

# Introduzione al sistema operativo Unix

In termini assai riduttivi possiamo dire che un sistema operativo è il software che, caricato in memoria ed eseguito al momento dell'accensione del calcolatore (bootstrap), permette ai programmi applicativi di girare e di sfruttare le risorse hardware della macchina. Il sistema operativo in un certo senso impone anche all'utente il modo di operare sulla macchina, stabilendo ciò che l'utente può fare e ciò che invece non può fare, stabilendo l'ordine in cui devono essere eseguite certe operazioni e così via.

Di questi aspetti ci si accorge poco utilizzando su un personal computer un sistema operativo limitato come MSDOS (MicroSoft Disk Operating System): in questo caso infatti poche cose è consentito fare, ma l'utente le può eseguire tutte, senza alcuna restrizione.

MSDOS d'altra parte è stato progettato per girare su un personal computer, su una macchina che quindi sarebbe stata utilizzata da un unico utente per volta e spesso da un solo utente in assoluto. MSDOS è anche concepito per poter eseguire un unico programma (task) per volta mentre usiamo un editor, non possiamo contemporaneamente utilizzare un programma di comunicazione o eseguire un programma di utilità.

Il sistema operativo Unix nasce intorno agli anni '60 contemporaneamente all'introduzione dei mini computer, nuove macchine più compatte ed agili dei grandi mainframe prodotti fino ad allora. Le prerogative di Unix che ne determinarono subito un grande successo sono sostanzialmente quelle di essere un sistema operativo compatto e modulare, facilmente adattabile alle risorse hardware ed alle esigenze operative dell'ambiente in cui viene installato. Con l'introduzione di Unix e dei mini computer si diffonde l'uso dei terminali alfanumerici che consentono una comunicazione più facile ed efficiente tra l'utente ed il sistema (fino ad allora si erano usate quasi esclusivamente le schede perforate). Si fa largo anche il concetto di sistema aperto e di elaborazione distribuita: Unix è predisposto per comunicare facilmente con altri sistemi e ad essere interconnesso con altre macchine per operare su risorse non più centralizzate su un unico grosso sistema, ma distribuite su più macchine di media potenza.

Entrando solo per un attimo in dettagli più tecnici, possiamo accennare al fatto che Unix è basato su un nucleo (il kernel) che gestisce la comunicazione di basso livello con la macchina, l'esecuzione dei programmi e l'uso e la condivisione o la protezione delle risorse del sistema. Questa è la parte del sistema operativo più strettamente legata all'hardware del computer. Il resto del sistema è costituito da una serie di moduli aggiuntivi finalizzati alla gestione dei vari sottosistemi che completano l'ambiente operativo (il riconoscimento degli utenti, la visualizzazione a video, l'input da tastiera, l'esecuzione dei comandi impostati dall'utente, la posta elettronica, la stampa, ecc.); questi moduli sono spesso "portabili", cioè è possibile adattarli quasi senza nessuna modifica sostanziale per poterli utilizzare su macchine completamente diverse, corredate del loro specifico kernel. Questo fatto rende Unix un sistema operativo aperto e facilmente trasportabile su hardware diversi. È proprio per questa grande lungimiranza dei suoi progettisti iniziali che oggi possiamo disporre di una grande varietà di Unix differenti, utilizzabili su piattaforme diverse. D'altra parte la possibilità da parte di chiunque di sviluppare moduli aggiuntivi per il sistema operativo ha fatto sì che oggi non esista uno Unix, ma ne esistano diverse versioni che hanno finito per essere anche incompatibili tra di loro.

Ci accorgeremo presto che Unix è in un certo senso un sistema operativo assai sobrio.

Tutto è realizzato tenendo in primo piano la sintesi: tutto ciò che è inutile o ridondante è bandito. I messaggi del sistema vengono visualizzati solo quando è strettamente necessario: spesso un programma viene eseguito senza che l'utente legga sullo schermo nessuna comunicazione da parte della macchina; se l'esecuzione termina senza che sia stato visualizzato alcun messaggio vorrà dire che l'esecuzione del programma è andata a buon fine, mentre nel caso contrario il sistema visualizzerà un breve messaggio di errore. I comandi della shell sono compatti, spesso costituiti di soli due caratteri, ma hanno un'infinità di opzioni perché l'utente si deve sentire libero di modificare a proprio piacimento la modalità operativa del software che utilizza. Questo fa di Unix un sistema operativo estremamente duttile e versatile.

Come vedremo meglio nella prossima sezione, una delle caratteristiche fondamentali di Unix è quella di essere un sistema operativo multiutente : più persone possono usare il sistema contemporaneamente o in tempi diversi. Questo fa sì che sia necessario imparare anche un certo "galateo" che deve contraddistinguere gli utenti di un sistema multiutente, in cui le risorse sono condivise tra più persone. Il fatto stesso che lo scambio di messaggi dalla macchina all'utente sia ridotta al minimo indispensabile è un segno del fatto che ogni operazione che possa in qualche modo appesantire il sistema, rallentando il lavoro di un altro utente, è evitata accuratamente, in modo da lasciare libere quante più risorse è possibile all'elaborazione vera e propria dei programmi.

## 1.2. Multiutenza e multitasking

Un sistema operativo multitasking permette di eseguire più programmi contemporaneamente: se ad esempio viene chiesto al sistema di eseguire contemporaneamente due processi, A e B, la CPU eseguirà per qualche istante il processo A, poi per qualche istante il processo B, poi tornerà ad eseguire il processo A e così via. È il sistema operativo a controllare che la CPU ripartisca equamente le sue prestazioni tra tutti i processi attivi; è sempre il sistema operativo a far sì che quando un processo va in crash, quando si blocca in seguito al verificarsi di un errore, i restanti processi e l'intero sistema non subiscano alcun danneggiamento e possano proseguire senza conseguenze i compiti loro assegnati.

Un sistema multiutente può essere utilizzato contemporaneamente da utenti diversi. Sotto Unix ad ogni utente del sistema viene assegnato uno user name che lo identifica univocamente: quando si inizia una sessione di lavoro si deve "entrare" nel sistema tramite una procedura di login durante la quale dovremo farci riconoscere dal sistema mediante l'introduzione del nostro username pubblico e della nostra password segreta.

Dopo essere entrati nel sistema potremo lanciare i nostri processi (le applicazioni) che verranno eseguiti in multitasking insieme ai processi lanciati dagli altri utenti collegati in quel momento sulla nostra stessa macchina.

## Console, terminali e terminali grafici

Può essere utile imparare a distinguere il sistema hardware mediante cui si accede alla macchina Unix su cui operiamo. Infatti, mentre quando lavoriamo con il nostro personal in ambiente DOS, siamo gli unici utenti di quel sistema, ed il computer che stiamo usando è quello che abbiamo fisicamente davanti a noi, la situazione può essere molto diversa nel caso in cui si stia utilizzando un sistema Unix.

La console è la coppia tastiera/video collegata direttamente alla macchina. In ambiente DOS la tastiera ed il monitor del nostro PC sono in un certo senso la console del PC stesso. Visto che però ad una stessa macchina Unix possono accedere contemporaneamente più utenti, deve essere possibile il collegamento di più "tastiere e video" allo stesso computer.

Mentre in passato ad un'unica macchina unix venivano collegati "direttamente" numerosi terminali alfanumerici il lavoro contemporaneo di più utenti è oggi reso possibile sfruttando le potenzialità di connessione remota native di UNIX e opportuni software che vengono eseguiti sulle macchine locali.

I terminali del passato erano costituiti da una tastiera, un video, ed una piccolissima unità di elaborazione locale, che si occupava esclusivamente di gestire la comunicazione tra il terminale stesso e l'elaboratore a cui era collegato. In sostanza il terminale si limitava a visualizzare sullo schermo i messaggi del sistema e ad inviare i comandi digitati dall'utente sulla tastiera.

Oggi come terminali vengono usati dei normali personal computer, dotati di un opportuno software di comunicazione.

L'elevata capacità di elaborazione dei moderni personal computer permette inoltre di usarli come terminali grafici che permettono di utilizzare un'interfaccia grafica (GUI) per eseguire le operazioni di input/output e quindi consentono anche di visualizzare un output di tipo grafico (immagini, disegni, grafici) rendendo il lavoro a distanza uguale nella sostanza a quello che si eseguirebbe dalla console.

# Diritti ed attributi

In un sistema multiuser e' fondamentale che gli utenti non si disturbino tra di loro ad esempio monopolizzando le risorse del server a cui si collegano o modificando gli archivi degli altri.

Unix garantisce la libert  di azione di tutti gli utenti facendo in modo che nessun altro utente non autorizzato possa in qualche modo violare la nostra privacy o distruggere o manomettere le nostre informazioni. In questo modo viene anche garantita una certa sicurezza dell'intero sistema: nessun utente "normale" potr  manometterlo compromettendone il corretto funzionamento; ma non solo: nessun utente "normale" potr  commettere errori talmente gravi nell'uso del sistema tanto da danneggiare altri utenti o il sistema stesso.

Cosa vuol dire pi  esattamente tutto questo? Abbiamo visto che ogni utente   individuato univocamente all'interno del sistema mediante uno username. Gli utenti del sistema sono distribuiti in pi  gruppi ; ogni utente fa parte di un gruppo. Ad esempio nel sistema del laboratorio, alcuni utenti fanno parte del gruppo denominato "informatica", che raccoglie coloro che seguono il corso di Informatica, altri fanno parte del gruppo "ottica", che raccoglie coloro che seguono il corso di Ottica, e cos  via.

Esiste poi un utente privilegiato, il cui username   root , che viene assegnato all'amministratore del sistema, il cosiddetto system manager . Questi   una figura assai importante nella gestione di un sistema Unix.   colui che pu  modificare la configurazione dell'intero sistema e che ha la possibilit  di verificare ogni cosa all'interno del sistema stesso.   il caso di dire che   anche l'unico utente che possa combinare dei guai seri su una macchina Unix!

Come si traduce tutto questo in pratica? Innanzi tutto ad ogni utente viene assegnata una propria home directory nel filesystem della macchina. Ad utenti differenti corrispondono home directory differenti; questa directory   di propriet  dell'utente a cui   assegnata e di solito ha lo stesso nome dell'utente. Anche ogni altro file o directory nel filesystem ha un proprietario che, mediante un apposito comando, ne stabilisce le modalit  d'uso (i diritti di accesso) per gli altri utenti del sistema.   ad esempio possibile fare in modo che un certo file possa modificarlo solo il proprietario, che possano leggerlo tutti gli utenti del gruppo del proprietario e che gli altri utenti non possano n  leggerlo, n  modificarlo o cancellarlo. Lo stesso   possibile fare con le directory, di cui si possono stabilire i diritti di lettura, scrittura ed accesso, e per i file binari contenenti le applicazioni (i programmi), di cui   possibile stabilire i diritti di lettura, scrittura ed esecuzione.

L'utente root non ha questi vincoli, ma pu  accedere in qualsiasi modo a qualunque file o directory presente nel filesystem.

## Uno sguardo al filesystem

Abbiamo usato pi  volte il termine filesystem , ma cosa significa esattamente? Con questa parola si indica l'insieme dei supporti di memorizzazione, fisici o virtuali, collegati al sistema (in gergo: montati sul sistema). Chi ha avuto modo di usare un PC con il DOS sa bene che ogni unit  a disco   identificata da una lettera dell'alfabeto: A   il dischetto magnetico, C   il primo disco rigido, D   il secondo disco rigido e cos  via. In ambiente Unix la situazione cambia radicalmente. Esiste una unit  (un disco) principale (root, radice, da non confondersi per  con il system manager) a cui vengono "agganciate" come sottodirectory tutte le altre unit . L'insieme di tutte le unit  di memorizzazione, accorpate in un'unica grande struttura ad albero, costituiscono il filesystem.

Generalmente, su quasi ogni sistema Unix, sono presenti alcune directory che rivestono una certa importanza all'interno del sistema e che hanno quasi sempre lo stesso nome. A titolo di esempio consideriamo la seguente struttura ad albero che rappresenta parte di un ipotetico filesystem (assai ridotto, per la verit ):

```
\ ---+-- bin
```

```

|
+-- dev
|
+-- etc
|
+-- home -- elisa
|   |
|   +-- marco
|   |
|   +-- root
+-- lib
|
+-- proc
|
+-- tmp
|
+-- usr
    |
    +-- bin
    |
    +-- include
    |
    +-- lib
    |
    +-- local +-- bin
    |         |
    |         +-- etc
    |         |
    |         +-- lib
    +-- man
    |
    +-- spool

```

Diamo una rapidissima scorsa al contenuto delle directory elencate:

### **/bin**

Contiene file binari (eseguibili) legati alla funzionalità del sistema operativo.

### **/dev**

È una directory molto importante che contiene i device drivers delle unità hardware installate sul nostro sistema. Sono alcune di quelle estensioni del kernel di cui parlavamo nelle pagine precedenti, che permettono al sistema di gestire le unità ad esso collegate. Ad esempio il file `/dev/ttyS0` gestisce l'input/output attraverso il primo terminale collegato al sistema, mentre `/dev/console` gestisce la console del sistema.

### **/etc**

Contiene una serie di file legati alla configurazione del sistema.

### **/home**

Contiene le home directory degli utenti del sistema.

### **/lib**

Contiene le libraries , dei file che contengono parte di codice eseguibile e che vengono condivisi da più applicazioni. Questo consente di ridurre la dimensione dei programmi, inserendo nelle librerie parti di codice comuni a più applicazioni.

### **/proc**

È una directory piuttosto particolare: i file che contiene non sono memorizzati su disco, ma direttamente nella memoria dell'elaboratore; contengono i riferimenti ai vari processi attivi nel sistema e le informazioni utili per potervi accedere.

### **/tmp**

È la directory temporanea di default. Spesso le applicazioni devono scrivere dei dati su un file temporaneo, che al termine dell'esecuzione verrà cancellato; in questi casi spesso usano la directory /tmp, che è sempre presente sui sistemi Unix.

### **/usr**

Contiene numerose sottodirectory legate al funzionamento del sistema.

#### **/usr/bin/X11**

Contiene tutto ciò che riguarda l'interfaccia grafica X Window .

#### **/usr/bin**

Contiene file eseguibili (file binari) di uso comune agli utenti.

#### **/usr/include**

Contiene i file include per i programmi in linguaggio C.

#### **/usr/lib**

librerie a collegamento dinamico (shared libraries).

#### **/usr/local**

Contiene files tipici della nostra macchina; tipicamente applicazioni installate successivamente al sistema operativo.

#### **/usr/man**

Contiene le pagine di manuale (il sistema di help on-line del sistema Unix).

#### **/usr/spool**

Contiene i files in attesa di essere elaborati da altri programmi, ad esempio contiene la coda di stampa ed i messaggi di posta elettronica giacenti e non ancora letti dai rispettivi destinatari.

## Comandi fondamentali

In questo capitolo diamo un rapido sguardo ai comandi principali della shell del sistema operativo Unix. Questi comandi ci consentono di muoverci nel filesystem attraverso le directory e gestire facilmente i nostri files e le funzioni di base del sistema.

La shell è il programma che di solito viene eseguito automaticamente dal sistema quando un utente effettua il login . È l'interprete dei comandi del sistema operativo, in altre parole è un programma che ci permette di usare i comandi che ci apprestiamo ad illustrare. Dobbiamo osservare, che i comandi sono molti di più di quelli riportati nel seguito, e che la shell, oltre a questi, è in grado di eseguire dei piccoli programmi, chiamati shell script , scritti nel linguaggio della shell (un po' come i files batch del DOS). La shell svolge più o meno la stessa funzione del programma COMMAND.COM del DOS, ma è molto più potente e versatile. Esistono diverse shell sotto Unix: tra queste citiamo la sh , la prima in ordine

cronologico, la bash , Bourne again shell, la ksh , Korn shell, la csh , C shell, molto amata dai programmatori in quanto ha una sintassi del tutto simile a quella del linguaggio C; le differenze risultano evidenti solo scrivendo degli script, perché per il resto le varie shell sono del tutto simili fra loro.

È importante osservare che in ambiente Unix le lettere maiuscole e minuscole sono completamente diverse: sotto DOS scrivendo DIR e dir ci si riferisce allo stesso comando, come pure è possibile riferirsi ad un certo file chiamandolo pippo.txt o PIPPO.TXT indifferentemente. Sotto Unix non vale la stessa cosa: i nomi dei comandi devono essere digitati rispettando le maiuscole e le minuscole e lo stesso vale per i nomi dei file e delle directory. Inoltre non esistono le limitazioni imposte dal DOS sui nomi dei file: questi possono essere anche molto lunghi e non è necessaria l'`estensione"; ad esempio potremmo chiamare il file che contiene il testo di questo capitolo guida.capitolo\_2.

## 2.1. Accesso al sistema e chiusura della sessione

Abbiamo già detto che per accedere al sistema si deve effettuare una procedura di `riconoscimento" detta login . In sistema ci chiede innanzi tutto di inserire il nostro username e poi la password . Per capire meglio vediamo un esempio:

```
Compaq Tru64 UNIX V5.1 (Rev. 732) (curie.pd.astro.it) (pts/10)
```

```
login: infoxx
```

```
Password:
```

```
Last login: Wed Dec 18 09:53:13 CET 2002 from curie.pd.astro.it
```

```
Compaq Tru64 UNIX V5.1 (Rev. 732); Thu Oct 18 11:04:56 CEST 2001
```

```
curie.pd.astro.it>_ La password non viene visualizzata a video quando la inseriamo, per impedire a chi ci sta guardando di scoprirla e poter quindi accedere al sistema a nome nostro. Il messaggio `Last login..." ci comunica la data e l'ora dell'ultima volta che siamo entrati nel sistema ed il nome del terminale da cui ci siamo collegati, in questo caso il terminale ttyS2.
```

La scritta "curie.pd.astro.it" che viene visualizzata davanti al cursore è il prompt , ossia l'indicazione che la shell è pronta ad accettare un nostro comando immesso da tastiera; quello riportato in queste pagine è solo un esempio: il prompt potrebbe variare da sistema a sistema o essere configurato diversamente dall'utente.

Quando abbiamo terminato di lavorare, prima di andarcene, dobbiamo scollegarci, effettuare cioè la procedura di logout che serve proprio per comunicare al sistema che abbiamo terminato di lavorare e che quindi può rimettersi in attesa del prossimo utente. Per scollegarci possiamo usare il comando exit, ma andrà bene anche il comando bye o logout o CTRL-D.

Di norma i personal computer usati per la connessione al sistema vengono lasciati sempre accesi, ma, salvo esplicita controindicazione, è possibile anche spegnerli dopo che si è effettuato il logout. È fondamentale invece non spegnere mai la macchina che ospita il sistema: solo root può effettuare la procedura di shutdown al termine della quale sarà possibile spegnere il sistema. Non si deve mai spegnere un sistema Unix senza prima aver completato la procedura di shutdown, pena la perdita irreparabile dei dati del filesystem.

## 2.2. Muoversi nel filesystem

Appena entrati nel sistema ci troviamo all'interno della nostra home directory . Per verificare il nome della directory possiamo usare il comando pwd (print work directory).

```
urie.pd.astro.it> pwd /home/informatica/infoxx_curie.pd.astro.it _ Di sicuro la prima cosa che ci viene in mente è quella di visualizzare il contenuto della nostra directory. Per far questo si deve usare il comando ls (list), equivalente al comando dir del DOS. Ad esempio potremmo ottenere il seguente output:
```

```
curie.pd.astro.it>ls _lettera      mail      pippo.c   progetto  tesi_libro
```

```
pippo      pippo.zip  src_curie.pd.astro.it>_ Vengono visualizzati nove nomi, ma non è chiaro se
```

siano nomi di files di dati, di sottodirectory o di programmi. Abbiamo accennato precedentemente alle numerose opzioni che generalmente consentono di modificare opportunamente l'esecuzione di un certo comando o programma. Ad esempio potremmo provare la seguente variante del comando ls:

```
curie.pd.astro.it>ls -F _lettera mail/ pippo.c progetto@ tesi/_libro/
```

pippo\* pippo.zip src/\_curie.pd.astro.it> Accanto ad alcuni nomi compare un simbolo che non fa parte del nome, ma che serve ad indicare di che tipo di file si tratta: lo slash "/" indica che è una directory, l'asterisco "\*" indica che si tratta di un file eseguibile, mentre la chiocciola "@" sta ad indicare che quel file (o directory) non è fisicamente presente nella nostra directory, ma è un link, un rimando, ad un file (o ad una directory) che si trova da un'altra parte nel filesystem.

Un'altra versione del comando ls si ottiene aggiungendo l'opzione "-l"; visto che è possibile specificare più opzioni su uno stesso comando, vediamo quale potrebbe essere l'output del comando "ls -lF" (che equivale anche ad "ls -l -F", visto che in molti casi le opzioni possono essere "raggruppate"):

```
curie.pd.astro.it>ls -lF _rw-r-r- 1 infoxx users 937 Apr 23 12:43 lettera _drwxr-xr-
x 2 infoxx users 1024 Apr 10 16:04 libro/_drwx--- 2 infoxx users 1024 Feb 01 09:32 bin/_-rwxr-
x-- 1 infoxx users 37513 Mar 10 11:55 pippo*_-rw-r-r-
1 infoxx users 18722 Mar 10 11:30 pippo.c _-rw-r-r- 1 infoxx users 23946 Mar 10 12:03 pippo.zip
_lrwrxrwx- 1 infoxx users 8 Apr 04 18:16 progetto -> /home/elisa/proj_drwxrwx--
2 infoxx users 1024 Mar 10 08:47 src/_drwxr-r- 2 infoxx users 1024 Feb 12 15:29 tesi/
```

\_curie.pd.astro.it> Illustriamo brevemente il significato delle numerose informazioni presentate dal comando "ls -l". Il primo carattere di ogni riga può essere "d" per indicare che si tratta di una directory, "l" per indicare un link o "-" per indicare che si tratta di un normale file. I successivi nove caratteri rappresentano in forma sintetica le proprietà dei files; devono essere letti raggruppandoli a tre a tre. I primi tre caratteri indicano i diritti del proprietario di tale file sul file stesso; i successivi tre caratteri indicano i diritti degli altri utenti facenti parte dello stesso gruppo del proprietario del file, mentre gli ultimi tre caratteri rappresentano i diritti di tutti gli altri utenti del sistema. Una "x" sta ad indicare il diritto di esecuzione di tale file (mentre è chiaro cosa significa eseguire un programma, è opportuno chiarire che "eseguire" una directory significa poterci entrare dentro). Il carattere "r" sta ad indicare il diritto di leggere tale file (read), mentre "w" indica il diritto di poterci scrivere sopra (write), modificandolo o cancellandolo.

Di seguito viene riportato lo username del proprietario del file (es.: infoxx) ed il nome del gruppo di appartenenza (es.: users). Viene poi visualizzata la dimensione del file, la data e l'ora in cui è stato creato ed infine il nome.

Nell'esempio il file pippo.zip può essere letto e modificato dal proprietario (infoxx), mentre può essere soltanto letto dagli altri utenti del sistema; è stato creato il 10 Marzo dell'anno in corso alle 12:03. Il file progetto è un link alla directory /home/informatica/infoxx/proj e può essere utilizzato in lettura, scrittura ed esecuzione solo dal proprietario e dagli utenti del suo stesso gruppo, ma non dagli altri utenti del sistema che possono solo accedervi in lettura.

Delle numerosissime opzioni del comando ls ci limitiamo ad illustrarne solo un'altra, che ci permette di introdurre qualche altro concetto.

Cominciamo con il solito esempio, osservando l'output prodotto dal comando "ls -a":

```
curie.pd.astro.it>ls -aF ./ .bash_history* .login* bin/ test/
./ .cshrc* .profile* letizia/ test23
.TTauthority .dt/ .sm* pippo
.Xauthority* .dtprofile* .sysman/ sara/
curie.pd.astro.it>_
```

Abbiamo alcuni strani file preceduti da un punto: per convenzione sotto Unix (ma anche sotto DOS) il nome "." (punto) sta ad indicare la directory corrente, mentre con ".." si indica la directory "madre" nell'albero del filesystem.

Gli altri file preceduti dal punto sono dei normali file (o directory, ma non nel nostro caso) che però non vengono visualizzati dal comando `ls` proprio perché hanno un nome che comincia con il punto. In questo modo possiamo evitare di visualizzare alcuni file che devono essere presenti nella nostra directory (ad esempio i files di configurazione), ma che non utilizzeremo direttamente quasi mai e dei quali dimenticheremo presto la presenza. Visualizzarli ogni volta renderebbe l'output del comando `ls` eccessivamente ridondante e quindi tali file vengono, in un certo senso, "nascosti".

Per spostarsi attraverso le directory del filesystem si usa il comando `cd` (change directory). Ad esempio per "scendere" nella directory `src` si dovrà digitare il comando `cd src`, mentre per risalire nella directory madre (la nostra home directory) dovremo digitare il comando `cd ..`. A proposito della home directory è utile osservare che ci si può riferire ad essa mediante l'uso del simbolo `~` (tilde), mentre per riferirsi alla home directory dell'utente "infoxx" si potrà usare la stringa `~infoxx` (ad esempio si potrà digitare `cd ~infoxx` per spostarci nella sua directory: questo modo di impostare il comando è sicuramente più sintetico del canonico `cd /home/informatica/infoxx`). Visto che si presenta frequentemente la necessità di rientrare nella propria home directory, il comando `cd` dato senza nessun parametro assolve efficacemente tale scopo.

## 2.3. Gestione di file e directory

Abbiamo visto nella sezione precedente che ad ogni file sono associati degli attributi per limitarne l'accesso ad altri utenti del sistema. Per modificare gli attributi di un file si deve utilizzare il comando `chmod`, che ha la seguente sintassi:

```
chmod attributi file
```

Il parametro attributi può avere varie forme, ma forse la più semplice di tutte è quella numerica, secondo la seguente regoletta. Si associa valore 100 al diritto di eseguire il file per il proprietario, 10 per gli utenti del gruppo e 1 per tutti gli altri; si associa 200 al diritto di accesso in scrittura al file per il proprietario, 20 per gli utenti del gruppo e 2 per tutti gli altri; infine si associa 400 al diritto di accesso in lettura al file per il proprietario, 40 per gli utenti del gruppo e 4 per gli altri. Sommando questi numeri si ottiene l'attributo da assegnare a quel file. Così, ad esempio, il numero 700 equivale a stabilire che l'unico a poter leggere, scrivere ed eseguire il file è il proprietario ( $100+200+400=700$ ), mentre con 644 si indica il fatto che il file può essere letto e modificato dal proprietario, ma può essere soltanto letto dagli altri utenti; in quest'ultimo caso, volendo associare tale attributo al file `pippo.c` daremo il comando

```
$ chmod 644 pippo.c
```

Esiste anche un modo più semplice da ricordare per ottenere lo stesso risultato che usa le lettere al posto dei numeri.

Si useranno cioè "u" per user "g" per gruppo e "o" per gli altri nonché "w" per write, "r" per read e "x" per i diritti di esecuzione e i simboli "+" e "-" per dare o togliere i diritti e "=" per assegnarli.

Il comando sopra visto diventerà perciò

```
curie.pd.astro.it>chmod u=rwx pippo.c
```

```
curie.pd.astro.it>chmod g=r pippo.c
```

```
curie.pd.astro.it>chmod o=r pippo.c
```

Per creare una directory useremo il comando `mkdir` (make directory); ad esempio con il comando

```
curie.pd.astro.it>mkdir esempio
```

possiamo creare una nuova directory nella directory di lavoro.

Per rimuovere una directory vuota (che non contiene alcun file) si deve usare invece il comando `rmdir` (remove directory), ad esempio:

```
curie.pd.astro.it>rmdir esempio
```

che viene prontamente eseguito dal sistema senza chiederci ulteriore conferma.

Per copiare un file da una directory ad un'altra si deve usare il comando `cp` (copy); ad esempio volendo copiare il file `pippo.c` nella directory `src` dovremo dare il comando:

```

curie.pd.astro.it> cp pippo.c src
curie.pd.astro.it>_ Sintassi analoga ha anche il comando mv (move) che serve a spostare i file da una
directory all'altra, ad esempio per spostare tutti i files il cui nome termina con ``.c" dalla directory src
alla directory tesi potremo dare il seguente comando:
curie.pd.astro.it>mv src/*.c tesi
curie.pd.astro.it>_ Il comando mv è anche utile per cambiare il nome ad un file, ad esempio se
volessimo modificare il nome del file pippo.c e chiamarlo pluto.c, potremmo usare il comando:
curie.pd.astro.it>mv pippo.c pluto.c
curie.pd.astro.it>_ Infine per cancellare un file si usa il comando rm (remove); si deve porre
particolare attenzione nell'uso di questo comando, perché se non si specifica l'opzione ``-i"
(interactive), il sistema eseguirà la cancellazione dei files senza chiedere ulteriori conferme. Ad
esempio il seguente comando cancella ogni file contenuto nella directory tesi:
curie.pd.astro.it>rm tesi/*
curie.pd.astro.it>_ Usando l'opzione ``-i" il sistema ci chiede conferma dell'operazione:
curie.pd.astro.it>rm -i pippo.c
rm: remove pippo.c? y
curie.pd.astro.it>_

```

## 2.4. Visualizzazione e stampa di file

È molto utile poter visualizzare e stampare il contenuto di un file di testo. A questo scopo sono stati implementati diversi comandi Unix: ne vedremo solo alcuni, quelli che riteniamo veramente indispensabili.

Il primo, ed il più elementare, è il comando cat, che serve per concatenare due file, ma che può essere usato anche per visualizzare il contenuto di un file di testo. Ad esempio il seguente comando

```

Curie.pd.astro.it>cat pippo.c
#include <stdio.h>
main()
{
printf("Hello\n");
return;
}

```

curie.pd.astro.it>\_ visualizza sul terminale il contenuto del file pippo.c (un banale programmino in C). Se il file è molto lungo e non può essere visualizzato all'interno di una schermata del terminale, l'output scorrerà sullo schermo fino all'ultima riga del file, senza che possiamo leggerne il contenuto. Per ovviare a questo problema si usa il comando more , che, come il cat, visualizza il contenuto del file sullo schermo, ma alla fine di ogni schermata rimane in attesa che l'utente batta un tasto, prima di visualizzare la schermata successiva. Il comando more si comporta diversamente a seconda del tasto che viene battuto:

### **spazio**

l'output scorre in avanti di una schermata;

### **invio**

l'output scorre in avanti di una riga;

### **b**

l'output scorre indietro (back) di una schermata;

### **q**

il comando more viene interrotto (quit);

more può anche essere utilizzato per filtrare verso il terminale l'output proveniente da un altro

comando, attraverso la possibilità di effettuare il piping dei comandi messa a disposizione dalla shell. Ad esempio se l'output del comando `ls -l` dovesse essere troppo lungo potremmo utilizzare il comando `ls -l | more` che prima di inviare l'output al terminale lo filtra attraverso il more, visualizzandolo una schermata alla volta.

## 2.5. Le pagine di manuale

La fonte di informazione principale sui comandi Unix e sui programmi installati sul proprio sistema è costituita dalle cosiddette man pages , o pagine di manuale, che formano, in unione ad un comodo programma di consultazione, una vera e propria biblioteca di manuali on-line, sempre pronti ad essere consultati. L'insieme delle man pages è suddiviso per argomento in nove sezioni:

1. User command, dove sono riportati tutti i comandi utili per l'utente;
2. System calls, vengono descritte tutte le funzioni standard del linguaggio C per effettuare delle chiamate alle funzioni del sistema (utile soprattutto ai programmatori);
3. Subroutines, vengono descritte le subroutines e le funzioni del sistema di sviluppo (linguaggio C);
4. Devices, dove sono riportate le descrizioni dettagliate dei devices installati sul sistema;
5. File Formats, vengono riportati i formati dei principali file di configurazione del sistema;
8. System administration, descrizione dei comandi per l'amministratore del sistema (root);

Ogni paragrafo del manuale è contenuta in un file diverso ed i file appartenenti ad una stessa sezione sono contenuti nella stessa directory; ad esempio i files con le descrizioni dei comandi della prima sezione possono trovarsi nella directory `/usr/man/man1`.

Se è presente nel sistema la pagina di manuale relativa ad un certo programma, potremo visualizzarla mediante il comando `man` seguito dal nome del programma; ad esempio con `man mkdir` si visualizza la pagina di manuale relativa al comando `mkdir`:

```
curie.pd.astro.it> man mkdir
```

```
mkdir(1)
```

```
mkdir(1)
```

```
NAME
```

```
mkdir - Makes a directory
```

```
SYNOPSIS
```

```
mkdir [-m mode] [-p] directory...
```

## STANDARDS

Interfaces documented on this reference page conform to industry standards as follows:

mkdir: XCU5.0

Refer to the standards(5) reference page for more information about industry standards and associated tags.

## OPTIONS \_ -m mode

Sets the file permissions to mode, a symbolic mode string as defined for chmod, after creating the specified directory. The mode argument can be either an absolute mode string or a symbolic mode string as defined for chmod. See the chmod(1) reference page.

In symbolic mode strings, the operation characters + and - are interpreted relative to an assumed initial mode of a=rwx, A + adds permissions to the default mode, whereas a - deletes permissions from the default mode.

-p Creates intermediate directories as necessary; otherwise, the full path name prefix to directory must already exist. The user must have mkdir write permission in the parent directory.

Each component of directory that does not name an existing directory is created with mode 777, modified by the current file mode creation mask (umask). The equivalent of chmod u+w is performed on each component to ensure that mkdir can create lower directories regardless of the setting of umask. Each component of directory that names an existing directory is ignored without error. If an intermediate path name component exists, but permissions are set to prevent writing or searching, mkdir fails and returns an error message. The mode argument does not

...

...

## SEE ALSO

Commands: `chmod(1)`, `rm(1)`, `rmdir(1)`, Bourne shell `sh(1b)`, POSIX shell `sh(1p)`, `umask(1)`

Functions: `mkdir(2)`

Standards: `standards(5)`

`curie.pd.astro.it>_`

La visualizzazione mediante il comando `man` è filtrata automaticamente attraverso il more e quindi potremo scorrere facilmente il testo della pagina, anche se questa dovesse risultare molto lunga.

Le `man` pages hanno tutte un formato piuttosto simile: viene sempre riportato il nome del comando illustrato seguito da una brevissima descrizione del comando stesso; viene poi descritto in modo sintetico la sintassi del comando (`synopsis`) ed infine vengono elencate e descritte dettagliatamente le opzioni che è possibile specificare insieme al comando. Di solito alla fine della pagina è riportato l'elenco dei files di configurazione del programma stesso (`files`), l'elenco di eventuali comandi correlati (`see also`), eventuali problemi noti, riscontrati in situazioni particolari, nell'uso del programma (`bugs`) ed il nome dell'autore del programma (`authors`).

## Gestione dei processi

In ambiente Unix si parla di processo per indicare un programma in esecuzione. Sappiamo che Unix è un sistema operativo multitasking, ma fino ad ora abbiamo sfruttato questa caratteristica solo grazie alla multiutenza, che ci permetteva di "toccare con mano" il fatto che la macchina, avendo più utenti collegati contemporaneamente, stava effettivamente elaborando più di un programma.

Per sfruttare pienamente e con comodità il multitasking si deve disporre di un terminale grafico che ci permetta di aprire sullo schermo più finestre in cui lanciare i diversi processi; è possibile fare altrettanto su un normale terminale alfanumerico, ma anche in questo caso si deve disporre di un programma (ad esempio `screen`) che ci consenta di simulare un ambiente con più finestre. Vedremo in maggiore dettaglio l'ambiente X Window in seguito, per ora ci limiteremo a dire che è possibile lanciare delle applicazioni che lavorano autonomamente ed indipendentemente dalle altre in una regione dello schermo (finestra) riservata ad ognuna di esse.

Per lanciare un'applicazione, come al solito è necessario dare un comando al prompt della shell, ma così facendo, come abbiamo visto fino ad ora, perdiamo temporaneamente l'uso della shell, fino a quando non saremo usciti dal programma in esecuzione. Esiste un modo per sganciare il processo "figlio" (il programma da eseguire) dal processo "padre" (la shell da cui si lancia il programma) rendendo i due processi indipendenti ed autonomi. Se aggiungiamo una `&` (e commerciale) alla fine della riga che invoca un certo comando, tale comando sarà eseguito in una "sessione" separata e quindi potremo utilizzare contemporaneamente la shell (da cui possiamo per esempio lanciare altri processi) ed il programma che abbiamo lanciato.

Vediamo un esempio molto semplice che possiamo provare anche su un normale terminale alfanumerico. Il comando `yes` visualizza una serie di "y" sullo schermo fino a quando non viene interrotto dall'utente. Per interrompere l'esecuzione del programma dobbiamo battere **CTRL-c** (`break`). Se invece di interromperlo volessimo solo sospenderne temporaneamente l'esecuzione, invece di `break` dovremmo battere **CTRL-z** (`stop`):

`curie.pd.astro.it>yes _yes _yes _yes _...`

## CTRL-z

```
[1]+ Suspended      yes _curio.pd.astro.it>_
```

Il sistema ci informa che il processo numero 1, che è stato lanciato con il comando `yes`, è stato momentaneamente interrotto. Il comando `jobs` ci permette di visualizzare la lista dei processi lanciati da quella shell; nel caso dell'esempio precedente avremmo il seguente output:

```
curie.pd.astro.it>jobs [1]+ Suspended      yes _curie.pd.astro.it>_
```

In questo momento abbiamo due processi attivi: la shell ed il programma `yes` che è momentaneamente interrotto. In particolare diremo che il processo attivo, la shell, è in foreground, mentre l'altro processo è in background; in uno stesso momento può esserci un solo programma in foreground, ma anche molti programmi in background. Per riportare in foreground il programma `yes` (e mandare quindi in background la shell) si deve usare il comando `fg` (foreground) seguito dal numero del processo:

```
curie.pd.astro.it>fg 1 _yes _yes _yes _...
```

È possibile indirizzare l'output di una applicazione verso una unità diversa dal device di output standard (il video del terminale) mediante l'uso del carattere `>` (maggiore). Proviamo a indirizzare l'output del programma `yes` verso l'unità `null` che semplicemente prende i dati e li scarta, dando il comando `yes > /dev/null`. Il programma è in esecuzione, ma sullo schermo non appare nulla, neanche il prompt della shell per poter lanciare altri programmi nel frattempo. Interrompiamo l'esecuzione di `yes` battendo **CTRL-c** e proviamo a riavviarlo con il comando `yes > /dev/null &`:

```
curie.pd.astro.it> yes > /dev/null & [1] 143 _curie.pd.astro.it>jobs
_[1] 143 Running      yes >/dev/null & _curie.pd.astro.it>ps
_ PID TTY STAT TIME COMMAND _ 67 1 S 1:32 bash _ 143 1 R 0:00 yes
_ 152 1 R 0:00 ps _curie.pd.astro.it>_
```

Con l'aggiunta del simbolo `&` alla fine della linea di comando, abbiamo lanciato l'applicazione in background, mantenendo l'uso della shell per impostare altri comandi. Con il messaggio `[1] 143` il sistema ci comunica che l'applicazione `yes` è il primo processo lanciato da questa shell e, nella tabella di tutti i processi del sistema, gli è stato assegnato il numero 143 (attenzione: questo non vuol dire che ci siano 143 processi attivi). Con il comando `jobs` verificiamo gli stessi dati ed in più il sistema ci comunica che l'applicazione è attualmente in esecuzione (running). Il comando `ps` ci fornisce delle informazioni su tutti i nostri processi attivi, non solo quelli lanciati attraverso una certa shell; l'esempio mostra che nel nostro caso sono attivi tre processi, tutti sul terminale `tty1`: `bash`, la shell è attiva da un'ora e 32 minuti ed è momentaneamente sospesa (`S`) perchè sta eseguendo il comando `ps`, che è stato appena attivato ed è in esecuzione (`R`), come pure il programma `yes`, attivo anche questo solo da qualche istante.

Con il comando `kill` è possibile interrompere forzatamente l'esecuzione di un processo in background (come la pressione dei tasti **CTRL-c** per i processi in foreground). Insieme al `kill` si deve specificare il PID (Process ID) dell'applicazione che vogliamo terminare o il suo job number, preceduto però dal simbolo di percentuale `%`. Ad esempio per interrompere il programma `yes` dell'esempio precedente i due comandi `kill 143` e `kill %1` sono equivalenti.

## Alcuni comandi utili

I comandi disponibili sul mio sistema Unix sono alcune centinaia: oltre ad essere impossibile riportarli tutti, sarebbe anche abbastanza inutile, visto che sono moltissimi i comandi che non ho mai usato e che probabilmente non userò mai. Ci limitiamo in questa sezione a descrivere per sommi capi alcuni comandi non indispensabili, ma che possono essere di una certa utilità. Per quanto riguarda gli altri, quelli che non hanno trovato spazio in queste pagine, il consiglio è il seguente: quando sentirete la necessità di un comando per svolgere un particolare compito, cercatelo, magari "sfogliando" le pagine di manuale, perché quasi sicuramente già esiste!

È forse il caso di chiarire che quando diamo un comando, la shell va a cercare il file eseguibile con quel nome in alcune directory del filesystem; queste directory sono quelle specificate nel path : è una variabile di ambiente della shell in cui è riportato l'elenco delle directory del nostro sistema che contengono file eseguibili. Per visualizzare l'elenco delle variabili di sistema si deve usare il comando `set` . Per sapere in quale directory è collocato fisicamente il file eseguibile di un certo programma si deve usare il comando `which programma` .

Abbiamo spesso usato, negli esempi della pagine precedenti, gli operatori di redirectione dell'input/output. Più in dettaglio possiamo dire che il simbolo `>>` serve per inviare l'output standard (quello che normalmente finisce sul video del terminale) su un file o su un altro device; il simbolo `<<` serve invece per leggere l'input standard (quello che altrimenti sarebbe inserito manualmente dall'utente mediante la tastiera) da un file o da un altro device. Infine il simbolo `|` (pipe) serve ad indirizzare l'output di un programma verso l'input di un secondo programma. Il seguente esempio utilizza tutti gli operatori di redirectione dell'I/O:

```
$ cat < lista | sort > lista.ordinata
```

\$ \_ Il programma `cat` (che, come abbiamo visto, serve a visualizzare il contenuto di un file) riceve l'input dal file `lista`; l'output di `cat` viene inviato mediante il piping al programma `sort` (che serve ad ordinare i dati contenuti in una lista) che li invia in output al file `lista.ordinata`. Al termine dell'esecuzione di questo comando il file `lista.ordinata` conterrà gli stessi dati di `lista`, ma ordinati alfabeticamente. Riguardo al programma `sort` è forse opportuno aggiungere che i comandi `sort < lista > lista.ordinata` e `sort lista -o lista.ordinata` avrebbero svolto efficacemente lo stesso compito del comando riportato nell'esempio precedente al solo scopo di illustrare l'uso di tutti i simboli di redirectione dell'I/O.

Con il comando `date` il sistema ci fornisce la data e l'ora corrente, mentre con `cal` viene visualizzato un sintetico calendario del mese corrente; per avere il calendario completo di un intero anno basterà specificare l'anno desiderato di seguito al comando `cal`, ad esempio `cal 1492`.

Il comando `tar` (tape archive) è molto usato in ambiente Unix e di solito serve per includere (senza comprimere) in un file unico più files, magari sparsi in directory differenti. In particolare è molto usato per effettuare il backup dei files del sistema. I files archiviati con `tar` hanno estensione `tar`.

Per creare un archivio `tar` si usa il comando `tar cfv nomefile.tar`, per visualizzare il contenuto di un file archiviato con `tar` si deve usare il comando `tar tfv nomefile.tar`, mentre per estrarre effettivamente i files si usa il comando `tar xfv nomefile.tar`.

Per archiviare su nastro (lo scopo principale del comando `tar`) si possono usare come nome di file i seguenti file speciali che unix usa per accedere alle periferiche a nastro:

```
/dev/tape/tape0
```

```
/dev/ntape/tape0
```

(`tape0` e' il nome in TruUnix64 v5.1 del primo dispositivo a nastro collegato alla macchina che nel caso di curie e' una unita DAT).

La differenza tra i due file speciali e' che con il primo ogni comando che accede al dispositivo riavvolge la cassetta DAT mentre il secondo non lo fa e, partendo dal punto in cui si era fermato con il comando precedente permette di scrivere piu' archivi `tar` uno di seguito all'altro sulla stessa cassetta.

Per visualizzare lo spazio su disco occupato dalla directory corrente e da tutte le sue sottodirectory si deve usare il comando `du` (disk usage). Per vedere invece quanta parte dei dischi montati nel filesystem è occupata si deve usare il comando `df` (disk free). Infine per visualizzare le risorse di memoria ancora libere si usa il comando `free` .

Come il comando `ps` visualizza i processi attivi sul sistema, il comando `lpq` (line printer queue) visualizza i `job pendenti` nella coda di stampa ; se si desidera interrompere una stampa, bisogna eliminare il relativo job dalla coda di stampa con il comando `lprm` seguito dal job-id indicato dal comando `lpq`.

# L'editor vi

vi (si pronuncia "vuai") è sicuramente l'editor più diffuso sotto Unix ed anche uno degli editor più potenti in assoluto. Con vi si può fare praticamente tutto, l'unico difetto è che è un po' ostico da utilizzare e completamente non standard rispetto agli altri editor (o meglio, vista la sua importanza, vi costituisce uno standard a sé).

Il programma è caratterizzato da due modalità operative completamente differenti: la modalità "comando" e la modalità "inserimento". Nella prima è possibile impostare i comandi generali per la gestione dell'intero file (in ambiente vi il file caricato si chiama buffer), mentre nella seconda è possibile scrivere e modificare il testo.

Appena caricato il vi ci troviamo in modalità comando, in cui possiamo muoverci all'interno del buffer utilizzando i tasti di spostamento del cursore o i tasti **h** per spostare il cursore verso sinistra, **j** per spostarlo in basso, **k** per muoverlo verso l'alto e **l** per muoverlo verso destra.

Dalla modalità comando il tasto **i** ci permette di passare alla modalità inserimento. Per tornare in modalità comando si deve battere il tasto **Esc** (che su alcuni sistemi è sostituito dai tasti **CTRL-]**). Il programma non visualizza nessuna indicazione sulla modalità attiva, quindi si deve porre una certa attenzione quando si commuta da una modalità all'altra.

Entrati in modalità inserimento possiamo digitare il testo da inserire, proprio come su qualsiasi altro programma di videoscrittura. Ogni modifica, cancellazione o spostamento, avviene però sul singolo carattere: per operare su interi blocchi di testo ci si deve portare in modalità comando, dove si può, ad esempio, cancellare una intera linea del buffer battendo due volte il tasto **d**, oppure un singolo carattere sotto al cursore, battendo **x**. Per tornare in modalità inserimento, invece del tasto **i** che ci permette di inserire il testo a partire dalla posizione attuale del cursore (insert), possiamo battere **A** (add) che ci permette di aggiungere il testo alla fine della linea su cui si trova il cursore.

La modalità comando mette a disposizione una vasta gamma di istruzioni, costituite da una o due lettere, con cui è possibile operare anche sull'intero buffer. Per utilizzarle si deve battere **:** (due punti) e poi digitare il comando. Vediamone alcuni:

co

(copy) Per copiare il testo che va dalla riga x alla riga y, dopo la riga z, si deve digitare il comando ``:x,y co z";

mo

(move) Per spostare dopo la riga z il testo che va dalla riga x alla riga y, si deve digitare il comando ``:x,y mo z";

d

(delete) Per cancellare il testo dalla riga x alla riga y si digiti il comando ``:x,yd";

r

(read) Per inserire il contenuto del file nome dopo la riga n, si digiti il comando ``:n r nome";

w

(write) Per salvare sul file nome il testo dalla riga x alla riga y, si deve digitare il comando ``:x,y w nome"; per salvare l'intero file si digiti semplicemente ``:w", oppure ``:w nome";

q

(quit) Per uscire dal programma si digiti ``:q"; per uscire senza salvare su file le modifiche apportate al testo si digiti ``:q!", mentre per uscire e salvare il testo su file si deve dare il comando ``:wq".

Per spostarsi direttamente su una certa linea all'interno del testo si può digitare ``:n", dove n è il numero di riga; per andare alla fine del file si può battere più semplicemente **G**.

Come in ogni editor che si rispetti, anche in vi è possibile cercare una certa stringa di caratteri all'interno del buffer: dalla modalità comando basta battere / (slash) seguito dalla stringa da cercare; una volta trovata la prima occorrenza della stringa, per cercarne un'altra occorrenza basterà battere **n** o **p**, senza specificare nessuna stringa, che prosegue la ricerca in avanti (next) o indietro (previous).

Per utilizzare i comandi di vi è estremamente utile poter identificare i numeri di linea del testo inserito; per questo è opportuno, appena avviato il programma, dare il comando ``:set number" che attiva la numerazione automatica delle linee. Per far sì che vi abiliti automaticamente questa funzione, si deve inserire il comando ``set number" nel file ``.exrc" che contiene i comandi che devono essere eseguiti da vi all'inizio della sessione di lavoro; come molti altri file di configurazione, anche questo comincia con un punto e deve essere inserito nella nostra home directory .

vi ha un gran numero di comandi, alcuni dei quali (pochissimi, a dire il vero) sono stati descritti in queste pagine; per avere un riferimento più dettagliato sulle varie possibilità offerte dal programma è opportuno riferirsi alla documentazione del proprio sistema.

## L'interfaccia grafica X Window

Unix nasce negli anni '60, quando i grandi elaboratori centralizzati, i mainframe , lavoravano soprattutto mediante schede perforate, ed i terminali con video e tastiera, così come li conosciamo oggi, erano appannaggio di pochi fortunati. L'intero sistema operativo è stato sviluppato quindi in modo tale da prescindere completamente dal tipo di terminale con cui l'utente avrebbe utilizzato il sistema. Negli ultimi dieci anni hanno avuto un grosso sviluppo le tecnologie per la gestione del video e della cosiddetta interfaccia utente, ossia quell'insieme di dispositivi hardware e software che permettono all'utente di comunicare (in gergo, di interfacciarsi) con la macchina. Grazie al lavoro svolto da un gruppo di ricercatori ``visionari" (almeno per quei tempi, oggi possiamo dire lungimiranti) del Palo Alto Research Center (PARC) della Xerox, hanno avuto un particolare sviluppo quelle che oggi chiamiamo GUI , Graphical User Interface, che ci permettono di operare su un terminale grafico, interagendo col sistema principalmente mediante il mouse e l'uso estensivo di rappresentazioni grafiche. Ci stiamo riferendo a quello che sicuramente ognuno di noi ha visto almeno una volta sul proprio personal computer: l'ambiente Microsoft Windows , il Finder del Macintosh e la WorkPlace Shell di OS/2 , solo per citare alcuni esempi.

Anche Unix, ormai da diversi anni, è dotato di una propria interfaccia grafica standard, denominata X Window . In questo capitolo cercheremo di farci un'idea piuttosto sommaria ed un po' superficiale, di cosa ci offre in più l'uso di questo ambiente, rispetto al terminale alfanumerico.

## X Window e i window manager

X Window è un insieme di funzioni di ``basso livello" (cioè, che comunicano molto da vicino con l'hardware del sistema) per la gestione di alcune funzionalità di base dell'interfaccia utente grafica (GUI). Per alcuni aspetti, il rapporto che c'è tra Unix ed X Window, è abbastanza simile a quello che c'è tra il DOS e Microsoft Windows: si tratta in entrambi i casi di una interfaccia grafica che si appoggia sul sistema operativo e ne sfrutta le funzionalità ed i servizi. Nel caso di OS/2 la situazione è un po' diversa, perché nel Presentation Manager sono incluse alcune funzionalità fondamentali che in modalità ``a carattere" non sono previste; nel Macintosh , l'interfaccia utente costituisce parte integrante del sistema operativo da cui è inscindibile, anzi, alcune funzioni di base dell'interfaccia grafica sono addirittura incluse su ROM e fanno parte del firmware dei computer Apple Macintosh.

Non dobbiamo però confondere X Window con il window manager che si occupa della presentazione a video delle finestre e della gestione di alcune funzionalità che non sono incluse nel set di base fornito da X Window stesso. Infatti, mentre con Microsoft Windows l'aspetto delle finestre è sempre lo stesso, sopra ad X Window si appoggia un altro programma, il window manager, appunto, che definisce l'aspetto esteriore delle finestre e gestisce il desktop . Uno stesso sistema può quindi essere configurato in modo tale da consentire ai propri utenti di scegliere il window manager preferito, con cui lavorare durante le sessioni al terminale grafico.

Più in dettaglio possiamo dire che il window manager definisce l'aspetto della barra del titolo della finestra, degli eventuali bottoni collocati su di essa, delle icone e dei menù, che solitamente vengono visualizzati clickando sullo sfondo dello schermo (desktop). Si occupa anche della modalità operativa che l'utente deve adottare per operare sulle finestre: i tre pulsanti del mouse, ad esempio, svolgono funzionalità diverse a seconda del window manager adottato.

A differenza di quanto avviene con Microsoft Windows , per selezionare una finestra e renderla attiva (per poter operare al suo interno), non è necessario effettuare un click del mouse su di essa, ma (in generale) basta che il puntatore del mouse si trovi sulla finestra stessa; l'input digitato sulla tastiera sarà diretto all'applicazione eseguita nella finestra selezionata.

Alcuni window manager hanno la cosiddetta gestione del desktop "virtuale": lo schermo a disposizione dell'utente viene reso virtualmente molto più grande di quello fisicamente visibile attraverso il monitor; mediante una finestra particolare (chiamata pager ) che rappresenta in scala ridotta l'intero grande schermo, è possibile selezionare la zona da "inquadrare" nel video del terminale.

I window manager più diffusi sono quattro, ma molti altri ne esistono o sono in via di sviluppo o di definizione:

Tab Window Manager (twm ), il primo e forse il più "spartano" fra i window manager; non è molto sofisticato e tuttavia spesso è apprezzato proprio per la sobrietà con cui visualizza le finestre. Twm si limita a visualizzare una barra del titolo sopra ad ogni finestra con un bottone per "iconizzare" la finestra stessa ed un'altro per modificarne la dimensione; per spostare la finestra la si deve trascinare con il mouse clickando sulla barra del titolo.

F(?) Virtual Window Manager (fvwm ), è uno dei più diffusi window manager per X Window e nasce come evoluzione di twm. Oltre alla barra del titolo, fvwm visualizza anche un bordo intorno alle finestre e dà al tutto un aspetto tridimensionale. È possibile inserire diversi bottoni sulla barra del titolo a cui associare funzionalità diverse: rendere massima la dimensione della finestra, ridurla ad icona, ecc. Il bottone nell'angolo in alto a sinistra permette di accedere ad un menù per la gestione della finestra stessa. Con fvwm è disponibile la gestione dello schermo "virtuale" più grande di quello reale.

Open Look Window Manager (olwm ), è il window manager che Sun Microsystems ha sviluppato inizialmente solo per le proprie workstation, ma che ultimamente ha anche reso disponibile, almeno in parte, per ogni altro sistema. Personalmente ritengo che sia il window manager più elegante tra quelli disponibili; sicuramente ha un livello di sofisticazione superiore a quello offerto dagli altri prodotti. Degni di nota sono i menù a tendina "staccabili" che è possibile fissare sul desktop nella posizione più comoda per l'utente. Esiste anche una versione di Open Look dotata di desktop virtuale (olvwm ).

Motif Window Manager (mwm), il window manager di riferimento per molti sistemi UNIX (IBM AIX, HP-UX, Digital UNIX, ecc.); l'aspetto delle finestre e le funzionalità disponibili sono simili a quelle di fvwm.

Common Desktop Environment e' il window manager sviluppato nell'ambito del progetto OSF che coinvolgeva numerosi produttori di UNIX negli anni 80.

KDE e GNOME sono diventati molto famosi negli ultimi tempi grazie al successo di LINUX uno unix in grado di essere eseguito praticamente su tutte le piattaforme hardware di computer e distribuito liberamente (cioè, gratis e senza costi aggiuntivi).

## Xterm

Mediante l'uso di X Window si può sperimentare e trarre profitto con maggiore facilità dalla grande potenza del multitasking di Unix. Spesso utilizzeremo il terminale grafico per compiti che avremmo potuto svolgere ugualmente su un normale terminale alfanumerico, ma che grazie ad X Window possiamo svolgere con maggiore comodità ed efficienza. Il programma che di solito è più usato sotto X

è xterm , una finestra di emulazione di terminale , dove viene eseguita la shell da cui è possibile lanciare altri programmi.

In pratica xterm riproduce in una finestra lo schermo di un terminale alfanumerico, aggiungendo però numerose funzionalità rese disponibili dall'ambiente X Window. Aprendo più finestre di xterm sullo schermo, potremo operare contemporaneamente, grazie al multitasking, su più applicazioni. Dalla shell attivata nella finestra di xterm è possibile lanciare anche applicazioni grafiche indipendenti dalla finestra stessa.

La shell che ci viene resa disponibile mediante xterm sarà il nostro programma chiave per controllare il sistema durante la sessione di lavoro al terminale grafico. Cerchiamo di entrare un po' più in dettaglio cominciando con qualche esempio. Per sperimentare subito il multitasking potremmo provare ad aprire un altro xterm, dando semplicemente il comando ``xterm" al prompt della shell. Dopo pochi istanti il sistema apre una seconda finestra con un'altra shell attiva al suo interno. Ci accorgiamo subito però che la shell della prima finestra non è più attiva, e questo perché è stata ``congelata" temporaneamente per poter eseguire l'applicazione che da essa è stata lanciata (il secondo xterm). Dove è finito il multitasking? La risposta è semplice: non l'abbiamo sfruttato a dovere. Portando il mouse all'interno della finestra xterm appena attivata, digitiamo il comando ``exit" e vediamo ricomparire il prompt della shell che fino ad ora era rimasta inattiva. Come avevamo visto nel paragrafo è necessario aggiungere il simbolo ``&" alla fine della linea di comando per attivare il processo ``figlio" in una sessione separata da quella del processo ``padre". Per lanciare un secondo xterm è quindi necessario dare il comando ``xterm &": in questo modo viene aperta una seconda finestra e viene attivata una shell al suo interno. Ogni comando che digiteremo verrà indirizzato alla finestra attiva (quella in cui si trova il mouse), evidenziata in modo opportuno dal window manager. Ogni volta che da un xterm vorremo lanciare un'applicazione che deve essere eseguita in una sessione separata, dovremo ricordarci di aggiungere il simbolo ``&" alla fine della linea di comando.

Non sempre però è opportuno eseguire un'applicazione in una sessione separata: ad esempio se volessimo usare l'editor vi all'interno della nostra finestra xterm, non dovremo aggiungere la ``&" alla fine del comando, proprio perché questa volta desideriamo interrompere momentaneamente l'esecuzione della shell, che deve lasciare il posto, all'interno della stessa finestra, all'editor. Sperimentare direttamente queste differenze aiuterà a capire questi meccanismi molto più di quanto non possa fare io con le mie intricate spiegazioni.

È possibile lanciare il programma xterm specificando una serie di opzioni tipiche delle applicazioni che operano sotto X ampiamente documentate nelle pagine del man.

L'ultima funzionalità offerta da xterm, ma forse anche una delle più interessanti, è il copy & paste . È possibile selezionare una parte del testo visualizzato in una finestra xterm e poi incollarlo in un altro punto della stessa finestra o di una finestra differente, nella posizione occupata dal cursore. La selezione (copy) avviene mediante il mouse, tenendo premuto il pulsante sinistro; l'inserimento del testo evidenziato (paste) avviene cliccando il pulsante di centro del mouse: il testo selezionato sarà inserito a partire dalla posizione occupata dal cursore, come se venisse digitato da tastiera. Questa funzionalità arricchisce e semplifica notevolmente l'uso di un editor utilizzato all'interno della finestra xterm.

## Connessione a sistemi remoti: Telnet

Il comando telnet consente di trasformare il nostro terminale nel terminale di una macchina remota connessa alla rete Internet. Tutto ciò che noi faremo in una sessione telnet, non verrà elaborato sulla nostra macchina locale, ma impegnerà le risorse (CPU, memoria, dischi, ecc.) della macchina remota.

Ad esempio da MC-Link o Agorà (due sistemi telematici commerciali italiani) posso accedere mediante il telnet alla macchina del Dipartimento di Matematica semplicemente impostando il comando ``telnet venere.mat.uniroma1.it". Naturalmente devo avere un account sulla macchina a cui mi collego, altrimenti mi verrà rifiutata la connessione; vediamo un esempio:

```
$ telnet curie.pd.astro.it _Trying 193.206.243.129... _Connected to curie.pd.astro.it. _Escape
```

character is `^'].

Compaq Tru64 UNIX V5.1 (Rev. 732) (curie.pd.astro.it) (pts/4)\_login: infoxx\_Password:

Last login: Wed Dec 18 10:39:07 CET 2002 from clannad2.pd.astro.it

Compaq Tru64 UNIX V5.1 (Rev. 732); Thu Oct 18 11:04:56 CEST 2001

Curie.pd.astro.it>\_

I messaggi che il sistema ci presenta a video sono gli stessi di quando ci connettiamo al sistema stesso da un terminale direttamente collegato ad esso. Alla fine il sistema ci presenta il prompt da cui, come al solito, potremo impartire i comandi necessari, che però, invece di essere seguiti sulla nostra macchina, saranno eseguiti sul sistema remoto. Per sconnettersi dal sistema si userà il consueto comando exit o logout .

Il messaggio ``Escape character is..." indica che, in questo esempio, la sequenza di tasti **CTRL-]** ci permette di rientrare nel sistema locale per chiudere la connessione, sospenderla, ecc. Premendo **CTRL-]** ci viene presentato il prompt ``telnet>" del programma telnet; ad esempio per chiudere la sessione a questo punto possiamo usare il comando close. È opportuno riferirsi alla documentazione del sistema per maggiori informazioni sui comandi specifici che possono essere utilizzati.

## Scambio di files con sistemi remoti: FTP

FTP significa File Transfer Protocol e serve appunto per trasferire file da una macchina all'altra attraverso la rete Internet. Anche in questo caso, connettendosi con un computer remoto per prelevare o depositare files mediante FTP, viene richiesto all'utente di farsi riconoscere mediante il suo username e la sua password. Spesso però è utile attingere alle risorse di un archivio pubblico residente su un sistema su cui non abbiamo un account; per questo motivo, in alcuni casi, è ammesso l'uso dello username anonymous che deve essere usato in unione alla password costituita dal nostro indirizzo di posta elettronica. In questo modo possiamo accedere ugualmente ad un certo server FTP, prelevando o depositando files in un'area pubblica in cui chiunque può avere accesso. È questo uno dei principali canali di scambio di informazioni, guide, manuali e software di pubblico dominio, che arricchisce e rende per certi versi unica la comunità degli utenti Internet.

All'interno di una sessione FTP si possono fare poche cose: sostanzialmente spostarsi all'interno del filesystem della macchina remota, prelevare o depositare files. Per far questo si deve utilizzare un ristretto set di comandi; riportiamo di seguito solo i principali, rimandando alla documentazione del proprio sistema per una illustrazione più chiara ed esauriente di tutti gli altri.

cd

ha lo stesso uso e significato dell'omonimo comando della shell Unix: serve per cambiare la directory corrente;

pwd

serve per visualizzare il nome della directory corrente (print work directory);

get

serve per trasferire un file dal sistema remoto al nostro computer; la sintassi è ``get fileremoto filelocale", dove fileremoto indica il nome del file che si trova sul server FTP remoto, e filelocale (che può anche essere omesso) indica il nome da assegnare al file quando sarà memorizzato nel disco del nostro sistema locale;

put

svolge la funzione inversa del comando get, trasferendo sul sito remoto un file residente sul disco del nostro sistema locale; la sintassi è ``put filelocale fileremoto";

dir

elenca i files contenuti nella directory corrente.

Un esempio di una sessione FTP è il seguente, che riporta un collegamento con il server FTP ``nic.switch.ch'', uno dei più grandi siti europei:

```
$ ftp nic.switch.ch _Connected to nic.switch.ch _220-
Hello user at curie.pd.astro.it [193.206.243.129] _220- _220-
Welcome to the SWITHinfo ftp archive. _220- _220-
*** Access to this FTP service is exclusively allowed for *** _220- *** -
Swiss universities, schools & organisations with a *** _220-
*** SWITCH service contract *** _220- *** -
foreign education & research organisation *** _220- ... _220- nic.switch.ch FTP server
(Version 4.1259 switchinfo@switch.ch) ready. _220- There are 26 (max 50) archive users in
your class at the moment. _220- Local time is Wed May 3 17:12:39 MET _220 _Name
(nic.switch.ch:anonymous): anonymous _331 Guest login ok, give your e-mail address
(user@domain) as password. _Password: user@pd.astro.it _230- Guest
`liverani@mat.uniroma1.it' login ok ... _Remote system type is UNIX _Using binary mode to
transfer files. _ftp> cd mirror/os2 _250 CWD command successful. _ftp> dir _200 PORT
command successful. _150 opening ASCII mode data connection for . _-rw-rw-r-
 1 24 16 906 Mar 31 06:35 .message _-rw-rw-r-
 1 488 16 235980 Jul 23 06:20 00index.txt _-rw-rw-r-
 1 488 16 5742 Jul 22 19:27 0readme _drwxrwxr-x 11 24 16 512 Jul 24 01:10 1_x ...
_-rw-rw-r- 1 24 16 377 Mar 31 06:32 cdrom.txt ... _226 Transfer complete. _ftp> get
cdrom.txt _200 PORT command successful. _150 Opening BINARY mode data connection for
/mirror/os2/cdrom.txt _226 Transfer complete. _377 bytes received in 0.3 seconds (1.2
Kbytes/sec) _ftp> quit _221 Goodbye. _curie.pd.astro.it>_
```

Nell'esempio ci siamo collegati con il sistema entrando come utenti ``anonimi'', siamo entrati nella directory ``/mirror/os2'', contenente files e programmi riguardanti il sistema operativo OS/2 ed abbiamo prelevato il file *cdrom.txt*. Come al solito sperimentare questi comandi di persona aiuterà a capire il funzionamento di FTP molto più di qualsiasi altra spiegazione.

## Sintesi dei comandi principali

bash

Bourne-Again Shell è un interprete di comandi compatibile con sh che esegue i comandi letti da tastiera o da un file. bash incorpora anche alcune delle caratteristiche della Korn shell e della C shell.

bg

Continua in background l'esecuzione di un processo precedentemente interrotto battendo **CTRL-z**. come argomento del comando bg può essere specificato il job number (preceduto dal simbolo ``%") o il process ID del processo da mandare in background. Ad esempio: ``bg %3".

bye

Termina l'esecuzione della shell di login chiudendo anche la sessione di lavoro dell'utente (logout).

cal

Visualizza un calendario. Senza alcun parametro visualizza il calendario del mese corrente; si può anche specificare un particolare mese ed anno. Ad esempio: ``cal 7 1492".

cat

Concatena i files specificati come argomento del comando e li invia allo standard output. Ad esempio: ``cat file1 file2 file3".

chmod

Cambia i permessi di accesso ai file. Ad esempio: ``chmod 644 pippo".

cp

Copia i file. Il formato del comando è ``cp [opzioni] sorgente destinazione", dove sorgente è il nome del (dei) file da copiare, mentre destinazione è il nome del file o della directory in cui copiare il (i) file sorgente. Ad esempio: ``cp pippo.\* ./src".

csh

C Shell. Come bash e ksh è un interprete di comandi compatibile con sh che esegue i comandi letti da standard input o da un file.

date

Visualizza la data e l'ora corrente.

df

Visualizza lo spazio ancora libero nel filesystem.

du

Visualizza l'occupazione (in Kbyte) della directory corrente e delle sue sottodirectory.

exit

Termina l'esecuzione della shell. Se la shell è una shell di login, exit provoca il logout dal sistema.

fg

Porta in foreground il processo specificato mediante il suo process ID o il suo job number (preceduto dal simbolo ``%"). Ad esempio: ``fg 143".

ftp

È il programma che permette di fruttare il File Transfer Protocol che consente all'utente di trasferire files sulla rete dalla propria macchina ad un sito remoto e viceversa. Ad esempio: ``ftp ftp.cnr.it".

jobs

Visualizza la tabella dei processi attivi nella shell corrente.

kill

Provoca l'interruzione dell'esecuzione del processo specificato mediante il suo Process ID o mediante il job number (preceduto dal simbolo ``%"). Ad esempio: ``kill %2".

ksh

Korn Shell. Come bash e csh è un interprete di comandi compatibile con sh che esegue i comandi letti da standard input o da un file.

latex

Sistema di editoria elettronica. Converte un file di testo scritto in formato LaTeX in un file DVI (device independent). Ad esempio: ``latex tesi.tex".

logout

Termina l'esecuzione della shell di login e conclude la sessione di lavoro dell'utente, scollegandolo

dal sistema.

lpq

Visualizza la coda di stampa; ad ogni elemento della coda di stampa è associato un job number ed un proprietario.

lpr

Invia il file specificato alla coda di stampa; tramite il piping può anche essere utilizzato come filtro dell'output di un altro programma. Ad esempio: ``lpr pippo.txt" oppure ``cat pippo.txt | lpr".

lprm

Rimuove dalla coda di stampa un file ancora non stampato; si deve specificare come parametro il job number del file da eliminare, ottenuto con il comando lpq. Ad esempio: ``lprm 127".

ls

Visualizza i files contenuti nella directory specificata.

man

Visualizza la pagina di manuale del comando specificato; ad esempio: ``man mail".

mkdir

Crea la directory specificata. Ad esempio: ``mkdir tesi".

more

Visualizza il file specificato, introducendo una pausa alla fine di ogni schermata; mediante il piping può anche essere usato come filtro dell'output di un altro programma. Ad esempio: ``more pippo.txt" oppure ``cat pippo.txt | more".

mv

Sposta o rinomina il file specificato. Il formato del comando è ``mv [opzioni] sorgente destinazione", dove sorgente è il nome del (dei) file da spostare, mentre destinazione è il nome del file o della directory in cui spostare il (i) file sorgente. Ad esempio: ``mv pippo.\* ./src".

pine

Sistema per la gestione della posta elettronica.

ps

Visualizza l'elenco dei processi attivi.

pwd

Visualizza il nome della directory corrente.

rm

Cancella i file specificati. Ad esempio: ``rm ./src/\*".

rmdir

Rimuove la directory specificata, che deve essere vuota (non deve contenere file o subdirectory). Ad esempio: ``rmdir src".

sh

È la shell di riferimento. Come bash , ksh e csh è un interprete di comandi che esegue i comandi letti da standard input o da un file.

tar

Tape Archive. Consente di archiviare in un unico file (che di default viene memorizzato su nastro) più file e directory. Con tar è anche possibile effettuare l'operazione inversa, cioè estrarre da un file

di archivio (in genere con estensione ``.tar") i files originali. Ad esempio: ``tar xfv pippo.tar".

telnet

È il programma con cui è possibile sfruttare il protocollo TELNET per utilizzare un sistema remoto collegato in rete (ad esempio su Internet). Ad esempio: ``telnet mclink.mclink.it".

tex

È il comando per eseguire il sistema di editoria elettronica TeX. Converte un file di testo in formato plain TeX in un file in formato DVI (device independent). Ad esempio: ``tex tesi.tex".

vi

Editor di file di testo.

who

Visualizza l'elenco degli utenti collegati al sistema.

whoami

Visualizza lo username dell'utente.

xterm

Emulazione di terminale alfanumerico sotto X Window.

yes

Visualizza indefinitamente il carattere ``y".