Esercizi pratici

L'uso di un sistema Unix attuale, attraverso l'interfaccia grafica consueta, può essere relativamente facile; tuttavia, per apprendere le basi del funzionamento di un sistema Unix è necessario conoscere l'uso dei comandi, impartiti da tastiera, secondo la modalità tradizionale.

Gli esempi di questo capitolo sono mostrati in modo da essere abbastanza vicini all'interazione che avviene effettivamente tra l'utente e il sistema operativo.

Gli esercizi proposti si riferiscono a un sistema GNU/Linux con caratteristiche abbastanza comuni, ma va tenuto conto che il sistema nel quale si tentano di svolgere potrebbe avere delle differenze o potrebbe non consentire di svolgere certe operazioni. In particolare, ci sono esercizi che richiedono di disporre dei privilegi dell'utente amministratore. A questo proposito, a margine si usa un'icona che rappresenta un dollaro ($) quando si tratta di operazioni che si possono svolgere in qualità di utente comune, oppure un cancelletto (#) quando occorrono i privilegi amministrativi. Inoltre, ci sono esercizi che si riferiscono espressamente a un kernel Linux ed esercizi che presumono di interagire direttamente con la console del sistema. In tutti questi casi appare un'icona appropriata a margine.

Se non si può disporre di un elaboratore con un sistema Unix è sempre possibile registrare un'utenza presso un servizio remoto che consenta l'accesso tramite il protocollo SSH; per esempio uno di quelli seguenti. Naturalmente, in questo caso, non si possono avere i privilegi amministrativi e gli esercizi contrassegnati con il cancelletto (#) non si possono svolgere.

- *SDF Public Access UNIX System* http://www.freeshell.org/

- *Elitter.net: Free UNIX Shell accounts!* http://elitter.net/
- *CJB Management, Inc.* http://www.cjb.net/shell.html

A titolo di esempio, viene mostrato un video con il procedimento per registrare un'utenza presso *http://www.cjb.net* . Come spesso accade in questi casi, per ottenere un accesso occorre disporre di una casella di posta elettronica, nella quale si riceve un codice di attivazione: *http://www.youtube.com/watch?v=ExO0FDvq9KI* . Successivamente alla registrazione si può accedere al servizio attraverso il protocollo SSH; per questo ci si può avvalere del programma PuTTY, disponibile per MS-Windows e altre piattaforme (*http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html*), come già appare alla fine del video.

4.1 Console virtuali

«

Un sistema GNU/Linux, installato in modo normale, consente l'utilizzo di diverse console virtuali (di solito sono sei) a cui si accede con la combinazione [*Alt Fn*] (dove ***n*** è un numero da uno a sei).[1]

[*Alt F2*]

In questo modo si passa alla seconda console virtuale e su questa si può eseguire un'attività differente. Le attività svolte nelle varie console virtuali sono indipendenti, come se avvenissero attraverso terminali fisicamente distinti.

[*Alt F1*]

4.2 Accesso al sistema e conclusione dell'attività

Per utilizzare il sistema occorre accedere attraverso un processo di identificazione. Per poter essere identificati e accettati occorre essere stati registrati in un'utenza, rappresentata in pratica da un nominativo-utente e da una parola d'ordine. È importante rammentare che l'uso di lettere maiuscole o minuscole non è equivalente. Nell'esempio proposto si suppone di accedere utilizzando il nominativo '**tizio**', scritto così, con tutte le lettere minuscole, e la parola d'ordine '**tazza**'.

Si comincia dall'inserimento del nominativo, **volontariamente errato**.

```
login: tizia[Invio]
```

Anche se il nominativo indicato non esiste, viene richiesto ugualmente l'inserimento della parola d'ordine (o almeno così dovrebbe essere). Si tratta di una misura di sicurezza, per non dare informazioni sull'esistenza o meno di un nominativo-utente determinato.

```
Password: tazza[Invio]

  Login incorrect
```

Naturalmente, l'inserimento della parola '**tazza**', in qualità di parola d'ordine, avviene alla cieca, nel senso che non appare come sembrerebbe dall'esempio. Ciò serve a evitare che un vicino indiscreto possa in seguito utilizzare tale informazione per scopi spiacevoli.

Se si sbaglia qualcosa nella fase di accesso (*login*), si deve ricominciare. Questa volta si suppone di eseguire l'operazione in modo

corretto.

```
login: tizio[Invio]

Password: tazza[Invio]

  Last login: Sun Nov 11 10:45:11 on tty1
```

Generalmente, dopo avere superato correttamente la procedura di accesso, si ottiene l'informazione sull'ultima volta che quell'utente ha fatto un accesso. Ciò permette di verificare in maniera molto semplice che nessuno abbia utilizzato il sistema accedendo con il proprio nominativo.

Successivamente si ottiene l'invito della shell (il *prompt*) che sta a indicare la sua disponibilità a ricevere dei comandi.

```
$
```

L'invito, rappresentato in questo esempio da un simbolo dollaro, può essere più o meno raffinato, con l'indicazione di informazioni ritenute importanti dall'utente. Infatti si tratta di qualcosa che ogni utente può configurare come vuole, ma ciò va oltre lo scopo di queste esercitazioni.

Spesso, per tradizione, l'invito termina con un simbolo che cambia in funzione del livello di importanza dell'utente: se si tratta di '**root**' si usa il cancelletto (ovvero '**#**'), altrimenti il dollaro, come in questo esempio.

Video: *http://www.youtube.com/watch?v=256e0rMV5qE*

4.2.1 Cambiamento di identità

Quando la stessa persona dispone di più di un'utenza, può essere opportuno, o necessario, agire sotto una diversa identità rispetto a quella con cui si accede attualmente. Questa è la situazione tipica in cui si trova l'amministratore di un sistema: per le operazioni diverse dall'amministrazione vera e propria dovrebbe accedere in qualità di utente comune, mentre negli altri casi deve utilizzare i privilegi riservati all'utente '**root**'.

Ci sono due modi fondamentali: concludere la sessione di lavoro e accedere con un altro nominativo-utente, oppure utilizzare il comando '**su**' cambiando temporaneamente i propri privilegi.

```
$ su caio [Invio]

Password: ciao [Invio]
```

Se la parola d'ordine è corretta si ottengono i privilegi e l'identità dell'utente indicato, altrimenti tutto resta come prima.

Video: *http://www.youtube.com/watch?v=RiOuYRAdVv0*

4.2.2 Chi sono?

Quando la stessa persona può accedere utilizzando diversi nominativi-utente, potrebbe essere necessario controllare con quale identità sta operando. Negli esempi che seguono si suppone che sia riuscita l'esecuzione del comando '**su caio**' mostrato in precedenza.

```
$ whoami [Invio]

  caio
```

Il comando '**whoami**' («chi sono») permette di conoscere con quale identità si sta operando.

```
$ logname [Invio]

  tizio
```

Il comando '**logname**' permette di conoscere con quale identità ci si è presentati inizialmente, nel momento dell'accesso.

Video: *http://www.youtube.com/watch?v=3ZKtVZh8jik*

4.2.3 Terminare una sessione di lavoro

«

Per terminare una sessione di lavoro è sufficiente concludere l'attività della shell, ovvero di quel programma che mostra l'invito.

Se la situazione è quella degli esempi precedenti, si stava operando come utente '**caio**' dopo un comando '**su**', mentre prima di questo si stava usando l'identità dell'utente '**tizio**'.

```
$ whoami [Invio]

  caio

$ exit [Invio]
```

In tal caso, il comando '**exit**' appena eseguito fa tornare semplicemente alla situazione precedente all'esecuzione di '**su**'

```
$ whoami [Invio]

  tizio
```

Il comando '**exit**' che chiude l'ultima shell, termina l'accesso al sistema.

```
$ exit [Invio]

login:
```

Si ripresenta la richiesta di identificazione della procedura di accesso.

Video: *http://www.youtube.com/watch?v=n5XSu2Fc1Wc*

4.2.4 Spegnimento

Lo spegnimento dell'elaboratore può avvenire solo dopo che il sistema è stato fermato, generalmente attraverso il comando '**shutdown**' che però è accessibile solo all'utente '**root**'.

```
login: root [Invio]

Password: ameba [Invio]

# shutdown -h now [Invio]

  System is going down NOW!!
  …
```

Inizia la procedura di arresto del sistema, che si occupa di eliminare gradualmente tutti i servizi attivi. In un sistema GNU/Linux, alla fine viene visualizzato un messaggio simile a quello seguente:

```
  Power down
```

Quando questo appare, ammesso che l'elaboratore non si sia già spento da solo, è possibile spegnerlo manualmente o riavviarlo.

Se si vuole utilizzare '**shutdown**' attraverso il comando '**su**' in modo da non dovere uscire e rifare un *login*, è possibile agire come di seguito.

```
$ su [Invio]
```

Quando si utilizza il comando '**su**' senza argomenti si indica implicitamente che si vuole ottenere l'identità dell'utente '**root**'.

```
Password: ameba [Invio]

# shutdown -h now [Invio]
```

4.2.5 Diventare temporaneamente amministratore

«

Il comando '**su**', appena mostrato, quando viene usato per acquisire i privilegi dell'utente '**root**', mette a disposizione solo un ambiente limitato (si vuole fare riferimento alle variabili di ambiente e al loro contenuto). Per esempio, se l'utente '**caio**' diventa temporaneamente '**root**', il contenuto della variabile di ambiente ***PATH*** cambia in modo strano:

```
$ whoami [Invio]

  caio

$ echo $PATH [Invio]

  /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:↩
 ↪/opt/prova/bin:/bin:./bin

$ su [Invio]

Password: ameba [Invio]

# whoami [Invio]

  root

# echo $PATH [Invio]
```

```
/sbin:/bin:/usr/sbin:/usr/bin:/usr/bin/X11:/usr/local/sbin:/usr/local/bin
```

Come si può osservare nell'esempio, nel percorso che si ottiene in qualità di utente '**caio**' esistono anche le directory '`/usr/games/`', '`/opt/prova/bin/`' e '`./bin/`'. Quando si diventa utenti '**root**' in questo modo, si ottengono percorsi aggiuntivi, ma ne spariscono altri, che in questo caso si presume siano accessibili in condizioni normali (con un accesso normale). Per risolvere il problema basta usare '**su**' con un'opzione speciale: '**-**'.

```
# exit [Invio]

$ su - [Invio]

Password: ameba [Invio]

# whoami [Invio]

root

# echo $PATH [Invio]

/sbin:/usr/sbin:/usr/local/sbin:/usr/local/bin:↩
↪/usr/bin:/bin:/usr/bin/X11:/usr/games:↩
↪/opt/prova/bin:/bin:./bin
```

In effetti, la differenza sta nel fatto che, usando '**su**' senza il trattino, si ottiene una shell interattiva normale, mentre con il trattino si ottiene una «shell di *login*». Dal lato pratico, la differenza sta nel fatto che nel secondo caso vengano eseguiti script che nel primo caso sono ignorati, ma quello che conta è sapere che la differenza esiste e dipende dalla configurazione del sistema operativo.

4.2.6 Conclusione

«

Il meccanismo attraverso cui si accede al sistema deve essere chiaro, prima di poter affrontare qualunque altra cosa. Per poter proseguire occorre essere certi che gli esempi visti fino a questo punto siano stati compresi, soprattutto, in seguito non viene più mostrato il modo con cui accedere, terminare una sessione di lavoro o cambiare identità.

È fondamentale tenere bene a mente che l'elaboratore non può essere spento prima di avere completato la procedura di arresto del sistema con '**`shutdown`**'.

In caso di dubbio è meglio ripetere l'esercitazione precedente.

4.3 Gestione delle parole d'ordine

«

La prima regola per una parola d'ordine sicura consiste nel suo aggiornamento frequente. Quando si cambia la parola d'ordine, viene richiesto inizialmente l'inserimento di quella precedente, quindi se ne può inserire una nuova, per due volte, in modo da prevenire eventuali errori di battitura. Di norma non vengono accettate le parole d'ordine troppo semplici (solo l'utente '**`root`**' dovrebbe avere la possibilità di assegnare parole d'ordine banali).

4.3.1 L'utente «root» che cambia la parola d'ordine di un utente comune

«

L'utente '**`root`**' può cambiare la parola d'ordine di un altro utente. Questa è la situazione comune di quando si crea una nuova utenza: è l'utente '**`root`**' che assegna la prima volta la parola d'ordine per quel nuovo utente.

```
# passwd tizio[Invio]
```

Trattandosi dell'utente '**root**' che cambia la parola d'ordine di un altro, viene richiesto semplicemente di inserire quella nuova (l'utente '**root**' non ha la necessità di conoscere la vecchia parola d'ordine di un altro utente).

```
New UNIX password: 123[Invio]
```

La parola d'ordine inserita (che nella realtà non si vede) è troppo breve e anche banale. Il programma potrebbe avvertire di questo, ma non dovrebbe opporsi:

```
  BAD PASSWORD: it's a WAY too short
Retype new UNIX password: 123[Invio]

  passwd: all authentication tokens updated successfully
```

La parola d'ordine è stata cambiata.

4.3.2 L'utente comune che cambia la propria parola d'ordine

L'utente comune può cambiare la propria parola d'ordine, solo la propria, ma di norma non gli è consentito di assegnarsi una parola d'ordine troppo semplice. Nell'esempio, l'utente è '**tizio**'.

```
$ passwd[Invio]
```

Prima di accettare una nuova parola d'ordine, viene richiesta quella vecchia:

```
  Changing password for tizio
(current) UNIX password: 123[Invio]
```

Quindi viene richiesta quella nuova:

```
New UNIX password: albero[Invio]

  BAD PASSWORD: it is based on a (reversed) dictionary word
  passwd: Authentication token manipulation error
```

Come si vede in questo caso, la parola d'ordine '**albero**' viene considerata troppo semplice e il programma si rifiuta di procedere. Si decide allora di usare qualcosa di più complesso, o semplicemente più lungo.

```
$ passwd[Invio]

  Changing password for tizio
(current) UNIX password: 123[Invio]

New UNIX password: fra martino campanaro[Invio]
```

Si è optato per una parola d'ordine lunga. Occorre tenere a mente che conta la differenza tra maiuscole e minuscole e anche il numero esatto di spazi inseriti tra le parole.

```
Retype new UNIX password: fra martino campanaro[Invio]

  passwd: all authentication tokens updated successfully
```

A seconda della configurazione del sistema e dell'aggiornamento delle librerie, può darsi che sia perfettamente inutile utilizzare delle parole d'ordine più lunghe di otto caratteri, nel senso che quanto eccede i primi otto caratteri potrebbe essere semplicemente ignorato. Si può provare a verificarlo; seguendo l'esempio appena visto, potrebbe essere che la parola d'ordine risultante sia solo '**`fra mart`**'.

4.3.3 Conclusione

«

Il cambiamento della parola d'ordine in un sistema Unix deve essere considerato una cosa abituale, anche per gli utenti comuni. Di norma, le parole d'ordine troppo semplici non sono accettabili.

4.4 Navigazione tra le directory

«

I dati contenuti in un file system sono organizzati in modo gerarchico attraverso directory e sottodirectory. Prima di iniziare questa esercitazione è conveniente rivedere la sezione 3.21.

L'utente a cui ci si riferisce negli esempi è '**`tizio`**'.

Video: *http://www.youtube.com/watch?v=iDBtqtqFln8*

4.4.1 Directory corrente

«

Mentre si utilizza il sistema, i comandi che si eseguono risentono generalmente della ***posizione corrente*** in cui ci si trova, ovvero della directory attuale, o attiva. Tecnicamente non c'è bisogno di definire una directory corrente: tutte le posizioni nell'albero del file system

potrebbero essere indicate in maniera precisa. In pratica, però, la presenza di questa directory corrente semplifica molte cose.

```
$ cd /usr/bin[Invio]
```

Eseguendo il comando precedente, la directory attuale dovrebbe divenire '`/usr/bin/`'. Per controllare che ciò sia avvenuto si utilizza il comando seguente:

```
$ pwd[Invio]

  /usr/bin
```

4.4.2 Spostamenti assoluti e relativi

«

$ Il comando '**cd**' può essere utilizzato per cambiare la directory corrente, sia attraverso l'indicazione di un percorso assoluto, sia attraverso un percorso relativo. Il percorso assoluto parte dalla directory radice, mentre quello relativo parte dalla posizione corrente.

```
$ cd /usr/local[Invio]
```

Il comando soprastante cambia la directory corrente in modo che diventi esattamente '`/usr/local/`'. Il percorso indicato è assoluto perché inizia con una barra obliqua che rappresenta la directory radice.

```
$ pwd[Invio]

  /usr/local
```

Quando si utilizza l'indicazione di un percorso che non inizia con una barra obliqua, si fa riferimento a qualcosa che inizia dalla posizione corrente.

```
$ cd bin [Invio]
```

Con questo comando si cambia la directory corrente, passando in ‘`bin/`’ che discende da quella attuale.

```
$ pwd [Invio]

  /usr/local/bin
```

4.4.3 Spostamenti a ritroso

Ogni directory contiene due riferimenti convenzionali a due sotto-directory speciali. Si tratta del riferimento alla directory stessa che lo contiene, rappresentato da un punto singolo (‘`.`’), assieme al riferimento alla directory genitrice, rappresentato da due punti in sequenza (‘`..`’). Questi simboli (il punto singolo e quello doppio) sono nomi di directory a tutti gli effetti.

```
$ cd .. [Invio]
```

Cambia la directory corrente, facendola corrispondere alla genitrice di quella in cui ci si trovava prima; in altri termini si potrebbe definire questa directory come quella che precede la posizione di partenza. Si tratta di un percorso relativo che utilizza, come punto di inizio, la directory corrente del momento in cui si esegue il comando.

```
$ pwd [Invio]

  /usr/local
```

Gli spostamenti relativi che fanno uso di un movimento all’indietro possono essere più elaborati.

```
$ cd ../bin [Invio]
```

In questo caso si intende indietreggiare di una posizione e quindi entrare nella directory ‘`bin/`’.

```
$ pwd[Invio]

  /usr/bin
```

Lo spostamento a ritroso può essere anche cumulato a più livelli.

```
$ cd ../../var/tmp[Invio]
```

In questo caso si indietreggia due volte prima di riprendere un movimento in avanti.

```
$ pwd[Invio]

  /var/tmp
```

Gli spostamenti all’indietro si possono usare anche in modo più strano e apparentemente inutile.

```
$ cd /usr/bin/../local/bin/..[Invio]
```

Indubbiamente si tratta di un’indicazione poco sensata, ma serve a comprendere le possibilità date dall’uso del riferimento alla directory precedente.

```
$ pwd[Invio]

  /usr/local
```

4.4.4 Riferimento preciso alla directory corrente

« $ La directory corrente può essere rappresentata da un punto singolo all’inizio di un percorso.[2] In pratica, tutti i percorsi relativi potrebbero iniziare con il prefisso ‘**./**’ (punto, barra obliqua). Per quanto

riguarda lo spostamento all'interno delle directory, ciò serve a poco, ma ritorna utile in altre situazioni.

```
$ cd ./bin[Invio]
```

A partire dalla directory corrente si sposta nella directory 'bin/'.

```
$ pwd[Invio]

  /usr/local/bin
```

4.4.5 Directory corrente e «.»

La directory corrente non corrisponde esattamente al concetto legato al riferimento '.': il «file» '.' è un cortocircuito che ha ogni directory. Per esempio, il percorso 'uno/due/./tre' è perfettamente equivalente a 'uno/due/tre'. In pratica, quel punto che appare nel primo caso, rappresenta semplicemente uno spostamento nullo nella stessa directory 'uno/due'. Lo si può verificare facilmente:

```
$ cd /usr/./local/./lib[Invio]

$ pwd[Invio]

  /usr/local/lib
```

Quando si indica un percorso relativo, è come inserire implicitamente all'inizio la directory corrente; pertanto, scrivere './uno/due', significa indicare un concetto equivalente a '***directory_corrente***/./uno/due'. In questo senso, **solo** quando il punto si trova all'inizio di un percorso (che quindi risulta essere relativo), il punto rappresenta la directory corrente.

4.4.6 Directory personale, o directory «home»

«

$ Ogni utente ha una directory personale, conosciuta come directory *home*, destinata a contenere tutto ciò che riguarda l'utente a cui appartiene. Usando il comando '**cd**' senza argomenti, si raggiunge la propria directory personale, senza bisogno di indicarla in modo esplicito.

```
$ cd [Invio]

$ pwd [Invio]

  /home/tizio
```

Alcune shell sostituiscono il carattere tilde ('**~**'), all'inizio di un percorso, con la directory personale dell'utente che lo utilizza.

```
$ cd ~ [Invio]

$ pwd [Invio]

  /home/tizio
```

Nello stesso modo, un nominativo-utente preceduto da un carattere tilde, viene sostituito dalla directory personale dell'utente stesso.[3]

```
$ cd ~caio [Invio]

$ pwd [Invio]

  /home/caio
```

Prima di proseguire si ritorna nella propria directory personale.

```
$ cd [Invio]
```

4.4.7 Conclusione

La directory corrente è un punto di riferimento importante per i programmi e il cambiamento di questa posizione avviene attraverso il comando '**cd**'. Per conoscere quale sia la directory corrente si utilizza '**pwd**'. Si può fare riferimento alla directory genitrice di quella attuale con una sequenza di due punti ('`..`'), mentre quella attuale (nell'ambito del contesto in cui ci si trova) si può individuare con un punto singolo ('`.`').

4.5 Contenuti

La navigazione all'interno delle directory, alla cieca, come visto negli esempi dell'esercitazione precedente, è una cosa possibile ma insolita: normalmente si accompagna con l'analisi dei contenuti di directory e file.

4.5.1 Contenuto delle directory

Le directory si esplorano con il comando '**ls**'

```
$ ls /bin[Invio]

arch          dd              gzip      nisdomainname tar
ash           df              hostname  ping          touch
awk           dmesg           kill      ps            true
basename      dnsdomainname   ln        pwd           umount
bash          doexec          login     rm            uname
bsh           domainname      ls        rmdir         vi
cat           echo            mail      rpm           view
chgrp         egrep           mkdir     sed           vim
chmod         ex              mknod     sh            zcat
chown         false           more      sleep
cp            fgrep           mount     sort
```

```
cpio          gawk          mt        stty
csh           grep          mv        su
date          gunzip        netstat   sync
```

Il comando '**ls /bin**' visualizza il contenuto della directory '/bin/'. I nomi che vengono elencati rappresentano file di qualunque tipo (sottodirectory incluse).

Una visualizzazione più espressiva del contenuto delle directory può essere ottenuta utilizzando l'opzione '**-l**'.

```
$ ls -l /bin[Invio]

-rwxr-xr-x  1 root   root      2712 Jul 20 03:15 arch
-rwxrwxrwx  1 root   root     56380 Apr 16  1997 ash
lrwxrwxrwx  1 root   root         4 Oct 21 11:15 awk -> gawk
-rwxr-xr-x  1 root   root     18768 Apr 18  1997 basename
-rwxrwxrwx  1 root   root    412516 Jul 17 21:27 bash
lrwxrwxrwx  1 root   root         3 Oct 21 11:15 bsh -> ash
-rwxr-xr-x  1 root   root     22164 Mar 14  1997 cat
-rwxr-xr-x  1 root   root     23644 Feb 25  1997 chgrp
-rwxr-xr-x  1 root   root     23960 Feb 25  1997 chmod
-rwxr-xr-x  1 root   root     23252 Feb 25  1997 chown
-rwxr-xr-x  1 root   root     61600 Feb 25  1997 cp
-rwxr-xr-x  1 root   root    296728 Apr 23  1997 cpio
…
```

In questo caso, è stato ottenuto un elenco più dettagliato che in particolare consente di distinguere il tipo di file, i permessi e l'appartenenza all'utente e al gruppo.

In precedenza è stato spiegato che ogni directory contiene due riferimenti convenzionali rappresentati da un punto singolo e da due punti in sequenza ('.' e '..'). Negli esempi appena visti, questi non sono apparsi. Ciò accade perché i file il cui nome inizia con un punto non

vengono presi in considerazione quando non si fa riferimento a loro in modo esplicito.

```
$ cd [Invio]
```

```
$ ls [Invio]
```

La directory personale di un utente potrebbe sembrare vuota, utilizzando il comando '**ls**' appena visto. Con l'opzione '**-a**' si visualizzano anche i file che iniziano con un punto (si osservi che in precedenza non sono apparsi i riferimenti alle voci '.' e '..').

```
$ ls -a [Invio]
```

```
.                .bash_profile    .riciclaggio
..               .bashrc          .screenrc
.Xdefaults       .fvwm2rc95       .twmrc
.bash_history    .mc.ext          .xfm
.bash_logout     .mc.ini          .xinitrc
```

Video: *http://www.youtube.com/watch?v=53iL53FccZPNY*

4.5.2 Contenuto dei file

Anche il contenuto dei file può essere analizzato, entro certi limiti, soprattutto quando si tratta di file di testo. Per visualizzare il contenuto di file di testo si utilizzano generalmente i comandi '**cat**' e '**more**'.

```
$ cat /etc/fstab [Invio]
```

```
/dev/hda3  /            ext3         defaults        1  1
/dev/hda2  none         swap         sw
proc       /proc        ignore
/dev/hda1  dos          vfat         quiet,umask=000
/dev/hdc   /mnt/cdrom   iso9660      ro,user,noauto
/dev/fd0   /mnt/floppy  vfat         user,noauto,quiet
```

Con il comando appena indicato si ottiene la visualizzazione del contenuto del file '`/etc/fstab`', che ovviamente cambia a seconda della configurazione del proprio sistema operativo.

Il comando '**`cat`**', usato così, non si presta alla visualizzazione di file di grandi dimensioni. Per questo si preferisce usare '**`more`**', oppure il più raffinato '**`less`**'.

```
$ more /etc/services[Invio]
```

```
# /etc/services:
# $Id: services,v 1.4 1997/05/20 19:41:21 tobias Exp $
#
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a
# single well-known port number for both TCP and UDP; hence,
# most entries here have two entries even if the protocol
# doesn't support UDP operations. Updated from RFC 1700,
# ``Assigned Numbers'' (October 1994).  Not all ports are
# included, only the more common ones.

tcpmux          1/tcp         # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
```

```
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp        # message send protocol
msp             18/udp        # message send protocol
--More--(5%)
```

[*barra-spaziatrice*]

```
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
fsp             21/udp          fspd
ssh             22/tcp          # SSH Remote Login Protocol
ssh             22/udp          # SSH Remote Login Protocol
telnet          23/tcp
# 24 - private
smtp            25/tcp          mail
# 26 - unassigned
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/udp          resource # resource location
nameserver      42/tcp          name     # IEN 116
whois           43/tcp          nicname
re-mail-ck      50/tcp          # Remote Mail Checking Protocol
re-mail-ck      50/udp          # Remote Mail Checking Protocol
domain          53/tcp          nameserver # name-domain server
domain          53/udp          nameserver
mtp             57/tcp                     # deprecated
bootps          67/tcp                     # BOOTP server
bootps          67/udp
--More--(9%)
```

Come mostrato, per passare alla schermata successiva, basta premere la [*barra−spaziatrice*]. Per terminare, anche se non è stato visualizzato tutto il file, basta usare la lettera «q».

[*q*]

Il comando '**less**' funziona in modo analogo, con la differenza che si può scorrere il file anche all'indietro, usando intuitivamente la tastiera.

I comandi '**more**' e '**less**' sono descritti meglio nella sezione 5.1.

Video: *http://www.youtube.com/watch?v=8FTPsd1slSs*

4.5.3 Determinazione del tipo

«

$ La tipologia dei file può essere determinata attraverso il comando '**file**', senza doverne visualizzare il contenuto. Ciò è molto importante, specialmente nelle situazioni in cui visualizzare un file è inopportuno (si pensi a cosa accadrebbe tentando di visualizzare un file eseguibile binario).

Il comando '**file**' si basa su un elenco di stringhe di riconoscimento chiamate ***magic number*** (una sorta di «impronta»), definite in base alla tradizione dei sistemi Unix.

```
$ file /etc/*[Invio]

  /etc/DIR_COLORS:        English text
  /etc/HOSTNAME:          ASCII text
  /etc/X11:               directory
  /etc/adjtime:           ASCII text
  /etc/aliases:           English text
  /etc/aliases.db:        Berkeley DB Hash file …
  /etc/at.deny:           ASCII text
```

```
/etc/bashrc:            ASCII text
/etc/cron.daily:        directory
/etc/cron.hourly:       directory
/etc/cron.monthly:      directory
/etc/cron.weekly:       directory
/etc/crontab:           ASCII text
/etc/csh.cshrc:         ASCII text
/etc/dosemu.conf:       English text
/etc/dosemu.users:      ASCII text
…
```

Il comando indicato come esempio visualizza l'elenco dei file contenuti nella directory '`/etc/`', dove a fianco di ogni file appare la definizione del tipo a cui questo appartiene.

Questo metodo di riconoscimento dei dati non è infallibile, ma è comunque di grande aiuto.

4.5.4 Spazio utilizzato e spazio disponibile

Per controllare lo spazio disponibile nel disco (o nei dischi) si utilizza il comando '**df**'.

```
$ df [Invio]
```

Il risultato del comando potrebbe essere qualcosa di simile a quanto segue.

```
Filesystem   1024-blocks  Used Available Capacity Mounted on
/dev/hda4        648331  521981     92860      85%   /
/dev/hda1         41024   38712      2312      94%   /dos
```

Per controllare lo spazio utilizzato in una directory si può usare il comando '**du**'.

```
$ du /bin [Invio]
```

```
3168    /bin
```

In questo caso, si determina che la directory ‘`/bin/`’ contiene file per un totale di 3 168 Kibyte.

4.5.5 Conclusione

«

L’analisi del contenuto di directory e file è un’operazione elementare, ma essenziale per la determinazione delle azioni da compiere in funzione di quanto si rivela in questo modo.

4.6 Creazione, copia ed eliminazione di file

«

La creazione, la copia e l’eliminazione dei file sono operazioni elementari, ma importanti e delicate. Questa esercitazione deve essere fatta con cura e attenzione.

4.6.1 Creazione di un file

«

$ Esistono vari modi per creare un file. Il modo più semplice per creare un file vuoto è quello di usare il comando ‘**`touch`**’. Prima di tutto ci si sposta nella propria directory personale, che è il luogo più adatto per questo genere di esercizi.

```
$ cd [Invio]

$ touch pippo [Invio]
```

Dopo aver usato il comando ‘**`touch`**’ per creare il file ‘`pippo`’ non si ottiene alcuna conferma dell’avvenuta esecuzione dell’operazione. Questo atteggiamento è tipico dei sistemi Unix i cui comandi tendono a non manifestare il successo delle operazioni eseguite. Si può comunque verificare.

```
$ ls -l pippo [Invio]

-rw-rw-r--   1 tizio    tizio           0 Dec 23 10:49 pippo
```

Il file è stato creato.

In questa fase degli esercizi, in cui non è ancora stato descritto l'uso di un programma per creare o modificare file di testo, è possibile vedere un metodo semplice per creare un file del genere. Si utilizza il comando '**cat**' in un modo un po' strano che viene chiarito più avanti.

```
$ cat > pippo2 [Invio]
```

Da questo momento inizia l'inserimento del testo come nell'esempio mostrato qui di seguito.

```
Esiste anche un modo semplice di scrivere [Invio]

un file di testo. [Invio]

Purtroppo si tratta di una scrittura a senso unico. [Invio]

[Ctrl d]
```

L'inserimento del testo termina con la combinazione [*Ctrl d*].[4]

Si può verificare che il file sia stato creato e contenga il testo digitato.

```
$ cat pippo2 [Invio]

Esiste anche un modo semplice di scrivere
un file di testo.
Purtroppo si tratta di una scrittura a senso unico.
```

Video: *http://www.youtube.com/watch?v=UXI2o22po22BM*

4.6.2 Copia di file

«

$ La copia dei file può essere fatta attraverso l'uso del comando '**cp**'.

```
$ cp pippo2 pippo3 [Invio]
```

Eseguendo il comando appena mostrato, si ottiene la copia del file 'pippo2' per generare il file 'pippo3'. Come al solito, se tutto va bene non si ottiene alcuna segnalazione.

La copia di un gruppo di file può avvenire solo quando la destinazione (l'ultimo nome indicato nella riga di comando) è una directory già esistente.

```
$ cp pippo pippo2 pippo3 /tmp [Invio]
```

Con il comando precedente si copiano i file creati fino a questo punto nella directory '/tmp/'. La stessa cosa si può fare in modo più semplice utilizzando i metacaratteri.

```
$ cp pippo* /tmp [Invio]
```

Video: *http://www.youtube.com/watch?v=ArYg6vRqFVE*

4.6.3 Eliminazione di file

«

$ L'eliminazione dei file avviene normalmente per mezzo di '**rm**'. L'uso di questo comando richiede molta attenzione, specialmente quando si agisce con i privilegi dell'utente '**root**'. Infatti, la cancellazione avviene senza obiezioni e senza bisogno di conferme. Può bastare un errore banale per cancellare tutto ciò a cui si può accedere.

```
$ rm pippo pippo2 [Invio]
```

Il comando appena mostrato elimina **definitivamente** e senza possibilità di recupero i file indicati: ‘`pippo`’ e ‘`pippo2`’.

La cancellazione dei file può avvenire anche indicandone un gruppo attraverso l’uso dei metacaratteri. L’uso di questi simboli rappresenta un rischio in più. Generalmente, quando non si ha ancora una buona preparazione e si può essere incerti sull’effetto di un comando di eliminazione, conviene prima controllare il risultato, per esempio attraverso ‘**`ls`**’.[5]

Volendo cancellare tutti i file il cui nome inizia per ‘**`pippo`**’, si potrebbe utilizzare il modello ‘**`pippo*`**’. Per sicurezza si verifica con ‘**`ls`**’.

```
$ ls pippo* [Invio]

  pippo3
```

Risulta corrispondere al modello solo il file ‘`pippo3`’. Infatti, poco prima sono stati cancellati ‘`pippo`’ e ‘`pippo2`’. In ogni caso, si vede che il modello è corretto e si procede con la cancellazione (tuttavia si deve fare attenzione ugualmente).

```
$ rm pippo* [Invio]
```

L’uso distratto di questo comando di eliminazione, può produrre danni gravi. Si pensi a cosa può accadere se, invece di digitare ‘**`rm pippo*`**’ si inserisse accidentalmente uno spazio tra la parola ‘**`pippo`**’ e l’asterisco. Il comando sarebbe ‘**`rm pippo *`**’ e produrrebbe l’eliminazione del file ‘`pippo`’ (se esiste) e successivamente anche di tutti i file contenuti nella directory corrente (questo è ciò che rappresenta l’asterisco da solo). Come è già stato spiegato, ‘**`rm`**’ non fa domande, così come accade con gli altri comandi, nel rispetto

delle tradizioni Unix: quello che è cancellato è cancellato.

Video: *http://www.youtube.com/watch?v=oOSx5q3qBcY*

4.6.4 Conclusione

«

La creazione di file, normalmente vuoti, la copia e l'eliminazione, sono operazioni elementari ma fondamentali. Nella loro semplicità si tratta comunque di funzionalità che richiedono un po' di attenzione, soprattutto quando si interviene con i privilegi dell'utente '**root**': con la copia si potrebbero sovrascrivere file già esistenti, con la cancellazione si potrebbe intervenire in un ambito diverso da quello previsto o desiderato.

4.7 Creazione, copia ed eliminazione di directory

«

Le directory possono essere viste come contenitori di file e di altre directory. La copia e l'eliminazione di directory ha delle implicazioni differenti rispetto alle stesse operazioni con i file normali. Continua a valere la raccomandazione di svolgere l'esercitazione con cura.

Video: *http://www.youtube.com/watch?v=gqkbjHnrPDw*

4.7.1 Creazione di una directory

«

$ La creazione di una directory è concettualmente simile alla creazione di un file vuoto. Quando la directory viene creata è sempre vuota: si riempie utilizzandola. Una directory viene creata con il comando '**mkdir**'.

Prima di procedere ci si sposta nella propria directory personale e quindi si crea la directory ‘`mia/`’ discendente dalla posizione corrente.

```
$ cd[Invio]

$ mkdir mia[Invio]
```

Si può verificare con il comando ‘**`ls`**’.

```
$ ls -l[Invio]

  …
  drwxr-xr-x 8 tizio  tizio   1024 Dec 23 12:11 mia
  …
```

La lettera ‘**`d`**’ all’inizio della stringa che identifica i permessi indica chiaramente che si tratta di una directory.

4.7.2 Copia di directory

La copia delle directory avviene attraverso il comando ‘**`cp`**’ con le opzioni ‘**`-r`**’ oppure ‘**`-R`**’, tra le quali c’è una differenza sottile che però qui non viene approfondita.

```
$ cp -r mia mia2[Invio]
```

Con il comando appena visto, si ottiene la copia della directory ‘`mia/`’ in ‘`mia2/`’. La copia è ricorsiva, nel senso che comprende tutti i file contenuti nella directory di origine, assieme a tutte le eventuali sottodirectory, compreso il loro contenuto.

4.7.3 Eliminazione di directory

«

$ Normalmente, le directory si possono cancellare quando sono vuote, per mezzo del comando '**rmdir**'.

Valgono le stesse raccomandazioni di prudenza fatte in precedenza in occasione degli esercizi sulla cancellazione di file.

```
$ rmdir mia2 [Invio]
```

Il comando appena mostrato elimina la directory '`mia2/`'.

L'eliminazione delle directory fatta in questo modo, cioè attraverso il comando '**rmdir**', non è molto preoccupante, perché con esso è consentito eliminare solo directory vuote: se ci si accorge di avere eliminato una directory di troppo, si riesce facilmente a ricrearla con il comando '**mkdir**'.

Tuttavia, spesso si eliminano interi rami di directory, quando con un comando si vuole eliminare una o più directory e con esse il loro contenuto di file ed eventuali altre directory. Si dice in questo caso che si esegue una cancellazione ricorsiva.

Prima di proseguire, si prova a creare una struttura articolata di directory.

```
$ mkdir carbonio [Invio]

$ mkdir carbonio/idrogeno [Invio]

$ mkdir carbonio/ossigeno [Invio]

$ mkdir carbonio/idrogeno/elio [Invio]
```

Si dovrebbe ottenere una struttura organizzata nel modo seguente:

```
$ tree carbonio [Invio]
```

```
carbonio
|-- idrogeno
|   `-- elio
`-- ossigeno

3 directories, 0 files
```

Se si tenta di eliminare tutta la struttura che parte da ‘`carbonio/`’ con il comando ‘**`rmdir`**’, si ottiene solo una segnalazione di errore.

```
$ rmdir carbonio[Invio]

rmdir: carbonio: Directory not empty
```

Per questo bisogna utilizzare il comando ‘**`rm`**’ con l’opzione ‘**`-r`**’. Tuttavia, il comando ‘**`rm`**’ applicato in questo modo ricorsivo è **particolarmente pericoloso** se utilizzato in modo distratto.

```
$ rm -r carbonio[Invio]
```

La directory ‘`carbonio/`’ e tutto ciò che da essa discendeva non c’è più.

Si provi a pensare cosa può accadere quando si utilizzano i metacaratteri: si cancellano indifferentemente file e directory che corrispondono al modello. C’è però ancora qualcosa di peggiore: l’insidia dei nomi che iniziano con un punto.

4.7.4 Eliminazione di directory il cui nome inizia con un punto

La cancellazione di directory il cui nome inizia con un punto è un’operazione estremamente delicata che merita una discussione a parte. Generalmente, quando si utilizzano i metacaratteri per identificare un gruppo di nomi di file e directory, questi simboli non corrispon-

dono mai ai nomi che iniziano con un punto. Questa convenzione è stata definita per evitare che con i metacaratteri si possa intervenire involontariamente con i riferimenti standard delle directory: '`.`' (la directory stessa che lo contiene) e '`..`' (la directory genitrice).

A questo fatto si è aggiunta la convenzione di nominare in questo modo (con un punto iniziale) file e directory che rappresentano la configurazione particolare di ogni utente. In tal modo, è come se tali file e directory fossero nascosti, per cui l'utente non risulta infastidito da questi che così non possono nemmeno essere cancellati involontariamente.

Potrebbe sorgere il desiderio di eliminare tutti questi file e tutte queste directory, utilizzando il modello '`.*`' (punto, asterisco), ma in questo modo si includerebbero anche i riferimenti standard: '`.`' e '`..`', eliminando così anche la directory corrente, ma soprattutto quella genitrice (con tutto il suo contenuto).

Se il comando viene dato da un utente comune, questo riesce a eliminare solo i dati a cui può accedere, mentre se lo sbaglio venisse fatto dall'utente '**`root`**', tutto ciò che è stato selezionato erroneamente potrebbe essere perduto.

Si osservi che in un sistema GNU/Linux tipico, la directory personale dell'utente '**root**' è '`/root/`'; pertanto, un comando del genere viene dato presumibilmente proprio quando la directory corrente è la directory personale, che, come si vede, discende immediatamente dalla radice. Pertanto, un utente '**root**' che fa uno sbaglio del genere, potrebbe cancellare tutto il file system principale con il contenuto di tutti gli altri dischi che vi si trovano innestati.

Per concludere, il comando da evitare assolutamente è '**rm -r .***'. **Attenzione a non usarlo mai!**

4.7.5 Conclusione

Quando si copiano e si eliminano le directory, sorge spontaneo il desiderio di intervenire in modo ricorsivo su tutto il contenuto della directory di partenza. I problemi maggiori cui si va incontro sono legati alla cancellazione ricorsiva, specialmente quando si pretende di eliminare i file e le directory il cui nome inizia con un punto, in modo globale, attraverso un modello fatto di metacaratteri.

4.8 Spostamenti e collegamenti di file e directory

Negli ambienti Unix, lo spostamento e il cambiamento del nome di file e directory sono la stessa cosa. Un'altra particolarità dei sistemi operativi Unix è la possibilità di gestire i collegamenti a file e directory.

4.8.1 Spostamento e ridenominazione

«

$ Lo spostamento di file e directory avviene per mezzo di '**mv**'. Per esercitarsi con questo comando si preparano alcuni file e alcune directory.

```
$ touch alfa [Invio]

$ touch beta [Invio]

$ mkdir gamma [Invio]
```

Come sempre è bene controllare.

```
$ ls -l [Invio]

  ...
  -rw-rw-r--   1 tizio    tizio           0 Dec 25 12:46 alfa
  -rw-rw-r--   1 tizio    tizio           0 Dec 25 12:46 beta
  drwxrwxr-x   2 tizio    tizio        1024 Dec 25 12:46 gamma
  ...
```

Si procede rinominando il file '`alfa`' in modo che diventi '`omega`'.

```
$ mv alfa omega [Invio]

$ ls -l [Invio]

  ...
  -rw-rw-r--   1 tizio    tizio           0 Dec 25 12:46 omega
  ...
```

Volendo spostare file e directory in gruppo, è necessario che la destinazione sia una directory. Con il comando seguente si spostano i due file creati poco prima nella directory '`gamma/`'.

```
$ mv omega beta gamma [Invio]
```

```
$ ls -l gamma [Invio]

  -rw-rw-r--   1 tizio    tizio           0 Dec 25 12:46 beta
  -rw-rw-r--   1 tizio    tizio           0 Dec 25 12:46 omega
```

Generalmente, lo spostamento (o il cambiamento di nome) non fa differenza tra file normali e directory.

```
$ mv gamma /tmp [Invio]
```

Il comando precedente sposta la directory ‘`gamma/`’ in ‘`/tmp/`’.

È importante tenere presente che il comando ‘**`mv`**’ non può cambiare un gruppo di nomi in modo sistematico. Per esempio, non si può cambiare ‘**`*.mio`**’ in ‘**`*.tuo`**’.

Video: *http://www.youtube.com/watch?v=FMqmTqDhPVs*

4.8.2 Collegamenti

La creazione di un collegamento è un’operazione simile alla copia, con la differenza che invece di creare un duplicato di file e directory, si genera un riferimento agli originali. Ne esistono due tipi: collegamenti simbolici e collegamenti fisici (questi ultimi conosciuti di solito come *hard link*). In questa esercitazione vengono mostrati solo collegamenti simbolici.

```
$ pwd [Invio]

  /home/tizio
```

Il comando utilizzato per creare questi collegamenti è ‘**`ln`**’; dal momento che si intendono mostrare solo quelli simbolici, viene usata sempre l’opzione ‘**`-s`**’.

Per esercitarsi con questo comando si preparano alcuni file e directory.

```
$ touch uno[Invio]

$ touch due[Invio]

$ mkdir tre[Invio]
```

Come sempre è bene controllare.

```
$ ls -l[Invio]

  …
  -rw-rw-r--   1 tizio    tizio           0 Dec 25 12:46 due
  drwxrwxr-x   2 tizio    tizio        1024 Dec 25 12:46 tre
  -rw-rw-r--   1 tizio    tizio           0 Dec 25 12:46 uno
```

Come già si accenna all'inizio, la creazione di un collegamento è un'operazione simile alla copia.

```
$ ln -s uno uno.bis[Invio]
```

Con il comando mostrato sopra, si ottiene un collegamento simbolico, denominato 'uno.bis', al file 'uno'.

```
$ ls -l[Invio]

  …
  lrwxrwxrwx 1 tizio  tizio    3 Dec 25 12:47 uno.bis -> uno
```

Da questo momento si può fare riferimento al file 'uno' utilizzando il nome 'uno.bis'.

La creazione di un collegamento a una directory può avvenire nello stesso modo visto per i file (a patto che si tratti di collegamenti simbolici).

```
$ ln -s /tmp miatemp [Invio]
```

Se il comando appena visto ha successo si può raggiungere la directory '`/tmp/`' anche attraverso il riferimento '`miatemp`'.

La creazione di un gruppo di collegamenti con un solo comando, può avvenire solo quando la destinazione (l'ultimo nome sulla riga di comando) è una directory. In questo modo si ottiene la creazione dei collegamenti al suo interno.

```
$ ln -s /home/tizio/uno* /home/tizio/due tre [Invio]
```

In questo caso, si generano i collegamenti per tutti i file i cui nomi iniziano per '`uno`' e anche per il file '`due`' nella directory '`tre/`'.

Nell'esempio mostrato sopra, i file per i quali si vogliono creare dei collegamenti simbolici sono stati indicati con il loro percorso assoluto, pur immaginando che la directory '`/home/tizio/`' fosse quella corrente. In tal senso, la directory di destinazione è stata indicata semplicemente in modo relativo. Quando si creano dei collegamenti simbolici in una directory come in questo modo, è necessario indicare anche il percorso adeguato nei file (o nelle directory) di origine, perché i nomi ai quali si vuole fare riferimento sono trattati come stringhe.

```
$ ls -l tre [Invio]

lrwxrwxrwx 1 tizio tizio 15 Dec 25 15:21 due -> /home/tizio/due
lrwxrwxrwx 1 tizio tizio 15 Dec 25 15:21 uno -> /home/tizio/uno
lrwxrwxrwx 1 tizio tizio 19 Dec 25 15:21 uno.bis -> /home/tizio/uno.bis
```

Si può osservare che è stato creato anche un collegamento che punta a un altro collegamento.

Se si cancellano questi collegamenti simbolici nella directory ‘`tre/`’, si può provare a vedere cosa può accadere se non si indica un percorso assoluto:

```
$ rm tre/*[Invio]
```

Intuitivamente si può ritenere che possa essere corretta la creazione dei collegamenti simbolici in questo modo:

```
$ ln -s uno* due tre[Invio]
```

Se però si va a controllare il contenuto della directory ‘`tre/`’, si può notare una cosa strana: i collegamenti simbolici puntano a loro stessi.

```
$ ls -l tre[Invio]

  lrwxrwxrwx 1 tizio tizio 15 Dec 25 15:25 due -> due
  lrwxrwxrwx 1 tizio tizio 15 Dec 25 15:25 uno -> uno
  lrwxrwxrwx 1 tizio tizio 19 Dec 25 15:25 uno.bis -> uno.bis
```

Inizialmente è difficile capire questa cosa. Conviene provare in modo ancora diverso:

```
$ rm tre/*[Invio]

$ ln -s ./uno* ./due tre[Invio]

$ ls -l tre[Invio]

  lrwxrwxrwx 1 tizio tizio 15 Dec 25 15:30 due -> ./due
  lrwxrwxrwx 1 tizio tizio 15 Dec 25 15:30 uno -> ./uno
  lrwxrwxrwx 1 tizio tizio 19 Dec 25 15:30 uno.bis -> ./uno.bis
```

Se non è ancora chiaro, si provi questo:

```
$ rm tre/*[Invio]
```

```
$ ln -s nero/marrone rosso/arancio giallo/verde tre[Invio]

$ ls -l tre[Invio]

  lrwxrwxrwx 1 tizio tizio 15 Dec 25 15:35 marrone -> nero/marrone
  lrwxrwxrwx 1 tizio tizio 15 Dec 25 15:35 arancio -> rosso/arancio
  lrwxrwxrwx 1 tizio tizio 19 Dec 25 15:35 verde -> giallo/verde
```

Si intende che i file ‘`nero/marrone`’, ‘`rosso/arancio`’ e ‘`giallo/verde`’ non esistono; tuttavia, i collegamenti simbolici vengono creati ugualmente.

Video: *http://www.youtube.com/watch?v=BpForhhHhtI*

4.8.3 Conclusione

Lo spostamento di file e directory avviene in modo simile alla copia, solo che l’origine viene rimossa. Lo spostamento di directory attraverso unità di memorizzazione differenti non è possibile. Lo spostamento erroneo può essere dannoso: se non si fa attenzione si può sovrascrivere qualcosa che ha già lo stesso nome dei file o delle directory di destinazione. Questo è lo stesso tipo di problema che si rischia di incontrare con la copia.

I collegamenti a file e directory permettono di definire percorsi alternativi agli stessi. I collegamenti simbolici vengono creati analizzando i nomi senza verificare che appartengano effettivamente a file o directory reali.

4.9 La shell

La shell è il mezzo attraverso cui si interagisce con il sistema. Il modo di inserire i comandi può cambiare molto da una shell all'altra. Gli esercizi proposti in questa sezione sono stati realizzati in particolare con la shell Bash, ma gran parte di questi possono essere validi anche per altre shell.

4.9.1 Completamento automatico

Il completamento automatico è un modo attraverso cui alcune shell aiutano l'utente a completare un comando. La richiesta di completamento viene fatta attraverso l'uso del tasto [*Tab*]. Si preparano alcuni file di esempio. I nomi utilizzati sono volutamente lunghi.

```
$ touch microinterruttore[Invio]

$ touch microscopico[Invio]

$ touch supersonico[Invio]
```

Supponendo di voler utilizzare questi nomi all'interno di una riga di comando, si può essere un po' infastiditi dalla loro lunghezza. Utilizzando il completamento automatico si risolve il problema.

```
$ ls sup[Tab]
```

Dopo avere scritto solo '**sup**', premendo il tasto [*Tab*] si ottiene il completamento del nome, dal momento che non esistono altri file o directory (nella posizione corrente) che inizino nello stesso modo. L'esempio seguente mostra lo stesso comando completato e terminato.

```
$ ls sup[Tab]ersonico[Invio]
```

Il completamento automatico dei nomi potrebbe essere impossibile. Infatti, potrebbe non esistere alcun nome che coincida con la parte iniziale già inserita, oppure potrebbero esistere più nomi composti con lo stesso prefisso. In questo ultimo caso, il completamento si ferma al punto in cui i nomi iniziano a distinguersi.

```
$ ls mic[Tab]ro
```

In questo caso, il completamento si spinge fino a '**micro**' che è la parte comune dei nomi '`microinterruttore`' e '`microscopico`'. Per poter proseguire occorre aggiungere un'indicazione che permetta di distinguere tra i due nomi. Volendo selezionare il primo di questi nomi, basta aggiungere la lettera '**i**' e premere nuovamente il tasto [*Tab*]. L'esempio seguente rappresenta il procedimento completo.

```
$ ls mic[Tab]roi[Tab]nterruttore[Invio]
```

Video: *http://www.youtube.com/watch?v=emZxXy81Ox810*

4.9.2 Sostituzione

L'utilizzo di metacaratteri, rappresenta una forma alternativa di completamento dei nomi. Infatti è compito della shell la trasformazione dei simboli utilizzati per questo scopo.

Per gli esercizi successivi si utilizzano inizialmente i file creati nella sezione precedente: '`microinterruttore`', '`microscopico`' e '`supersonico`'. In seguito ne vengono aggiunti altri quando l'occasione lo richiede.

4.9.2.1 Asterisco

«

$ L'asterisco rappresenta una sequenza indefinita di zero o più caratteri di qualunque tipo, esclusa la barra obliqua di separazione tra le directory. Per cui, l'asterisco utilizzato da solo rappresenta tutti i nomi di file disponibili nella directory corrente.

```
$ ls [Invio]
```

Il comando '**ls**' appena mostrato serve a elencare tutti i nomi di file e directory contenuti nella directory corrente.

```
$ ls * [Invio]
```

Questo comando è un po' diverso, nel senso che la shell provvede a sostituire l'asterisco con tutto l'elenco di nomi di file e directory contenuti nella directory corrente. Sarebbe come se il comando fosse '**ls microinterruttore microscopico**'...

In tal senso, anche il comportamento di '**ls**' cambia: non si limita a elencare il contenuto della directory corrente, ma (eventualmente, se ce ne sono) anche quello di tutte le directory contenute in quella corrente.

L'asterisco può essere utilizzato anche assieme a parti fisse di testo.

```
$ ls micro* [Invio]
```

Questo comando è composto in modo che la shell sostituisca '**micro***' con tutti i nomi che iniziano per '**micro**'.

```
microinterruttore microscopico
```

È stato precisato che l'asterisco può essere sostituito anche con la stringa nulla. Per verificarlo si crea un altro file.

```
$ touch nanomicro [Invio]
```

Con il comando seguente si vogliono elencare tutti i nomi che contengono la parola '**micro**'.

```
$ ls *micro* [Invio]

microinterruttore microscopico nanomicro
```

Video: *http://www.youtube.com/watch?v=dbpuvdo5hU4*

4.9.2.2 Punto interrogativo

Il punto interrogativo rappresenta esattamente un carattere qualsiasi.

Prima di proseguire si aggiungono alcuni file con nomi adatti agli esempi seguenti.

```
$ touch xy123j4 [Invio]

$ touch xy456j5 [Invio]

$ touch xy789j111 [Invio]

$ touch xy78j67 [Invio]
```

Con il comando seguente si vuole intervenire su tutti i file lunghi esattamente sette caratteri che contengono la lettera '**j**' nella sesta posizione.

```
$ ls ?????j? [Invio]

xy123j4 xy456j5
```

Diverso sarebbe stato usando l'asterisco: non si può limitare il risultato ai file che contengono la lettera '**j**' nella sesta posizione, ma nemmeno la lunghezza del nome può essere presa in considerazione.

```
$ ls *j*[Invio]
```

In questo modo si ottiene l'elenco di tutti i nomi che contengono la lettera '**j**', senza specificare altro.

```
xy123j4 xy456j5 xy789j111 xy78j67
```

Video: *http://www.youtube.com/watch?v=aBOUhuqjVic*

4.9.2.3 Parentesi quadre

«

$ Le parentesi quadre vengono utilizzate per delimitare un elenco o un intervallo di caratteri. Rappresentano un solo carattere tra quelli contenuti, o tra quelli appartenenti all'intervallo indicato.

```
$ ls xy????[4567]*[Invio]
```

```
xy123j4 xy456j5 xy78j67
```

Il comando appena indicato è stato scritto in modo da fornire a '**ls**', come argomento, l'elenco di tutti i file i cui nomi iniziano per '**xy**', proseguono con quattro caratteri qualunque, quindi contengono un carattere da '**4**' a '**7**' e terminano in qualunque modo. Lo stesso risultato si potrebbe ottenere indicando un intervallo nelle parentesi quadre.

```
$ ls xy????[4-7]*[Invio]
```

Video: *http://www.youtube.com/watch?v=f6_FhpGFu1w*

4.9.3 Protezione

Il fatto che la shell sostituisca alcuni caratteri complica il loro utilizzo nei nomi di file e directory. Se esiste la necessità, è possibile evitare la sostituzione di questi caratteri facendoli precedere da una barra obliqua inversa, che funge da carattere di escape, ovvero, da simbolo di protezione.

```
$ touch sei\*otto [Invio]

$ ls [Invio]

  ...
  sei*otto
```

In questo modo è possibile includere nel nome di un file anche lo spazio.

```
$ touch sei\ bella [Invio]

$ ls [Invio]

  ...
  sei bella
  sei*otto
```

È possibile ottenere un risultato simile delimitando il testo che deve essere interpretato come un oggetto unico attraverso apici singoli o apici doppi:

```
$ touch "sei*sei" [Invio]

$ touch 'tre*due' [Invio]

$ ls [Invio]
```

```
…
tre*due
…
sei bella
sei*otto
sei*sei
```

```
$ touch "sei alta"[Invio]

$ ls[Invio]

…
tre*due
…
sei alta
sei bella
sei*otto
sei*sei
```

È bene ricordare che l'uso degli apici singoli o degli apici doppi non è sempre equivalente, ma questo e altri dettagli vengono approfonditi nel capitolo dedicato alla shell POSIX (parte 17).

Video: *http://www.youtube.com/watch?v=TYgVKugeEs8*

4.9.4 Verifica della sostituzione

«

$ L'uso di metacaratteri può essere pericoloso quando non si ha un'esperienza sufficiente a determinare l'effetto esatto del comando che ci si accinge a utilizzare. Ciò soprattutto quando si utilizzano per cancellare. Il modo migliore per verificare l'effetto della sostituzione dei metacaratteri è l'uso del comando '**echo**', che si occupa semplicemente di visualizzare l'elenco dei suoi argomenti.

Per esempio, per sapere quali file e directory vengono coinvolti dal modello '**micro***', basta il comando seguente:

```
$ echo micro*[Invio]

  microinterruttore microscopico
```

Anche l'uso di '**ls**', come comando non distruttivo, può essere di aiuto per determinare l'estensione di un modello fatto di metacaratteri. Ma '**ls**', in condizioni normali, mostra anche il contenuto delle directory che vengono indicate tra gli argomenti, distraendo in questo caso l'utilizzatore. Pertanto, per evitare l'aggiunta di informazioni non richieste, '**ls**' andrebbe corredato con l'opzione '**-d**'.

4.9.5 Ridirezione

La ridirezione dirotta i dati in modo da destinarli a un file o da prelevarli da un file.

```
$ ls -l > elenco[Invio]
```

Questo comando genera il file 'elenco' con il risultato dell'esecuzione di '**ls**'. Si può controllare il contenuto di questo file con '**cat**'.

```
$ cat elenco[Invio]
```

Anche l'input può essere ridiretto, quando il comando al quale si vuole inviare è in grado di riceverlo. Il comando '**cat**' è in grado di emettere ciò che riceve dallo standard input.

```
$ cat < elenco[Invio]
```

Si ottiene in questo modo la visualizzazione del contenuto del file ‘`elenco`’, esattamente come succede con il comando precedente, ma in tal caso, il file da visualizzare viene ottenuto dallo standard input per mezzo dell’attività della shell.

La ridirezione dell’output, come è stata vista finora, genera un nuovo file ogni volta, eventualmente sovrascrivendo ciò che esiste già con lo stesso nome. Sotto questo aspetto, la ridirezione dell’output è fonte di possibili danni.

La ridirezione dell’output può essere fatta in aggiunta, creando un file se non esiste, o aggiungendovi i dati se invece esiste già.

```
$ ls -l /tmp >> elenco[Invio]
```

In tal modo viene aggiunto al file ‘`elenco`’ l’elenco dettagliato del contenuto della directory ‘`/tmp/`’.

```
$ cat elenco[Invio]
```

4.9.6 Condotti

«

$ Il condotto, ovvero la *pipeline*, è una forma di ridirezione in cui la shell invia l’output di un comando come input del successivo.

```
$ cat elenco | sort[Invio]
```

In questo modo, ‘**`cat`**’ legge il contenuto del file ‘`elenco`’, ma questo, invece di essere visualizzato sullo schermo, viene inviato dalla shell come standard input di ‘**`sort`**’ che lo riordina e poi lo emette sullo schermo.

Un condotto può utilizzare anche la ridirezione, per cui, il comando visto precedentemente può essere trasformato nel modo seguente:

```
$ cat < elenco | sort[Invio]
```

Naturalmente, il comando si può semplificare come indicato sotto, ma in tal caso non si tratta più di condotto:

```
$ sort < elenco [Invio]
```

Video: *http://www.youtube.com/watch?v=F0vkUfj6WF8*

4.9.7 Alias

La creazione di un alias è un metodo che permette di definire un nome alternativo per un comando preesistente.

```
$ alias elenca='ls -l' [Invio]
```

Dopo aver definito l'alias '**elenca**', come indicato nel comando precedente, utilizzandolo si ottiene l'equivalente di '**ls -l**'. Basta provare.

```
$ elenca [Invio]
```

Ma l'alias permette di utilizzare argomenti, come se si trattasse di comandi normali.

```
$ elenca micro* [Invio]
```

Quello che si ottiene corrisponde al risultato del comando '**ls -l micro***':

```
-rw-rw-r--  1 tizio  tizio  0 Dec 26 10:19 microinterruttore
-rw-rw-r--  1 tizio  tizio  0 Dec 26 10:19 microscopico
```

Gli alias tipici che vengono creati sono i seguenti. Servono per fare in modo che le operazioni di cancellazione o sovrascrittura vengano eseguite dopo una richiesta di conferma.

```
$ alias rm='rm -i' [Invio]
```

```
$ alias cp='cp -i' [Invio]

$ alias mv='mv -i' [Invio]
```

Si può provare a eliminare un file per vedere cosa accade.

```
$ rm microinterruttore [Invio]

  rm: remove 'microinterruttore'?:
n [Invio]
```

In questo modo, il file non è stato cancellato.

Per l'eliminazione di un alias si procede intuitivamente con il comando '**unalias**':

```
$ unalias elenca [Invio]
```

In questo modo, si elimina l'alias '**elenca**' selettivamente, mentre con il comando seguente si eliminano tutti gli alias ancora esistenti:

```
$ unalias -a [Invio]
```

A ogni modo, alla fine della sessione di lavoro con la shell, gli alias vengono perduti.

4.9.8 Conclusione

«

Il completamento dei nomi e i metacaratteri sono gli strumenti operativi più importanti che una shell fornisce. Tuttavia, l'uso di modelli con metacaratteri può essere fonte di errori anche gravi, pertanto, prima di utilizzarli in comandi distruttivi, conviene verificare l'effetto di questi modelli con '**echo**'.

La ridirezione e il condotto sono un altro strumento importante che permette di costruire comandi molto complessi a partire da comandi elementari.

4.10 Controllo dei processi

In presenza di un ambiente in multiprogrammazione è importante il controllo dei processi in esecuzione. Un processo è un singolo eseguibile in funzione, ma un comando può generare diversi processi.

Video: *http://www.youtube.com/watch?v=XGRB3ONy8_M*

4.10.1 Visualizzazione dello stato dei processi

Il comando fondamentale per il controllo dei processi è '**ps**'.

```
$ ps x [Invio]

  PID TTY STAT  TIME COMMAND
  077   1 SW   0:01 (login)
  078   2 SW   0:01 (login)
  091   1 S    0:01 -bash
  132   2 S    0:01 -bash
  270   1 R    0:00 ps
```

In questo caso '**ps**' mostra che sono in funzione due copie di '**bash**' (la shell Bash), ognuna su un terminale differente (in questo caso si tratta della prima e della seconda console virtuale di un sistema GNU/Linux): TTY 1 e 2. L'unico programma in esecuzione è lo stesso '**ps**', che in questo esempio è stato avviato dal primo terminale.

Attraverso l'opzione '**f**', si può osservare la dipendenza tra i processi.

```
$ ps xf [Invio]

  PID TTY STAT  TIME COMMAND
  077   1 SW   0:01 (login)
  091   1 S    0:01  \_ -bash
  275   1 R    0:00      \_ ps -f
  078   2 SW   0:01 (login)
  132   2 S    0:01  \_ -bash
```

In un sistema GNU/Linux esiste il programma '**pstree**' che offre un modo graficamente più aggraziato di osservare la dipendenza tra i processi.

LINUX

```
$ pstree [Invio]

 init-+-crond
      |-kflushd
      |-klogd
      |-kswapd
      |-login---bash
      |-login---bash---pstree
      |-4*[mingetty]
      |-4*[nfsiod]
      |-portmap
      |-rpc.mountd
      |-rpc.nfsd
      |-syslogd
      `-update
```

Mentre prima si vedevano solo i processi connessi ai terminali, adesso vengono visualizzati tutti i processi in funzione in modo predefinito. L'elenco cambia a seconda della configurazione del proprio

sistema.

4.10.2 Eliminazione dei processi

I processi vengono eliminati automaticamente una volta che questi terminano regolarmente. A volte ci può essere la necessità di eliminare forzatamente un processo.

Per verificare questa situazione è necessario disporre di un secondo terminale di accesso al sistema, come potrebbe essere una seconda console virtuale di un sistema GNU/Linux (combinazione di tasti [*Alt F2*]), o una seconda connessione a un sistema remoto. Dal secondo terminale si avvia un programma inutile che viene poi eliminato attraverso il primo terminale.

Se nel secondo terminale fosse necessario eseguire l'accesso, è questo il momento di farlo. Quindi si dà il comando «inutile»:

```
$ yes [Invio]

y
y
y
y
...
```

Attraverso '**yes**' si ottiene un'emissione continua di lettere «y». Si può passare al primo terminale e osservare la situazione.

[*Alt F1*]

```
$ ps x [Invio]
```

```
  PID TTY STAT  TIME COMMAND
  077   1 SW   0:01 (login)
  078   2 SW   0:01 (login)
  091   1 S    0:01 -bash
  132   2 S    0:01 -bash
  311   2 R    0:26 yes
```

Si decide di eliminare il processo generato da '**yes**' attraverso l'invio di un segnale di conclusione.

```
$ kill 311 [Invio]
```

Il numero 311 è il numero abbinato al processo, o PID, che si ottiene osservando le informazioni emesse da '**ps**'. Tornando sul terminale da cui è stato eseguito '**yes**' si può osservare che questo ha smesso di funzionare.

[*Alt F2*]

```
…
y
y
Terminated
```

4.10.3 Processi sullo sfondo

«

$ Per mezzo della shell è possibile avviare dei comandi sullo sfondo, ovvero in *background*, in modo che si renda nuovamente disponibile l'invito per inserire altri comandi.

```
$ yes > /dev/null & [Invio]
```

Questo comando avvia '**yes**' dirottando l'output nel file '`/dev/null`' che in realtà è un dispositivo speciale paragonabile a una pattumiera senza fondo (tutto ciò che vi viene scritto è eliminato). Il

simbolo e-commerciale (‘**&**’), posto alla fine del comando, dice alla shell di eseguirlo sullo sfondo.

Naturalmente, ha senso eseguire un comando sullo sfondo quando questo non richiede input da tastiera e non emette output sul terminale.

4.10.4 Conclusione

«

Il controllo dei processi avviene essenzialmente attraverso ‘**ps**’ e ‘**kill**’. La shell fornisce generalmente una forma di controllo sui comandi avviati attraverso di essa; questi vengono definiti normalmente *job* di shell.

Il capitolo 10 tratta meglio di questo argomento.

4.11 Permessi

«

I permessi definiscono i privilegi dell’utente proprietario, del gruppo e degli altri utenti nei confronti dei file e delle directory.

La sezione 3.21 introduce i problemi legati ai permessi, in particolare spiega il modo in cui si rappresentano in forma numerica.

4.11.1 Permessi sui file

«

Sui file possono essere regolati tre tipi di permessi: lettura, scrittura ed esecuzione. Mentre il significato del permesso di esecuzione è abbastanza logico (riguarda i file eseguibili e gli script), così come lo è anche quello in lettura, quello in scrittura potrebbe fare pensare che permetta di evitarne la cancellazione. Non è così: la possibilità di cancellare un file dipende dai permessi della directory.

```
$ touch mio_file[Invio]
```

```
$ chmod -r mio_file[Invio]
```

In questo modo è stato tolto il permesso di lettura a tutti gli utenti, compreso il proprietario.

```
$ ls -l mio_file[Invio]

--w--w----   1 tizio    tizio        0 Dec 26 10:24 mio_file
```

Si può vedere che dalla stringa dei permessi è sparita la lettera '**r**'.

```
$ cat mio_file[Invio]

cat: mio_file: Permission denied
```

L'emissione sullo schermo del file è impossibile perché non c'è il permesso di lettura (in questo caso il file è vuoto e non c'è proprio nulla da visualizzare, ma qui conta il fatto che il sistema si opponga alla lettura).

```
$ chmod +r mio_file[Invio]
```

Prima di verificare cosa accade togliendo il permesso di scrittura conviene ripristinare il permesso di lettura, con il comando appena visto.

```
$ chmod -w mio_file[Invio]
```

In questo modo viene tolto il permesso di scrittura, cosa che impedisce la modifica del file, ma non la sua cancellazione.

```
$ ls > mio_file[Invio]

bash: mio_file: Permission denied
```

Un tentativo di sovrascrittura genera una segnalazione di errore, come nell'esempio appena visto, così come qualunque altro tentativo

di modificare il suo contenuto.

```
$ mv mio_file tuo_file[Invio]
```

Lo spostamento o il cambiamento del nome è possibile.

```
$ ls -l tuo_file[Invio]

-r--r--r--   1 tizio    tizio        0 Dec 26 10:24 tuo_file
```

Anche la cancellazione è ammissibile; probabilmente si ottiene un avvertimento, ma niente di più.

```
$ rm tuo_file[Invio]

rm: remove 'tuo_file', overriding mode 0444?
y[Invio]
```

Il file, alla fine, viene cancellato.

Video: *http://www.youtube.com/watch?v=Ce23Fp23BV5kg*

4.11.2 Permessi sulle directory

Sulle directory possono essere regolati tre tipi di permessi: lettura, scrittura e accesso (ovvero «attraversamento», che corrisponde al permesso di esecuzione dei file). Per chi non conosce già un sistema operativo Unix, il significato potrebbe non essere tanto intuitivo.

```
$ mkdir provedir[Invio]

$ touch provedir/uno[Invio]

$ touch provedir/due[Invio]
```

Togliendo il permesso di lettura si impedisce la lettura del contenuto della directory, cioè si impedisce l'esecuzione di un comando come

‘**ls**’, mentre l’accesso ai file continua a essere possibile (purché se ne conoscano i nomi).

```
$ chmod -r provedir[Invio]

$ ls provedir[Invio]

  ls: provedir: Permission denied
```

Prima di proseguire si ripristinano i permessi di lettura.

```
$ chmod +r provedir[Invio]
```

I permessi di scrittura consentono di aggiungere, eliminare e rinominare i file (comprese le eventuali sottodirectory).

```
$ chmod -w provedir[Invio]
```

Questo comando toglie il permesso di scrittura della directory ‘`provedir/`’.

```
$ rm provedir/uno[Invio]

  rm: provedir/uno: Permission denied
$ cp provedir/uno provedir/tre[Invio]

  cp: cannot create regular file ‘provedir/tre’: ↩
 ↪Permission denied
$ mv provedir/uno provedir/tre[Invio]

  mv: cannot move ‘provedir/uno’ to ‘provedir/tre’: ↩
 ↪Permission denied
```

Prima di proseguire si ripristina il permesso di scrittura.

```
$ chmod +w provedir[Invio]
```

Il permesso di accesso è il più strano. Impedisce l'accesso alla directory e a tutto il suo contenuto. Ciò significa che non è possibile accedere a file o directory discendenti di questa.

Viene creata una directory discendente da '`provedir/`'.

```
$ mkdir provedir/tmp [Invio]
```

Si crea un file al suo interno, per poter verificare in seguito quanto affermato.

```
$ touch provedir/tmp/esempio [Invio]
```

Si tolgono i permessi di accesso a '`provedir/`' per vedere cosa accade.

```
$ chmod -x provedir [Invio]
```

Da questo momento, '`provedir/`' e tutto quello che ne discende è inaccessibile.

```
$ cd provedir [Invio]

  bash: cd: provedir: Permission denied
$ cat provedir/tmp/esempio [Invio]

  cat: provedir/tmp/esempio: Permission denied
$ touch provedir/tmp/esempio2 [Invio]

  touch: provedir/tmp/esempio2: Permission denied
```

Video: *http://www.youtube.com/watch?v=Yew-Sn2SV2w*

4.11.3 Maschera dei permessi: umask

«

$ La maschera dei permessi, ovvero la maschera *umask*, determina i permessi che devono essere tolti quando si crea un file o una directory e non si definiscono esplicitamente i loro permessi. Nello stesso modo, quando si attribuiscono dei permessi senza definire a quale livello si riferiscono (all'utente, al gruppo o agli altri, come è stato fatto nelle sezioni precedenti), vengono tolti quelli della maschera dei permessi. Per conoscere il valore di questa maschera basta il comando seguente:

```
$ umask [Invio]

  0002
```

Ciò che si ottiene dipende dalla configurazione del sistema; frequentemente, il valore della maschera dei permessi è 0022_8.

Il numero due rappresenta un permesso di scrittura, nel caso del risultato 0002_8 si riferisce solo agli utenti differenti dal proprietario e dal gruppo di appartenenza. Questo significa che il permesso viene tolto in modo predefinito. Se invece il valore fosse 0022_8, anche al gruppo verrebbe tolto il permesso di scrittura.

Si può ottenere una rappresentazione della maschera dei permessi più espressiva con l'opzione '**-S**'.

```
$ umask -S [Invio]

  u=rwx,g=rwx,o=rx
```

In tal caso si è ottenuta la rappresentazione dei permessi che vengono concessi in modo predefinito.

Si suppone, per esercizio, di trovarsi nella situazione di volere difendere i propri dati da qualunque accesso da parte degli altri utenti (a eccezione dell'utente '**root**' al quale nulla può essere impedito).

```
$ umask 0077 [Invio]
```

Il numero sette rappresenta tutti i permessi (lettura, scrittura ed esecuzione-accesso) e questi vengono tolti sistematicamente al gruppo e agli altri utenti. Per verificarlo si può provare a creare un file.

```
$ touch segreto [Invio]

$ ls -l segreto [Invio]

-rw-------   1 tizio    tizio         0 Dec 27 11:10 segreto
```

Il comando '**touch**' non ha tentato di attribuire dei permessi di esecuzione, quindi questo permesso non appare tra quelli dell'utente proprietario.

```
$ mkdir segreta [Invio]

$ ls -l [Invio]

...
drwx------   2 tizio    tizio      1024 Dec 27 11:14 segreta
...
```

Come si vede dall'esempio, anche la creazione di directory risente della maschera dei permessi.

Video: *http://www.youtube.com/watch?v=5e7SPYTJi5A*

4.11.4 Conclusione

«

Il significato dei permessi di file e directory non è necessariamente intuitivo o evidente. Un po' di allenamento è necessario per comprenderne il senso.

La maschera dei permessi, o *umask*, è un mezzo con cui filtrare i permessi indesiderati nelle operazioni normali, quelle in cui questi non vengono espressi in modo esplicito.

4.12 Creazione e modifica di file di testo

«

$ In tutti i corsi di Unix si mostra l'uso di un applicativo storico piuttosto spartano, per la creazione e la modifica di file di testo: VI. Questo poi si concretizza in pratica nell'eseguibile '**`vi`**'. La necessità di imparare a usare questo programma, almeno in modo elementare, sta nel fatto che utilizza poche risorse di memoria e spesso fa parte dell'insieme di programmi di servizio di sistemi di emergenza su dischetti o altre unità esterne di memorizzazione con poca capacità.

Video: *http://www.youtube.com/watch?v=XrUi3kwnDYY*

4.12.1 Modalità di comando e di inserimento

«

L'uso di VI è difficile perché si distinguono diverse modalità di funzionamento. In pratica si separa la fase di inserimento del testo da quella in cui si inseriscono i comandi.

Per poter inserire un comando occorre sospendere l'inserimento con la pressione di [*Esc*]. Per poter ritornare alla modalità di inserimento occorre dare un comando apposito.

Il tasto [*Esc*] può essere usato anche per annullare un comando che non sia stato completato. Se premuto più del necessario non produce alcun effetto collaterale.

4.12.2 Creazione

Si crea un nuovo file semplicemente avviando il programma senza argomenti.

```
$ vi [Invio]
```

Appena avviato, VI impegna tutto lo schermo.

```
_
~
~
~
~
~
/tmp/vi.OX7AXb: new file: line 1
```

I simboli tilde ('**~**') rappresentano righe nulle (inesistenti).

In questo momento il programma si trova in ***modalità di comando*** e accetta comandi espressi attraverso lettere o simboli della tastiera.

4.12.3 Inserimento di testo

Con il tasto [*i*], che rappresenta il comando di inserimento (*insert*), si passa alla modalità di inserimento attraverso la quale si può digitare del testo normalmente. Il testo viene inserito di proposito senza lettere accentate, perché il comportamento del programma, in presenza di caratteri al di fuori della codifica ASCII pura e semplice, dipende da diversi fattori.

[*i*]

GNU/Linux e' un sistema operativo completo [*Invio*]

il cui kernel e' stato scritto da [*Invio*]

Linus Torvalds e altri collaboratori. [*Invio*]

Quello che si vede sullo schermo dovrebbe apparire come l'esempio che segue, con il cursore alla fine dell'ultima frase digitata.

```
GNU/Linux e' un sistema operativo completo
il cui kernel e' stato scritto da
Linus Torvalds e altri collaboratori._
~
~
~
```

Si termina la modalità di inserimento e si torna a quella di comando attraverso la pressione del tasto [*Esc*].

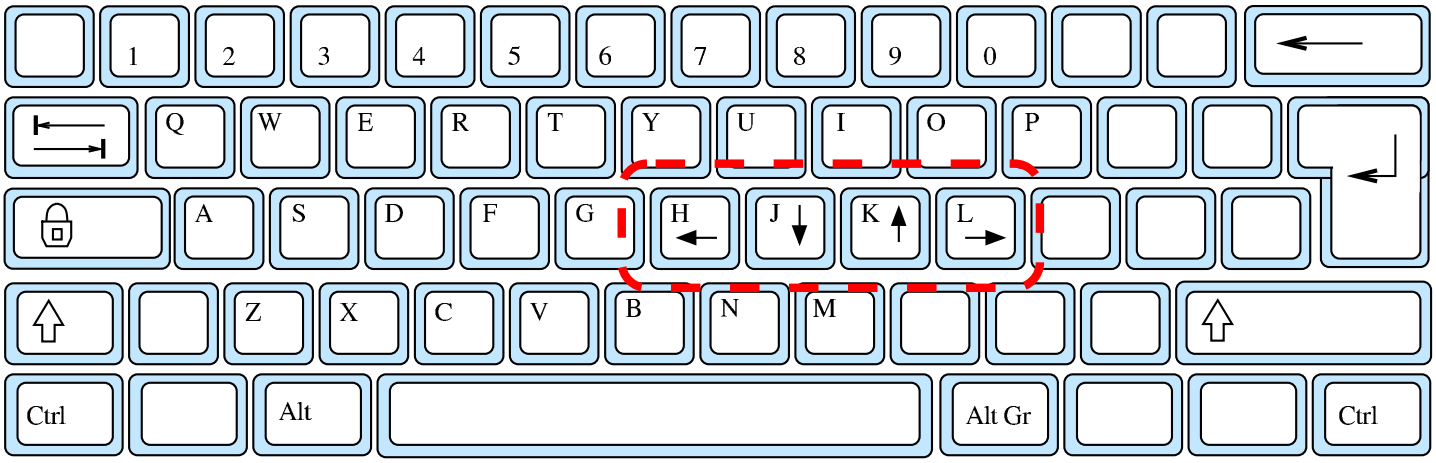
[*Esc*]

Probabilmente, dopo la pressione del tasto [*Esc*], il cursore si sposta sotto al punto che conclude la riga, dal momento che a destra, dove si trovava prima, non c'è alcun carattere, nemmeno lo spazio.

4.12.4 Spostamento del cursore

«

$ Lo spostamento del cursore attraverso il testo avviene in modalità di comando, con i tasti [*h*], [*j*], [*k*] e [*l*] che corrispondono rispettivamente allo spostamento a sinistra, in basso, in alto e a destra. Nella maggior parte delle situazioni possono essere utilizzati i tasti freccia, anche durante la fase di inserimento.



Si decide di spostare il cursore davanti alla parola «completo» della prima riga.

[*h*][*h*]

[*k*][*k*]

In pratica si sposta il cursore a sinistra di due posizioni e in alto di due.

```
GNU/Linux e' un sistema operativo_completo
il cui kernel e' stato scritto da
Linus Torvalds e altri collaboratori.
~
~
~
```

4.12.5 Cancellazione

La cancellazione di testo in modalità di comando avviene attraverso l'uso del tasto [*x*]. Si ottiene la cancellazione del carattere che si trova in corrispondenza del cursore, avvicinando il testo rimanente dalla destra.

Nella maggior parte dei casi può essere usato anche il tasto [*Canc*] con tale scopo, che in particolare, dovrebbe funzionare sia in modalità di comando, sia in inserimento.

Si decide di cancellare la parola «completo», assieme allo spazio che la precede; si osservi che al termine, il cursore potrebbe spostarsi automaticamente sotto la lettera «o» della parola «operativo»:

[*x*][*x*][*x*][*x*][*x*][*x*][*x*][*x*][*x*]

```
GNU/Linux e' un sistema operativo
il cui kernel e' stato scritto da
Linus Torvalds e altri collaboratori.
~
~
~
```

La cancellazione di una riga intera si ottiene con il comando '**dd**' ovvero con la pressione del tasto [*d*] per due volte di seguito.

Si decide di cancellare l'ultima riga. Per prima cosa si sposta il cursore sopra con il tasto [*j*], premuto per due volte, quindi si procede con la cancellazione.

[*j*][*j*]

[*d*][*d*]

```
GNU/Linux e' un sistema operativo
il cui kernel e' stato scritto da
~
~
~
~
```

4.12.6 Salvataggio e conclusione

Il salvataggio del testo in un file si ottiene attraverso un comando più complesso di quelli visti finora. Dalla modalità di comando si preme il tasto [*:*] che inizia un comando speciale, detto *colon* o ***ultima riga***, perché appare sull'ultima riga dello schermo.

[*:*]

```
GNU/Linux e' un sistema operativo
il cui kernel e' stato scritto da
~
~
~
~
:_
```

Il comando per salvare è il seguente:

```
:w nome_file
```

Si decide di salvare con il nome '**miotesto**':

```
:w miotesto
```

Sullo schermo dovrebbe apparire come si vede di seguito:

```
GNU/Linux e' un sistema operativo
il cui kernel e' stato scritto da
~
~
~
~
:w miotesto_
```

Si conclude con la pressione del tasto [*Invio*].

[*Invio*]

La conclusione del funzionamento di VI si ottiene con il comando '**:q**'. Se si pretende di terminare senza salvare occorre imporre il comando con l'aggiunta di un punto esclamativo ('**:q!**').

:q[*Invio*]

4.12.7 Apertura di un file esistente

«

$ Per avviare l'eseguibile '**vi**' in modo che questo apra immediatamente un file già esistente per permetterne la modifica, basta indicare il nome di questo file nella riga di comando.

$ **vi miotesto**[*Invio*]

```
GNU/Linux e' un sistema operativo
il cui kernel e' stato scritto da
~
~
~
~
miotesto: unmodified: line 1
```

In alternativa si può utilizzare il comando '**:e**' con la sintassi seguente:

```
:e nome_file
```

Il risultato è lo stesso.

4.12.8 Conclusione

VI è un applicativo per la creazione e la modifica di file di testo, molto poco elaborato esteticamente e piuttosto complicato da utilizzare. Tuttavia è necessario saperlo usare nelle occasioni in cui non è disponibile un programma migliore, o non è possibile usare altro a causa delle ristrettezze del sistema.

Questo esercizio sull'uso di VI è solo un minimo assaggio del funzionamento del programma, che, al contrario di quanto possa sembrare, offre molti accorgimenti e potenzialità che alla lunga possono rivelarsi veramente utili. La sezione 22.15 mostra un po' meglio le possibilità di questo programma.

4.13 File eseguibili

I file normali che hanno i permessi di esecuzione, sono intesi dal sistema operativo come dei programmi che possono essere messi in funzione. Naturalmente, perché si possa trattare effettivamente di programmi è necessario che il sistema sia in grado di eseguire il loro contenuto.

4.13.1 Avvio di un programma e variabile di ambiente «PATH»

Video: *http://www.youtube.com/watch?v=rT-NlaUFPhY*

In linea di principio, l'avvio di un programma richiede l'indicazione del percorso, relativo o assoluto:

```
$ /bin/uname -a [Invio]

  Linux dinkel 2.2.15 #1 Thu Aug 31 15:55:32 CEST 2000 i586 ↩
 ↪unknown
```

Tuttavia, spesso non si sa precisamente dove sia collocato questo o quel programma eseguibile, ma di solito è possibile avviarlo ugualmente lasciando che sia il sistema stesso a trovarlo automaticamente:

```
$ uname -a [Invio]

  Linux dinkel 2.2.15 #1 Thu Aug 31 15:55:32 CEST 2000 i586 ↩
 ↪unknown
```

Questo automatismo dipende dalla configurazione della variabile di ambiente ***PATH***, la quale serve a elencare i percorsi degli eseguibili:

```
$ echo $PATH [Invio]

  /usr/local/bin:/usr/bin:/bin:/usr/bin/X11
```

I percorsi elencati, separati dai due punti verticali, rappresentano le directory in cui vengono cercati i file eseguibili quando per questi non è stato specificato il percorso (l'elenco del proprio sistema potrebbe essere molto più lungo).

Generalmente, nel percorso di avvio degli eseguibili è esclusa la directory corrente, che dovrebbe essere rappresentata con un punto singolo, '`.`'; per verificare questa cosa, si può copiare un programma noto nella propria directory personale.

```
$ cd [Invio]
```

```
$ cp /bin/uname ./mio_uname [Invio]
```

A questo punto, nella propria directory personale è stato copiato il programma '**uname**', chiamato localmente '**mio_uname**'. Si può verificare che sia ancora in grado di funzionare:

```
$ ./mio_uname -a [Invio]

  Linux dinkel 2.2.15 #1 Thu Aug 31 15:55:32 CEST 2000 i586 ↩
 ↪unknown
```

Tuttavia, non è possibile avviare il programma senza specificare il percorso:

```
$ mio_uname -a [Invio]

  bash: mio_uname: command not found
```

Esiste un buon motivo per evitare di avviare automaticamente i programmi esistenti nella directory corrente; tuttavia, sarebbe facile includere questa possibilità modificando il contenuto della variabile ***PATH***:

```
$ export PATH="$PATH:." [Invio]

$ echo $PATH [Invio]

  /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:.
```

A questo punto il programma '**mio_uname**' si può avviare automaticamente senza specificare il percorso:

```
$ mio_uname -a [Invio]

  Linux dinkel 2.2.15 #1 Thu Aug 31 15:55:32 CEST 2000 i586 ↩
 ↪unknown
```

Nell'ambito dei percorsi di ricerca elencati nella variabile di ambien-

te ***PATH***, ci potrebbero essere più programmi diversi con lo stesso nome. Per sapere quale di questi viene avviato per primo, basta verificare con '**which**':

```
$ which uname [Invio]

  /bin/uname

$ which mio_uname [Invio]

  ./mio_uname
```

Volendo modificare il nome della propria copia locale del programma, usando lo stesso nome originale, si può verificare quale dei due venga messo in funzione effettivamente, in mancanza dell'indicazione di un percorso:

```
$ mv mio_uname uname [Invio]

$ which uname [Invio]

  /bin/uname
```

Se si modifica l'ordine dei percorsi di ricerca nella variabile di ambiente ***PATH***, si può invertire il risultato. In tal caso si aggiunge la directory corrente all'inizio dell'elenco, restando anche l'indicazione finale che comunque è inutile:

```
$ export PATH=".:$PATH" [Invio]

$ echo $PATH [Invio]

  .:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:.

$ which uname [Invio]
```

```
./uname
```

4.13.2 Comandi interni di shell

Non sempre ciò che si avvia è un programma eseguibile; potrebbe essere un comando interno della shell. Per esempio, nel caso della shell Bash, si può usare il comando '**help**' per ottenere un elenco completo di questi:

```
$ help [Invio]
```

```
GNU bash, version 4.1.5(1)-release (i486-pc-linux-gnu)
These shell commands are defined internally.  Type 'help' to
see this list.
Type 'help name' to find out more about the function 'name'.
Use 'info bash' to find out more about the shell in general.
Use 'man -k' or 'info' to find out more about commands not
in this list.

A star (*) next to a name means that the command is
disabled.

 job_spec [&]                    history [-c] [-d offset] [>
 (( expression ))                if COMMANDS; then COMMANDS>
 . filename [arguments]          jobs [-lnprs] [jobspec ...>
 :                               kill [-s sigspec | -n sign>
 [ arg... ]                      let arg [arg ...]
 [[ expression ]]                local [option] name[=value>
 alias [-p] [name[=value] ..>    logout [n]
 bg [job_spec ...]               mapfile [-n count] [-O ori>
 bind [-lpvsPVS] [-m keymap]>    popd [-n] [+N | -N]
 break [n]                       printf [-v var] format [ar>
 builtin [shell-builtin [arg>    pushd [-n] [+N | -N | dir]>
 caller [expr]                   pwd [-LP]
 case WORD in [PATTERN [| PA>    read [-ers] [-a array] [-d>
```

```
cd [-L|-P] [dir]                 readarray [-n count] [-O o>
command [-pVv] command [arg>     readonly [-af] [name[=valu>
compgen [-abcdefgjksuv] [-o>     return [n]
complete [-abcdefgjksuv] [->     select NAME [in WORDS ... >
compopt [-o|+o option] [-DE>     set [--abefhkmnptuvxBCHP] >
continue [n]                     shift [n]
coproc [NAME] command [redi>     shopt [-pqsu] [-o] [optnam>
declare [-aAfFilrtux] [-p] >     source filename [arguments>
dirs [-clpv] [+N] [-N]           suspend [-f]
disown [-h] [-ar] [jobspec >     test [expr]
echo [-neE] [arg ...]            time [-p] pipeline
enable [-a] [-dnps] [-f fil>     times
eval [arg ...]                   trap [-lp] [[arg] signal_s>
exec [-cl] [-a name] [comma>     true
exit [n]                         type [-afptP] name [name .>
export [-fn] [name[=value] >     typeset [-aAfFilrtux] [-p]>
false                            ulimit [-SHacdefilmnpqrstu>
fc [-e ename] [-lnr] [first>     umask [-p] [-S] [mode]
fg [job_spec]                    unalias [-a] name [name ..>
for NAME [in WORDS ... ] ; >     unset [-f] [-v] [name ...]>
for (( exp1; exp2; exp3 ));>     until COMMANDS; do COMMAND>
function name { COMMANDS ; >     variables - Names and mean>
getopts optstring name [arg>     wait [id]
hash [-lr] [-p pathname] [->     while COMMANDS; do COMMAND>
help [-dms] [pattern ...]        { COMMANDS ; }
```

Dal momento che un sistema ben equipaggiato deve poter consentire l'uso di shell differenti, si affiancano spesso dei programmi eseguibili equivalenti a comandi interni di shell già disponibili. Per esempio, può esistere il programma '**`echo`**', benché la shell Bash lo fornisca come comando interno.

```
$ help echo[Invio]
```

```
echo: echo [-neE] [arg ...]
    Output the ARGs.  If -n is specified, the trailing
    newline is suppressed.  If the -e option is given,
    interpretation of the following backslash-escaped
    characters is turned on:
        \a      alert (bell)
        \b      backspace
        \c      suppress trailing newline
        \E      escape character
        \f      form feed
        \n      new line
        \r      carriage return
        \t      horizontal tab
        \v      vertical tab
        \\      backslash
        \num    the character whose ASCII code is NUM
                (octal).

    You can explicitly turn off the interpretation of the
    above characters with the -E option.
```

Quello che si vede è la sintassi del comando interno '**echo**', mentre il programma '**echo**' può essere leggermente differente:

```
$ /bin/echo --help[Invio]

  Usage: /bin/echo [OPTION]... [STRING]...
  Echo the STRING(s) to standard output.

    -n             do not output the trailing newline
    -e             enable interpretation of the
                     backslash-escaped characters listed
                     below
    -E             disable interpretation of those sequences
                     in STRINGs
```

```
       --help      display this help and exit (should be
                     alone)
       --version   output version information and exit
                     (should be alone)

  Without -E, the following sequences are recognized and
  interpolated:

    \NNN   the character whose ASCII code is NNN (octal)
    \\     backslash
    \a     alert (BEL)
    \b     backspace
    \c     suppress trailing newline
    \f     form feed
    \n     new line
    \r     carriage return
    \t     horizontal tab
    \v     vertical tab

  Report bugs to <bug-sh-utils@gnu.org>.
```

4.13.3 Script e permessi di esecuzione

«

$ Video: *http://www.youtube.com/watch?v=F-4QaR7Qh1o*

In un sistema Unix è facile realizzare dei programmi elementari in forma di raccolta di comandi, ovvero in forma di script di shell. Si può provare a realizzare il file ‘`mio_script`’ nella directory corrente con il contenuto seguente:

```
#!/bin/sh
echo Ciao a tutti!
```

Si tratta di un file di testo che si può costruire facilmente con VI. Se questo file viene salvato con il nome previsto, ovvero

'mio_script', ci si può aspettare di poterlo mettere in funzione con il comando seguente:

```
$ ./mio_script [Invio]

  bash: ./mio_script: Permission denied
```

Come si vede, lo script non viene avviato. Si può verificare che mancano i permessi necessari:

```
$ ls -l mio_script [Invio]

  -rw-rw-r--    1 tizio    tizio    20 mar 24 10:54 mio_script
```

Ecco che basta aggiungere i permessi mancanti e tutto funziona regolarmente:

```
$ chmod +x mio_script [Invio]

$ ls -l mio_script [Invio]

  -rwxrwxr-x    1 tizio    tizio    20 mar 24 10:54 mio_script

$ ./mio_script [Invio]

  Ciao a tutti!
```

4.13.4 Conclusione

Nei sistemi Unix si parla spesso di «comandi», in modo volutamente vago, per non dover specificare se si tratti di eseguibili veri e propri, o di comandi interni all'interprete (la shell).

Se si indica il nome di un comando senza specificare un percorso, si lascia fare la scelta all'interprete dei comandi, per cui viene cercato prima un comando interno, eventualmente un alias, quindi si cerca

un file eseguibile nell'elenco dei percorsi contenuti nella variabile ***PATH***.

Quando si realizza un programma o uno script di shell, occorre ricordare di dare i permessi di esecuzione perché questo possa funzionare.

4.14 Ricerche

« $ Le ricerche di file e directory sono molto importanti in presenza di un file system articolato come quello dei sistemi Unix.

Video: *http://www.youtube.com/watch?v=Iaf5D8gVea8*

4.14.1 Find

« $ Le ricerche di file e directory in base al nome e altre caratteristiche esterne, vengono effettuate attraverso il comando '**find**'.

```
$ find / -name bash -print [Invio]
```

Questo comando esegue una ricerca per i file e le directory denominati 'bash' all'interno di tutte le directory che si articolano a partire dalla radice.

```
/bin/bash
…
find: /var/run/sudo: Permission denied
find: /var/spool/at: Permission denied
find: /var/spool/cron: Permission denied
…
```

Il file viene trovato, ma tutte le volte che '**find**' tenta di attraversare directory per cui non si ha il permesso, si ottiene una segnalazione di errore.

Le ricerche basate sul nome possono impiegare anche modelli con metacaratteri, ma in tal caso deve essere '**find**' a gestirli e non la shell, di conseguenza si deve fare in modo che questa non intervenga.

```
$ find / -name \*sh -print [Invio]
```

L'uso della barra obliqua inversa prima dell'asterisco permette di evitare che la shell tenti di interpretarlo come metacarattere. Alla fine, '**find**' riceve l'argomento corretto, senza barra davanti all'asterisco.

```
/bin/bash
/bin/ash
/bin/sh
…
```

4.14.2 Grep

Per le ricerche all'interno dei file si utilizza '**grep**'.

```
$ grep tizio /etc/* [Invio]
```

```
/etc/group:tizio::500:tizio
/etc/passwd:tizio:Ide2ncPYY1234:500:500:Tizio Tizi:↩
↪/home/tizio:/bin/bash
grep: /etc/skel: Is a directory
grep: /etc/sudoers: Permission denied
…
```

Il risultato che si ottiene dal comando di esempio, sono i nomi dei file contenenti la parola «tizio» e la riga in cui questo appare. Anche in questo caso si possono incontrare file per i quali non si hanno i permessi, o directory, per le quali l'uso di '**grep**' non ha alcun significato.

4.14.3 Conclusione

I comandi '**find**' e '**grep**' sono fondamentali per le ricerche di file con i sistemi Unix. Questi due possono essere anche combinati insieme in modo da definire una ricerca in base a caratteristiche esterne e interne ai file. L'argomento viene trattato nel capitolo 23.

4.15 Memoria di massa e file system con i dischetti

La gestione delle memorie di massa nei sistemi Unix appare piuttosto laboriosa per chi si avvicina la prima volta alla sua filosofia. La sezione 3.3 introduce l'argomento.

In questa sezione si fa riferimento a dischetti, ammesso che se ne abbia ancora la disponibilità. Eventualmente, per questi esercizi i dischetti non devono essere protetti contro la scrittura.

Dal momento che i dischetti sono praticamente scomparsi, questa sezione può essere solo letta, cercando di comprendere gli esempi. Non vengono mostrati casi basati su unità di memorizzazione comuni, perché sarebbe facile inizializzare erroneamente un disco fisso o comunque qualcosa di importante, dato che i file di dispositivo usati per dischi fissi SATA, le unità di memoria solida e i dischi SCSI, sono dello stesso gruppo.

Video: *http://www.youtube.com/watch?v=m2y3N9edoaA*

4.15.1 Inizializzazione a basso livello

LINUX # L'inizializzazione o formattazione di un'unità di memorizzazione di massa, potrebbe richiedere due fasi. Nel caso di unità a disco, secondo la tecnologia degli anni 1980, si richiede un primo processo

di inizializzazione in cui si predispongono le tracce e i settori; in tutti gli altri casi, è sufficiente la sola predisposizione di un file system. La prima fase di inizializzazione è detta anche «inizializzazione a basso livello» e può riguardare soltanto i dischetti usati fino agli anni 1990.

Prima di procedere occorre ottenere i privilegi dell'utente '**root**'.

```
$ su [Invio]

Password: ameba [Invio]
```

Prima di procedere con l'inizializzazione a basso livello si deve verificare il nome del file di dispositivo utilizzato nel proprio sistema, infatti ci possono essere differenze sotto questo aspetto da un'installazione all'altra. Si presume di potere utilizzare dischetti da 3,5 pollici (9 cm) con un formato di 1440 Kibyte.

Si procede con l'inizializzazione a basso livello del dischetto su un sistema GNU/Linux, nel quale si utilizza il file di dispositivo '`/dev/fd0`' per questo:

```
# fdformat /dev/fd0 [Invio]

  Double-sided, 80 tracks, 18 sec/track. Total capacity
  1440 kB.
  Formatting ... done
  Verifying ... done
```

Se non esiste il programma eseguibile '**fdformat**', dovrebbe essere presente al suo posto '**superformat**', che può essere usato nello stesso modo, anche se i messaggi che genera sono differenti.

Se questo è l'esito che si ottiene, il dischetto è stato inizializzato con successo. Prima di procedere oltre è necessario preparare altri due

dischetti.

4.15.2 Predisposizione del file system

«

I dischetti inizializzati a basso livello non sono ancora adatti a contenere dati in forma di directory e file: occorre creare un file system (lo stesso varrebbe per altri tipi di unità di memorizzazione di massa, con la differenza che l'inizializzazione a basso livello non esiste oppure è già fatta e non può essere modificata). Il comando seguente riguarda un sistema GNU/Linux:

```
# mkfs.msdos /dev/fd0 [Invio]

  mkfs.msdos 2.9 (15 May 2003)
```

In questo modo è stato creato un file system di tipo Dos-FAT nel dischetto inizializzato precedentemente a basso livello. Il messaggio che si ottiene può variare da un'installazione di GNU/Linux a un'altra, ma questo non è molto importante.

Dopo avere sostituito il dischetto si esegue il comando seguente allo scopo di creare un file system Minix.

```
# mkfs.minix -n 14 /dev/fd0 [Invio]

  480 inodes
  1440 blocks
  Firstdatazone=19 (19)
  Zonesize=1024
  Maxsize=268966912
```

Dopo avere sostituito il dischetto si esegue il comando seguente allo scopo di creare un file system Ext2. Si osservi che in un dischetto non è possibile creare le estensioni relative al formato Ext3, per mancanza di spazio.

```
# mkfs.ext2 /dev/fd0 [Invio]

  Filesystem label=
  OS type: Linux
  Block size=1024 (log=0)
  Fragment size=1024 (log=0)
  184 inodes, 1440 blocks
  72 blocks (5.00%) reserved for the super user
  First data block=1
  1 block group
  8192 blocks per group, 8192 fragments per group
  184 inodes per group

  Writing inode tables: done
  Writing superblocks and filesystem accounting information:
  done

  This filesystem will be automatically checked every 33
  mounts or 180 days, whichever comes first.  Use tune2fs -c
  or -i to override.
```

Per proseguire l'esercizio si devono distinguere i tre dischetti appena preparati, in modo da sapere riconoscere quale utilizza il file system Dos-FAT, quale quello Minix e quale quello Ext2.

4.15.3 Innestare e staccare i dischetti

Nei sistemi Unix e derivati, per poter accedere a un'unità di memorizzazione occorre che il file system di questa sia ***innestato*** («montato») in quello globale. Non si può indicare semplicemente una directory o un file di un certo dispositivo. L'innesto è l'operazione con cui si inserisce il file system di un'unità di memorizzazione in corrispondenza di una directory del file system già attivo. La directory che si utilizza generalmente per innestare provvisoriamente le unità

esterne è ‘/mnt/’ (ma in molti casi si usa una struttura più articolata, discendente dalla directory ‘/mnt/’ stessa). Gli esempi di questa sezione si riferiscono a un sistema GNU/Linux.

Si procede inserendo il dischetto inizializzato con il formato Ext2 e innestandolo nella directory ‘/mnt/’.

```
# mount -t ext2 /dev/fd0 /mnt [Invio]
```

Da questo momento, la directory ‘/mnt/’ è l’inizio del dischetto.

```
# touch /mnt/super.ultra.mega.macro [Invio]
```

Con il comando appena visto, si vuole creare un file vuoto con un nome piuttosto lungo. Volendo si possono copiare dei file nel dischetto.

```
# cp /bin/bash /mnt [Invio]
```

Solitamente, l’invito della shell torna a disposizione prima che le operazioni di scrittura siano state completate, a causa della presenza di una «memoria trattenuta», o più precisamente di una memoria cache. Anche per questo motivo, il dischetto non può essere estratto semplicemente alla fine delle operazioni che con questo si vogliono svolgere.

Finché il dischetto risulta innestato, si può trattare come parte del file system globale.

```
# ls -l /mnt [Invio]

total 418
-rwxr-xr-x 1 root root 412516 Dec 28 13:10 bash
drwxr-xr-x 2 root root  12288 Dec 28 12:37 lost+found
-rw-r--r-- 1 root root      0 Dec 28 13:05 super.ultra.mega.macro
```

Si può osservare che il dischetto contiene il file creato all'inizio, l'eseguibile '**bash**' copiato in precedenza e una directory particolare: '`lost+found/`'. Questa viene creata automaticamente assieme al file system Ext2. Generalmente può essere cancellata se la sua presenza infastidisce. In un certo senso, la presenza di questa directory è utile per scorgere l'inizio di un file system innestato in quel punto particolare.

Si procede sostituendo il dischetto con quello contenente un file system Minix. Per fare questo occorre prima staccare il dischetto inserito attualmente e quindi innestare il secondo.

```
# umount /mnt [Invio]
```

A questo punto, al ritorno dell'invito della shell, si possono sostituire i dischetti.

```
# mount -t minix /dev/fd0 /mnt [Invio]
```

Per esercizio, si fanno le stesse operazioni di prima.

```
# touch /mnt/super.ultra.mega.macro [Invio]

# cp /bin/bash /mnt [Invio]

# ls -l /mnt [Invio]

 total 404
 -rwxr-xr-x  1 root  root  412516 Dec 28 13:31 bash
 -rw-r--r--  1 root  root       0 Dec 28 13:31 super.ultra.me
```

Come si può osservare, il file system Minix (precisamente si tratta di Minix 1) non prevede la presenza della directory '`lost+found/`' e consente l'uso di nomi con un massimo di 14 caratteri.

```
# umount /mnt [Invio]
```

A questo punto si ripetono le stesse cose con il dischetto Dos-FAT.

```
# mount -t msdos /dev/fd0 /mnt [Invio]

# touch /mnt/super.ultra.mega.macro [Invio]

# cp /bin/bash /mnt [Invio]

# ls -l /mnt [Invio]

 total 403
 -rwxr-xr-x   1 root     root   412516 Dec 28 14:02 bash
 -rwxr-xr-x   1 root     root        0 Dec 28 14:01 super.ult
```

Trattandosi di un dischetto con un file system Dos-FAT, le cose non sono andate come in precedenza. Prima di tutto, i permessi dei file non corrispondono agli esempi già visti: in pratica, tutti i file hanno gli stessi permessi. L'utente proprietario di tutti i file è '**root**' essendo stato lui a innestare il dischetto. I nomi dei file vengono troncati.

Volendo utilizzare un dischetto Dos-FAT per memorizzare nomi lunghi, nello stesso modo in cui fa MS-Windows lo si può innestare facendo riferimento al tipo di file system '**vfat**', mentre l'inizializzazione del dischetto avviene sempre nello stesso modo.

Prima di concludere l'esercizio, si stacca il dischetto.

```
# umount /mnt [Invio]
```

È il caso di ricordare che non è possibile staccare un disco se prima non è terminata l'attività con questo. Per cui, se la directory corrente è posizionata su una directory appartenente al file system del disco che si vuole staccare, non si riesce a eseguire l'operazione di distacco.

4.15.4 Conclusione

La gestione delle unità di memorizzazione può sembrare complicata fino a che non se ne comprende la logica. La cosa più importante da capire è che non si può accedere al contenuto di un disco o di altra unità di memorizzazione se prima questi non vengono innestati, così come non si può estrarre un disco o un'altra unità se prima non è stato eseguito il distacco.

Nel capitolo 19 vengono trattati questi argomenti.

È il caso di ricordare che l'esercizio è stato svolto operando come utente '**root**', per cui, prima di proseguire, è meglio ritornare allo stato normale.

```
# exit [Invio]
```

4.16 Dispositivi

Nei sistemi Unix, i dispositivi sono manifestati da file speciali collocati nella directory '/dev/'. L'utilizzo diretto dei dispositivi è spesso un'operazione delicata, che può essere eseguita solo dall'utente '**root**'. Alcuni esercizi di questa sezione vanno svolti come utente '**root**' e in tal caso si può notare l'invito della shell, che negli esempi viene rappresentato dal cancelletto (il simbolo '**#**').

4.16.1 File «/dev/null»

«

$ Il file di dispositivo ‘`/dev/null`’ corrisponde in lettura a un file vuoto e in scrittura a una sorta di buco senza fondo: tutto ciò che vi viene scritto è perduto. Questa particolarità è molto utile negli script in cui si vuole evitare che i comandi contenuti emettano segnalazioni all’utente.

```
$ ls /bin > /dev/null [Invio]
```

Il comando appena mostrato non emette nulla sullo schermo perché tutto viene ridiretto verso ‘`/dev/null`’. Si può verificare che in questo file non ci sia più alcuna traccia con il comando seguente:

```
$ cat /dev/null [Invio]
```

Non si ottiene alcun output.

4.16.2 Dispositivi di memorizzazione

«

La gestione diretta dei dispositivi di memorizzazione è un’operazione delicata e richiede i privilegi dell’utente ‘**root**’.

```
$ su [Invio]

Password: ameba [Invio]
```

I dispositivi di memorizzazione possono essere gestiti come se fossero dei file. In pratica, un dischetto da 1 440 Kibyte può essere trattato come se fosse un file della stessa dimensione.

Nell’esercizio sulle unità di memorizzazione sono stati inizializzati alcuni dischetti e vi è stato copiato dentro qualcosa.

Si procede in modo da generare un file-immagine di un dischetto di quelli preparati in precedenza. Si inserisce uno di quei dischi.

```
# cp /dev/fd0 disco.img [Invio]
```

Il dischetto di cui è stata fatta la copia in un file-immagine non è stato innestato in precedenza e tuttora non risulta innestato.

```
# ls -l disco.img [Invio]

-rw-r-----   1 root   root    1474560 Dec 28 14:59 disco.img
```

Volendo eseguire la copia del dischetto a cui appartiene questa immagine, basta sostituirlo con un altro che sia già stato inizializzato a basso livello (con '**fdformat**' per esempio, o con un altro sistema operativo con gli strumenti che questo mette a disposizione) e quindi copiare il file-immagine sul file di dispositivo corrispondente al dischetto.

Si sostituisce il dischetto e si procede.

```
# cp disco.img /dev/fd0 [Invio]
```

Il dischetto che si ottiene contiene la copia identica di quello di partenza.

4.16.3 Conclusione

L'accesso diretto ai file di dispositivo è un metodo utilizzato particolarmente per la riproduzione di dischi (prevalentemente dischetti o unità di memorizzazione di capacità ridotta) in modo da conservare tutte le informazioni in essi contenuti.

[1] È il caso di ricordare che, durante il funzionamento del sistema grafico, per tornare a una console virtuale non è sufficiente la combinazione [*Alt F**n***], ma serve invece la combinazione [*Ctrl Alt F**n***].

[2] In tal caso si tratta necessariamente di un percorso relativo

[3] Negli esempi che si vedono si presume di poter entrare nella directory personale di un altro utente. Tuttavia, tale possibilità dipende dai permessi che questo gli attribuisce.

[4] La combinazione di tasti [*Ctrl d*] termina l'inserimento di un testo se ciò viene fatto dopo aver concluso l'ultima riga, premendo [*Invio*], diversamente, la combinazione [*Ctrl d*] deve essere premuta due volte di seguito.

[5] Per la precisione, sarebbe più opportuno l'uso del comando '**`echo`**', che però non è ancora stato mostrato.