Corso di Unix per principianti

a cura di Paolo Mastroserio

INDICE

Introduzione

Nonostante ci siano numerosi scritti su *UNIX*, erano sempre molto frequenti le richieste sia da parte di studenti che da parte di docenti e ricercatori su qualcosa che potesse indirizzare chi per la prima volta si avvicinava a questo sistema operativo.

Ho pensato di scrivere queste pagine presupponendo un minimo di conoscenze d'informatica da parte del lettore ritenendo che alcuni termini, una volta usati soltanto dagli addetti ai lavori, facciano invece parte oggi del linguaggio comune.

I comandi in *UNIX* sono numerosi e talvolta molto complessi, di dialetti di questo sistema operativo ve ne sono svariati, abbiamo ad esempio *Ultrix* e *DEC-Unix* della DIGITAL, *AIX* dell'IBM, *HP-UX* dell'HP, *Sun OS* e *Solaris* della SUN, *Linux* ecc. In conseguenza di ciò lo scopo di questo corso non poteva e non voleva essere la trattazione completa di tutti i suoi comandi e su tutte le piattaforme principali.

Con questo corso ho volutamente trattato gli argomenti in maniera sintetica e certamente non esaustiva, prefiggendomi viceversa l'obiettivo di fornire strumenti sufficienti affinché il lettore acquisisca la capacità di muoversi da solo in quest'ambiente. Questo secondo me è indispensabile perché è mia convinzione che in *UNIX*, per risolvere alcuni problemi non c'è corso che tenga, e che solo l'esperienza consentirà di venirne eventualmente a capo. Ciò è dovuto all'estrema cripticità di questo sistema operativo nato e pensato da più menti e in periodi in cui la realtà era molto diversa da oggi. *UNIX* nacque infatti quando i processori non erano così veloci, quando le memorie sia centrali che di massa non erano così grandi e conseguentemente i comandi e le opzioni erano ridotti a poche lettere essenziali e quasi sempre incomprensibili.

Saranno naturalmente graditi commenti, richieste di aggiunte, suggerimenti e segnalazioni di eventuali errori. Per scrivere queste righe sono stato costretto a rimettermi a studiare, a fare delle prove e a consultare scritti altrui quali ad esempio *Shell Choice, A shell comparison (july 27, 1994)* di Arnaud Taddei, *The Little Gray Book: An Ultrix Primer* della Digital Equipment Corporation, *Introduzione all'uso del sistema DECsystem 5900* di Fabio Solarino, *UNIX* di Maurizio Matteuzzi e di Paolo Pellizzardi ecc.

CAPITOLO I

CENNI GENERALI

1.1 Come ci si collega *(login, password)*

La prima cosa da fare quando ci si collega ad un calcolatore con sistema operativo di tipo *Unix* è quella di fornirgli il nome con cui il gestore del sistema ha creato l'area di lavoro, che in gergo viene denominato *account,* ed il relativo codice di autorizzazione.

Se il calcolatore non ha un video grafico e quindi, escludendo che ci si trovi di fronte a una delle tante ed accattivanti schermate proposte nelle varie piattaforme, quello che apparirà all'atto del collegamento sarà una richiesta di account con la scritta:

```
login:
```

cui seguirà la richiesta del codice di autorizzazione con la scritta:

```
Password:
```

A queste domande l'utente dovrà rispondere tenendo conto che in *UNIX le lettere minuscole e le lettere maiuscole sono interpretate differentemente*.

Quando si è in rete e ci si collega da un calcolatore a un altro, si può riscontrare un'incompatibilità di tastiera; infatti talvolta, se si sbaglia a scrivere il nome dell'*account* o della *password* l'utilizzo del tasto *backspace* può provocare la visualizzazione di caratteri imprevisti quali ad esempio:

```
^[[D^[[D^[[D
```

a questo punto per ottenere l'effetto di *cancellazione arretrando* si può provare ad usare in luogo del backspace il caratere di controllo:

```
ctrl-H
```

a volte funziona.

Sempre nel caso in cui ci si colleghi da un calcolatore all'altro e magari con una linea di comunicazione non troppo veloce, se dopo aver digitato il nome dell'*account* si passa *troppo velocemente* a digitare *la propria password,* può verificarsi la seguente sgradevole situazione: che il calcolatore remoto non avendo ancora ricevuto il nome dell'account continuerà a visualizzare i caratteri che gli

provengono dalla tastiera; se a questo punto si digita la password essa comparirà regolarmente a video alla mercé di occhi indiscreti.

In realtà è solo dopo la comparsa della parola *Password* che il sistema operativo smetterà di visualizzare i caratteri che gli vengono dalla tastiera, e solo allora converrà digitare il proprio codice di autorizzazione.

1.2 Cambio della password *(passwd)*

Una volta collegati ci si ritroverà in un ambiente che sarà o in modalità testo o in modalità grafica con interfaccia utente amichevole.

La prima operazione da fare sarà *il cambio della password !*

I sistemi con una configurazione *UNIX* standard non hanno una gestione della sicurezza eccessivamente sofisticata per cui all'atto del primo collegamento non chiedono di cambiare la password, questa operazione dovrà quindi essere effettuata a cura dell'utente. Il comando per effettuare questo cambiamento è:

```
%    passwd
```

il simbolo di percentuale non va digitato, si tratta del cosiddetto *prompt* che viene utilizzato per convenzione in ambiente *UNIX*; in ambiente *VMS* viene usato il simbolo del dollaro *$* e in ambiente *DOS* viene generalmente usato il simbolo *C:\>*.

Il sistema operativo *Unix*, a meno che non siano stati installati programmi specifici, sia per il nome dell'account che per la password prevede la possibilità di utilizzare fino ad un *massimo di otto caratteri,* conseguentemente se al momento di cambiare la password verrà digitato un codice con più di otto caratteri, la parte eccedente verrà ignorata.

Sia *mastro* l'utente collegato, al comando *passwd* il computer risponderà nel seguente modo:

```
%    passwd
Changing password for mastro
Old password:
Enter new password:
Verify:
```

da quanto mostrato appare chiaro che bisognerà fornire prima la vecchia password, questo farà si che se il terminale rimane momentaneamente incustodito, solo chi la conosce potrà cambiarla e non una persona qualunque che in questo modo s'impadronirebbe dell'account.

Siccome durante la battitura la password non compare a video, il sistema ne chiede la ripetizione come verifica onde evitare che si siano battuti accidentalmente tasti diversi da quelli voluti.

1.3 Come ci si scollega *(logout, exit, ctrl-D)*

Per scollegarsi il comando *standard* è:

```
%       logout
```

si può usare anche *exit*; alcuni sistemi operativi quali l'*Ultrix* della Digital consentono per l'uscita l'utilizzo del codice di controllo *ctrl-D***.**

Va segnalato che in ambiente *UNIX non sono previste le abbreviazioni*, vale a dire che non esiste ad esempio la possibilità di usare il comando *lo* come abbreviazione di *logout* !

Qualora ne sorgesse la necessità, è possibile creare degli *alias*, vale a dire delle parole cui associare comandi *Unix* in modo tale da creare un ambiente personalizzato.

1.4 Ambiente di lavoro *(gli script)*

Una volta collegati si può dare inizio a tutte quelle attività che ci si aspetta di poter fare con un calcolatore per mezzo del suo sistema operativo. Un sistema operativo deve mettere a disposizione dell'utente la possibilità di dare comandi, di elaborare testi, di scrivere programmi, di compilarli, di eseguirli ecc. Una delle principali caratteristiche del sistema operativo *UNIX* è che questo immerge l'utente in un ambiente il cui insieme di comandi costituisce un linguaggio vero e proprio.

Per poter dire che l'affermazione precedente è vera è necessario ricorrere al teorema di *Boehm Jacopini* che dice che qualsiasi algoritmo è esprimibile facendo ricorso alla presenza delle tre condizioni di seguito riportate:

- la *sequenza* (eseguire una sequenza di comandi);
- l'*iterazione* (ad esempio *foreach ... end*);
- l'*alternativa* (ad esempio *if <statement> then <statement> ... fi)*

e siccome è possibile creare un file di comandi con tutte e tre le strutture ritenute minimali dal teorema appena citato possiamo dire certamente che le shell di *Unix* sono dei linguaggi a tutti gli effetti.

E' chiaro a questo punto che si possono creare *file di comandi* di qualunque complessità per svolgere operazioni di sistema, compilazione e linking di programmi, manipolazione di file ecc.

Questi file di comandi in *UNIX* si chiamano *script*.

A complicare la vita ci pensano i molteplici linguaggi esistenti; essi hanno caratteristiche diverse, non sono presenti su tutte le piattaforme, alcuni sono proprietari, altri sono *public domain*. Questi linguaggi vengono chiamati *shell (gusci),* e poggiano su un substrato comune a tutte le piattaforme detto *kernel (nocciolo)* e saranno oggetto del paragrafo successivo.

1.5 Kernel e shell di *Unix* *(sh, csh, ksh, tcsh, bash, zsh)*

I sistemi *UNIX* sono costituiti di due parti principali, l'una detta nocciolo o *kernel* e l'altra detta guscio o *shell*. Quando si accende un calcolatore inizia quella che in gergo viene detta *fase di boot*; in quel momento parte un programma che non si fermerà più se non allo spegnimento o se si vuole allo *shutdown* del sistema, fase opposta a quella di boot; questo programma è il *kernel*. Generalmente il *kernel* è un programma di nome *vmunix, esso* si trova nella directory principale; quest'ultima o viene chiamata *root* (radice), o viene indicata con la barra obliqua "/".

Il kernel è il cuore del sistema e la shell è un programma che interpreta i comandi dell'utente; detti comandi vengono acquisiti dalla shell e rigirati al kernel che ha il diretto controllo delle periferiche quali ad esempio i dischi e le stampanti. Si suole dire quindi che la shell (il guscio) avvolge il kernel (nocciolo) al suo interno.

Uno dei principali vantaggi di avere un sistema operativo organizzato in questo modo è quello di poter cambiare l'ambiente (shell) senza dover cambiare il kernel, ne è riprova la loro proliferazione. I vari punti di vista ci fanno raggruppare le shell come sotto riportato:

- per sintassi e grammatica:

  Bourne shell (AT&T): *sh, ksh, bash, zsh*;

  C shell: *csh, tcsh;*

- per supporto e sviluppo:

  supportate dai fornitori: *sh, csh, ksh*;

  public domain: *bash, tcsh, zsh*;

■ per ordine cronologico: *sh, csh, ksh, tcsh, bash, zsh*.

La *Bourne shell sh* (scritta da *S. R. Bourne)* fu sviluppata dall'AT&T ed è la capostipite di tutte le shell moderne, utilizza come *startup file* il file *.profile*.

La *C shell csh* (sviluppata *dall'Università della California di Berkley*), ha la sintassi e la grammatica molto vicini al linguaggio C. Le aziende di software hanno realizzato versioni senza usare gli stessi file di startup.

La *Korn shell ksh* (scritta da *David Korn*) è una estensione delle precedenti, assomiglia alla shell di POSIX, non è presente sulla piattaforma *SUN* che invece è la più venduta del mondo.

La Bash shell *bash* (realizzata dal progetto *GNU*) sviluppa le caratteristiche della Bourne shell e ha delle somiglianze con la csh e la tcsh.

La *tcsh* è una riscrittura della *csh* e ha lo scopo di correggerne le imperfezioni e incrementarne le caratteristiche; è *public domain*.

La *zsh* è considerata dagli esperti di shell *la più potente di tutte*, è presente su *tutte* le piattaforme, ha delle forti somiglianze con la *tcsh* e include sostanzialmente la maggioranza di tutte le caratteristiche delle altre shell finora descritte.

Senza scendere nei dettagli delle giustificazioni tecniche, si può affermare che le migliori shell sono la *TCSH* e la *ZSH*, e attualmente sono le più *consigliate negli ambienti della fisica delle alte energie (HEP)*.

CAPITOLO II

STRUTTURA DI *UNIX* E SUOI COMANDI PRINCIPALI

2.1 Personalizzazione dell'ambiente *(csh, i file di startup, alias, tipi di variabili, (un)set, echo, caratteri speciali e metacaratteri, rehash, concatenazione di stringhe)*

Il comando chsh. Si supponga che l'utente *mastro* si colleghi al computer denominato *dsna1* da un terminale *non grafico*, dopo aver digitato *login* e *password* gli si presenterà una schermata simile alla seguente:

```
Digital UNIX (dsna1.na.infn.it) (ttyp4)

login: mastro
Password:


Last login: Sat Jul 26 23:38:00 from osfna1.na.infn.i

Digital UNIX V4.0A(Rev. 464); Wed Oct 30 12:02:56 MET 1996

dsna1 /usr/users/mastro>
```

all'atto del collegamento va in esecuzione la shell impostata dal *gestore del sistema* quando ha creato l'account; questa può essere cambiata in qualunque momento in maniera temporanea o permanente; per cambiare shell basta digitare il comando:

```
%      chsh
```

alla richiesta *new shell* bisognerà indicare una delle shell previste dal sistema, ad esempio */bin/tcsh*.

I file di startup. Come citato in precedenza ogni shell ha i suoi file di startup, in ambiente *DOS* abbiamo il file *AUTOEXEC.BAT* mentre in *VMS* abbiamo il file *LOGIN.COM*. *Usando la C shell* in ambiente *DEC Unix,* all'atto del collegamento vengono eseguiti nell'ordine i seguenti script di startup:

- /etc/csh.login
- $HOME/.cshrc
- $HOME/.login

Il primo script non appartiene e non si trova nell'area del singolo utente; il suo contenuto viene deciso dal gestore del sistema e sarà eseguito ogni volta che un qualunque utente si collega; contiene le impostazioni generali che se non soddisfano le esigenze informatiche del singolo utente possono essere annullate e sostituite da altri comandi da inserire nei file di startup personali *.cshrc* e *.login*.

Quando nacquero i primi sistemi operativi di tipo *UNIX* i terminali video non erano così diffusi come oggi, conseguentemente per interagire con le *CPU* si usavano telescriventi lentissime; per accorciare i tempi d'interazione chi scriveva sistemi operativi, e quindi anche chi scrisse le prime versioni di *UNIX*, decise di adottare comandi brevissimi quali *ls* per list, *cp* per copy, *mkdir* per make directory ecc. e di usare singole lettere in minuscolo e in maiuscolo per le opzioni; il risultato finale è stato che oggi per chi non è pratico l'interfaccia utente dei sistemi *UNIX* risulta particolarmente criptica.

Nei file di startup .cshrc e .login, oltre a inserire comandi da far eseguire all'inizio del collegamento si possono impostare variabili d'ambiente e creare anche dei cosiddetti *alias* ovvero nomi secondari da dare ad altri comandi.

I comandi *UNIX* hanno la forma seguente:

```
comando [-opzione] [argomenti]
```

se si vuole la lista dei file contenuti nella propria directory il comando da adottare è *ls*, questo prevede, in ambiente *DEC UNIX* ad esempio, una quantità innumerevole di opzioni che per mera curiosità si riporta di seguito:

```
ls [-aAbcCdfFgilLmnopqrRstux1] [file | directory]
```

Vediamo ora un esempio di file di startup

```
umask 022
set notify
set ignoreoff
set editmode = emacs
set term = vt100
set path = ( . /sbin /usr/sbin /usr/bin )
alias dir ls
alias l 'ls -la | more'
alias lo logout
biff y
```

si ricorda ancora una volta che le lettere maiuscole *non* vengono interpretate allo stesso modo delle lettere minuscole.

Il comando alias. Digitando semplicemente il comando *ls* si avrà una lista dei file nel direttorio corrente, ma se si è abituati ad esempio ad usare il comando *dir* è possibile creare un'equivalenza tra la stringa di caratteri *dir* e la stringa *ls*. Questa equivalenza la si ottiene digitando il comando:

```
%    alias dir ls
```

tuttavia scollegandosi e ricollegandosi al sistema questo perderà la nozione di associare la parola *dir* al comando *ls* per cui sarà conveniente inserire la stessa riga nel file di startup come sopra mostrato. Come si è potuto notare quindi è possibile creare un ambiente costituito di termini familiari o di comodo; chi ad esempio si collega molto spesso con macchine di altri siti può creare una alias di poche lettere come ad esempio:

```
%    alias cern 'telnet vxcern.cern.ch'
```

in questo caso basterà digitare la parola *cern* per ottenere l'esecuzione del comando *telnet vxcern.cern.ch;* si noti che per associare alla parola *cern* un insieme di parole come *telnet* e *vxcern.cern.ch*, queste devono essere racchiuse all'interno della coppia di apici `''`.

Per conoscere tutti gli *alias* impostati basterà digitare il comando senza opzioni.

Tipi di variabili, i comandi set e unset. Abbiamo visto come si creano gli alias, vediamo ora come s'impostano le variabili d'ambiente o di shell. Le variabili di shell sono di due tipi:

- variabili binarie;
- variabili stringa.

Le *variabili binarie* possono avere due valori: *on* e *off*. Si supponga di aver scritto un programma in *C* di nome *myprog* e di volerlo mandare in esecuzione, ci sono due modi per farlo e sono riportati di seguito:

```
%    myprog
%    myprog &
```

nel primo caso il programma viene eseguito e l'utente non può fare nient'altro che aspettare che termini l'esecuzione per poter digitare altri comandi; in questo caso si dice che il programma gira in *foreground*; nel secondo caso il sistema *restituisce il*

prompt all'operatore che può digitare altri comandi o addirittura scollegarsi. Il programma continuerà a girare fino alla fine e in questo caso si dice che funziona in *background*. Quando un programma gira in *background* ha due possibilità al momento in cui termina: *notificare* o *non notificare* all'utente l'avvenuta conclusione. Questa notifica avviene o meno se *la variabile binaria notify* è impostata a *on* o a *off*. Per effettuare questa scelta bisogna dare uno dei due seguenti comandi:

```
%    set notify
%    unset notify
```

Naturalmente vi sono altre variabili di shell di tipo binario che si possono impostare o *on* o a *off* quali ad esempio *ignoreeof,* questa serve per indicare se il tasto *CTRL-D* può essere usato come l'equivalente del comando *logout* o meno, la variabile *noclobber* che serve per impedire o meno che un *programma ridiriga il suo output* su un file che già esiste. Sul concetto di *ridirigere l'output verso un file* da parte di un programma ci soffermeremo in seguito.

Per quanto riguarda le variabili stringa c'è da dire che queste possono essere al servizio o di programmi di sistema o di programmi applicativi. In precedenza abbiamo visto alcune righe che si possono trovare in un file di startup, una di queste è:

```
set path = (. /sbin /usr/sbin /usr/bin )
```

si tratta dell'assegnazione di un valore ad una variabile.

Il comando echo. Per leggere il contenuto della variabile *path* bisognerà digitare quanto seguente:

```
%    echo $path
```

da ciò si evince che il comando *echo* serve per visualizzare dati e il simbolo del dollaro posto davanti al nome delle variabili serve a indicarne il contenuto.

Caratteri speciali e metacaratteri. Gli utilizzatori del *VMS* dovranno tener conto che i nomi dei file in *UNIX* non possono contenere il simbolo del dollaro. Mentre in *VMS* il simbolo del dollaro è equivalente a un qualunque altro carattere, in *UNIX* esso ha una funzione ben precisa: se posto davanti al nome di una variabile servirà ad indicare il contenuto della stessa.

Se in seguito all'importazione di file da una macchina con sistema operativo *VMS* o per un qualunque altro motivo ci si ritrovi nella propria directory un file il cui nome contiene il dollaro o altri *caratteri non consentiti,* questi non potranno essere

richiamati semplicemente con il loro nome. Infatti se detto file si chiama *sys$user*, non sarà possibile dare il seguente comando:

```
%     ls -la sys$user          <sbagliato!>
```

perché in questo caso *Unix* riterrà di dover elencare un file il cui nome è formato dalla stringa *sys* più una seconda stringa contenuta nella variabile *user*!

Per poter richiamare tali file si dovrà anteporre la barra retroversa "\" al carattere in questione, in questo modo il sistema operativo lo tratterà come un *qualunque carattere*. Conseguentemente per riferirsi al file *sys$user* bisognerà digitare:

```
%     ls -la sys\$user          <corretto!>
```

Un *metacarattere* è un carattere che serve ad indicare altri caratteri. Ad esempio l'asterisco ***** serve a indicare *qualsiasi carattere in una qualsiasi quantità*, il punto interrogativo **?** rappresenta *uno ed un sol qualsiasi carattere*, due punti interrogativi **??** rappresentano *due e due solo qualsiasi caratteri* ecc.

Con il comando:

```
%     ls -lR p*
```

si elencheranno tutti e solo i file i cui nomi cominciano per *p*; grazie all'opzione *-R* (in maiuscolo e non in minuscolo altrimenti cambia tutto!) si elencheranno *ricorsivamente* anche tutte le directory e le sottodirectory che cominciano con la lettera p.

Con il comando:

```
%     ls -l p??
```

si elencheranno solo nomi di file di tre lettere i cui nomi cominciano con la lettera p.

Il comando rehash. Il contenuto della variabile *path* è al servizio del sistema operativo e serve a indicare in quali *directory* e con quale ordine deve andare a cercare i file che sono stati invocati per l'esecuzione. Poiché per un computer *nulla è ovvio,* se non si imposta nella variabile *path* anche *la directory corrente* che va indicata con il punto "**.**", possono succedere cose curiose: se si crea un programma di nome *myprog* e si tenta di lanciarlo con il comando:

```
%     myprog
```

si riceverà in risposta il messaggio inaspettato *file not found*. Molti per abitudine, non sapendo o non avendo voglia di verificare se la variabile *path* contiene o non il punto, lanciano i programmi direttamente nel seguente modo:

```
%     ./myprog
```

il punto e la barra servono ad indicare che il file va cercato nella directory corrente. Un'altra situazione in cui si può ricevere in risposta il messaggio inaspettato *file not found,* pur avendo una path contenente anche il punto, è quando si lancia un programma subito dopo averlo creato. In realtà succede che il sistema operativo anziché controllare il contenuto delle directory, verifica l'esistenza dei file leggendo da un'area di memoria temporanea creata in precedenza e che aggiorna solo periodicamente. Se detta area non è stata ancora aggiornata ci dirà che il file non esiste. Per forzare questo aggiornamento basta usare il seguente comando:

```
%     rehash
```

Concatenazione di stringhe. In precedenza abbiamo visto citare il file *$HOME/.cshrc*; il suo nome è composto dalla concatenazione di due stringhe: *$HOME* che indica la directory assegnata all'utente dal gestore del sistema al momento della creazione del suo account, e *.cshrc* che è il nome del file. Nel caso dell'utente *mastro* se il contenuto della variabile *$HOME* è */usr/users/mastro* vuol dire che il nome completo è */usr/users/mastro/.cshrc*.

Nei file di startup è possibile anche lanciare direttamente dei programmi, siano essi di sistema che applicativi, quali *date* che visualizza la data, il comando *biff y* che in questo modo fa si che il computer fornisca un messaggio quando arriva una *mail* in luogo di *biff n* che invece non lo fa emettere, il comando *umask 022* che imposta i cosiddetti *permessi di default* di cui parleremo successivamente ecc.

2.2 - Come farsi aiutare dal sistema *(man)*

Nei sistemi operativi di tipo *Unix* il comando che consente di consultare i manuali in linea è *man*. Alcuni ambienti proprietari forniscono anche altri tipi di aiuto quali il comando *help* che fornisce brevi spiegazioni sui singoli comandi e *apropos stringa* che fa una ricerca sulle occorrenze di una determinata stringa nei file di testo dei manuali presenti in macchina.

Anche in questo caso ci sono versioni differenti per quanto riguarda la modalità di consultazione dei manuali; le opzioni da utilizzare possono differire a seconda del sistema con cui ci si è collegati e anche la posizione stessa dei manuali può variare.

Generalmente il comando *man* fornisce l'accesso alle pagine in questione attingendo dalla directory /usr/share/man ma è possibile creare anche aree aggiuntive in cui sistemare ulteriori pagine per specifiche esigenze di gruppi di lavoro che sviluppano nuovo software cui allegano documentazione sotto forma di *man*.

Abitualmente il comando *man,* per accedere al database contenente i manuali, segue prima il percorso /usr/share/man e successivamente il percorso /usr/local/man. Lo studioso lettore può provare a dare il comando *man man* con cui verranno visualizzate le funzioni del comando in questione quali ad esempio la possibilità di indicare pagina o sezione del comando di cui si ha bisogno di spiegazioni.

Un output parziale del comando *man ls* è il seguente:

```
     NAME

    ls list file and directory names and attributes

     SYNOPSIS

ls [-AabCcdFfgikLlmnopqRrstux1] [-X attr] [pathname]

     DESCRIPTION

    ls lists files  and directories. If  the pathname is a
file, ls displays information on the file according to
the requested options. If it is a directory, ls displays
information on the files and subdirectories therein. You
may obtain information on a directory itself using the  -
d option.
```

2.3 - Alcuni comandi base *(ls, nomi di file, cat, more)*

Il comando ls. Come si è potuto osservare nel paragrafo precedente, il comando per elencare l'insieme dei file contenuti nella propria directory è *ls* e si è visto anche che ad esso è possibile associare una quantità innumerevole di opzioni per ottenere gli output più svariati. Un esempio di output dovuto alla digitazione del semplice comando *ls* è il seguente:

```
%     ls

README README.PS a2ps nsmail paw pine pride printscreen.ps
```

mentre aggiungendo, precedute dal segno meno, le opzioni *l* per *long* e *a* per *all*, si ha il seguente output:

```
%     ls -la

drwxr-x--x  20 mastro   system      2048 Sep 27 17:02 .
drwxr-xr-x  94 root     system      2048 Mar 26  1997 ..
-rw-------   1 mastro   system       955 Sep 17 10:34 .Xauthority
```

```
-rw-r--r--   1 mastro   system       417 Jul 22 11:21 .Xdefaults
-rw-r--r--   1 mastro   system     18844 May 23 19:17 README
-rw-r--r--   1 mastro   system     82665 Jun 10 16:29 a2ps
drwx------   2 mastro   system       512 Sep 10  1996 nsmail
lrwxr-xr-x   1 mastro   system      2860 Apr 11 11:08 paw
-rwxr-xr-x   1 mastro   system   3219065 Jun  9 12:47 pine
drwxr-xr-x   2 mastro   system       512 Nov 10  1995 pride
-rw-r--r--   1 mastro   system     55149 Jul 17 10:39 printscreen.ps
```

nel primo caso, ci troviamo di fronte ad un semplice elenco di file proposti uno dietro l'altro sullo stesso rigo, mentre nel secondo caso abbiamo i file disposti uno per ogni rigo e con delle informazioni aggiuntive. Quest'ultimo elenco inoltre contiene due file in più e sono indicati rispettivamente con un punto e con due punti. Il file indicato con un punto contiene le informazioni inerenti la directory corrente con le indicazioni di tutti i file in essa contenuti ed eventuali sottodirectory. Il file indicato con i due punti contiene invece le informazioni inerenti la directory superiore. Tramite l'opzione *a* vengono evidenziati file come .Xauthority e come .Xdefaults i cui nomi cominciano con il punto.

Si può notare che sulla metà di destra della schermata appaiono anche informazioni quali la data e l'ora dell'ultima modifica del file e la dimensione in byte, sulla sinistra appaiono altre informazioni su cui ci soffermeremo successivamente.

Come abbiamo visto il comando *ls* prevede tante opzioni, possiamo ricordare la *p*, questa se usata distingue le directory dagli altri file facendo seguire al nome del file la barra obliqua /, la *R*, che visualizza ricorsivamente tutte le directory e sottodirectory a partire dal punto indicato, la *u*, che visualizza la data di ultimo accesso al file invece che la data di ultima modifica ecc.

Nomi di file. Mentre in *DOS* i nomi dei file sono composti da una stringa iniziale di al massimo otto caratteri più eventualmente un'altra stringa di altri tre caratteri separata da un punto, in *UNIX* si ha la possibilità di assegnare nomi di qualunque lunghezza. Non essendo inoltre il punto considerato un carattere speciale, è possibile avere nomi di file del tipo *printscreen.ps.save*. In *Unix*, per indicare un insieme di file che hanno una parte del nome uguale, è possibile l'uso dell'asterisco e quindi si possono dare comandi quali:

```
%     ls -la p*
```

che producono output del tipo:

```
lrwxr-xr-x   1 mastro   system      2860 Apr 11 11:08 paw
-rwxr-xr-x   1 mastro   system   3219065 Jun  9 12:47 pine
drwxr-xr-x   2 mastro   system       512 Nov 10  1995 pride
-rw-r--r--   1 mastro   system     55149 Jul 17 10:39 printscreen.ps
```

si può notare che a differenza del *DOS*, non essendo i nomi costituiti di desinenza e tema, detti comandi sortiscono effetti differenti.

I comandi cat e more. Il comando *cat* serve per dirigere sul video il contenuto di un file; il comando *more*, a differenza del precedente, ne ferma a fondo video la visualizzazione se esso è costituito di un numero di righe superiore a quelle disponibili sullo schermo; con alcuni sistemi operativi più moderni c'è la possibilità di scorrere avanti e indietro lo stesso utilizzando i tasti *i, j, f, b* ecc..

2.4 - Ridirezione dell'I/O, filtri e pipe *(| , >, <, il comando grep)*

Un'interessante peculiarità del sistema operativo *UNIX* è la possibilità di poter ridirigere l'*input* e l'*output* di un programma, tuttavia uno dei concetti più importanti che fanno di *UNIX* un sistema operativo originale e sofisticato è la concatenabilità dei comandi per mezzo della barra verticale | detta *pipe*.

Si supponga di aver dato il comando *ls -la*, il suo output verrà diretto verso il terminale su cui si sta lavorando, quest'ultimo viene chiamato *standard output*; qualora si voglia ridirigere detto output verso un file denominato *uscita* basterà dare il seguente comando:

```
%    ls –la > uscita
```

se il file *uscita* già esiste il suo contenuto verrà *cancellato* e sostituito dall'output del comando *ls -la* ma se si vuole *appendere* questo output al file già esistente si dovrà allora usare il simbolo >> dando il comando:

```
%    ls –la >> uscita
```

Si supponga di dover far girare un programma di nome *myprog* che chiede diversi dati da tastiera e che questa operazione debba essere fatta più volte magari perché il programma è in corso di sviluppo; può risultare comodo mettere questi dati provenienti da quello che chiameremo lo *standard input* direttamente in un file di nome *dati* e lanciare il programma così come di seguito riportato:

```
%    myprog < dati
```

Nei sistemi operativi di tipo *UNIX* oltre a parlare di *standard input* e di *standard output* si parla anche di *standard error*, il modo di ridirigere l'output di un programma verso quest'ultimo cambia a seconda della shell in cui ci si trova.

Veniamo ora alla ***caratteristica più originale di UNIX*** che è quella di poter dare comandi concatenati e tali che ***l'output di un programma costituisca l'input di un altro programma***. Si prenda in considerazione la seguente concatenazione di comandi tramite la *pipe* |:

```
%    cat my_file | sort
```

il risultato finale sarà che l'output del processo *my_file* verrà dato in input al processo *sort* che effettuerà un ordinamento alfabetico. In questo caso il processo *cat my_file* viene detto *filtro;* si lascia al lettore l'immaginazione di quanto possa essere utile la possibilità di far scambiare dati tra programmi senza dover ricorrere all'utilizzo di file temporanei.

Questo ponte, o meglio questo *canale* creato tra i due programmi dal sistema operativo *UNIX* viene chiamato *pipe* e si tratta di un *buffer* allocato in memoria centrale. Il processo *cat my_file* si metterà in attesa quando detto buffer sarà pieno ovvero fintanto che il processo *sort* non avrà terminato di leggerne il contenuto, una volta liberato il processo *cat my_file* riprenderà a scrivere sul buffer e così via finché il file *my_file* non verrà completamente scandito. Una volta terminato il processo, il sistema operativo cancellerà il *buffer* utilizzato senza che l'utente debba far nulla ed in maniera del tutto trasparente.

Il comando grep. Una tipica applicazione dei concetti appena esposti può essere rappresentata dalla seguente concatenazione di comandi:

```
%    cat my_file | grep Unix | sort >> your_file
```

il comando *grep* riceve in input per mezzo della *pipe* l'output del comando precedente, la sua funzione è di ricercare tutte quelle righe che contengono la parola *Unix*, l'output così ottenuto viene ancora dato in input al comando *sort* che dopo averne fatto il riordino alfabetico appenderà il suo output nel file *your_file*.

Anche il comando *grep* ha molte opzioni, ad esempio con il comando

```
%    cat my_file | grep -v Unix
```

è possibile *catturare* tutte le linee contenute nel file *my_file tranne* quelle che contengono la stringa *Unix*; altra opzione interessante è la *-i* che consente di effettuare la ricerca della stringa indipendentemente se compare in maiuscolo o in minuscolo ecc.

2.5 - Protezione e link di file *(chown, chgrp, protezioni, ln, protezioni, chmod, umask)*

Precedentemente abbiamo visto che il comando `ls -la p*` aveva prodotto il
seguente output:

```
lrwxr-xr-x 1 mastro system    2860 Apr 11 11:08 paw ->/usr/cern/pro/bin/paw
-rwxr-xr-x 1 mastro system 3219065 Jun  9 12:47 pine
```

```
drwxr-xr-x 2 mastro system     512 Nov 10  1995 pride
-rw-r--r-- 1 mastro system   55149 Jul 17 10:39 printscreen.ps
```

e che necessitava ancora di alcuni chiarimenti.

Ad ogni utente registrato su una macchina *UNIX* viene associato un nome e un numero detto *UID (user identifier)*, detto utente viene anche associato a un gruppo di utenti che viene anch'esso identificato con un nome e un numero detto *GID (group identifier)*. Possiamo notare che i file dell'esempio appena riportato appartengono tutti all'utente mastro e al gruppo system:

UID (user identifier) = mastro
GID (group identifier) = system

Ogni qual volta che l'utente *mastro* crea un file, questo apparterrà all'utente *mastro,* tutti gli appartenenti al suo stesso gruppo (*system)* potranno o non potranno utilizzare questo file a secondo dell'insieme di permessi che ad esso verrà associato.

I comandi chown e chgrp. Si supponga di avere un file di nome *pride* appartenente all'utente *mastro* e al gruppo *system* e che gli si voglia cambiare proprietario e gruppo in *mario* e *users*; per fare ciò basteranno i due seguenti comandi:

```
%    chown mario pride
%    chgrp users pride
```

dove *chown* sta per *change owner* e *chgrp* sta per *change group*.

Abbiamo visto che l'output del comando *ls -la* presenta all'estrema sinistra di ciascun file una stringa di dieci caratteri, il primo ne indica il tipo; i tipi di file che il nostro utente incontrerà più frequentemente sono di seguito riportati:

```
d    directory
-    file generico
l    link simbolico
```

i primi due si commentano da soli, il terzo rappresenta un riferimento ad un file che può trovarsi altrove o nella medesima directory ma con un nome diverso.

Il comando ln. Si noti che nell'elencazione precedente appare un file di nome *paw:* questo non è un file vero e proprio ma contiene un collegamento ad un altro file anch'esso chiamato *paw* e che si trova nella directory */usr/cern/pro/bin*:

```
lrwxr-xr-x 1 mastro system 25 Apr 11 11:08 paw ->/usr/cern/pro/bin/paw
```

detto collegamento si chiama *link simbolico*. Se si digita il comando *paw* verrà quindi eseguito il file cui esso punta. Per creare un *link simbolico* come nell'esempio sopra riportato basta dare il seguente comando:

```
%     ln -s /usr/cern/pro/bin/paw     paw
```

Volendo questa trattazione rivolgersi solo a principianti accenneremo soltanto che si possono incontrare anche altri tipi di file quali quelli indicati con le lettere *c*, *b* ecc. denominati rispettivamente file di *caratteri,* file di *blocchi* ecc.

Le protezioni dei file. Finora, della stringa di dieci caratteri iniziali associati a ogni file, abbiamo parlato solo del primo. Le altre tre terne di caratteri stanno a indicare se i file in questione sono ***r***eadable, ***w***riteable o e***x***ecutable rispettivamente dal proprietario (***u****ser)* oppure dagli utenti del gruppo (***g****roup*) e infine se lo sono da chiunque altro (***o****thers)***.** In caso negativo al posto delle lettere troveremo un trattino. Riprendiamo come esempio il programma *paw*:

```
lrwxr-xr-x 1 mastro system 25 Apr 11 11:08 paw ->/usr/cern/pro/bin/paw
```

possiamo notare che per lo *user* è impostata la stringa ***rwx*** ovvero che sono consentiti la lettura la scrittura e l'eseguibilità del file mentre sia per gli utenti dello stesso gruppo che per tutti gli altri è impostata la stringa ***r-x*** ovvero a questi è consentito di leggere, di eseguire ma non di modificare e/o cancellare il file.

Il comando chmod. Qualora si vogliano cambiare i permessi di un singolo file occorre usare il comando *chmod*. Per usare questo comando bisogna indicare:

a chi si riferisce la protezione da impostare:

u	al proprietario del file	(user)
g	agli utenti del gruppo	(group)
o	a tutti gli altri utenti	(others)
a	a tutti	(all)

se togliere o mettere una protezione:

+	per assegnare un permesso
-	per togliere un permesso
=	per assegnarlo a uno e toglierlo agli altri

ed infine a quali protezioni ci si sta riferendo:

r	alla lettura
w	alla scrittura

x all'esecuzione

Chiariamo le cose direttamente con un esempio e vediamo come si fa a togliere a tutti, compreso il proprietario il permesso di scrittura sul file *paw:*

```
%    chmod ugo-w paw
```

conseguentemente al comando:

```
%    ls -la paw
```

si avrà il seguente output:

```
lr-xr-xr-x 1 mastro system 25 Apr 11 11:08 paw ->/usr/cern/pro/bin/paw
```

se si vuol dare *accesso pieno a chiunque* lo si può fare con uno dei seguenti comandi:

```
%    chmod ugo+rwx paw
```

oppure

```
%    chmod 777 paw
```

e l'output del comando *ls -la paw* sarà:

```
lrwxrxxrwx 1 mastro system 25 Apr 11 11:08 paw ->/usr/cern/pro/bin/paw
```

Il comando umask. In precedenza abbiamo visto il comando *umask 022* inserito nel file .cshrc che è uno degli script che vengono lanciati dal sistema appena l'utente si collega.

Questo comando permette di specificare quali protezioni devono essere impostate automaticamente dal sistema operativo quando un file viene creato.

Il valore a fianco del comando è in ottale; valori comuni sono ad esempio 002 che da l'accesso al proprietario e agli utenti del gruppo e assegnando i soli permessi di lettura ed esecuzione a gli *others*. Con 022 si da l'accesso al solo proprietario assegnando i soli permessi di lettura ed esecuzione al gruppo e agli others. .

2.6 - Struttura dei file *(od, I-NUMBER, I-LIST, I-NODE)*

Pur non essendo il caso di richiamare i concetti di file e directory in quanto oramai sono noti a chiunque, può essere invece il caso di spendere alcune parole su termini che si sentono nominare da chi lavora in ambiente *Unix*.

Si supponga che al comando *ls -l* corrisponda il seguente output:

```
drwxr-x--x  20 mastro   system      2048 Sep 27 17:02 .
drwxr-xr-x  94 root     system      2048 Mar 26  1997 ..
-rwxr-xr-x   1 mastro   system      1995 Oct  8 12:41 .cshrc
-rwxr-xr-x   1 mastro   system      1892 Mar 18  1997 .login
-rw-r--r--   1 mastro   system     18844 May 23 19:17 README
lrwxr-xr-x   1 mastro   system      2860 Apr 1.1 11:08 paw
-rwxr-xr-x   1 mastro   system   3219065 Jun  9 12:47 pine
drwxr-xr-x   2 mastro   system       512 Nov 10  1995 pride
```

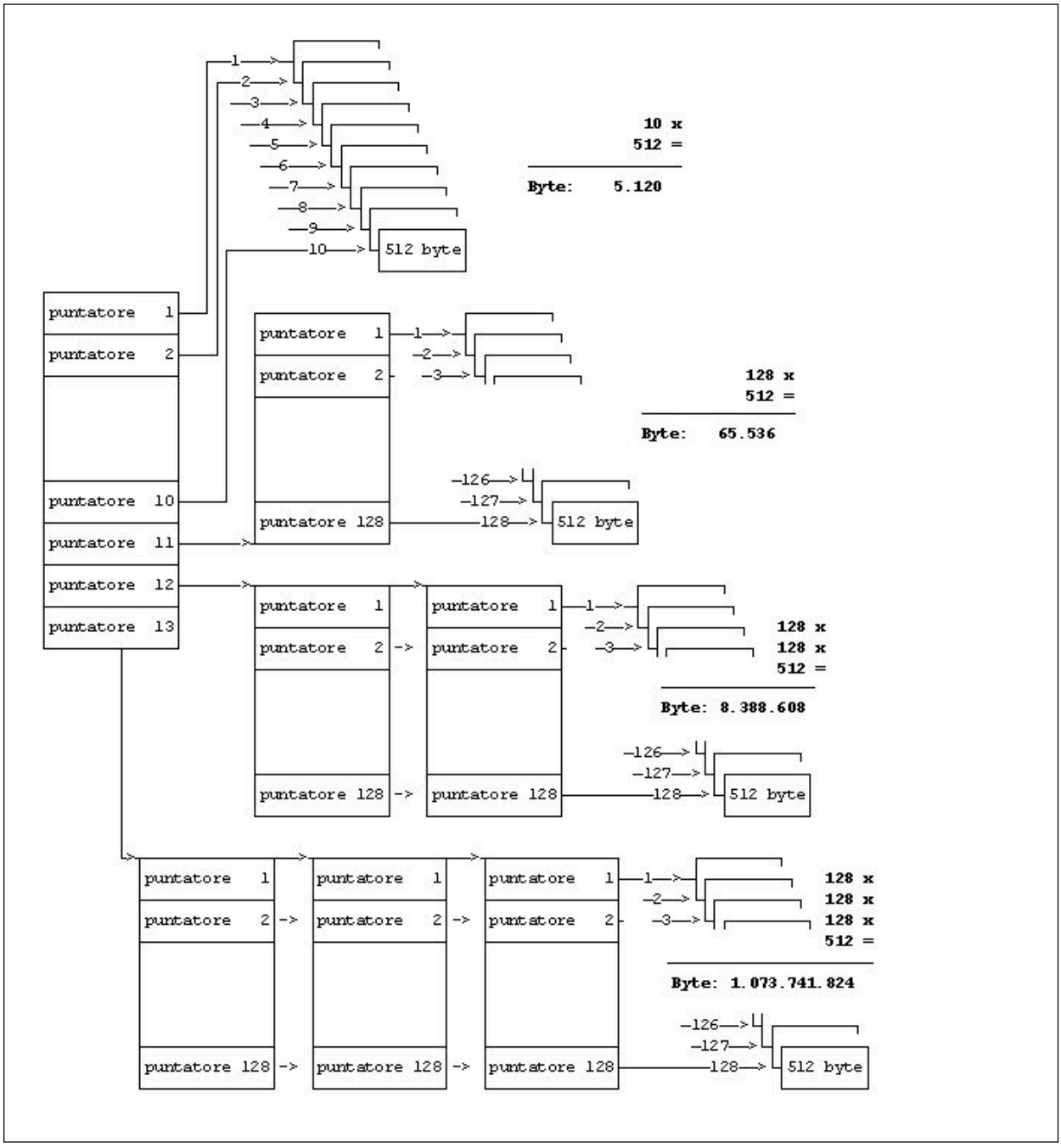
si può notare che tra gli altri file ne compaiono due nominati, rispettivamente `.` e `..` e che corrispondono rispettivamente alla directory corrente e alla directory padre di quella corrente. Se si prova a dare il comando:

```
%    od -bc .
```

dove *od* sta per *octal dump, b* e *c* indicano la richiesta di un output in *ottale* e l'altro in *ASCII* e il punto è il nome del file che contiene le informazioni della directory corrente, si ottiene la seguente uscita:

```
0000000  000 210 002 000 014 000 001 000 056 000 000 000 000 264 000 000
          \0 210 002  \0  \f  \0 001  \0   .  \0  \0  \0  \0 264  \0  \0
0000020  014 000 002 000 056 056 000 000 001 210 002 000 020 000 006 000
          \f  \0 002  \0   .   .  \0  \0 001 210 002  \0 020  \0 006  \0
0000040  056 154 157 147 151 156 000 155 002 210 002 000 020 000 006 000
           .   l   o   g   i   n  \0   m 002 210 002  \0 020  \0 006  \0
0000060  056 143 163 150 162 143 000 155 003 210 002 000 020 000 006 000
           .   c   s   h   r   c  \0   m 003 210 002  \0 020  \0 006  \0
0000100  122 105 101 104 115 105 000 155 004 210 002 000 014 000 003 000
           R   E   A   D   M   E  \0   m 004 210 002  \0  \f  \0 003  \0
0000120  160 141 167 000 005 210 002 000 020 000 004 000 160 151 156 145
           p   a   w  \0 005 210 002  \0 020  \0 004  \0   p   i   n   e
0000140  000 164 151 155 173 262 002 000 020 000 005 000 160 162 151 144
          \0   t   i   m   { 262 002  \0 020  \0 005  \0   p   r   i   d
0000160  145 000 141 164 007 210 002 000 214 001 006 000 160 151 160 160
           e  \0   a   t     210 002  \0 214 001 006  \0   p   i   p   p
0000200  157 061 000 164 000 000 000 000 000 000 000 000 000 000 000 000
           o   1  \0   t  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000220  000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
          \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0001000
```

Senza voler scendere nei dettagli inerenti il significato dei singoli campi numerici, il prospetto mostrato è un esempio di come *Unix* organizza le informazioni riguardando i file contenuti nelle proprie directory. Ad ogni file viene associato un nome ed un numero intero detto *I-NUMBER,* questo fa riferimento ad una tabella di sistema denominata *I-LIST* che a sua volta contiene le informazioni dei singoli file; le informazioni di un singolo file vengono raccolte in un file detto *I-NODE,* in esso vengono indicati il proprietario il gruppo e la stringa delle protezioni; ci sono altri dati quali la dimensione del file, le date di creazione, dell'ultima modifica dell'ultimo accesso, il numero di link e se il file in questione è a sua volta una directory o un file speciale.



- Struttura dei file in ambiente Unix; facendo la somma dei blocchi di 512 byte che si possono indirizzare ovvero 5.120 + 65.536 + 8.388.608 + 1.073.741.824 si ottiene il numero massimo di byte di cui un file può essere composto, ovvero 1.082.201.088.

L'*I-NODE* contiene anche le informazioni che consentono d'individuare la posizione dei file sul disco. In *Unix* un file su disco occupa porzioni costituite da 512 byte non necessariamente contigue, esse sono individuabili da una struttura a puntatori che ci si appresta a descrivere.

Si comincia con un insieme di 13 puntatori di cui i primi 10 contengono l'indirizzo fisico di altrettanti blocchi di 512 byte per un totale di 5.120 byte. Se il file in questione è più grande si ricorre ad un undicesimo puntatore che anziché puntare ad un altro blocco di 512 byte punta ad un insieme di 128 puntatori che a loro volta indicano l'indirizzo di altrettanti blocchi di 512 byte per un totale di 65.536 byte che sommati ai precedenti 5.120 consentono l'allocazione di un file di al massimo 70.656 byte. Il dodicesimo puntatore indicherà un insieme di 128 puntatori ciascuno dei quali punterà ad un altro insieme di 128 puntatori che a loro volta indicheranno l'indirizzo di blocchi di 512 byte. Infine il tredicesimo è un puntatore di puntatori di puntatori di puntatori di blocchi di 512 byte. Facendo un po' di conti si può verificare che questo marchingegno consente l'allocazione di un file la cui dimensione massima è di 1.082.201.088 byte. I moderni sistemi operativi tuttavia consentono in maniera trasparente la possibilità di gestire singoli file di dimensioni maggiori usando tecnologie proprietarie. Nella pagina successiva è mostrata la struttura dei file in ambiente *Unix* appena descritta

La struttura a blocchi dei file in *Unix* comportava rispetto ad altri sistemi operativi contemporaneamente un vantaggio e uno svantaggio: da un lato non era più necessario stabilire a priori la dimensione massima dei file ma dall'altro lato la non contiguità dei blocchi determinava un rallentamento nelle fasi di lettura e scrittura essendo la testina del disco costretta a saltare da un punto all'altro dello stesso. Si ovviò a questa limitazione utilizzando i *buffer* di memoria configurabili da sistema, vale a dire che l'accesso ad un file, sia in lettura che in scrittura, veniva effettuato utilizzando un'area d'appoggio in memoria centrale e scaricando solo periodicamente i dati sul disco, sfruttando il fatto che le operazioni fatte in *ram* sono ovviamente più veloci di quelle fatte sul disco meccanico. Il rovescio di questa medaglia sta nel fatto che se c'è un'improvvisa caduta di tensione, il mancato aggiornamento dei dati sul disco può arrecare danni irreversibili agli stessi. Vengono in soccorso allora comandi quali *sync* che forza lo scarico su disco del buffer in memoria centrale e programmi che cercano di recuperare file *corrotti* a causa di impreviste *cadute di sistema*.

2.7 - Gestione dei file *(mkdir, rmdir, date, du, file, find, which, cd, pwd, cp, mv, rm, touch, wc, head, tail, cmp, diff, dxdiff, pack, unpack, compress, uncompress, uuencode, uudecode, tar, lpr, lpq, lprm)*

I comandi mkdir, rmdir. Per creare una directory si usa il comando *mkdir nome_directory*, quest'ultima avrà i permessi di default che si hanno con il comando *umask* visto in precedenza, tuttavia se nello specifico si vuol dare alla nuova directory un set di permessi diverso si può dare il comando *mkdir -m mode nome_directory* dove *mode* è lo stesso argomento che viene utilizzato con il comando *chmod*. Per esempio, con il comando:

```
%    mkdir –m ugo+rw scratch
```

si crea una directory di nome *scratch* con i permessi di lettura e scrittura per chiunque.

Con il comando *rmdir nome_directory* si cancella una directory di nome nome_directory purché questa sia vuota.

Il comando date. Ai file creati viene associata una data di creazione che si ottiene consultando l'orologio del calcolatore. Il comando per modificare l'orologio di sistema è:

```
%    date aammgghhmm
```

il significato dei campi è evidente. L'utente generico non potrà usare il comando *date* ma potrà farlo solo il gestore del sistema.

L'esigenza di conoscere la data di creazione o di ultima modifica di un file può avere le motivazioni più varie: da essa si può stabilire quando un sistema di acquisizione ha cominciato a ricevere dati, quando un utente si è collegato, quando è arrivata una mail ecc.

Diventa invece per alcuni software una necessità irrinunciabile che un insieme di calcolatori abbia gli orologi sincronizzati quando in base alla data dell'ultima modifica di un file devono decidere un trasferimento di dati o meno da un calcolatore a un altro. E' una tipica problematica che si presenta con i *file system distribuiti* quali ad esempio *afs* (*Andrew File System)* o *dce/dfs (distributed computing environment / distributed file system)*, in questi casi i calcolatori vengono messi in comunicazione con uno o più *nntp-server (Network Time Protocol)*, ovvero dei calcolatori che sono a loro volta collegati con orologi di grande precisione. Il *client*, ovvero il calcolatore che chiede di conoscere l'orario, provvederà a correggere progressivamente con una frequenza prefissata in fase di configurazione il ritardo o l'anticipo riportato dal proprio orologio.

Il comando du. Il comando *du pathname* consente di conoscere l'occupazione dell'area su disco indicata dall'argomento *pathname* comprese tutte le eventuali subdirectory in blocchi di 1024 o di 512 byte qualora l'opzione *k* venga specificata o meno. Tra le opzioni più usate si ricorda la *a* che determina la visualizzazione di ogni singolo file mentre la *s* soltanto il totale.

Il comando file. Il comando *file nome_file* analizza il contenuto del file nome_file e fornisce informazioni sulla natura dello stesso, vale a dire se si tratta di un file di testo o di un eseguibile ecc. A volte per conoscere il contenuto di un file si usa il comando *cat*, tuttavia questo non è consigliabile se non si conosce la natura del file in questione. Dando il comando *cat executable* dove executable è il nome di un file eseguibile, il suo contenuto viene riversato sullo standard output, generalmente lo schermo e i caratteri da visualizzare vengono preventivamente interpretati; fintanto che si tratta di ascii la loro visualizzazione non comporta problemi di sorta tuttavia possono esserci caratteri che se interpretati assolveranno a ben precise funzioni dalle conseguenze anche devastanti, del tutto impreviste e che possono nel migliore dei casi bloccare la sessione di lavoro.

Esplorare i file in maniera controllata con un comando apposito come il comando *file* che risolve questo problema.

Il comando find. Dato un file di nome *myprog*, si supponga di avere una struttura di directory e sottodirectory abbastanza complessa e si vuol conoscere in quale di queste si trova. Un comando che risolve questo problema è *find*. Un possibile suo utilizzo è dato dall'esempio di seguito riportato:

```
%    find . -print | grep myprog
```

in questo caso, a partire dalla directory corrente (`.`) e in maniera ricorsiva verranno scanditi tutti i file esistenti e l'output verrà dato in input al comando *grep* che estrarrà le sole occorrenze della stringa *myprog*.

Il comando which. Sempre per sapere dov'è un file esiste anche il comando *which nome_file*, esso fornisce il percorso completo del file *nome_file* esaminando solo le directory contenute nella variabile *PATH*.

I comandi cd e pwd. I comandi *cd* e *pwd* rispettivamente consentono di spostarsi da una directory all'altra e di conoscere in quale directory ci si trova.

I comandi cp, rm e mv. Il comando *cp* viene utilizzato per copiare uno o più file. La sua forma più semplice è la seguente:

```
%    cp file_origine file_destinazione
```

Il seguente comando:

```
%    mv file_origine file_destinazione
```

serve sia per spostare un file da una directory all'altra che per cambiargli nome.

Il seguente comando:

```
%    rm nome_file
```

serve per cancellare un file. Anche questi comandi hanno innumerevoli opzioni che il lettore interessato potrà esaminare con il comando *man*.

Le difficoltà che s'incontrano quando si lavora al computer sono innumerevoli. Un problema che si presenta abbastanza spesso al principiante può essere il seguente. Si supponga di aver creato *involontariamente* un file con un nome formato di caratteri proibiti, come ad esempio il trattino. Si supponga che detto file si chiami *-prova* e che lo si voglia cancellare; è noto che il trattino viene usato nei comandi per indicare le opzioni per cui è chiaro che al comando:

```
%    rm -prova
```

il calcolatore risponderà picche ovvero mostrerà il messaggio:

```
Unknown option -p
Usage: rm [-firRv] file ...
```

come si fa allora a cancellare un file il cui nome contiene uno o più caratteri speciali ? Una possibilità, ma non è l'unica, è ad esempio offerta dal comando:

```
%    rm -i *
```

l'opzione *-i* attiva una procedura interattiva in cui è il calcolatore stesso a leggere e visualizzare uno per uno tutti i file contenuti nella directory corrente chiedendo se li si vuole cancellare. A questo punto basterà rispondere con la lettera *y* nel solo caso in cui si presenterà il file *-prova* e con la lettera n in tutti gli altri casi e il problema sarà risolto. Un altro espediente può essere il comando:

```
%    rm *prova
```

ma è chiaro che in questo caso bisognerà fare attenzione a che nessun file abbia il nome che termini con la stringa prova.

Il comando touch. Se non esiste un file di nome *my_file,* con il comando *touch my_file* ne verrà creato uno nuovo e vuoto, se il file my*_file* esiste ne verrà aggiornata la data di ultima modifica. E' molto utile nel caso di alcuni software che

necessitano della presenza di un file anche se vuoto e che falliscono in caso di mancanza dello stesso.

Il comando wc. Con il comando *wc nome_file* vengono visualizzati del file *nome_file* il numero di righe, il numero di parole e il numero di byte.

I comandi head e tail. I comandi *head nome_file* e *tail nome_file* consentono di visualizzare rispettivamente le prime dieci e le ultime dieci righe del file *nome_file*. Può essere comodo quando il file da esaminare è molto grande e si può anche cambiare il default di dieci con un'opzione opportuna. Il comando:

```
%    tail -f nome_file
```

può essere molto utile per monitorare la crescita di file, esso verifica ogni due secondi se il file in oggetto è cresciuto e ne visualizza le righe nuove.

I comandi cmp, diff e dxdiff. Può tornare utile conoscere la differenza tra due file e per questo arriva in aiuto il comando *cmp*; esistono comunque comandi più complessi quali *diff, diffh, bdiff* e *vdiff* che confrontano anche gruppi di file in una directory con i file dello stesso nome di un'altra directory e che generano sullo *standard output* le differenze riscontrate.

In un'epoca in cui tutto è in formato grafico non poteva mancare un comando che in maniera agevole e immediata mostrasse la differenza tra due file, detto comando si chiama *dxdiff*.

I comandi (un)pack, (un)compress. Altra tematica importante legata alla gestione dei file è il loro trasporto da un computer all'altro. Quando lo spostamento di un file utilizza mezzi trasmissivi lenti e costosi, quale può essere ad esempio una linea telefonica, sono molto apprezzati software che *comprimono* i file riducendone le dimensioni. Facciamo un esempio esplicativo per chi non ha mai sentito parlare di *compressione dei file*. Si supponga di avere un file dati che rappresenta l'immagine di un quadrato nero su fondo bianco; è evidente che si tratta di un file costituito semplicemente da varie successioni di molti zeri e di molti uni; un file del genere, oltre che rappresentato da una banale ma molto lunga elencazione di byte può essere rappresentato con una semplice formula. Il risultato che si ottiene è un file più piccolo dell'originale per il quale occorrerà meno tempo per un eventuale trasferimento da un calcolatore ad un altro; la compressione di un file torna utile anche per risolvere momentanei problemi di spazio disco.

Sono state ideate varie tecniche per comprimere i file e in ambiente *Unix* sono abbastanza noti programmi quali *pack* e *unpack, compress* e *uncompress* ma in giro se ne vedono molti altri ancora e che non stiamo qui a citare. Il dato statistico dice che i file dati, quali ad esempio i sorgenti dei programmi, si riescono a comprimere fino ad un quarto delle dimensioni iniziali mentre gli eseguibili fino alla metà.

I comandi uuencode e uudecode. Molte delle tecniche citate e delle relative problematiche vengono sempre di più coperte e rese trasparenti all'utente dai moderni sistemi operativi. Un altro problema che si presenta è quando si vuol trasferire un file eseguibile via posta elettronica da un computer all'altro. Oggi chi ha software grafico sofisticato può farlo senza neanche accorgersi di cosa stia succedendo, ma c'è ancora una fetta di utenza che deve porsi il problema che un file eseguibile, potendo avere al suo interno un qualunque tipo di carattere, non può essere spedito così facilmente da un punto ad un altro senza incontrare problemi. Uno dei possibili problemi è che il file giunga a destinazione non per intero ma in parte, e questo può succedere se il file contiene caratteri di controllo che alcuni software interpretano come *carattere di fine file*.

In soccorso ai problemi appena esposti vengono incontro i comandi *uuencode* e *uudecode*, che rispettivamente codificano e decodificano un file assegnato. Detti programmi trasformano il file originale in un altro file costituito di soli caratteri *stampabili*. Un file siffatto può essere spedito tranquillamente via posta elettronica da un calcolatore all'altro senza subire danni. Tuttavia questa trasformazione causa un incremento medio del 35% delle dimensioni del file originale per cui quest'ultimo andrebbe prima compresso e poi codificato. Il file così ottenuto è bello e pronto per essere spedito al destinatario che per poterlo leggere dovrà prima decodificarlo, poi scomprimerlo e poi utilizzarlo.

Quando gli utilizzatori erano solo gli esperti della materia, tutti questi passaggi non costituivano un problema, ma oggigiorno, con la diffusione dei computer anche in ambienti non costituiti di esperti, il fatto di dover effettuare tutte queste operazioni non risulta molto gradito, conseguentemente si è reso necessario, come citato precedentemente, lo sviluppo di programmi che per la loro facilità d'uso agevolano sempre di più il nostro utente finale sgravandolo da inutili e tortuose manovre che esulano dalle sue competenze.

Il comando tar. In ambiente *Unix*, è possibile raggruppare un insieme di file, directory e sottodirectory in un unico file in un formato classico detto *tar*. L'omonimo comando sta per *tape archiver*, esso consente la registrazione su nastro e su disco del file prodotto. La sintassi di questo comando è la seguente:

```
%    tar -ctxvf archivefile.tar file ...
```

il segno meno non è obbligatorio, le opzioni indicate sono solo alcune tra tutte quelle possibili ed hanno i seguenti significati:

- c crea il tar file (Create);
- t mostra il contenuto del tar file (Table);
- x estrae dal tar file (eXtract);
- v mostra le operazioni sul tar file (Verbose)

- f segue il nome del file tar (File);

archivefile.tar è il file che verrà creato e *file ...* sono l'insieme dei file che si vuole raggruppare in *archivefile.tar*; passiamo ora alla spiegazione del comando direttamente con un esempio.

Si supponga che tutti i file della directory corrente e indicati quindi con l'asterisco debbano essere fatti confluire in un unico file in formato *tar* con il nome *dati.tar*; si tenga conto che l'estensione *.tar* aggiunta al nome del file si usa per convenzione. In questo caso il comando da eseguire sarà:

```
%    tar cvf dati.tar *
```

Per visualizzare il contenuto di un file in formato tar bisognerà dare il seguente comando:

```
%    tar tvf dati.tar
```

Per poterne estrarre il contenuto ed esploderlo nella directory corrente bisognerà dare invece il comando:

```
%    tar xvf dati.tar
```

Affinché il lettore si renda conto con quanta ***ineguagliabile cripticità*** sia stato scritto il sistema operativo *Unix* e tutti i suoi comandi annessi e connessi valga per sempre il chiarimento di seguito riportato.

Il lettore inesperto, se da un lato avrà compreso il significato delle opzioni *c, t, x* e *v,* dall'altro lato probabilmente non avrà completamente chiaro in mente il significato dell'opzione *f,* conseguentemente ci si appresta a spiegarne meglio il significato.

L'opzione *f* serve a indicare che la parola successiva indicherà *il nome del tar file*, ovvero su quale file andranno a confluire *i file di origine*. In altre parole sia l'opzione *f* che il file di destinazione *dati.tar* devono essere citati insieme. Si supponga di scrivere il comando:

```
%    tar cv *
```

ovvero senza avergli dato né l'opzione *f* né il nome del *tar file* dati.tar. In questo caso, non essendo stato citato il nome del *tar file,* il sistema operativo dovrà utilizzare il *file di default*; in *Unix* il file di default del comando tar è *l'unità a nastro* ed ha il seguente nome:

```
/dev/rmt0h
```

lo zero indica la prima unità a nastro installata, se si ha una seconda unità bisognerà indicarla con */dev/rmt**1**h*; a questo punto, con il comando *tar cv ** il nostro utente salverà su nastro i propri dati. Per leggere e caricare un file in formato tar da una unità a nastro basterà usare il comando:

```
%     tar xv
```

dopodiché ci si troverà nella directory corrente il contenuto del nastro. Qualora si abbiano ***due*** unità a nastro installate e si voglia copiare dal nastro posto nella seconda unità, un comando che può venire in mente di digitare è:

```
%     tar xv /dev/rmt1h          <sbagliato!>
```

ma è sbagliato. Per indicare qualunque posizione del tar file diversa dalla *posizione di default* ovvero diversa da */dev/rmt0h,* oltre a scrivere specificatamente il nome del *tar file* bisogna indicare anche l'opzione *f* così come riportato in quest'esempio:

```
%     tar xvf /dev/rmt1h         <corretto!>
```

Da come si è visto, in *Unix* un canale di ingresso o di uscita come l'unità a nastro è trattato a tutti gli effetti come *un file!* La stessa cosa la si è potuta vedere quando si è parlato dello standard input, dello standard output e dello standard error.

I comandi lpr, lpq ed lprm. Sono svariati i comandi che al momento di studiarli sembra non debbano mai originare problemi ma che poi all'atto pratico causano notevoli perdite di tempo. Le operazioni di stampa, per motivi direttamente o indirettamente collegati ai comandi relativi, possono presentare non poche difficoltà. E' compito del gestore del sistema configurare le stampanti e assegnare loro un nome; dette stampanti, a seconda di quanto siano più o meno sofisticate, possono essere configurate in modi diversi. Il comando più semplice per stampare un file in *Unix* è il seguente:

```
%     lpr nome_file
```

Il sistema operativo *Unix* prevede per il comando *lpr* una quantità innumerevole di opzioni; in questo caso vengono utilizzate tutte le impostazioni di default, ivi compresa la stampante. Qualora si voglia usare una stampante specifica diversa da quella di sistema va usata l'opzione *P* seguita dal nome della stampante:

```
%     lpr -Pnome_stampante nome_file
```

si può chiedere, qualora la stampante lo preveda, che la stampa avvenga su ambo i lati del foglio, si può scegliere tra l'orientamento verticale oppure orizzontale, si possono avere più copie di stampa di un singolo file ecc.

Vediamo ora cosa si deve fare per interrompere una stampa. Si supponga che sia in corso la stampa di un file di nome *manpage* su una stampante di nome *lntex;* l'utente dovrà individuarne il *numero del job* con il seguente comando:

```
%    lpq -Plntex
```

esso produrrà un output simile a quello sotto riportato contenente il nome della stampa in corso più quelle eventualmente in attesa;

```
lntex is ready and printing
Rank      Owner     Job  Files     Total Size
active    mastro    108  manpage   123456
1st       plr       109  sample1     1261
2st       plr       110  README     71429
```

da questo si evince facilmente che il numero del job è il 108, a questo punto basterà dare il comando:

```
%    lprm -Plntex 108
```

e la stampa verrà interrotta, il sistema operativo passerà al job successivo, nel nostro caso inizierà la stampa del file *sample1*.

2.8 - Esecuzione di programmi *(ctrl-C, ctrl-Z, &, jobs, ps, kill , nice, at)*

Si supponga di aver scritto un programma in *C*, e di averlo compilato ad esempio con il comando:

```
%    cc prog.c -o prog
```

se la compilazione è andata a buon fine il programma eseguibile *prog* conterrà nella stringa dei permessi la lettera *x;* per lanciarlo basterà semplicemente scrivere:

```
%    prog
```

e a questo punto il comincerà a girare e non restituirà il *prompt* finchè l'elaborazione non sarà finita.

Qualora l'utente voglia bloccare l'esecuzione del programma potrà digitare il carattere di controllo *ctrl-C*. Chi proviene dall'ambiente VMS è abituato, per fare la stessa cosa, a usare il carattere *ctrl-Z*; questo carattere in *Unix* assolve ad un'altra funzione, che è la sospensione momentanea e non definitiva del programma. Ritorneremo sull'argomento tra un momento.

I comandi bg e fg. In *Unix* è possibile lanciare dei programmi facendosi restituire il *prompt* e poter così digitare anche altri comandi senza che il programma lanciato smetta di girare. E' possibile anche scollegarsi e ricollegarsi dopo qualche tempo e controllare che lo stesso stia effettuando le operazioni programmate. Un programma lanciato in questa modalità viene detto *background job*, e per farlo basta dare il comando seguito dal simbolo *&* come di seguito riportato:

```
%    prog &
```

un programma lanciato invece con la modalità inizialmente descritta viene detto *foreground job*. Per passare dalla modalità foreground alla modalità background bisogna *sospendere* il job lanciato in foreground con il carattere di controllo *ctrl-Z* dopodiché si dovrà digitare il comando *bg*, rivediamo il tutto con un esempio:

```
%    prog
ctrl-Z
Stopped
%    bg
[1]  prog
%
```

per far tornare un job in foreground bisognerà utilizzare il comando *jobs* per visualizzare tutti i job in background ed individuarne il numero identificativo, quindi si dovrà dare il comando *fg %numero_del_job*; rivediamo di nuovo il tutto con un esempio:

```
%    prog &
%    jobs
[1]  Running   prog
%    fg %1
prog
```

Qualora si abbia involontariamente sospeso dei programmi con il carattere di controllo *ctrl-Z*, cosa che succede spesso agli utenti che hanno esperienza con il sistema operativo *VMS*, e poi ci si voglia scollegare ad esempio con il comando *logout*, il sistema non si scollegherà subito ma mostrerà il seguente messaggio:

```
%    logout
there are stopped jobs
```

l'avvertimento servirà per non dimenticare di chiudere correttamente i job sospesi.

I comandi ps e kill. Per individuare e poi invece interrompere definitivamente un programma in esecuzione si utilizzano i comandi *ps* e *kill*. Il comando *ps* che sta

per *process status* pone sullo standard output l'elenco dei processi in corso; anch'esso dispone di innumerevoli opzioni, tuttavia, digitato semplicemente fornisce un prospetto simile al seguente:

```
%     ps
PID   TTY      S           TIME COMMAND
571   console I  +     0:00.05 /usr/sbin/getty console vt100
15190 lat/629 I  +     0:00.34 login rossi
24879 ttyp4    S       0:00.22 mypr
24912 ttyp4    R  +     0:00.02 ps
%
```

Tra le varie informazioni che vengono fornite risultano necessarie nel nostro caso il nome del processo, ad esempio *mypr* ed il suo numero identificativo detto *PID* o process identifier: A questo punto, per interrompere il processo *mypr* si dovrà dare il comando:

```
%     kill -9 24879
```

dove 24879 è il PID di *mypr* e `-9` è il *signal_number* che viene trasferito al comando kill per indicare che si tratta di un processo da *uccidere*. Facendo *man kill* si potranno leggere tutte le possibilità offerte dal comando kill.

Il comando nice. Si supponga ora di dover lanciare un programma che faccia calcolo intensivo sfruttando appieno la cpu e che rallenti quindi i tempi di risposta di altri programmi; può nascere l'esigenza di *abbassargli la priorità* ovvero di fargli utilizzare meno risorse. In *Unix* questo lo si fa con il comando *nice*. In genere la compilazione di un programma o di un insieme di programmi utilizza massicciamente le risorse del computer, conseguentemente è buona norma compilare i propri programmi con priorità basse. Un possibile comando è il seguente:

```
%     nice  cc  -c  *.c  &
```

La priorità di lavoro standard è fissata con un parametro il cui valore è 0, la priorità più bassa ha come valore 19, priorità elevate hanno valori che vanno da 0 a -20; solo il gestore di sistema può attribuire quest'ultime. Per compilare ad esempio tutti i programmi in C all'interno della propria directory, con priorità bassa, diciamo 15, e in background basta dare il comando:

```
%     nice -n 15  cc  -c  *.c  &
```

Si supponga di voler controllare le priorità dei propri programmi, il comando per ottenere questo è il seguente:

```
%     ps -l
```

e l'output sarà:

```
      F S      UID   PID  PPID %CPU PRI  NI  RSS WCHAN    TTY          TIME COMMAND
80808001 I  +     0   571     1  0.0  42   0  40K ttyin    console      0:00.05 getty
80808001 I  +     0 15190     1  0.0  44   0 736K tty      lat/629      0:00.34 login
80808001 S        0 24879 24849  0.0  44  15 392K pause    ttyp4        0:00.23 mypr
80808001 R  +     0 24893 24879  0.0  44   0 176K -        ttyp4        0:00.01 ps
```

Si osservi la colonna indicata con NI che sta per nice, si può notare che il programma myprog ha la priorità 15 così come assegnato precedentemente mentre tutti gli altri hanno la priorità 0.

Il comando at. Si supponga che un programma di nome *mypr* debba girare per parecchie ore, che richieda molte risorse di sistema e che non lo si voglia far girare durante il normale orario di lavoro, momenti in cui il calcolatore serve a fare altre cose, ma ad esempio, durante il fine settimana. Una risposta a questa esigenza è data dal comando *at* che consente di lanciare un programma ad una certa ora di un determinato giorno:

```
%     echo  mypr | at 23:00 pm January 24
```

se si vuole lanciare il programma alle ore 20 dello stesso giorno si può dare il comando:

```
%     echo  mypr | at 20:00
```