sequenza di operazioni da effettuare per scrivere, compilare ed eseguire un programma.

La creazione di un programma scritto in un **linguaggio ad alto livello**, come il C ed il fortran, prevede diverse fasi per arrivare ad ottenere un codice eseguibile.

- Creazione del **codice sorgente**, ovvero scrittura, tramite un editor, del/dei files contenenti le istruzioni che il programma dovrà eseguire.
- Compilazione del codice sorgente. In questa fase, il compilatore individua e segnala gli eventuali errori di sintassi. È allora necessario utilizzare l'editor per correggere gli errori, salvare il file corretto e quindi ricompilare il programma. Una volta eliminati tutti gli errori, il compilatore traduce il codice sorgente in **codice oggetto**.
- Linking (collegamento) di tutti i file oggetto e delle librerie che concorrono a formare il **programma eseguibile**.

A questo punto, sarà possibile eseguire il programma.

Esempio di un semplice programma C

1. Utilizzando un editor creare il file `gr2rad.c`, contenente le seguenti righe di codice sorgente:

```
/* Questo programma converte la misura di un angolo da gradi e radianti */
#include <stdio.h>
int main ()
{
    float Pi = 3.14159;
    float gradi, radianti;
    printf("Misura dell'angolo in gradi?  ");
    scanf("%f", &gradi);
    radianti = Pi*gradi/180.;
    printf("%f gradi = %f radianti.\n", gradi, radianti);
}
```

Può essere conveniente aprire la finestra di editing in background, utilizzando, ad esempio, il comando

```
emacs gr2rad.c&
```

In questo modo, la shell resterà attiva e sullo schermo saranno disponibili due finestre: la finestra di editing, per correggere gli errori e salvare il file modificato e la shell per compilare.

2. Per compilare utilizziamo *GNU project C and C++ Compiler*, richiamabile con il comando:

```
gcc gr2rad.c
```

Verranno segnalati eventuali errori, che dovranno essere corretti.

gcc eseguirà quindi le operazioni di compilazione e di linking e salverà il programma eseguibile nel file `a.out`.

È possibile assegnare un nome diverso da `a.out` al programma eseguibile, utilizzando l'opzione `-o` (*output*) nel comando di compilazione:

```
gcc gr2rad.c -o gr2rad
```

In questo caso, il programma eseguibile verrà salvato nel file `gr2rad`.

3. Per eseguire il programma, digitare semplicemente il nome del file:

```
gr2rad
```

Se la directory corrente non è compresa fra le directory contenute nella variabile d'ambiente PATH, sarà necessario specificare il nome del file completo del *path*:

```
./gr2rad
```

Esempio di un semplice programma fortran

1. Utilizzando un editor creare il file `rad2gr.f90`, contenente le seguenti righe di codice sorgente:

```
! Questo programma converte la misura di un angolo da radianti a gradi
Program rad2gr
  implicit none
  real, parameter :: Pi = 3.14159
  real :: gradi, radianti
  print *, "Misura dell'angolo in radianti? "
  read *, radianti
  gradi = 180.*radianti/Pi
  print *, radianti, " radianti = ",gradi," gradi"
end program rad2gr
```

Può essere conveniente aprire la finestra di editing in background, utilizzando, ad esempio, il comando

```
emacs rad2gr.f90&
```

In questo modo, la shell resterà attiva e sullo schermo saranno disponibili due finestre: la finestra di editing, per correggere gli errori e salvare il file modificato e la shell per compilare.

2. Per compilare, utilizziamo *Intel Fortran Compiler*, richiamabile con il comando:

```
f90 rad2gr.f90
```

Verranno segnalati eventuali errori, che dovranno essere corretti.

f90 eseguirà quindi le operazioni di compilazione e di linking e salverà il programma eseguibile nel file `a.out`.

È possibile assegnare un nome diverso da `a.out` al programma eseguibile, utilizzando l'opzione `-o` (*output*) nel comando di compilazione:

```
f90 rad2gr.f90 -o rad2gr
```

In questo caso, il programma eseguibile verrà salvato nel file `rad2gr`.

3. Per eseguire il programma, digitare semplicemente il nome del file:

```
rad2gr
```

Se la directory corrente non è compresa fra le directory contenute nella variabile d'ambiente PATH, sarà necessario specificare il nome del file completo del *path*:

```
./rad2gr
```



```

$$
\begin{array}{l}
\left\{
\begin{array}{l}
h = (b-a)/n \quad ;\;;\;;\;;\;;\;;\; a,b \in \mathbb{R}, \quad ;\;;\;;\; \mbox{ con } a < b \\
x_0 = a \quad \backslash\backslash \\
x_{i+1} = x_i + h \quad ;\;;\;;\;;\;;\;;\; \mbox{ per } i=0,\ldots,n-1
\end{array}
\right.
\end{array}
$$

%% APPENDICE
\appendix
\chapter{Esempio di Appendice}
Questa \ 'e la prima appendice
\chapter{Altro Esempio di Appendice}
Questa \ 'e la seconda appendice

%% INDICE
\tableofcontents
\end{document}

```

Può essere conveniente aprire la finestra di editing in background, utilizzando, ad esempio, il comando

```
emacs report.tex&
```

In questo modo, la shell resterà attiva e sullo schermo saranno disponibili due finestre: la finestra di editing, per modificare e salvare il sorgente e la shell per compilare, visualizzare, stampare.

2. Processare il file con il comando:

```
latex report.tex
```

Risolti gli eventuali errori, verrà generato il file `report.dvi`, che potrà essere visualizzato con il comando:

```
xdvi report.dvi&
```

L'utilizzo del carattere `&`, permette di aprire il visualizzatore `xdvi` in background, mantenendo attiva la shell.

3. Il file `report.dvi` può essere convertito in formato PostScript, con il comando:

```
dvips report.dvi -o report.ps
```

Sarà quindi possibile visualizzare il file ps con il comando:

```
gv report.ps &
```

ed inviarlo alla stampante con il comando:

```
lpr -Pnome-stampante report.ps
```

È possibile, chiedere a LaTeX di generare il file di output in formato pdf (*Portable Document Format*), piuttosto che in formato dvi. In questo caso, il documento sorgente dovrà essere processato con il comando:

```
pdflatex report.tex
```

Il file `report.pdf` potrà essere visualizzato con il comando

```
acroread report.pdf&
```

e potrà essere stampato, utilizzando le voci "File" → "Print" del menù di acroread.

SEMPLICI ESEMPI DI SHELL SCRIPT

Come già accennato nei paragrafi precedenti, la shell può essere utilizzata come linguaggio di programmazione, ed in quanto tale, dispone di variabili, di operatori e di strutture logiche. Uno **shell script** è un file di testo contenente una sequenza di comandi, che consentono di eseguire rapidamente operazioni complesse e/o ripetitive. Come per i programmi scritti in un linguaggio ad alto livello, il file di testo contenente un programma shell può essere scritto utilizzando un qualsiasi editor. Diversamente dai programmi in linguaggio ad alto livello però, non è necessario utilizzare un compilatore per convertire il programma sorgente in linguaggio macchina. La shell infatti agisce come **interprete**: legge, traduce ed esegue direttamente le istruzioni contenute nello script.

Per eseguire uno shell script, è sufficiente digitarne il nome al prompt dei comandi. Se la directory contenente lo script, non è fra quelle comprese nella variabile d'ambiente PATH, sarà necessario digitare il nome completo del percorso. Se ad esempio lo shell script si trova nella directory corrente, e se quest'ultima non è compresa nella variabile PATH, dovremo inviare il comando:

```
./nome-shell-script
```

Attenzione: affinché il programma shell possa essere eseguito dal sistema, è necessario che l'utente che invia il comando abbia "diritto di esecuzione" sul file contenente lo script. Come illustrato nei paragrafi precedenti, si possono utilizzare i comandi "ls -l" e "chmod" per vedere ed eventualmente modificare i diritti di accesso sul file.

Solo a titolo di esempio, si riportano due semplici programmini shell. È importante sapere che la sintassi dei comandi, dipende dalla shell utilizzata. I due script riportati di seguito utilizzano la *bash* shell.

Per maggiori dettagli sui comandi utilizzati negli esempi e per una descrizione completa della sintassi da utilizzare in uno shell script bash, consultare il manuale in linea ("man bash").

Esempio 1: shell script **esegui**

```
1. #!/bin/bash
2. #
3. # esegui: questo script manda in esecuzione tutti i
4. # programmi eseguibili contenuti nella directory,
5. # passata come parametro di input $1.
6. #
7. if [ $1 ] ; then
8.     echo "Esecuzione di tutti i file contenuti nella dir $1"
9.     echo ""
10.    for i in $1/* ; do
11.        if [ -f $i ] && [ -x $i ] ; then
12.            echo "`basename $i` in esecuzione ....."
13.            $i
14.            echo ""
15.        fi
16.    done
17. else
18.     echo "Indicare il nome della dir contenente i programmi da eseguire"
19. fi
20. exit
```

Questo semplice script permette di eseguire in sequenza tutti i programmi contenuti nella directory indicata in input. Alla riga 7, il programma controlla che sia stato specificato in input il nome della directory ed in caso contrario (righe 17,18) invia un messaggio di errore. Le righe 10-16 eseguono un ciclo su tutti i file contenuti nella directory specificata in input: lo script controlla se il file preso in esame è un file regolare (-f) e se è eseguibile (-x) (riga 11) ed in caso affermativo, invia a schermo un messaggio (riga 12) e lo esegue (riga 13).

Nell'istruzione alla riga 12 vengono utilizzati i comandi "echo" e "basename" e l'operatore *backtick*:

il comando `echo` scrive i suoi argomenti su standard output (a video);

il comando `basename` restituisce il nome di un file, privo del percorso;

l'operatore *backtick* esegue un comando (nel nostro caso, esegue il comando `basename $i`) e ne restituisce l'output.

L'effetto dell'istruzione a riga 12 sarà quindi quello di produrre a video il messaggio:

```
nome-file in esecuzione .....
```

Esempio 2: shell script `txt2ps`

```
1. #!/bin/bash
2. #
3. # txt2ps: questo script trasforma tutti i file con estensione
4. # .txt, presenti nella directory corrente, in file .ps
5. #
6. error_file=/tmp/txt2ps_errore
7. lista='ls *.txt 2> $error_file'
8. if [ -s $error_file ]; then
9.     echo "Nessun file da convertire in ps"
10. else
11.     for i in $lista ; do
12.         j='echo $i | sed -e 's:txt$:ps:' `
13.         a2ps $i -o $j
14.         echo "trasformato $i in $j"
15.     done
16. fi
17. rm $error_file
18. exit
```

Questo semplice script trasforma tutti i file con estensione `.txt` contenuti nella directory corrente, in file postscript. Alla riga 6, viene definita la variabile `error_file` contenente il nome del file utilizzato per memorizzare eventuali segnalazioni di errore. Alla riga 7, viene definita la variabile `lista` contenente l'output del comando `"ls *.txt"`, ovvero l'elenco dei nomi dei file `.txt` contenuti nella directory corrente. Se nella directory corrente non ci sono file `.txt`, la segnalazione viene indirizzata sul file `error_file`. Alla riga 8 il programma controlla se il file `error_file` ha dimensione maggiore di 0 (`-s`), ovvero se non è vuoto e quindi se contiene una segnalazione di errore. In caso positivo (riga 9), invia a schermo il messaggio "Nessun file da convertire in ps". Se invece il file `error_file` è vuoto, il programma esegue un ciclo (righe 11–15) su tutti i file compresi nella variabile `lista`: alla riga 12 viene definita la variabile `j` contenente il nuovo nome da assegnare al file ed alla riga 13 il file viene convertito in postscript. Prima della chiusura, lo script cancella il file `error_file` (riga 17).

Nell'istruzione alla riga 12, per definire il nuovo nome da assegnare al file preso in esame vengono utilizzati l'operatore *backtick*, l'operatore di concatenazione *pipe* (`|`), i comandi `echo` e `sed`:

l'operatore *backtick* esegue un comando e ne restituisce l'output;

l'operatore *pipe* concatena due comandi, passando l'output del primo comando in input al secondo comando;

il comando `echo` invia i suoi argomenti su standard output;

il comando `sed` è uno *Stream Editor* e viene utilizzato per manipolare la stringa contenente il nome del file.

Cerchiamo di capire come si combinano tutti questi ingredienti: utilizzando l'operatore *backtick*, la variabile `j` viene definita dall'output del comando

```
echo $i | sed -e 's:txt$:ps'
```

In questa istruzione, l'operatore di concatenazione *pipe* (`|`), passa in input allo *stream editor* "sed", l'output del comando `"echo $i"`, ovvero il nome del file preso in esame. `sed` applica sulla stringa che riceve in input il comando di editing `'s:txt$:ps:'` specificato tramite l'opzione `-e`, ovvero sostituisce (`s`) la stringa `txt` che chiude il nome del file, con la stringa `ps`. Il carattere `$` dopo la stringa `txt` (`txt$`) serve ad indicare che solo l'occorrenza della stringa `txt` che chiude il nome del file deve essere sostituita con la stringa `ps` e non altre eventuali occorrenze comprese nel nome del file. Se, ad esempio, si fosse scritto semplicemente `'s:txt:ps:'`, un file dal nome `txttag.txt` sarebbe stato rinominato in `ps tag.txt`, poichè sarebbe stata sostituita la prima occorrenza di `txt`. Se invece fosse stato scritto `'s:txt:ps:g'`, allora tutte (`g` sta per *global*) le occorrenze della stringa `txt` sarebbero state sostituite dalla stringa `ps` ed un file con nome `txttag.txt` sarebbe stato rinominato in `ps tag.ps`.

L'effetto dell'istruzione a riga 12 sarà quindi quello di prendere la variabile `i`, contenente il vecchio nome del file, sostituire l'estensione `txt` con l'estensione `ps` ed assegnare il nuovo nome alla variabile

`j`.

CENNI SULL'UTILIZZO DI SSH ED SFTP

- **ssh** (Secure SHell) è un programma che permette di connettersi e di eseguire il *login* su una macchina remota, per poter eseguire comandi su di essa.
- **sftp** (Secure File Transfer Program) è un programma che consente il trasferimento di files da e verso una macchina remota sulla rete.

Il protocollo di comunicazione SSH, a differenza di altri protocolli utilizzati per le comunicazioni su Internet (POP, IMAP, HTTP, SMTP, TELNET, etc.) che sono fundamentalmente insicuri, fornisce una infrastruttura per connessioni crittografate, che prevedono autenticazione fra host e host e fra utente ed host.

Informazioni dettagliate sul protocollo SSH e sul funzionamento dei comandi `ssh` ed `sftp`, si possono ottenere dal manuale in linea ("`man ssh`" e "`man sftp`") e dall'abbondante documentazione disponibile sulla rete (vedere ad esempio <http://www.cert.garr.it/documenti/ssh/>).

Riportiamo di seguito solo alcuni semplici esempi di utilizzo.

- Esempio di apertura shell remota via ssh:

```
archimede%ssh giulia@giove.mat.uniroma1.it
giulia@giove.mat.uniroma1.it's password: *****
Last login: Mon Mar 27 2006 07:55:34
Sun Microsystems Inc. SunOS 5.7 Generic October 1998
GIOVE (Enterprise 250 del Dipartimento di Matematica)
IP 151.100.50.7 giove.mat.uniroma1.it
.....
.....
giove%qui si possono inviare comandi (ls, cd, lpr, etc.)
.....
.....
giove%exit (per chiudere)
```

Il comando "`ssh giulia@giove.mat.uniroma1.it`" apre una connessione per l'utente specificato (nell'esempio `giulia`) verso l'host specificato (nell'esempio `giove.mat.uniroma1.it`). Viene richiesto all'utente di autenticarsi digitando la propria password. Quando l'identità dell'utente è stata accertata dall'host remoto (`giove.mat.uniroma1.it`), questo fornisce all'utente una shell dalla quale sarà possibile eseguire comandi. Tutte le comunicazioni fra la macchina locale e la shell remota sono crittografate. Se il nome dell'utente sulla macchina remota è uguale al nome dell'utente sulla macchina locale, esso può essere omesso, quindi ad esempio, se Giulia Bianchi è identificata dallo username `giulia` sia sul *local host* `archimede.mat.uniroma1.it`, sia sul *remote host* `giove.mat.uniroma1.it`, la connessione potrà essere attivata digitando semplicemente "`ssh giove.mat.uniroma1.it`".

- Esempio di apertura connessione via sftp per trasferimento files:

```
archimede%sftp giulia@giove.mat.uniroma1.it
Connecting to giove.mat.uniroma1.it...
giulia@giove.mat.uniroma1.it's password:
sftp> .....
.....
.....
sftp> quit (per chiudere)
```

Attivata la connessione, sarà possibile trasferire files da/verso l'host remoto. Seguono alcuni dei comandi utilizzabili in una sessione sftp:

- <code>ls [nome-dir]</code>	per avere l'elenco dei files sulla directory remota <code>nome-dir</code>
- <code>lls [nome-dir]</code>	per avere l'elenco dei files sulla directory locale <code>nome-dir</code>
- <code>cd nome-dir</code>	per spostarsi sulla directory remota <code>nome-dir</code>
- <code>lcd nome-dir</code>	per spostarsi sulla directory locale <code>nome-dir</code>
- <code>pwd</code>	restituisce il nome della directory corrente su <i>remote host</i>
- <code>lpwd</code>	restituisce il nome della directory corrente su <i>local host</i>

- get nome-file per copiare il file nome-file da *remote host* a *local host*
- put nome-file per copiare il file nome-file da *local host* a *remote host*
- rm nome-file per cancellare il file remoto nome-file

Ecco un esempio nel quale vengono utilizzati alcuni dei comandi sopra riportati:

```
archimede@sftp giulia@giove.mat.uniroma1.it
Connecting to giove.mat.uniroma1.it...
giulia@giove.mat.uniroma1.it's password:
sftp> cd /tmp        | si sposta sulla directory remota /tmp
sftp> lcd /tmp      | si sposta sulla directory locale /tmp
sftp> ls            | elenca i files sulla directory remota
drwxrwxrwx    7 sys        sys            928 Apr 18 10:31 ./
drwxr-xr-x   46 root      root           1536 Apr 10 9:02 ../
-rw-rw-rw-    1 root      root           2395 Apr 18 5:36 archivio.tar.gz
sftp> get archivio.tar.gz
Fetching /tmp/archivio.tar.gz to archivio.tar.gz
sftp> quit
```