Corso di laurea in Ingegneria Informatica

Corso di Sistemi Operativi

Prof. G. Piscitelli Ing. M. Ruta

Linux e la Shell Bash

2.0

Emanuele Mottola **Pietro Russo**

Copyright, Licenza, Termini d'uso

Prefazione

Il presente documento si propone di raccogliere e spiegare i comandi più utilizzati e le relative nozioni, per capirne il funzionamento e la logica, al fine di utilizzare la shell Bash su un sistema GNU/Linux.
Tutto il contenuto del documento, i comandi, la teoria e gli esempi si riferiscono all'uso della shell Bash sul sistema operativo GNU/Linux, che è l'*opera collettiva* frutto della più grande, complessa e duratura *collaborazione volontaria* mai realizzata nella storia dell'uomo.

Nello sviluppo di questo testo è stato utilizzato un sistema GNU/Linux con kernel della serie 2.6.x e una shell Bash versione 3.2_p33.
Tuttavia salvo piccoli dettagli quanto scritto è applicabile anche ai sistemi con versione di kernel Linux e shell Bash differente, senza nessuna modifica sostanziale.
Questo documento è stato interamente realizzato utilizzando la suite freeware opensource OpenOffice.org, in particolare il programma Writer.
Per l'esportazione nel formato *pdf* si è adoperato il tool integrato in OpenOffice.org.
Questo documento è stato elaborato su sistemi Gentoo GNU/Linux, utilizzando quindi solo software opensource, senza violare alcun copyright.

Nel documento col termine *shell* si fa riferimento alla shell *Bash*, con il termine *superuser* si indica equivalentemente l'amministratore di sistema cioè l'utente *root*.
Si utilizza inoltre il termine *Linux* come abbreviativo di *GNU/Linux*.
Ben consapevoli che Linux è in realtà il nucleo (kernel) del sistema operativo, dell'importanza ed essenzialità sia storica che tecnica delle librerie e software GNU, si è utilizzata questa semplificazione al solo fine di rendere più leggibile il testo.

Fonte iniziale di questo testo è stato il corso di Sistemi Operativi del **Prof. Giacomo Piscitelli** tenuto presso il Politecnico di Bari nell'A.A. 2006/07, con il contestuale laboratorio di GNU/Linux tenuto dall'**Ing. Michele Ruta**.
Inoltre esprimiamo tutta nostra stima e riconoscenza a tutti coloro che hanno contribuito allo sviluppo e diffusione di GNU/Linux e dell'opensource, da chi scrive codice a chi mette le proprie conoscenze al servizio degli altri su IRC e forum.

INDICE GENERALE

INDICE ANALITICO DEI COMANDI

LA SHELL

INTRODUZIONE

La shell è l'*interprete dei comandi* di un sistema operativo, un programma, cioè, che permette ad un utente di interagire con un sistema operativo leggendo ed interpretando i comandi che vengono inseriti dall'utente.

Dal punto di vista pratico, il funzionamento di un sistema *Unix-like* dipende molto dalla shell utilizzata, di conseguenza, la scelta di quest'ultima è molto importante.
La shell tipica dei sistemi Unix è una shell derivata da quella di Bourne, possibilmente aderente allo standard POSIX: la shell BASH.

Tecnicamente Bash, acronimo di ***B****ourne* ***a****gain* ***sh****ell*, è un clone evoluto della shell standard di Unix (*/bin/sh*), chiamata Bourne shell dal nome del suo autore originario. Bash è la shell ufficiale del progetto GNU, conforme allo standard POSIX, usata nei sistemi operativi Unix e specialmente in GNU/Linux.

Una shell Unix svolge, di base, i compiti seguenti:

- Mostra l'invito, o prompt, all'inserimento dei comandi
- Interpreta la riga di comando data dall'utente
- Esegue delle sostituzioni, in base ai caratteri jolly e alle variabili di ambiente
- Mette a disposizione alcuni comandi interni
- Permette l'esecuzione di programmi
- Gestisce la ridirezione dell'input e dell'output
- É in grado di interpretare ed eseguire dei file script in linguaggio di shell
- Fornisce la cronologia dei comandi immessi
- Dispone della funzione di completamento automatico
- Permette il controllo dei processi.

GNU/Linux è, tra l'altro, un sistema multiutente e multitasking. Multiutente indica che può essere utilizzato da più utenti contemporaneamente, attraverso vari terminali; multitasking significa invece che permette l'esecuzione di più processi contemporaneamente. Da ciò consegue che sul sistema possono essere attive simultaneamente diverse shell.

La shell può essere ampiamente personalizzata modificando le informazioni fornite da prompt, i colori, definizione *statica* degli alias e molto molto altro.
Ogni utente di un sistema Linux ha un file di configurazione (nascosto) per questo scopo, cioè *.bashrc* oppure *.bash_profile,* a seconda delle distribuzione utilizzata. Questi file non sono altro che script di configurazione, contengono cioè una serie di comandi che verranno eseguiti al login, settando le variabili d'ambiente come il prompt o i colori, e quanto altro specificato.
Modificando questo file è possibile quindi adattare il più possibile la shell alle proprie esigenze e abitudini, riutilizzando il file di configurazione su altri sistemi.

Sempre nella home-directory di ogni utente è presente un file (nascosto) che contiene la cronologia dei comandi immessi: *.bash_history*.

TERMINALE

In campo informatico, un terminale è un dispositivo hardware elettronico o elettromeccanico che viene usato per inserire dati all'interno di un computer o di un sistema di elaborazione e riceverli per la visualizzazione.
Per estensione, il termine indica anche il dispositivo equivalente realizzato dal sistema operativo o tramite un emulatore software (terminale *virtuale*).

Il concetto di terminale, nelle sue diverse implementazioni, costituisce l'interfaccia tra le applicazioni con interfaccia testuale e i molteplici dispositivi di visualizzazione che queste possono usare. I primi terminali erano *telescriventi* (*TTY*), ovvero apparecchiature che stampavano i risultati dell'elaborazione su carta; successivamente si sono diffuse versioni a schermo.

A conferma di ciò i terminali nei sistemi Unix e derivati vengono tutt'ora identificati con l'acronimo *TTY* seguito dal numero del terminale (es. *tty3*). Ovviamente i anche i terminali di Linux sono terminali virtuali, quindi si utilizza comunemente la nozione *terminale* come abbreviativo di *terminale virtuale.*
Si può definire quindi un terminale, in Unix, come un'interfaccia, testuale o grafica, tramite la quale gli utenti possono interagire col sistema.
Normalmente una distribuzione Linux per sistemi desktop avvia sette terminali di cui i primi sei sono shell testuali, mentre il settimo è dedicato al terminale che ospita la sessione grafica. Quando si avviano più sessioni grafiche, queste risiederanno nei terminali grafici successivi al settimo.

Ci si può spostare tra i terminali premendo **ALT**+**Fx**, dove "***x***" è il numero del tasto funzione sulla tastiera e "**+**" significa *contemporaneamente*.
Se si è nella sessione grafica, cioè si utilizza un'istanza del *server X*, ci si può recare in un altro terminale aggiungendo alla sequenza precedente il tasto **CTRL**, poiché le combinazioni **ALT+Fx** sono utilizzate per altri scopi.
Riepilogando quindi:

- **ALT**+**Fx** Per muoversi tra i terminali
- **CTRL+ALT**+**Fx** Per spostarsi dalla sessione grafica ad un altro terminale.

Nella sessione grafica è possibile utilizzare il terminale tramite un programma che lo emuli, consentendo le stesse operazioni del terminale *standard*.
Utilizzando un programma di emulazione terminale, che è largamente personalizzabile, si ha la possibilità di utilizzare anche il mouse per inserire comandi ed estrarre del testo, ma si ha una velocità d'esecuzione inferiore rispetto al terminale *standard,* dove è comunque possibile utilizzare il mouse con opportune impostazioni di *termcap*.
I programmi di emulazione terminale più utilizzati nella sessione grafica sono:

- *xterm* Lo standard, disponibile su tutti i desktop environment
- *gnome-terminal* Programma terminale del desktop Gnome
- *kterm* Programma terminale del desktop Kde.

Con il mouse si può inserire del testo nella shell o estrarlo per copiarlo da qualche altra parte, anche nella shell stessa o in un editor di testi. Ciò che si sottolinea viene automaticamente copiato mentre per incollare quanto copiato si utilizza il tasto centrale del mouse, in assenza emulato premendo insieme i tasti sinistro e destro.

TIPI DI COMANDI

Esistono tre tipi di comandi che la shell può eseguire:

- **interni** (**built-in**) Funzioni elementari eseguite direttamente dalla shell, la loro esecuzione non provoca la creazione di un nuovo task poiché il codice fa parte dell'eseguibile stesso della shell
- **esterni** File contenenti programmi in formato eseguibile la cui esecuzione provoca la creazione di un nuovo task *figlio* della shell da cui sono avviati
- **script** File testuali in linguaggio shell, con permessi di esecuzione, contenente una serie di istruzioni e comandi interpretabili dalla shell.

Quando si digita un comando, la shell verifica se si tratta di un comando interno (built-in). In caso positivo lo esegue, altrimenti crea un nuovo processo che esegue il comando. Questo processo (task) sarà *figlio* della shell da cui è stato eseguito il comando; nella sezione “Gestione Processi” ci sono maggiori dettagli.

METACARATTERI

La shell Bash mette a disposizione dei metacaratteri per poter lavorare con file e directory, ecco come vengono tradotti:

- ***** Una qualsiasi stringa di caratteri
- **?** Un solo carattere, qualsiasi
- **[a-h]** Un carattere compreso, nel set ASCII, tra '**a**' ed '**h**', cioè tra i valori ASCII di **a** e **h**
- **[aei]** Un carattere che sia tra '**a**' o '**e**' o '**i**', i caratteri sono in OR tra loro.
- **[!aei]** Un carattere che non sia '**a**' o '**e**' o '**i**'.

Altri metacaratteri sono:

- **>** Ridirezione dello standard output su file
- **<** Ridirezione dello standard input su file
- **>>** Ridirezione in modalità *append* dello standard output su file
- **&** Richiesta di esecuzione di un comando in background, il simbolo va messo subito dopo il comando
- **;** separatore di comandi all'interno di una stessa riga.

Il metacarattere punto e virgola (**;**) serve a separare i comandi permettendo di eseguire più comandi in maniera sequenziale, cioè terminato uno inizia l'esecuzione del successivo.
La ridirezione ed i processi in background sono trattati nelle relative sezioni. Notare che i metacaratteri, anche se utilizzano notazione simile alle espressioni regolari, sono diversi da queste, poiché i metacaratteri vengono interpretati diversamente. Questi servono normalmente per lavorare con file e directory, e sono interpretati ed utilizzati dalla direttamente dalla shell, mentre le espressioni regolari servono per definire un modello complesso di ricerca sopratutto all'interno dei file, e vengono utilizzate da particolari comandi chiamati *filtri*, che svolgono funzioni di ricerca e filtraggio dell'input.

FLAG

I flag, uno dei concetti fondamentali per l'utilizzo dei comandi nella shell, sono delle opzioni aggiuntive con cui eseguire un comando, per modificarne l'effetto rispetto a quello di default. Questi flag non sono altro che lettere che si inseriscono dopo il nome del comando e separate da spazi.
Normalmente l'ordine relativo di inserimento dei flag è indifferente.

Di solito ogni comando ha dei suoi flag, ed ogni flag può essere:

- **semplice** Una lettera, in genere preceduta da un trattino (es. -**R**)
- **estesa** Una parola preceduta da due trattini (es. **--all**); oppure più parole, sempre precedute da due trattini ma separate tra loro da un trattino, senza spazi (es. --**no-spinner**)
- **con argomento** Un flag semplice o estesa con in aggiunta un argomento di seguito (es. -**f** *file*, oppure **--max-depth** *3*).

I flag, i comandi e tutto il resto del mondo Unix, sono *case sensitive*, cioè **X** è ben diverso da **x**. Non è raro che un comando abbia flag maiuscoli e minuscoli dello stesso carattere ma con effetto totalmente differente.
Esistono, per alcuni comandi, anche dei flag letterali che si usano senza anteporre il trattino, quindi bisogna prestare molta attenzione alle pagine di manuale.
Per usare più flag contemporaneamente, si può specificarle tutte separandole da uno o più spazi (es. **-X -y -Z**) oppure con un solo trattino iniziale se il comando prevede i flag anteponendo il trattino, ma in ogni caso senza spazi tra di loro (es. **-XyZ**).

LE PIÙ FREQUENTI

Ci sono alcuni flag che vengono utilizzate in maniera identica da tantissimi comandi.
Flag equivalenti che richiedono la visualizzazione dell'*help* del colmando:

- **--help** Presente normalmente in ogni comando
- **-h**
- **-H**

I comandi che accettano necessariamente un argomento su cui operare, se lanciati senza l'argomento, mostrano (oltre che un errore di mancato inserimento) l'help dei comandi, esattamente come con il flag –help.

Flag equivalenti che richiedono un output (sullo standard output) più dettagliato delle operazioni eseguite:

- **--verbose**
- **-v**

Flag frequenti, il primi due (-R e -f) specie nei comandi che gestiscono i file.

- **-R** (**--recursive**) richiede di eseguire l'operazione ricorsivamente all'interno delle eventuali subdirectory
- **-f** (**--force**) forza l'esecuzione dell'operazione in presenza di file o directory con lo stesso nome
- **-a** (**--all**) nei comandi che mostrano informazioni su *qualcosa*, è usata per utilizzare insieme tutti i possibili flag.

Inoltre, nella maggior parte dei comandi esterni è spesso presente il flag **--version** che mostra la versione del programma, anziché eseguirlo.

SCORCIATOIE

La shell Bash offre diverse scorciatoie per renderne l'utilizzo decisamente più agevole e veloce:

- Tasto **Tab** Completa ciò che si sta scrivendo, se è univoco
- Tasti **Freccia** Il tasto **su** per navigare nell'elenco dei comandi digitati, tornando indietro cronologicamente, **giù** per scorrere in avanti la cronologia. Viene mostrata una riga per volta
- **CTRL**+**R** Cerca dinamicamente nella history (trattata in seguito) le righe che contengono la stringa digitata
- **CTRL**+**H** Cancella il carattere a sinistra del cursore, equivalente del tasto backspace
- **CTRL**+**W** Cancella l'ultima parola inserita
- **CTRL**+**U** Cancella l'intera riga
- **CTRL**+**L** Pulisce lo schermo (si può anche digitare il comando **clear**).

Conviene abituarsi presto ad un uso *intensivo* della funzione di autocompletamento, mediante la pressione del tasto Tab. Oltre che a velocizzare enormemente la scrittura di comandi e percorsi, questa la funzione permette di non sbagliare mai il nome di un comando o di un percorso, poiché se viene completato col tasto Tab, allora è sicuramente corretto, viceversa c'è di certo qualcosa di sbagliato.
Se il tasto Tab non completa il comando o il percorso vuol dire che i caratteri fin ora digitati non rendono univoca la scelta del percorso o del comando, oppure che non esiste un comando o percorso che comincia per i caratteri fin ora digitati.
Premendo il tasto Tab due volte di seguito, come se si facesse *doppio click,* verranno mostrate, se presenti, le possibili scelte disponibili, sempre tenendo in partendo da quanto digitato finora.

La shell riesce a completare il nome di un comando perché nella variabile d'ambiente PATH (si veda la sezione apposita) sono definiti tutti i percorsi dei programmi eseguibili a cui si vuole fare accesso scrivendone anche solamente il nome, cioè dei comandi. Quindi la shell non completa genericamente tutti i nomi dei comandi, ma solo quelli che sono nelle locazioni specificate nella variabile PATH.
E' ovvio che si può sempre specificare il percorso completo.
Ad esempio se si scrive un qualsiasi percorso fino ad una directory e si preme due volte di seguito Tab, verranno mostrati i file in essa contenuti. Se invece si aggiunge allo stesso percorso una lettera, e si preme due volte il tasto Tab, verranno mostrati tutti i file che iniziano per quella lettera, se ne esistono.
E' chiaro che una volta capito il meccanismo non bisognerà più ricordare quasi niente a memoria, al massimo le iniziali dei percorsi o dei comandi, poi basta il tasto Tab e eventualmente l'help dei comandi per trovare rapidamente ciò che serve.

Esistono inoltre delle scorciatoie relative alla gestione dei processi:

- **CTRL+C** Termina forzatamente l'esecuzione del processo
- **CTRL+Z** Ferma temporaneamente il processo, cioè lo mette in background
- **CTRL+D** Termina l'input (esce dalla shell se la si sta usando)
- **CTRL+S** Interrompe l'output a schermo
- **CTRL+Q** Riattiva l'output a schermo.

Si veda la sezione relativa alla gestione dei processi per maggiori dettagli.

SINTASSI DEL DOCUMENTO

Di seguito è illustrato il modello utilizzato per rappresentare le informazioni riguardo ad un generico comando.

- **comando**
  (eventuale significato acronimo)
  Breve descrizione di quello che fa il comando specificato.

  SINTASSI
  - comando [-flag] *argomento* [*file*]

  FLAG
  - **-X** Descrizione dell'effetto del flag **X** sul comando
  - **-t** *stringa* Descrizione dell'effetto del flag **t** seguita dal suo argomento *stringa.*

  ESEMPI
  1. **comando** -X *argomento*
     Descrizione dell'effetto del comando con il flag x.

La voce *SINTASSI* descrive il modo, anche più di uno, con cui si può utilizzare **comando**. Rifacendoci allo standard di visualizzazione delle pagine di manuale nei sistemi Linux, tutto ciò che in questo documento è racchiuso tra [parentesi quadre] è da intendersi come opzionale.
L'*argomento* del comando o di un suo flag è in *corsivo*, cosi come lo è l'eventuale *file*, con o senza il suo *path* (percorso) completo */percorso/file*. L'argomento è in genere una stringa se si tratta di comandi che si comportano da filtri o nella gestione degli utenti, oppure un file, completo del suo percorso (assoluto o relativo) se si tratta di comandi che operano su file o directory.

La voce *FLAG* elenca le opzioni maggiormente utilizzate del rispettivo comando, secondo l'esperienza degli autori. Tuttavia la maggior parte dei comandi descritti in questo documento contengono altri flag, per un utilizzo più approfondito è sempre necessario consultare l'unica fonte esaustiva: la **pagina man** del comando.
Spesso anche i flag hanno argomenti, questo accade quando il comando può fare un'azione abbastanza complessa e quindi avere più argomenti per le varie opzioni disponibili.

In questo documento sono spesso forniti *ESEMPI* di utilizzo dei comandi.
In genere gli esempi sono omessi quando si tratta di comandi semplici o che non hanno particolarità di utilizzo.
Gli esempi forniti risultano altre sì utili per comprendere sia le sottili differenze che possono portare ad un comportamento molto diverso del comando in questione, sia i vari modi di ottenere lo stesso effetto.
Quando si è ritenuto importante ai fini della comprensione specificare l'output che il comando mostra sulla shell, esso è mostrato dopo il relativo esempio, riga per riga, sotto la voce *OUTPUT.*
Nel caso di comandi interattivi è presente un breve elenco sotto la voce *COMANDI.*

DOCUMENTAZIONE IN LINEA

- **man**

  (manual)
  Formatta e mostra le pagine di guida in linea del comando specificato. Le pagine di manuale hanno delle sottosezioni, i cui nomi e ordine sono stabiliti per tutte le pagine. L'ordine in genere è:
  - *NAME* Nome comando e breve descrizione
  - *SYNOPSIS* Sintassi del comando
  - *DESCRIPTION* Descrizione dettagliata
  - *OPTIONS* Lista di tutte i flag con relativa descrizione.

  Diversi file di configurazione del sistema hanno una loro pagina di manuale accessibile digitando *man nome_file_configurazione.*

  SINTASSI
  - ***man*** -[fkKw] [*sezione o* -S *sezione*] [-P *paginatore*] [-m *sistema*] [-p *stringa*] *nomecomando*

  FLAG
  - **-f** Equivalente a **whatis**
  - **-k** Equivalente ad **apropos**
  - **-K** Ricerca una stringa specifica in *tutte* le pagine della guida, può risultare lento
  - **-w** Non mostra subito la pagina di guida, ma stampa la posizione dei file che dovrebbero essere formattati o mostrati:
    - Equivalente a --path
  - **-P** Specifica il programma di impaginazione da usare. Per default il comando **man** usa */usr/bin/**less-is***
  - **-S***x* Mostra la pagina di manuale della sezione *x,* con *x* numero intero*.* Di seguito le sezioni:
    1 - è quella relativa ai comandi per l'utente
    2 - alle chiamate di sistema
    3 - alle librerie delle funzioni C
    4 - ai devices e file speciali
    5 - ai formati dei file e convenzioni
    6 - ai giochi
    7 - ad altri argomenti
    8 - a strumenti per la manutenzione del sistema e ai demoni.
    La sezione può essere anche specificata solo con ***-x.***

  ESEMPI
  1. **man** *comando1*
     Mostra le pagine di manuale di *comando1*
  2. **man** -S1 *comando1*
     Mostra la sezione 1 del manuale di *comando1*
  3. **man** -1 *comando1*
     Equivalente all'esempio precedente.

- **info**

  Mostra la documentazione in formato *Info* del comando specificato. Info è stato pensato per sostituire man, ma man è ancora molto usato.

  SINTASSI
  - **info** *nomecomando*

- **whatis**

  Ricerca una parola chiave nel suo database contenente brevi descrizioni dei comandi di sistema, ossia solo i campi NOME (NAME) delle rispettive pagine man di tutti i comandi. Ne mostra quindi i risultati sullo standard output, cercando solo le corrispondenze esatte della parola chiave. Il database di whatis è creato usando il comando **makewhatis** *(/usr/sbin/makewhatis*).

  ESEMPIO
  1. **whatis** *man*

  OUTPUT
  - man (1) - format and display the on-line manual pages
  - man (1p) - display system documentation
  - man (7) - macros to format man pages
  - man.conf [man] (5) - configuration data for man

- **apropos**

  Opera come *whatis*, la differenza è che mostra tutte le corrispondenze trovate, non solamente quella esatta nel nome del comando, ma anche le parole che contengono la stringa specificata, sia nel nome del comando sia nella loro descrizione.

  ESEMPI
  1. **whatis** *grep*

  OUTPUT
  - grep (1) - print lines matching a pattern
  - grep (1p) - search a file for a pattern

  2. **apropos** *grep*

  OUTPUT
  - egrep [grep] (1) - print lines matching a pattern
  - fgrep [grep] (1) - print lines matching a pattern
  - grep (1) - print lines matching a pattern
  - grep (1p) - search a file for a pattern
  - msggrep (1) - pattern matching on message catalog
  - pgrep (1) - look up or signal processes based on name

- **history**

  Mostra sullo standard output lo storico dei comandi, di default gli ultimi 500, digitati nella shell dall'utente correntemente *loggato*, numerandoli progressivamente.
  Permette inoltre molte funzioni di manipolazione ed elaborazione della cronologia dei comandi.

ESEMPI

1. **history**
   Mostra gli ultimi 500 comandi digitati
2. **history -c**
   Elimina la cronologia dei comandi, ricominciando la numerazione.

E' possibile utilizzare nella shell delle scorciatoie collegate a history:

- **!n** Per riutilizzare il comando **n**-esimo nella history
- **!-n** Per riutilizzare il comando già usato **n** comandi prima
- **!string** Per riutilizzare il comando più recente contenente *sting*

- **uname**
  Mostra alcune informazioni sul sistema.

SINTASSI

- ***man*** -[a] -[snrvmpio]

FLAG

- **-a** (**--all**) Mostra tutte le informazioni dei flag successivi, nell'ordine in cui sono indicate i flag
- **-s** (**--kernel-name**) Nome del kernel in uso
- **-n** (**--nodename**) *Hostname* del calcolatore, il nome del suo nodo nella rete
- **-r** (**--kernel-release**) Release del kernel in uso
- **-v** (**--kernel-version**) Versione del kernel in uso
- **-m** (**--machine**) Nome dell'architettura della CPU (es. i686)
- **-p** (**--processor**) Tipo di processore o “unknown” se non viene riconosciuto automaticamente
- **-i** (**--hardware-platform**) Tipo di piattaforma hardware o “unknown” se non viene riconosciuta automaticamente
- **-o** (**--operating-system**) Sistema operativo (es. GNU/Linux).

ESEMPI

1. **uname -p**

OUTPUT

AMD Athlon(tm) 64 X2 Dual Core Processor 6200+

2. **uname -roms**

OUTPUT

- Linux 2.6.27-gentoo-r2 x86_64 GNU/Linux
  Notare che l'ordine di inserimento dei flag non viene rispettato nell'output, poiché le informazioni vengono sempre elencate nell'ordine in cui sono stati elencati sopra i flag, indipendentemente quindi dall'ordine con cui sono utilizzate nel comando.

PERMESSI DI ACCESSO

TEORIA

In Unix *tutto è un file*, directory comprese, ed ogni file ha una **ACL** (Access Control List), una stinga di dieci caratteri logicamente ordinati che definisce i permessi di accesso al file.
Ogni file ha, inoltre, un utente proprietario ed un gruppo proprietario.
Al momento della creazione di un file, esso viene collegato all'utente che lo ha generato e al gruppo principale a cui appartiene l'utente.

Una ACL potrebbe essere: **d rwx r-x r--**
La stringa è logicamente suddivisa in 4 parti: il primo carattere e a seguire tre blocchi da tre caratteri, anch'essi ordinati.

Il primo carattere dell'ACL descrive i tipo di file:

- **d** Directory
- **-** File ordinario
- **b** **File speciale a blocchi**, è un tipo di file che fornisce un buffer I/O di accesso per dispositivi di dimensione fissa come gli hard disk
- **c** **File speciale a caratteri**, non fornisce un buffer I/O di accesso (il trasferimento dati va verso o proviene direttamente dal disco), ha dimensione variabile per ogni dispositivo
- **l** Link simbolico, a cui è dedicato l'omonima sezione, realizza collegamenti tra file
- **p** Per una **FIFO**(chiamate anche *pipe con nome*), hanno le stesse caratteristiche delle pipe (vedi sezione sul concatenamento dei comandi), ma sono accessibili attraverso un inode sul filesystem
- **s** Per un **socket**. Questo è un canale di comunicazione fra due processi, su cui si possono leggere e scrivere dati analogo a quello di una pipe, ma non sono limitati alla comunicazione fra processi che girano sulla stessa macchina, è possono realizzare la comunicazione anche attraverso la rete.

I permessi di accesso in Unix e derivati sono definiti per tre soggetti:

1. utente proprietario del file (o directory)
2. gruppo proprietario del file (o directory)
3. tutti gli altri utenti.

I permessi di ognuno di questi soggetti sono descritti appunto da uno dei blocchi di tre caratteri, nell'ordine specificato.

Ogni blocco contiene quindi le informazioni relative ai permessi presenti per un determinato soggetto. I permessi descritti da ogni blocco sono le possibili azioni consentite sul file o directory, sono di tre tipi:

1. Lettura (**r**)
2. Scrittura (**w**)
3. Esecuzione (**x**).

Se è presente il permesso ci sarà la lettera corrispondente (**r**, **w**, **x**), sempre nell'ordine indicato, viceversa al suo posto ci sarà un trattino (**–**).
Significato dei permessi:

- file ordinari
  1. **r** Leggere il contenuto
  2. **w** Modificare il contenuto
  3. **x** Eseguire il file (ha senso per file binari e script).

- file speciali
  1. **r** Leggere dal device (input)
  2. **w** Scrivere sul device (output)
  3. **x** Non significativo.

- directory
  1. **r** Leggere il contenuto della directory
  2. **w** Modificare la directory: aggiungere, rimuovere o rinominare file al suo interno
  3. **x** Accesso (scansione) della directory: leggere, modificare o eseguire i file in essa contenuti.

ESEMPI

1. **- rwx rw- r--**
   Sul file ordinario (**-** iniziale) l'utente proprietario (**rwx**, primo blocco da tre caratteri) può leggere (**r**), scrivere (**w**) ed eseguire il file (**x**).
   Il gruppo proprietario (**rw-**, secondo blocco da tre caratteri) può leggere (**r**) e scrivere (**w**) il file ma non eseguirlo (**-**).
   Gli altri utenti (**r--**, terzo blocco da tre caratteri) possono solamente leggere(**r**) il file.

2. **d rwx r-x r-x**
   Sulla directory (**d** iniziale) l'utente proprietario (**rwx**, primo blocco da tre caratteri) può leggere (**r**), scrivere (**w**) ed eseguire il file (**x**).
   Sia i membri del gruppo proprietario che tutti gli altri utenti (**r--**, secondo e terzo blocco da tre caratteri) possono solamente leggere(**r**) il file.

Esiste un altro modo per rappresentare equivalentemente i tre blocchi da tre caratteri della ACL, esso consiste nell'utilizzare solamente tre cifre ottali (da 0 a 7), es 775.

Ognuna di queste tre cifre corrisponde ai tre blocchi precedentemente descritti, sempre nello stesso ordine, di conseguenza indica in maniera equivalente i permessi di accesso per ognuno dei tre soggetti.
Ogni cifra ottale si può espandere quindi nella corrispondente sequenza di tre cifre binarie, del tipo *010*.

Le tre cifre possono ovviamente essere solo 0, ad indicare la mancanza del permesso, oppure 1 se questo è presente. Ognuna di queste tre cifre è chiaramente associata ad uno dei tre possibili permessi precedentemente illustrati, sempre nell'ordine **rwx**. Quindi ad esempio *010* equivale a **-w-**, in quanto la cifra che indica la presenza del permesso (*1*) è presente solo nella seconda posizione, quella occupata appunto da **w**, il permesso di modifica/scrittura.

Per tradurre questa la stringa di tre cifre binarie nella corrispondente cifra ottale bisogna sommare tre cifre che sono:

1. $2^2 =$ **4** Se è presente il permesso di lettura (**r**)
2. $2^1 =$ **2** Se è presente il permesso di scrittura (**w**)
3. $2^0 =$ **1** Se è presente il permesso di esecuzione (**x**).

La somma ottenuta è una combinazione unica dei tre numeri precedenti e conterrà quindi l'informazione dei permessi consentiti al soggetto in questione.
Le tre cifre ottali, come specificato in precedenza, descriveranno quindi univocamente i permessi di accesso al file relativi: la prima al proprietario, la seconda al gruppo proprietario, la terza a tutti gli altri utenti.

ACL di default dopo la creazione di:

- File ordinari **644** cioè **- rw- r-- r--**
- Directory **755** cioè **d rwx r-x r-x**

ESEMPI

1. **764**
   L'utente proprietario, la cui cifra associata è **7** (4+2+1) corrispondente a **rwx**, può leggere (**r**), scrivere (**w**) ed eseguire (**x**) il file.
   Il gruppo proprietario, la cui cifra associata è **6** (4+2+0) corrispondente a **rw-**, può leggere (**r**) e scrivere (**w**) il file ma non eseguirlo.
   Gli altri utenti, la cui cifra associata è **4** (4+0+0) corrispondente a **r--**, possono solamente leggere (**r**) il file.
2. **644**
   L'utente proprietario, la cui cifra associata è **6** (4+2+0) corrispondente a **rw-**, può leggere (**r**) e scrivere (**w**) il file ma non eseguirlo.
   Il gruppo proprietario così come gli altri utenti, la cui cifra associata ad entrambi è **4** (4+0+0) corrispondente a **r--**, possono solamente leggere (**r**) il file.
3. **755**
   L'utente proprietario, la cui cifra associata è **7** (4+2+1) corrispondente a **rwx**, può leggere (**r**), scrivere (**w**) ed eseguire (**x**) il file.
   Il gruppo proprietario cosi come gli altri utenti, la cui cifra associata ad entrambi è **5** (4+0+1) corrispondente a **r-x**, può leggere (**r**) ed eseguire (**x**) il file ma non modificarlo.

COMANDI

- **chmod**

  (change mode)
  Cambia i permessi di accesso ai file.

  SINTASSI
  - **chmod** [-R] *modo file*

    L'argomento *modo* può essere o in formato simbolico oppure ottale.
    Nel modo simbolico si aggiunge un permesso con più (**+**) e si toglie con meno (**-**) .
    I soggetti a cui attribuire o togliere i permessi sono:
    - **u** (users) Utente proprietario
    - **g** (group) Gruppo proprietario
    - **o** (others) Tutti gli altri utenti
    - **a** (all) Tutti i tre soggetti precedenti.

  FLAG
  - **-R** Cambia i permessi ricorsivamente, cioè proseguendo l'operazione nelle eventuali subdirectory e relativi file.

  ESEMPI
  1. **chmod** u+w */percorso/file_esempio1*
     Il proprietario di *file_esempio1* avrà ora il permesso di scrittura su detto file
  2. **chmod** g-x */percorso/file_esempio2*
     Il gruppo proprietario di *file_esempio2* non avrà più il permesso di esecuzione su *file_esempio2*
  3. **chmod** o+r */percorso/file_esempio3*
     Viene aggiunto il permesso di lettura su *file_esempio4* a tutti gli altri utenti del sistema
  4. **chmod** a-w */percorso/file_esempio4*
     Viene tolto il permesso di scrittura su *file_esempio4* a tutti gli utenti del sistema, proprietario e gruppo proprietario compresi
  5. **chmod** 754 */percorso/file_esempio5*
     Vengono settati i seguenti permessi: utente proprietario (**7**) può leggere scrivere ed eseguire *file_esempio5*, il gruppo proprietario (**5**) può leggere ed eseguire, gli altri utenti (**4**) possono leggere *file_esempio5.*

- **chown**

  (change owner)
  Cambia l'utente proprietario, il gruppo proprietario (o entrambi) dei file specificati.
  Per cambiare anche o solo il gruppo proprietario bisogna farlo precedere da un punto (**.**).

  SINTASSI
  - **chown** [-R] [*utente*] [.] [*gruppo*] *file*

  FLAG
  - **-R** Cambia i permessi ricorsivamente, cioè proseguendo l'operazione nelle eventuali subdirectory.

  ESEMPI
  1. **chown** *utente1 /percorso/file_esempio*
     file_esempio avrà come proprietario *utente1*
  2. **chown** -R *utente1.gruppoB /percorso/directory_esempio*
     directory_esempio e tutti i suoi file e subdirectory avranno come proprietario *utente1* e *gruppoB* come gruppo proprietario
  3. **chown** *.gruppoB /percorso/file_esempio*
     file_esempio avrà come gruppo proprietario *gruppoB.*

- **chgrp**

  (change group)
  Cambia il gruppo proprietario dei file specificati.

  SINTASSI
  - **chgrp** [-R] *nuovo_gruppo lista_file.*

  FLAG
  - **-R** Cambia i permessi ricorsivamente, cioè proseguendo l'operazione nelle eventuali subdirectory.

  ESEMPIO
  1. **chgrp** *gruppoB /percorso/file_esempio*
     file_esempio avrà come nuovo gruppo proprietario *gruppoB.*

GESTIONE FILE E DIRECTORY

DIRECTORY DI LINUX

In Linux le directory sono dei file che contengono tanti oggetti, cioè tanti *directory-entry* quanti sono i file in esso contenuti, sempre seguendo la logica Unix: "*tutto è un file*".

Questo semplice ma efficacissimo meccanismo, unito alla possibilità di effettuare dei collegamenti tra di esse (vedesi paragrafo Link) ed avere permessi particolareggiati su ognuna, permette una estrema flessibilità di gestione dell'albero delle directory, nonché di ospitare strutture molto complesse di file ed un numero elevato di subdirectory, il tutto molto semplicemente.

In ogni directory sono presenti due file nascosti, che hanno come nome *punto* (**.**) e *doppio punto* (**..**). Il file punto (**.**) è un riferimento alla directory stessa, mentre il file doppio punto (**..**) è un riferimento alla relativa directory *padre*.

L'unica eccezione è rappresentata dalla directory radice, o root-directory (**/**) nella quale il file punto e doppio punto sono il realtà lo stesso file, dato che non esiste per definizione una directory padre di quella radice.

Le directory utilizzate normalmente da Linux, tenendo presente che qualcosa può cambiare tra una distribuzione e l'altra, sono:

- */* Directory radice del sistema, chiamata root-directory
- */boot* Contiene l'occorrente per l'avvio di Linux
- */bin* Eseguibili di sistema, accessibili da tutti gli utenti
- */dev* File speciali per i dispositivi
- */etc* File di configurazione dei programmi e servizi di sistema
- */home* Contiene le home-directory degli utenti del sistema
- */media* Mount-point di device a blocchi (hard-disk removibili, CD-ROM, etc.)
- */mnt* Come */media*
- */opt* Software non pacchettizzato e *macrosoftware*
- */proc* File virtuali di informazioni sul sistema
- */root* Home-directory del superuser
- */sbin* Comandi e programmi riservati al superuser
- */tmp* File temporanei
- */usr* Tutti i programmi installati
- */var* Dati temporanei che vengono modificati quando il sistema lavora.

Ecco nel dettaglio cosa contiene ogni sub-directory di quella radice in un sistema Linux:

- */*

  Directory radice nella quale viene montata all'avvio la partizione principale del sistema.

- **/boot**
  Contiene il file immagine del kernel ed il necessario per la parte iniziale dell'avvio di Linux.
  A seconda del boot loader utilizzato ci sarà una subdirectory *Lilo* o *Grub* in cui è presente, tra l'altro, il file di configurazione del menù di avvio, essenziale per l'avvio di Linux in quanto ci sono i riferimenti al file dell'immagine del kernel presente nella directory */boot*.
  Normalmente è preferibile far risiedere la director */boot* in una partizione separata per evitare cancellazioni accidentali dell'immagine del kernel.

- **/bin**
  Qui ci sono gli eseguibili dei programmi comuni utilizzati da tutti gli utenti (come *cd*, *chmod*, ...) e alcuni script di shell.

- **/dev**
  In questa directory ci sono i device di sistema, cioè tutti i file speciali associati a periferiche hardware o virtuali. Ci sono anche i file speciali associati agli standard input/output/error e il file */dev/null* che funge sia “generatore di zero”, che da “pozzo senza fondo”, poiché se vi si scrive qualcosa, questa verrà irrimediabilmente persa.

- **/etc**
  Contiene i file di configurazione del sistema, dei servizi di sistema, e i file di configurazione generali dei programmi.
  Sono presenti diverse subdirectory tra cui *init.d* che contiene gli script di avvio, *conf.d* che contiene anche i file di configurazione degli script in *init.d* ed in generale i file di configurazione dei servizi di sistema (daemon).

- **/home**
  Ci sono tutte le home-directory degli utenti tranne quella del superuser.

- **/media**
  Contiene le directory di mount delle altre partizioni come quelle di altri sistemi operativi, dei CD/DVD-ROM, degli hard disk removibili etc.

- **/mnt**
  Uso equivalente delle directory */media*.
  Dipende dalle distribuzioni quella utilizzata, in generale prima si usava solo */mnt*, ora si preferisce usare */media* poiché il Filesystem Hierarchy Standard della Linux Standards Base raccomanda appunto */media* invece di */mnt*.

- **/opt**
  In questa directory normalmente c'è il software non installato tramite il sistema di pacchettizzazione (cioè il sistema di pacchetti binari previsto dalla distribuzione in uso per installare il software), al fine di tenere staccato da tutto il resto quello che non viene gestito direttamente dal pacchettizzatore ufficiale. Di solito qui viene anche installata la suite OpenOffice.org, Acrobat Reader, Matlab, Opera ecc.

- **/proc**
  Il kernel scrive i propri messaggi in questa directory, dalla quale è possibile ottenere diverse informazioni sullo stato del sistema.

- **/root**
  Home-directory del superuser, tenuta separata per motivi di sicurezza, infatti gli utenti non possono visualizzarne il contenuto.

- **/sbin**
  Contiene comandi e programmi riservati al superuser, come i comandi per la gestione dei moduli nel kernel e per il partizionamento dell'hard disk.

- **/usr**
  Contiene tutti i programmi installati. Praticamente il sistema risiede per lo più qui dentro. Alcune subdirectory:
  - */usr/bin/* e */usr/sbin/* Contengono file eseguibili usabili dagli utenti (bin) e dall'amministratore (bin e sbin), che non sono strettamente legati al funzionamento base. Quando si installa un programma qualsiasi è molto facile che il suo eseguibile finirà proprio in */usr/bin/*
  - /usr/doc/ e /usr/man/ Pagine di documentazione e manuale
  - /usr/local/ Qui dovrebbero essere installati (dall'amministratore) tutti quei programmi locali che non appartengono alla distribuzione.

- **/var**
  Contiene i dati che vengono modificati quando il sistema lavora normalmente, come i file di log, le directory per le mail di sistema, le code di stampa, i file temporanei e le cache dei programmi.

- **/tmp**
  Directory dei file temporanei, generalmente viene cancellata ad ogni avvio.

Per aumentare la sicurezza, la flessibilità e la velocità del sistema si può effettuare un partizionamento avanzato per poter utilizzare partizioni dedicate ad alcune directory particolari (vedi paragrafo Partizionamento). Questo accorgimento deve essere preso in fase di installazione del sistema, una volta create le partizioni si dovranno inserire i mount-point nel file /etc/fstab (vedi paragrafo Partizionamento).

Le directory che può convenire utilizzare usando partizioni separate sono:

- /dati E' sempre conveniente utilizzare una partizione apposita per i dati, preservandone l'integrità in caso di malfunzionamenti al filesystem principale, questa è un ipotesi rara, ma comunque possibile.
- /tmp Per impedire a malfunzionamenti, bug ed eventuali attacchi di saturare la partizione principale, perché qui vi sono file temporanei e questa directory è scrivibile da tutti gli utenti.
- /var Oltre ai motivi di /tmp è fortemente consigliato utilizzare una partizione apposita per questa directory perché qui ci sono i file temporanei utilizzati durante l'esecuzione di quasi tutti i programmi, compilazioni e sopratutto i log dei servizi di sistema. Si evita così che un eventuale bug di un programma, come un loop di scrittura infinito, possa saturare il filesystem principale.
- /home specie nei sistemi multiutente, per separare lo spazio utenti dal sistema, impedendo quindi di saturare la partizione principale.

Percorsi assoluti e relativi

Un percorso nella shell si può scrivere in due modi:

- percorso **assoluto** specificando il percorso completo dalla directory radice
- percorso **relativo** scrivendo il percorso a partire dalla directory corrente.

Ad esempio se si è nella directory */usr* e si vuole raggiungere la directory */usr/local/bin* è equivalente scrivere:

- **cd** local/bin solo se si è nella directory /usr
- **cd** /usr/local/bin da qualsiasi posizione nel sistema.

E' chiaro che specificando un percorso assoluto si può indicare un qualsiasi file o directory mentre col percorso relativo si è *limitati* a quanto contenuto *a partire* dalla directory di lavoro corrente.

Nei percorsi relativi può essere utile utilizzare il file doppio punto (**..**) presente in ogni directory per specificare un percorso relativo che fa riferimento ad esempio alla directory precedente.

Per esempio se si è nella directory */usr/local* e si vuole andare nella directory */etc* è equivalente scrivere:

- **cd** ../../etc con il primo doppio punto si torna a */usr*, col secondo a */*
- **cd** /etc da qualsiasi posizione nel sistema.

COMANDI

- **pwd**

  (print working directory)
  Mostra il percorso completo della directory di lavoro corrente.

- **touch**

  Cambia l'orario di accesso e/o modifica di ogni file specificato (di default entrambi), aggiornandolo all'orario corrente.
  Se il file non esiste, ne crea uno vuoto.

  SINTASSI
  - **touch** [-acm] *file_esempio*

  FLAG
  - **-a** Cambia l'orario di accesso di *file_esempio*
  - **-c** Non crea il file, aggiorna solo la data di ultimo accesso
  - **-m** Cambia l'orario di modifica di *file_esempio.*

- **cd**

  (change directory)
  Cambia la directory corrente in quella specificata.

  SINTASSI
  - **cd** [*nome_directory*]

  Ci sono altri 4 modi per usare il comando **cd**.

  ESEMPI
  1. **cd**
     Se non si specifica la directory, la directory di lavoro diventerà la *home-directory* dell'utente loggato. Equivalente di: **cd ~**
  2. **cd -**
     La directory di lavoro diventerà quella precedentemente utilizzata, questa è memorizzata nella variabile d'ambiente OLDPWD
  3. **cd .**
     La directory di lavoro resterà quella corrente
  4. **cd ..**
     La directory di lavoro diventerà quella padre di quella corrente.

- **ls**

  (list)
  Lista il contenuto della directory corrente o di quella specificata.

  SINTASSI
  - **ls** [-aFlrtu] [*percorso*]

  FLAG
  - **-a** Include nell'elenco anche i file che iniziano con **.** (punto), cioè file e directory nascoste
  - **-F** Aggiunge ai nomi dei file un carattere che ne indica il tipo:
    - * per i file eseguibili
    - / per i file-directory
    - @ per i link simbolici
    - | per i FIFO
  - **-l** Mostra le informazioni dettagliate: Access Control List, numero di link al file, utente e gruppo proprietario, dimensione in byte, ultimo accesso e nome del file
  - **-r** Inverte l'ordinamento alfabetico
  - **-t** Ordina i file per data di ultima modifica
  - **-u** Ordina i file per data di ultimo accesso
  - **-1** Mostra l'elenco in un unica colonna.

- **mkdir**

  (make directory)
  Crea una nuova directory in quella corrente o in quella specificata.

  SINTASSI
  - **mkdir** [-m *Modo*] [*percorso*] *directory*

  FLAG
  - **-m** Imposta anche i permessi di acceso alla directory. *Modo* è una stringa di tre cifre ottali, di default questa è 755
  - **-v** Mostra sullo standard output le operazioni eseguite.

  ESEMPIO
  1. **mkdir** -m 755 ~/nuova_dir
     Crea la directory *nuova_dir* nella home-directory dell'utente loggato con i permessi **rwx r-x r-x**.

- **rmdir**

  (remove directory)
  Rimuove le directory specificate purché vuote.

  SINTASSI
  - **rmdir** [--ignore-fail-on-non-empty] *directory_1* [*directory_2 directory_3*]

  FLAG
  - **--ignore-fail-on-non-empty** Rimuove la directory specificata anche se non è vuota
  - **-v** Verbose mode.

- **cp**

  (copy)
  Copia i file/file-directory da un percorso di partenza ad uno di destinazione.
  Mentre il percorso di partenza deve necessariamente comprendere il nome del file/file-directory da copiare, il percorso di destinazione può comprendere solo la directory di destinazione, o anche il nuovo nome del file/file-directory, in quest'ultimo caso si effettuerà contemporaneamente la copia e la rinomina del file/file-directory di partenza.

  SINTASSI
  - **cp** [-fipRv] *percorso_partenza percorso_destinazione*

  FLAG
  - **-f** Forza la sovrascrittura di eventuali file di destinazione esistenti
  - **-i** In caso di file di destinazione esistenti richiede conferma interattivamente
  - **-p** Preserva le caratteristiche originali del file di partenza:
    - utente proprietario
    - gruppo proprietario
    - permessi di accesso
    - data ultima modifica e ultimo accesso
  - **-R** Copia ricorsivamente, cioè proseguendo la copia nelle eventuali subdirectory
  - **-v** (verbose) mostra sullo standard output le operazioni eseguite.

- **rm**

  (remove)
  Rimuove file o directory.

  SINTASSI
  - **rm** [-filRv] *percorso_partenza percorso_destinazione*

  FLAG
  - **-f** Non mostra nulla sullo standard output, nemmeno in caso di file inesistenti
  - **-i** Chiede conferma interattivamente per ogni file
  - **-I** Chiede conferma interattivamente quando si vuole rimuovere più di tre file o quando si rimuove ricorsivamente
  - **-R** Rimuove ricorsivamente, cioè proseguendo la rimozione delle eventuali subdirectory
  - **-v** Mostra sullo standard output le operazioni eseguite.

  ESEMPI
  1. **rm** *file1 file2*
     Rimuove *file1* e *file2*
  2. **rm** *directoryA*
     Operazione non consentita

3. **rm** *directoryA/**
   Rimuove tutti i file, directory escluse, contenuti in *directoryA*
4. **rm** -R *directoryA*
   Rimuove *directoryA* e qualsiasi subdirectory e file essa contenga.

- **mv**

(move)
Sposta i file o directory da un da un percorso di partenza ad uno di destinazione.
Mentre il percorso di partenza deve necessariamente comprendere il nome del file o directory da copiare, il percorso di destinazione può comprendere il nuovo nome del file o solo la directory di destinazione.

SINTASSI

- **mv** [-fipRv] *percorso_partenza percorso_destinazione*

FLAG

- **-f** Forza la sovrascrittura di eventuali file di destinazione esistenti
- **-i** In caso di file di destinazione esistenti richiede conferma interattivamente
- **-p** Preserva le caratteristiche originali del file di partenza:
  - Utente proprietario
  - Gruppo proprietario
  - Permessi di accesso
  - Data ultima modifica e ultimo accesso
- **-R** Sposta ricorsivamente, cioè proseguendo l'azione nelle eventuali subdirectory
- **-v** Mostra sullo standard output le operazioni eseguite.

ESEMPI

1. **mv** *file1 file2*
   Rinomina *file1* in *file2*. Se quest'ultimo esiste viene sovrascritto da *file1*
2. **mv** *file1 directoryA*
   Sposta *file1* dentro *directoryA*
3. **mv** *directoryA directoryB*
   Sposta *directoryA* in *directoryB* se quest'ultima esiste già, viceversa rinomina *directoryA* in *directoryB*
4. **mv** *directoryA file1*
   Operazione non consentita.

- **file**

  Determina il tipo di uno o più file.

  SINTASSI
  - **file** [-k] *file_esempio*

  FLAG
  - **-k** Esegue una scansione più approfondita non fermandosi alla prima occorrenza.

  ESEMPI
  1. **file** */percorso/file_esempio*
     Mostra di che tipo è *file_esempio*
  2. **file** */percorso/**
     Mostra di che tipo sono tutti i file presenti nella directory *percorso.*

LINK

Un **hard link** è un puntatore all'inode di un file esistente, quindi non viene creato un nuovo file ma solo un *directory-entry* (*inode#*, nome_file) nella directory contenente il nuovo collegamento. L'*inode#* consente di accedere agli attributi del file.
I due riferimenti sono totalmente equivalenti perché puntano allo stesso inode, non è possibile distinguere quello creato per prima perché tutti gli attributi sono nell'inode.
Quando si crea un hard link, il link-count del file viene incrementato di 1, quindi il file esiste con nomi diversi. Riferendosi direttamente all'inode, non è possibile creare un hard link a directory.
Quando si cancella un file che ha più hard link il sistema in realtà decrementa solo il link-count di 1. Se nella cancellazione il link-count non è zero viene rimossa solo la *directory-entry*, se invece il link-count è 0 viene rimossa la directory entry e liberato l'inode del file, de-allocando quindi i blocchi di memoria.

Un **soft link** è un collegamento che può riferirsi sia a file che a directory. Viene creato un nuovo file di tipo speciale che contiene semplicemente un percorso. Ogni volta che il sistema incontra un soft link sostituisce il suo contenuto con il percorso che lo individua. Quando viene rimosso un soft link, si elimina solo il file speciale e non certo il file a cui esso si riferisce. I due file (speciale e sorgente) hanno numero di inode (inode#) distinti e il link-count rimane 1 per entrambi i file, che hanno ovviamente dimensione differente. Il comando per realizzare un link è **ln**.

- **ln**

  Realizza i collegamenti tra file. Di default realizza un hard link.

  SINTASSI
  - **ln** [-sn] *file_sorgente file_destinazione*
  - **ln** [-sn] *directory_sorgente file_destinazione.*

  FLAG
  - **-n** Non crea il link se *file_destinazione* esiste già
  - **-s** Crea un collegamento simbolico a *file_sorgente* o a *directory_sorgente* con il nome *file_destinazione.*

  ESEMPI
  1. **ln** -s /dir_dati /home/utente/dati
     Crea un link simbolico di nome dati che punta a */dir_dati*, quindi */home/utente/dati* sarà un collegamento a directory
  2. **ln** file1 /home/utente/file2
     Crea un hard link di nome file2 che punta allo stesso inode di file1, quindi il file esisterà con due nomi diversi
  3. dopo il comando dell'esempio 1: **rm** /home/utente/dati/
     Operazione non consentita, poiché lo slash dopo dati (*dati/*) identifica il link come directory
  4. dopo il comando dell'esempio 1: **rm** /home/utente/dati
     Rimuove il link simbolico */home/utente/dati*
  5. dopo il comando dell'esempio 2: **rm** file1
     Diminuisce di 1 il link-count del file.

RIDIREZIONE I/O

TEORIA

Ciascun programma, per poter funzionare correttamente, elabora i dati attraverso i seguenti canali di comunicazione:

- Dati in ingresso
- Dati in uscita.

In realtà, nei sistemi derivati da Unix, il programma può utilizzare anche un canale di errore, in cui scrivere i messaggi di errore.
Quindi c'è una separazione dei dati in uscita dalle segnalazione di errore.

I tre canali di comunicazione sono rappresentati da altrettanti file speciali:

- /dev/**sdtin** Standard *input*, canale di ingresso
- /dev/**sdtout** Standard *output*, canale di uscita
- /dev/**sdterr** Standard *error*, canale di errore.

Questi tre file speciali sono link simbolici ai tre primi descrittori di file associati ad un processo:

- /proc/self/fd/0 Link simbolico al dispositivo terminale che fornisce i caratteri al processo
- /proc/self/fd/1 Link simbolico al dispositivo terminale che stampa l'uscita del processo
- /proc/self/fd/2 Link simbolico al dispositivo terminale che stampa gli errori del processo.

I dispositivi terminali di output coincidono con il terminale attualmente attivo, per esempio:

- gnome-terminal
- kterm
- shell testuale.

Il dispositivo di input è normalmente associato alla tastiera, ma si può utilizzare anche un joypad, mouse etc.

I tre canali di comunicazione possono essere ridirezionati su un qualunque file, detto file prenderà le veci del dispositivo associato al canale di comunicazione.

Gli scopi primari della ridirezione sono:

- Prendere ingressi da file
- Scrivere uscite su file
- Scrivere errori su file diversi da quelli destinati all'uscita.

Descrittori di file per l'input e l'output:

- **0** standard *input*
- **1** standard *output*
- **2** standard *error*.

Operatori di ridirezione:

- **>** Equivalente a **1>**, redirige solo lo standard output su un file qualunque, sovrascrivendo il contenuto del file
- **2>** Redirige solo lo standard error su un file qualunque
  >> Redirige lo standard output su un file aggiungendo il contenuto in coda al file di ridirezione, cioè in ***append mode***
- **2>>** Redirige lo standard error su un file in *append mode*
- **<** Redirige lo standard input su un file qualunque, equivalente a **0<**
- n**>&**m Concatena i due canali di comunicazione associati ai descrittori di file *n* ed *m*, se non indicati.

Ciascun processo ha a disposizione 1024 descrittori di file (da 0 a 1023) che puntano tutti al terminale associato al processo. È possibile ridirezionare il descrittore **n**-esimo su un file mediante la stessa sintassi: **n>**, **n>>**, **n<**.

- **n>** Redirige il descrittore n-esimo su un file qualunque, sovrascrivendo il contenuto del file ridirezione
- **n>>** Redirige il descrittore n-esimo su un file aggiungendo il contenuto in coda al file di ridirezione, cioè in *append mode*.

ESEMPI

1. sort **<** file_esempio
   Ridireziona lo standard input di sort dandogli in pasto *file_esempio*, quindi ne verrà mostrato il contenuto ordinato da sort. Equivalenti:
   - sort **<** ./file_esempio
   - sort **0<** file_esempio
   - cat file_esempio **|** sort
2. ls /* **>** stdout.file
   Ridireziona lo standard output del comando *ls /** in *stdout.file,* creandolo o sovrascrivendo quello esistente. Equivalenti:
   - ls /* **1>** stdout.file
   - ls /* **>** ./stdout.file
   - ls /* **1>|** stdout.file indica esplicitamente di creare stdout.file
3. ls /* **2>** stderr.file
   Ridireziona lo standard error del comando *ls /** in *stderr.file*, creandolo o sovrascrivendo quello esistente.
4. ls /* **>>** stdout.file
   Ridireziona lo standard output del comando *ls /** in *stdout.file*, aggiungendo in coda al file *stdout.file* il risultato della ridirezione. Equivalente:
   - ls /* **1>>** stdout.file
5. ls /* **2>>** stderr.file
   Ridireziona lo standard error del comando *ls /** in *stdout.file*, aggiungengo in coda al file *stderr.file* il risultato della redirezione.
6. ls /* **>** stdout.file **2>** stderr.file
   Ridireziona lo standard output del comando *ls /** in *stdout.file* e lo standard error sempre del comando *ls /** nel file *stderr.file*. I file verranno creati o sovrascritti se esistenti. Equivalente:
   - ls /* **2>** stderr.file **>** stdout.file
7. ls /* **>>** stdout.file **2>>** stderr.file
   Ridireziona lo standard output del comando *ls /** in *stdout.file* e lo standard error sempre del comando *ls /** nel file *stderr.file*. Il risultato della ridirezione sarà aggiunto in coda ai rispettivi file. Equivalente:
   - ls /* **2>>** stderr.file **>>** stdout.file
8. ls /* **>** stdout_err.file **2>&1**
   Ridireziona lo standard output del comando *ls /** in *stdout_err.file* e lega lo standard error allo standar output. Equivalenti:
   - ls /* **1>** stdout_err.file **2>&1**
   - ls /* **2>** stdout_err.file **1>&2**
9. ls /* **1>>** stdout_err.file **2>&1**
   Come l'esempio 8, con l'unica differenza che il risultato della ridirezione viene aggiunto in coda al file *stdout_err.file*. Equivalente:
   - ls /* 2**>>** stdout_err.file **1>&2**
10. ls /* **2>&1 >** stdout.file
    Lega lo standard error allo standard output, che è il terminale poiché la shell, leggendo da sinistra verso destra, non ha ancora incontrato la seconda redirezione (**>** stdout.file). In altri termini ridireziona poi lo standard output in *stdout.file*. Equivalente:
    - ls /* **>** stdout.file

CONCATENAMENTO COMANDI

- **|**

  (**pipe**)
  Il simbolo di pipe (**|**) serve a ridirigere lo standard output del comando a sinistra della pipe nello standard input del comando a destra della pipe, creando così una *pipeline*, letteralmente un *tubo.*
  Questa connessione è effettuata prima di qualsiasi ridirezione specificata dal comando. Per poter parlare di pipeline basta anche un solo comando.
  Di norma il valore restituito dalla pipeline è quello dell'ultimo comando che viene eseguito all'interno di questa.
  Se all'inizio nella pipeline c'è un punto esclamativo (!) il valore restituito corrisponde alla negazione logica del risultato normale.
  La shell attende che tutti i comandi della pipeline siano terminati prima di restituire un valore, ognuno di questi comandi è eseguito come un processo separato, cioè in una sub-shell.

  SINTASSI
  - [!] *comando1* [**|** *comando2*]

- **;**

  I comandi separati da punto e virgola (**;**) sono eseguiti sequenzialmente. Può essere usato per separare una serie di comandi sulla stessa riga o per terminare una lista di comandi. Idealmente esso sostituisce il codice di interruzione di riga.

  ESEMPIO
  1. **touch** *file_esempio* **; chmod** u+x *file_esempio*

- **() o {}**

  (comandi racchiusi da parentesi tonde)
  Se si racchiude un comando o una lista di comandi tra parentesi tonde **()** o graffe **{}** questi verranno raggruppati per controllare la sequenza di esecuzione. Il valore di uscita di tale lista corrisponde a quello dell'ultimo comando della stessa che ha potuto essere eseguito.

- **&&**

  (AND)
  L'operatore di controllo **&&** si comporta come l'operatore booleano AND: se il valore di uscita di ciò che sta alla sua sinistra è zero (0 = *true*, ossia vero) viene eseguito anche ciò che sta alla sua destra. In sostanza viene eseguito il comando a destra solo se il primo ha terminato con successo.

  ESEMPIO
  1. **mkdir** dir_esempio **&& echo** “directory di prova creata”
     Se viene creata con successo la directory *dir_esempio* verrà mostrato il messaggio di conferma “directory di prova creata”.

- **||**

  (OR)
  L'operatore di controllo **||** si comporta come l'operatore booleano OR: se il valore di uscita di ciò che sta alla sua sinistra è zero (0 = true, ossia vero) non viene eseguito ciò che sta alla sua destra. In sostanza viene eseguito il comando a destra solo se il primo non ha terminato con successo oppure non si è potuto eseguire.

  ESEMPIO
  1. **mkdir** */root/dir_nopermessi* **||** **mkdir** */home/utente/dir_sipermessi*
     Si tenta di creare la directory */root/dir_nopermessi*, se il comando fallisce (ad esempio perché non si è loggati come superuser) verrà creata la directory */home/utente/dir_sipermessi*.

VISUALIZZAZIONE FILE

- **echo**

  Mostra sullo standard output una stringa.

  SINTASSI
  - **echo** [*stringa*]

  ESEMPI
  1. **echo** ~
     Mostra il percorso della home-directory dell'utente *loggato*
  2. **echo** “parola_esempio” >> *file_prova*
     Inserisce la stringa *parola_esempio* in coda al *file_prova* (vedere sezione ridirezione I/O).

- **cat**

  (concatenate)
  mostra sullo standard output il contenuto del file specificato. Se vengono specificati più file ne mostra il contenuto sempre sullo standard output ma concatenando il successivo dopo il precedente, da sinistra verso destra.
  Se non viene indicato nessun file viene utilizzato lo standard input.

  SINTASSI
  - **cat** [*file*]

  ESEMPI
  1. **cat** *file1*
     Mostra sullo standard output il contenuto di *file1*. Equivalente a: **cat** < file1
  2. **cat** *file1 file2*
     Mostra sullo standard output il contenuto di *file1* e di seguito quello di *file2*
  3. **cat** *file1 file2 > file3*
     Genera *file3* che è il risultato del concatenamento in sequenza di *file1* e *file2*
  4. **cat** *file4 file5 >> file3*
     Aggiunge in coda a *file3* il concatenamento in sequenza di *file4* e *file5.*

- **more**

  Formatta il testo in pagine per facilitarne la lettura a video.

  SINTASSI
  - **more** [*file*]

  COMANDI
  - **INVIO** Avanza la visualizzazione di una riga

- **SPAZIO** Avanza la visualizzazione di una pagina
- **b** Torna indietro alla precedente pagina mostrata
- **q** Termina la visualizzazione, di default essa termina quando viene mostrata l'ultima riga
- **h** Mostra la guida in linea dei comandi
- **/** *regexp* *E*segue una ricerca in avanti in base all'espressione regolare indicata in *regexp*
- **n** Ripete l'ultimo comando di ricerca.

ESEMPI

1. **more** */percorso/file_esempio*
   Mostra il contenuto di *file_esempio* usando il paginatore more
2. **cat** */percorso/file_esempio* | **more**
   Equivalente dell'esempio precedente
3. **cat** */percorso1/file_esempio1 /percorso2/file_esempio2* | **more**
   Mostra mediate il paginatore **more** il contenuto di *file_esempio2* concatenato al contenuto di *file_esempio1*. L'ordine di visualizzazione è quello con cui **cat** legge i file che gli sono forniti come argomento, ossia da sinistra a destra.

- **less**

Formatta il testo per facilitarne la lettura a video.

SINTASSI

- **less** [*file*]

COMANDI

- **FRECCIA giù** Avanza la visualizzazione di una riga
- **INVIO** Avanza la visualizzazione di una riga
- **FRECCIA su** Torna indietro di una riga
- **SPAZIO** Avanza la visualizzazione di una pagina
- **PAG giù** Avanza la visualizzazione di una pagina
- **PAG su** Torna indietro di una pagina
- **b** Torna indietro di una pagina
- **q** Termina la visualizzazione, non termina automaticamente quando viene mostrata l'ultima riga
- **h** Mostra la guida in linea dei comandi
- **/** *stringa* *E*segue una ricerca in avanti in base all'espressione regolare indicata
- **?** *stringa* *E*segue una ricerca indietro in base all'espressione regolare indicata
- **n** Ripete l'ultimo comando di ricerca.

ESEMPI

1. **less** */percorso/file_esempio*
   Mostra il contenuto di *file_esempio* usando il paginatore less
2. **cat** */percorso/file_esempio* | **less**
   Equivalente dell'esempio precedente.

ESPRESSIONI REGOLARI

- Regole di base
  a) Un qualsiasi carattere è già un'espressione regolare (es. caratteri speciali)
  b) Un carattere speciale può essere usato come carattere regolare anticipandolo con backslash (\) (es. * viene considerato *asterisco* con *****)
  c) L'espressione regolare va racchiusa tra apici (**'expr'**).

- Regole sintattiche
  1. **[xwyz]** Set di caratteri qualsiasi all'interno delle parentesi quadre, i caratteri sono in OR tra di loro
  2. **[^xwyz]** Un carattere qualsiasi esclusi quelli nelle parentesi
  3. **[a-f]** Un carattere qualsiasi tra **a** ed **f**, estremi inclusi. Vengono verificati i codici ASCII dei simboli specificati, quindi [11-17] equivale a 1[1-7], poiché 1 è fisso (fuori dalle parentesi quadre), viene considerato sempre
  4. **[[:alnum:]]** Tutti i caratteri alfanumerici
  5. **[^[:alnum:]]** Tutti i caratteri tranne quelli alfanumerici
  6. **.** (punto) Carattere qualsiasi
  7. **^** (apice) Cerca il modello specificato dopo l'apice all'inizio di ogni riga
     es. ^[A-L] Ricerca le sequenza A-L solo ad inizio di riga
  8. **$** Carattere di posizionamento a fine riga
  9. **\<** Posizionamento a inizio parola, **\>** posizionamento a fine parola
     es. [sS]**\>** Ricerca una parola che finisce per s o S
     es **\<**[56] Ricerca una parola che inizia per 5 o 6
  10. Ripetizione dei caratteri
      10.1. ***** Ripetizione del carattere *y* 0 o più volte, se c'è una volta lo filtra: *y** equivalente a ***y{0,}***
      10.2. **?** Nessuna ripetizione o massimo una ripetizione del carattere *y*: *y?* equivalente a ***y{0,1}***
      10.3. **+** Almeno una ripetizione del carattere *y*: *y*+ equivalente a ***y{1,}***
      10.4. **{n}** Esattamente *n* ripetizioni del carattere y: *y{n}*
      10.5. **{n,}** Almeno *n* ripetizioni di y: *y{n,}*
      10.6. **{n,m}** Da *n* a *m* volte *y* (n<m): *y{n,m}*
  11. **expr1 expr2** É una sequenza di espressioni regolari, da cercare una di seguito all'altra (le espressioni sono in AND)
  12. \| simbolo di OR
  13. **&&** simbolo di AND.

- Classe di caratteri
  1. **[:upper:]** Lettere maiuscole
  2. **[:lower:]** Lettere minuscole
  3. **[:alpha:]** Lettere alfabetiche: unione di *upper* e *lower*
  4. **[:digit:]** Cifre numeriche
  5. **[:alnum:]** Cifre alfanumeriche: unione di *alpha* e *digit*
  6. **[:punct:]** Caratteri di punteggiatura
  7. **[:space:]** Caratteri definiti come «spazi bianchi»
  8. **[:blank:]** Comprende solo <space> e <tab>

Per esempi pratici consultare il comando **grep**, nella pagina successiva.

FILTRI E RICERCHE

I filtri sono programmi che trasformano il loro standard input nel loro standard output.
Questi comandi fanno il *post-processing* su comandi precedenti, alterando quindi lo stream di output della funzione precedente.

...=> F.D.T. => Out

La Funzione Di Trasferimento si specifica all'interno del comando.
es. sort => cat prova | sort
La F.D.T. di sort è *cablata* all'interno del programma sort.

- **sort**

  Ordina o fonde insieme il contenuto dei file.

  SINTASSI
  - **sort** [-cmbdfinr] [-o *file*] [*file1 file2*]

  FLAG
  - **-c** Controlla se i file indicati siano già ordinati. Se non lo sono viene emessa una segnalazione di errore e il programma mostra la prima riga che non rispetta l'ordine
  - **-m** Fonde insieme i file indicati che devono essere già ordinati
  - **-b** Ignora gli spazi vuoti iniziali
  - **-d** Ignora tutti i caratteri che non siano lettere, numeri o spazi
  - **-f** Non distingue tra lettere maiuscole e minuscole
  - **-i** Ignora i caratteri speciali al di fuori del set ASCII puro
  - **-n** Esegue un ordinamento numerico tenendo conto anche del segno meno e del punto decimale
  - **-r** Inverte l'ordine della comparazione
  - **-o** *file* Invece di utilizzare lo standard output, utilizza il file indicato per inserire il risultato dell'operazione.

- **grep**

  (global regular expression print)
  Mostra le linee che corrispondono al modello ricercato. Si può specificare uno o più file oppure usare lo standard input.

  SINTASSI
  - **grep** [-GEFecnvlLi] *modello* [*file_1 file_2*]

  FLAG
  - **-G** Utilizza l'espressione regolare elementare, il default
  - **-E** Utilizza l'espressione regolare estesa
  - **-F** Utilizza un modello fatto di stringhe fisse

- **-e** Specifica il modello di ricerca
- **-c** Mostra il numero di righe in cui è presente il modello
- ***-n*** Aggiunge ad ogni riga contenente il modello il numero di tale riga nel file
- **-v** Selezione inversa: mostra tutte le righe tranne quelle contenenti il modello
- **-l** Mostra il percorso dei file in cui è presente il modello
- **-L** Selezione inversa: mostra il percorso di tutti i file in cui non è presente il modello
- **-i** Non distingue tra caratteri maiuscoli e minuscoli di *modello.*

ESEMPI

1. **grep** '.*' *file_esempio*
   Mostra tutte le righe di *file_esempio*
2. **grep** '[uo]' *file_esempio*
   Mostra tutte le righe di *file_esempio* contenenti i caratteri *o*, oppure *u*, indipendentemente dalla loro collocazione nella stringa
3. **grep** '^ciao' *file_esempio*
   Mostra tutte le righe di *file_esempio* che iniziano con la stringa *ciao,* anche se seguita da altri caratteri
4. **grep** '\<[Ll]inu[xs]\>' *file_esempio*
   Mostra tutte le righe di *file_esempio* che contengono esattamente le parole: Linux, Linus, linux, linux, indipendentemente dalla loro posizione nella riga
5. **grep** '[Ll]i.*[Gg]nu.*[2-3]$' *file_esempio*
   Mostra tutte le righe di *file_esempio* che contengono parole che iniziano con *Li* oppure *li*, che contengono dopo i caratteri *Gnu* o *gnu*, e la riga termina con *2* o *3*: gentoo *linux Gnu 2.6.13*
6. **ls -1** /percorso/ | **grep** '[^[:alpha:]]$'
   Mostra tutti i file nella directory */percorso* che non finiscono con un carattere alfabetico.
   Equivalente di: **ls -1** /percorso/ | **grep** '[^A-Za-z]$'
7. **ls -1** /percorso/ | **grep** '^[[:digit:]].*[[:lower:]]\>'
   Mostra tutti i file nella directory */percorso/* il cui nome inizia con un numero, e nel cui nome è presente una parola che finisce con un carattere minuscolo.
   Equivalente di: **ls -1** /percorso/ | **grep** '^[0-9].*[a-z]\>'
8. **ls -1** /percorso/ | **grep** '[[:punct:]]\|[[:space:]]'
   Mostra tutti i file nella directory /percorso il che contengono nel nome o uno spazio o un carattere di punteggiatura.

- **head**

  Mostra le prime righe o byte di un file, di default le prime 10 righe.

  SINTASSI
  - **head** [-cn] [*file*]

  FLAG
  - **-c** *X* Mostra i primi X byte di un file
  - **-c -***X* Mostra i byte di un file esclusi gli ultimi *X* byte
  - **-n** *X* Mostra le prime X righe di un file
  - **-n -***X* Mostra le righe di un file escluse le ultime *X*.

  ESEMPI
  1. ***h*ead** *file_esempio*
     Mostra le prime 10 righe di *file_esempio*
  2. **head** -c 6 *file_esempio*
     Mostra i primi 6 byte di *file_esempio*
  3. **head** -n -15 *file_esempio*
     Mostra le righe di *file_esempio* escluse le ultime 15
  4. **cat** *file_esempio* | **head** -n 20
     Mostra le prime 20 righe di *file_esempio.*

- **tail**

  Mostra le ultime righe o byte di un file, di default le ultime 10 righe.

  SINTASSI
  - **tail** [-cn] [*file*]

  FLAG
  - **-c** *X* Mostra gli ultimi *X* byte di un file
  - **-c +***X* Mostra gli ultimi byte di un file a partire dal byte numero *X*
  - **-n** *X* Mostra le ultime *X* righe di un file
  - **-n +***X* Mostra le ultime righe di un file a partire dalla riga numero *X*
  - **-f** Mostra le righe aggiunte in coda ad un file man mano che cresce, utile per vedere in tempo reale i messaggi scritti in un file di log.

  ESEMPI
  1. **tail** *file_esempio*
     Mostra le ultime 10 righe di *file_esempio*
  2. **tail** -c 6 *file_esempio*
     Mostra gli ultimi 6 byte di *file_esempio*
  3. **tail** -n +10 *file_esempio*
     Mostra le ultime righe di *file_esempio* a partire dalla decima
  4. **cat** *file_esempio* | **tail** -n 20
     Mostra le ultime 20 righe di *file_esempio.*

- **uniq**

  Mostra od omette le linee ripetute, a seconda che si usi l'opzione **-d** o **-u**. Se non è specificato nessun flag si comporta come *cat*.

  SINTASSI
  - **uniq** [-cdfisuw] [*file_esempio*]

  FLAG
  - **-c** Antepone alle linee il numero di occorrenze
  - **-d** Mostra solo le linee duplicate
  - **-f** *N* Salta il confronto delle prime *N* parole
  - **-i** Nel confronto ignora la differenza tra lettere maiuscole e minuscole
  - **-s** *N* Salta il confronto dei primi *N* caratteri
  - **-u** Mostra solo le linee uniche, non duplicate
  - **-w** *N* Confronta non più di *N* caratteri per riga.

  ESEMPIO
  1. **uniq** /var/log/Xorg.0.log -dc
     Mostra solo le righe ripetute (**-d**) nel file */var/log/Xorg.0.log*, mostrando anche il numero di occorrenze (**-c**)

- **wc**

  (word count)
  Conta e mostra righe, parole e byte contati per ogni file.
  Di default mostra tutto ciò che conta, cioè: linee, parole e byte.

  SINTASSI
  - **wc** [-cmlw] [*file_esempio*]

  FLAG
  - **-c** Mostra i byte contati
  - ***-m*** Mostra i caratteri contati
  - **-l** Mostra le linee contate
  - **-w** Mostra le parole contate.

  ESEMPIO
  1. **wc** */var/log/dmesg* -lw
     Conta il numero di righe (-w) e di parole (-l) del file */var/log/dmesg*

- **find**

  Cerca dei file nella directory specificata e relative subdirectory.

  SINTASSI

  - **find** [-HLP] [*percorso*] [-name '*modello*'] [-user *utente_1*] [-group *gruppo_a*] [-perm +-*permessi*] [-max(min)depth *X*] [-mount] [-[mac]time *n*] [-[mac]min *n*] [-[nac]ewer *file*] [-empty] [-links *n*] [-inum *n*] [-size *n*[bckMG]]

  FLAG

  - **-H** Non segue i link simbolici tranne quando processa gli argomenti della riga di comando
  - **-L** Nella ricerca segue i link simbolici
  - **-P** non segue in nessun caso i link simbolici (default)
  - **-name** *espress.regol.* Restituisce i file aventi nel nome la stringa *espress.regol*.
  - **-iname** *espress.regol.* Come **-name** ma non distingue maiuscole e minuscole in *espress.regol.*
  - **-user** *utente_1* Cerca i file in percorso aventi come utente proprietario *utente_1*
  - **-group** *gruppo_a* Cerca i file in percorso aventi come gruppo proprietario g*ruppo_a*
  - **-perm** *permessi* Restituisce i file che hanno esattamente i permessi specificati, in modo ottale o simbolico
  - **-perm -** *permessi* Restituisce i file che hanno almeno i permessi indicati, in modo ottale o simbolico
  - **-perm +***permessi* Restituisce i file che hanno alcuni dei permessi specificati, in modo ottale o simbolico
  - **-maxdepth** *X* Prosegue la ricerca massimo fino al livello *X*, con *X* intero non negativo
  - **-mindepth** *X* Prosegue la ricerca a partire dal livello *X*
  - **-mount** Non esegue la ricerca in altre partizioni montate
  - **-**[**mac**]**time** *n* Trova i file o directory la cui data corrisponde a *n* giorni fa, la data in questione può essere la data di modifica (***m*time**), accesso (***a*time**) o creazione (***c*time**)
  - **-**[**mac**]**min** *n* Come l'opzione precedente ma *n* corrisponde a minuti
  - **-**[**nac**]**ewer** *file* Come l'opzione precedente con la differenza che il confronto viene fatto con il tipo di data relativa a *file*
  - **-empty** Solo file e directory non vuoti

- **-links** *n* Solo file che hanno esattamente *n* hard-link
- **-inum** *n* Solo il file che ha inode numero *n*
- **-size** *n*[bckMG] File che usano al massimo *n* unità di spazio, *n* è un intero seguito da una lettera che è:
  - b Per blocchi da 512 byte, default
  - c Per byte
  - k Per Kilobytes (unità da 1024 byte)
  - M Per Megabytes (unità da 1048576 byte)
  - G Per Gigabyte (unità da 1073741824 byte).

ESEMPI

1. **find** */* -iname “*rc” -mount -perm 755 -ctime 10
   Esegue una ricerca, a partire dalla directory radice, ed escludendo gli altri file system, dei file che hanno ACL “rwx r-x r-x” (*-perm 755*), che finiscono per “rc” indipendentemente dall'inizio del nome e senza distinguere maiuscole e minuscole, ed infine che sono stati creati 10 giorni fa
2. **find** */usr* -maxdepth 4 -name 'help**.txt*' -size 3k -group *portage*
   Cerca tutti i file, entrando in un livello massimo di 4 subdirectory a partire da */usr*, che cominciano esattamente con *help* e finiscono con *.txt*, che hanno come gruppo proprietario *portage* e che non siano più grandi di 3kB.

• sed

(stream editor)
(versione GNU sed > 3.01)
E' un editor di flusso molto veloce, esegue modifiche ai dati ricevuti dallo standard input, lavorando con stringhe di qualsiasi lunghezza.
Il comando è line-based, quindi i comandi vengono eseguiti riga per riga, in ordine, scrivendo il risultato nello standard output.
Il comando legge le righe dal pattern buffer del suo standard input, quindi nel caso fosse specificato un file, lo legge, lo elabora e ne mostra il risultato, ma non lo modifica.

SINTASSI

- **sed** [-n] [-e script] [*file*]

FLAG

- **-n** Il normale output non viene mostrato
- **-e** Viene usata l'espressione *script* per elaborare il testo.

ESEMPI

1. **sed** -e 'd' */percorso/file1*
   Legge file1 ed elimina tutte le righe (d)
2. **sed** -e '/regexp/d' */percorso/file1*
   Elimina tutte le righe corrispondenti all'espressione regolare *regexp*
3. **sed** -e -n '/regexp/p' */percorso/file1*

Vengono stampate (p) solo le linee (n) che soddisfano l'espressione regolare

4. **sed** -e -n '/begin/,/end/p' */percorso/file1*
   Blocco di testo che inizia con una riga contenente begin e finisce con end. Se *begin* viene trovato ma non viene trovata nessuna riga contenente *end* dopo di esso, verranno comunque stampate tutte le righe successive a *begin*. Questo accade perché sed è un editor di flusso, non può sapere se nella riga successiva apparirà end
5. **sed** -e 's/sbagliato/giusto/' *file1*
   Verrà mostrato nello standard output il contenuto di file1 con la prima ricorrenza di *sbagliato* di ogni riga, sostituite con la stringa *giusto*
6. **sed** -e 's/sbagliato/giusto/g' *file1*
   Come l'esempio precedente, ma la sostituzione è effettuata in maniera globale
7. **sed** -e '1,10d' */percorso/file1*
   Elimina le righe dalla 1 alla 10 incluse
8. **sed** -e '/^#/d' */percorso/file1*
   Elimina le righe che iniziano per #
9. **sed** -e '/^$/d' */percorso/file1*
   Elimina tutte le linee vuote
10. **sed** -e '1,10s/sbagliato/giusto/g' *file1*
    Come l'esempio precedente, ma verranno analizzate solo le righe dalla 1 alla 10 incluse
11. **sed** -e '/^080/,/^END/s/bari/Bari/g' *file1*
    Verrà mostrato nello standard output il contenuto di file1, *bari* verrà sostituito con *Bari* ma solo nei blocchi di testo che iniziano con una riga che inizia con *080* e terminano con una riga che inizia con *END*
12. **sed** -e 's:/usr/local:/usr:g' *file1*
    E' utilizzato il carattere due punti (:) come separatore, poiché la stringa da cercare ha troppi slash, quindi verrà mostrato il contenuto di file1 dove ogni occorrenza di /usr/local verrà sostituita con /usr
13. **sed** -e 's/<.*>//g' *file1*
    Verrà mostrato nello standard output il contenuto di file1, ma verranno individuate le stringhe che iniziano per “<” e terminano per “>”, contenenti un numero qualsiasi di carattere, sostituendole con una stringa vuota.

- **cut**

  Permette di estrarre sezioni di una riga di testo, il suo utilizzo consueto è quello di spezzettare la riga in più parti delimitate da un carattere separatore.

  SINTASSI
  - **cut** [-d separatore] [-f sottoinsieme]

  FLAG
  - **-d** *separatore* Indica di separare porzioni di riga delimitate da separatore (carattere o stringa)
  - **-f** *intervallo* Indica di considerare *l'intervallo* delle porzioni della riga.

  ESEMPI
  1. **cut** -d : */percorso/file1* -f 3
     Legge *file1*, considera ogni riga composta da più stringhe separate dal separatore due punti (:) e ne mostra, per ogni riga solo la terza (-f 3)
  2. **cut** -d : */percorso/file1* -f 1,3,5-9
     Come nell'esempio precedente, ma verrano mostrate di ogni riga la parte: prima, terza, dalla quinta alla nona (estremi inclusi).

- **diff**

  Trova differenze tra due file.
  Questo comando risulta molto utile per confrontare versioni diverse dello stesso file, oppure per confrontare due file di configurazione, uno funzionante e l'altro meno e capire quali sono le porzioni diverse.
  Se si sostituisce a uno dei file da confrontare il trattino (**-**) questo indica che il confronto verrà eseguito con lo standard input.

  SINTASSI
  - **diff** [-bBr] [--brief] [--ignore-case] *da_file1 a_file2*

  FLAG
  - **-b** Ignora differenze nella quantità di spazi bianchi
  - **-B** Ignora differenze che consistono solo in righe vuote
  - **--brief** Riferisce solo se i file sono diversi, senza mostrarne le differenze
  - **-r** Confrontando due directory, confronta ogni sottodirectory ricorsivamente.
  - **--ignore-case** Considera allo stesso modo maiuscole e minuscole, ignorando ogni relativa differenza.

- **colordiff**

  Il comando ha le stesse opzioni ed effetti di **diff** ma mostrare un output colorato che permette una maggiore leggibilità.

- **slocate**

  (secure locate)
  Versione migliorata e sicura del comando **locate**.
  Il comando fornisce un meccanismo sicuro per indicizzare e cercare quindi rapidamente i file. Usa la codifica incrementale per comprimere il suo database e fare delle ricerche veloci, ma a differenza di locate conserva anche l'ACL dei file, l'utente e il gruppo proprietario. In questo modo non consente agli utenti di mostrare tramite delle ricerche sul suo database file di cui non hanno sufficienti permessi. Chiaramente prima di poter utilizzare questo comando bisogna crearne il database.

  SINTASSI
  - **slocate** [-u] [-U *dir_st*] [-e *dir1,dir2,...*] [-f *tipo_fs1,...*] [-l *n*] [-iq] [-n *num*] [-r *regexp*] [-o *file_db*] [-d *path_db*]

  FLAG
  - **-u** Crea il *database* partendo dal percorso “*/*”, cioè dalla root directory
  - **-U** *dir_st* Crea il database partendo dal percorso *dir_st*
  - **-e** *dir1,...* Esclude le directory specificate (dir1,...) dal database
  - **-f** *tipo_fs1* Esclude i file che sono sui tipi di file system indicati
  - **-l** *n* Livello di sicurezza. Con n uguale a 1 il controllo della sicurezza è abilitata (il default), con n uguale a 0 è disabilitato
  - **-i** Non distingue maiuscole e minuscole
  - **-q** Non mostra i messaggi di errore
  - **-n** *num* Mostra al massimo *num* risultati
  - **-r** *regexp* Cerca nel database usando come espressione regolare *regexp*
  - **-o** *file_db* Specifica il nome del database da creare
  - **-d** *path_db* Specifica il percorso del database in cui cercare.

GESTIONE PROCESSI

GERARCHIA

I processi in Linux sono detti task, non c'è distinzione tra processi e *thread*. Questi ultimi, conosciuti anche come processi a *peso leggero* per via del fatto che non richiedono l'intervento del *process-manager* per il cambio di contesto computazionale (*context switch*), sono ottenuti derivando opportunamente un task figlio da un processo genitore, mediante la chiamata di sistema *clone().*

Tutti i processi derivano da un processo precedente, chiamato il loro *processo genitore*. Un processo oltre ad avere un process-identificator univoco (PID), conserva anche il PID del processo che lo ha generato (PPID, parent process ID).

Quando una shell esegue un comando esterno, non *built-in*, avvia il comando come un processo figlio, mediate la chiamata di sistema *fork()*. Questo eredita le variabili di ambiente del suo genitore e alcuni altri attributi, come la directory di lavoro attuale.
In generale non si ha bisogno di sapere se un comando è un comando interno o un vero programma; comunque, i comandi interni non saranno mostrati nell'output di **ps** o **top** dato che non sono processi separati. Essi sono solo parte della shell. Con la chiamata di sistema *exec()*, si può sostituire i dati ereditati dal processo padre con dei nuovi, rendendo così il nuovo task indipendente in termini di dati ed istruzioni.

Il più delle volte, quando un processo genitore termina, così fa anche quello figlio. Così si può *uccidere* un intero insieme di processi, *uccidendo* il processo genitore.

Nella shell quando si eseguono più comandi concatenati tra loro, tramite pipeline, viene associato a ciascuna pipeline un numero di *job*, mentre per ciascun comando sarà creato un task separato.
La shell ha inoltre la capacità di sospendere e continuare i task o i job selettivamente.

Per ogni processo attivo sono disponibili, instante per instante, diverse informazioni come: un codice che ne descrive lo stato, un tempo di vita, utente che lo ha eseguito, il comando dal quale è stato creato, eventuali sotto-processi e cosi via.

Il comando **ps** è uno strumento utile per mostrare le informazioni relative ai processi attivi e può essere anche utilizzato per esaminare le relazioni genitore-figlio. Il comando **pstree** è dedicato alla visualizzazione dell'albero, cioè la gerarchia di tutti i processi.

Foreground e Background

Tutti i task possono essere eseguiti o in foreground o in background.
Un processo eseguito in **foreground** interagisce con l'utente mediante input ed output, come un browser, un editor di testo eccetera.
Un sinonimo di foreground è *in primo piano*, che sta ad indicare appunto i processi *visibili*, che interagiscono in maniera *diretta* con l'utente.

Un processo in background, o *secondo piano*, è invece un processo che opera sullo *sfondo*, cioè non interagisce direttamente con l'utente.
Tipicamente i task in background sono quelli dei servizi di sistema (*daemon* in Linux), che appunto forniscono costantemente un servizio, ma non sono direttamente visibili all'utente.

1. **Foreground**
   Tutte le pipeline e in generale i comandi, se non specificato diversamente, vengono eseguiti in foreground.
   Le caratteristiche di una pipeline in foreground sono:
   - lo standard in del primo processo e lo standard out dell'ultimo processo sono associati ad un terminale
   - si possono dare delle sequenze di controllo tramite tastiera e ciascun processo della pipeline è in grado di riceverle separatamente.

   La sequenza **CTRL-Z** ferma l'esecuzione di tutti i processi della pipeline restituendo il controllo alla shell, il job in questione viene messo in background.

   Premendo invece **CTRL-C** viene terminato il comando o la pipeline attualmente in esecuzione. Se si preme la sequenza al prompt, senza aver inserito alcun comando, verrà terminato il processo che mostra il prompt e subito riavviato.

   Tutti i processi nella pipeline e il terminale di riferimento hanno un Process Group Identificator (PGID). Quando si invia una sequenza di controllo alla pipeline la shell trova tutti i processi con lo stesso PGID del terminale da cui è stata lanciata e invia loro il segnale corrispondente.

2. **Background**
   Per eseguire un comando (o una pipeline) in background da shell si aggiunge il carattere e-commerciale "**&**"alla fine della linea comando.

   Quando una pipeline viene lanciata in background, la shell non aspetta che la pipeline finisca la computazione prima di ritornare al prompt, il terminale risulta quindi libero per l'immissione di altri comandi mentre il comando precedente viene elaborato in background.

   Quando la pipeline è eseguita in background ha le seguenti caratteristiche:
   - Lo standard in del primo processo come lo standard out dell'ultimo non sono associati ad un terminale
   - Non si possono più dare sequenze di controllo dalla tastiera

Quando viene lanciata una pipeline in background vengono mostrati:

- il numero di job
- il PID dell'ultimo processo della pipeline

ESEMPI

1. yes | cat > /dev/null **&**
   Sul terminale si visualizzerà *[1] 32427* dove [1] è il numero di job e 32427 è il PID del processo

DAEMON

Con il termine *daemon* (demone) in Linux ci si riferisce ad un programma, generalmente di piccole dimensioni e dalle funzionalità ridotte che svolge un **compito specifico**, avviato in background. Si tratta quindi di processi non interattivi che permangono in esecuzione per tutto il periodo di funzionamento del sistema e generalmente si tratta di processi considerati *server*, cioè in grado di offrire un servizio quando ricevono una richiesta da un corrispondente processo client.

Il termine *demone* o *daemon* è tipico degli ambienti Unix, quindi GNU/Linux, in ambiente Windows vengono identificate con il nome di *servizi* o *servizi di sistema*.

Il sistema di avvio dei daemon utilizza un meccanismo derivato da Unix System V. All'interno della directory */etc/rc.d/init.d* (o simile) sono contenuti gli script (di shell) che gestiscono l'avvio e l'arresto dei daemon, uno script per ogni daemon.

Convenzionalmente tali script hanno il nome del servizio (o daemon) che lanciano in esecuzione e accettano diversi parametri:

- **start** avviare il daemon
- **stop** terminarne l'esecuzione
- **restart** riavviare il demone
- **status** mostra se il demone è in esecuzione o meno.

I parametri *start* e *stop* sono sempre presenti.

I demoni vengono quindi attivati dall'esecuzione del relativo script (con parametro *start*), questo meccanismo è utilizzato per automatizzare l'avvio e la chiusura dei vari servizi nei diversi runlevel del sistema.
In questa maniere risulta molto semplice gestire i daemon potendo con semplicità e precisione avere il controllo di ciò che si deve avviare o terminare ad ogni runlevel.

COMANDI

- **yes**

  Stampa il carattere “y”, o la stringa specificata indefinitamente sullo standard output, finché non viene terminato.

  SINTASSI
  - **yes** [*stringa*]

  ESEMPIO
  1. **yes** *stringa di prova*

  OUTPUT
  - stringa di prova
  - stringa di prova
  - stringa di prova
  - ....

- **fg**

  (foreground)
  Porta in foreground un job che prima era in background. Se non viene specificato il job su cui agire si intende quello attuale, ossia l'ultimo referenziato.

  SINTASSI
  - **fg** [*num_job* oppure *%num_job*]

  ESEMPI
  1. *yes | cat > /dev/null &*

  OUTPUT
  - *[1] 25178*

  2. **fg** 1

  OUTPUT
  - *yes | cat > /dev/null*

    Dopo aver eseguito il primo comando (*yes | cat > /dev/null &*), supposto l'output successivo, ossia che il numero del job restituito è 1, il successivo comando (***fg** 1*) porta in foreground tornando a visualizzare sull'output *yes | cat > /dev/null.*

- **bg**

  (background)
  Permette di far riprendere in background l'esecuzione di un job sospeso. Questo è possibile solo se il job in questione non è in attesa di un input o di poter emettere l'output. Se non si specifica il job si intende quello attuale, ossia l'ultimo referenziato.

  SINTASSI
  - **bg** [*num_job* oppure *%num_job*]

ESEMPI

1. *yes | cat > /dev/null*
   premendo CTRL-Z

OUTPUT

- *[4]+ Stopped yes | cat > /dev/null*

2. **bg** 4

OUTPUT

- *[4]+ yes |cat > /dev/null &*

  Se dopo la visualizzazione in foreground del job (*yes | cat > dev/null*) si preme CTRL+Z, si stoppa il processo ricevendo in output la stringa *[4]+ Stopped yes | cat > /dev/null*. Digitando quindi comando ***bg** 4* si rimanda il processo il cui job era appunto 4 in background e l'output diventa così *[4]+ yes | cat > /dev/null &.*

- **kill**

  Il comando **kill** permette di inviare un segnale ai processi di un job, indicando direttamente il PID del processo. Quando si vuole concludere direttamente un job non sempre è sufficiente l'invio di un segnale SIGTERM (15), che è default se non è specificato il tipo di segnale. In questi casi si può utilizzare, con cautela, il segnale SIGKILL (9).

  SINTASSI

  - **kill** [-s *nome_segnale*] [-*nome_segn*] [-*num_segn*] *num_PID*

  *SEGNALI PIÙ FREQ*UENTI

  - **1 : SIGHUP**
    Segnale di aggancio (Hang up). I processi possono ascoltare questo segnale e agire (o non agire) di conseguenza. Indica la caduta della "portante" della linea telefonica.
  - **2 : SIGINT**
    Segnale di interruzione. Questo segnale è inviato ai processi per interromperli. I processi possono elaborare questo segnale e agire di conseguenza.
  - **3 : SIGQUIT**
    Indica, come il precedente, la richiesta di terminazione del processo a cui è diretto, ma genera anche nella home directory dell'utente proprietario un file chiamato *core*, contenente l'immagine di memoria del processo in esecuzione al momento della ricezione del segnale
  - **9 : SIGKILL**
    Segnale di kill. Questo segnale provoca la conclusione immediata del processo dal kernel Linux. I programmi non possono captare questo segnale.
  - **15 : SIGTERM**
    Segnale di conclusione. Questo segnale è inviato ai processi per concluderli. I programmi possono elaborare questo segnale e agire di conseguenza. E' il segnale che si invia premendo **CTRL-C** nella finestra del terminale dove il

programma viene eseguito.

- **30: SIGPWR**
  È il segnale che viene inviato a tutti i processi attivi nel caso in cui si verifichi una caduta di tensione .

ESEMPI

1. **kill** -SIGTERM 25
   Invia al task con PID 25 il segnale di “terminazione” SIGTERM (15), equivalente di **kill** -15 25, oppure **kill** -s SIGTERM 25
2. **kill** -9 25
   Invia al task 25 il segnale di “terminazione forzata” SIGKILL.

- **killall**

Si comporta esattamente come il comando *kill* tranne che per il fatto che non riceve come argomento uno o più PID, bensì cerca i processi che contengono nel nome la stringa specificata e invia loro il segnale indicato. Come il comando *kill*, se non è specificato nessun segnale di default esso è SIGTERM (9).

SINTASSI

kill [-s *nome_segnale*] [-ui] [-*nome_segn*] [-*num_segn*] *num_PID*

FLAG

- **-u** [*utente*] Manda il segnale di terminazione a tutti i processi relativi all'utente specificato.
- **-i** Chiede conferma prima di inviare il segnale, processo per processo.

ESEMPI

1. **killall** *gdm*
   Termina tutti i processi della sessione grafica, questi infatti sono tutti figli del processo *gdm*, il login manager (il default se si utilizza Gnome)
2. **killall** -15 **-u** *utente1* **-i**

OUTPUT

- Kill gnome-keyring-d(9040) ? (y/N)
  Manda il segnale SIGTERM(15) a tutti i processi dell'utente *utente1* chiedendone preventivamente la conferma.

- **jobs**

Mostra lo stato dei job nella sessione corrente, etichetta inoltre il più recente ed il meno recente.

SINTASSI

- **jobs** [-lp]

FLAG

- **-l** Visualizzazione estesa delle informazioni di ogni processo di stato
- **-p** Mostra solo il PID del processo leader del job selezionato.

- **pstree**

  Mostra l'albero dei processi, ossia la gerarchia di tutti i task.

  SINTASSI
  - **pstree** [-hp]

  FLAG
  - **-h** Evidenzia il processo corrente
  - **-p** Mostra i PID dei processi.

- **ps**

  Riporta l'immagine (*snapshot*) dei processi attivi al momento. Le informazioni visualizzate variano in base alle flag utilizzate, possono essere relative anche ad un solo processo o ad un gruppo di essi.

  SINTASSI
  - **ps** [-rxeaf] [f] [-u *lista_user*] [-p *lista_PID*] [-t *lista_tty*]

  Codici dello stato dei processi:
  - **D** *Sleep* - non interrompibile (normalmente I/O)
  - **R** *Running* - o in coda di run
  - **S** *Sleep* - interrompibile (aspetta che un evento si verifichi)
  - **T** Stoppato o da un segnale di controllo del job o perché esso è tracciato
  - **W** Sta subendo la paginazione, non residente in memoria
  - **X** Morto (dead), non sarà mai più attivo
  - **Z** Zombie, cioè è terminato ma il suo codice d'uscita non è stato ancora restituito al processo padre
  - **N** Bassa priorità (frequente per gli utenti)
  - **<** Alta priorità (non frequente per gli utenti)
  - **L** Ha le pagine bloccate in memoria (per real-time e particolari I/O)
  - **s** Leader della sessione (session-leader)
  - **l** Processo multi-thread
  - **+** Nel gruppo dei processi in foreground.

  FLAG
  - **-u** [*lista_utenti*] Solo i processi relativi agli utenti in *lista_utenti*, se non specificato all'utente corrente. Equivalente a **--user**
  - **-r** Restringe la selezione solo ai processi in esecuzione
  - **-x** Tutti i processi che hanno una *tty*
  - **-e** Tutti i processi, equivalente a **-A**
  - **-a** Tutti i processi tranne il session-leader e i processi che non sono associati ad un terminale
  - **-f** Visualizzazione estesa
  - **f** Mostra l'associazione padre-figlio dei processi
  - **-p** [*lista_PID*] Solo le informazioni relative ai PID specificati, equivalente a **p** e –**pid**

- **-t** [*lista_tty*] Tutti i processi associati ai terminali in *lista_tty*, se questa è vuota verranno mostrati i processi associati al terminale corrente, equivalente a **-T**
- **U** [*utente*] Tutti i processi associati all'utente specificato
- **-C** [*comando*] Tutti i processi creati dal comando specificato.

ESEMPIO

1. ps ax

OUTPUT

```
  PID TTY STAT  TIME COMMAND
    1  ?  S    0:02 init [3]
    2  ?  SW   0:00 (kflushd)
    3  ?  SW<  0:00 (kswapd)
  105  ?  S    0:01 klogd
  116  ?  S    0:00 crond
  128  ?  S    0:00 inetd
  139  ?  S    0:00 lpd
  167  ?  S    0:00 httpd
  179  ?  S    0:00 smbd -D
  188  ?  S    0:00 nmbd -D
  202  2  S    0:00 /sbin/mingetty tty2
  208  ?  S    0:00 update (bdflush)
  274  ?  S    0:02 in.telnetd
  465  ?  S    0:00 bash
  714  ?  R    0:00 ps ax
```

- **sleep**

  Sospende la shell per i secondi specificati.

  SINTASSI

  - ***sleep*** *num_secondi*

- **wait**

  Quando una lista asincrona di comandi è stata lanciata dalla shell il PID dell'ultimo comando della lista è noto nelle variabili d'ambiente. Se è lanciato senza parametri la shell aspetta finché tutti i processi conosciuti, invocati dalla shell, sono terminati. Nel caso fosse specificato il PID, la shell aspetta in egual modo ma solo finché non termina il PID specificato.

  SINTASSI

  - **wait** [*PID*]

- **sh**

  (shell)
  L'interprete standard di comandi, permette quindi di eseguire comandi letti da una stringa a riga di comando (**-c**), dallo standard input (**-s**) o da un file specificato.

  SINTASSI

  - **sh** -c *stringa_comando*
  - **sh** -s
  - **sh** *file_script*

- **top**

  Mostra dinamicamente (in foreground), ad intervalli regolari, una serie di informazioni sui processi e sul sistema.

  Lo strumento top produce un output a monitor diviso in:
  - Una parte alta che contiene informazioni generali sul sistema
  - Una tabella centrale che mostra i processi che usano maggiormente la CPU.

  Nella prima riga vengono mostrati:
  - Orario
  - Tempo di attività del sistema, in formato HH:MM
  - Numero di utenti loggati
  - Carico medio di sistema nell'ultimo minuto, negli ultimi 5 minuti e 15 minuti.

  Dalla seconda alla quarta riga sono mostrati informazioni su:
  - Numero totale dei task con resoconto del loro stato
  - Distribuzione del carico di CPU tra processi di sistema, utente e altri
  - Occupazione della memoria principale con sintesi del suo utilizzo
  - Utilizzo della memoria swap.

  Nella parte centrale ci sono le informazioni relative ai processi, racchiuse in tabella avente una riga per ogni processo, dove vengono mostrati:
  - Numero del processo (PID)
  - Utente proprietario del processo (USER)
  - Priorità del processo (PR)
  - Valore di *nice* (NI)
  - Dimensione delle pagine di memoria centrale complessivamente assegnate al processo dal gestore della memoria virtuale (VIRT)
  - Dimensione della memoria fisica attualmente occupata dal processo (RES)
  - Dimensione delle pagine di memoria condivise con altri processi (SHR)
  - Stato del task (S)
  - Utilizzo del processore (%CPU)
  - Utilizzo della memoria principale (%MEM)
  - Tempo di attività del processo (TIME+)
  - Comando utilizzato per avviare il task (COMMAND).

  Sono disponibili una serie di comandi interattivi:
  - k Permette di inserire il PID del processo da terminare
  - r Consente di modificare il valore di *nice* di un processo, che influenza la sua priorità. I valori accettati variano da -20 (massima priorità) a 19 (minima)
  - f Dà la possibilità di aggiungere o togliere alcuni campi nella tabella dei processi
  - s Modifica l'intervallo di aggiornamento di top, di default è 3 secondi.

GESTIONE UTENTI E GRUPPI

TEORIA

In GNU/Linux le informazioni relative agli utenti e gruppi sono conservate nei seguenti file:

- ***/etc/passwd*** Informazioni sugli account degli utenti
- ***/etc/shadow*** Password cifrate degli utenti
- ***/etc/group*** Informazioni sui gruppi di appartenenza degli utenti
- ***/etc/default/useradd*** Directory contenenti le informazioni predefinite per la creazione di nuovi utenti
- ***/etc/skel*** Directory contenente i file usati per la personalizzazione di default della shell per gli utenti
- ***/etc/gshadow*** Informazioni e password cifrate dei gruppi.

Per poter effettuare il login su un sistema Linux è necessario disporre di un account utente.
Nei sistemi Unix oltre agli utenti ci sono i gruppi, che sono anche loro comproprietari dei file. Infatti nella ACL di ogni file sono specificati anche i permessi del gruppo proprietario.

Lo scopo principale dei gruppi è quello di raccogliere gli utenti per consentire loro di fare qualcosa, senza specificare i permessi ad ogni utente del sistema. Quindi se un utente appartiene ad un gruppo, può accedere ai file e ai dispositivi di cui il gruppo è proprietario.

Il gruppo principale associato ad ogni utente è uno solo, ma si può appartenere anche a più gruppi, estendendo così le possibilità dell'utente sul sistema.
A titolo di esempio si può considerare il gruppo "audio", tutti e soli i membri del gruppo audio avranno accesso ai dispositivi audio. Ciò è realizzato semplicemente facendo appartenere i file associati ai dispositivi audio al medesimo gruppo.

I gruppi non hanno un numero massimo di utenti membri. Bisogna comunque tener presente che il superuser appartiene al gruppo di nome *root* e può accedere a qualsiasi file o risorsa del sistema, senza che ne sia esplicitamente membro.

Il file /etc/passwd ha la seguente sintassi:

- *account*:*passwd*:UID:GID:GECOS:*homedir*:*shell*

  - *account* Nome utente con cui si accede alla macchina
  - *passwd* Password criptata oppure “**x**” se si utilizza il file */etc/shadow*
  - *UID* Valore numerico assegnato all'utente (User IDentificator)
  - *GID* Valore numerico del gruppo primario (Group IDentificator)
  - *GECOS* Informazioni generali dell'utente, generalmente vuoto
  - *homedir* Directory principale dell'utente, la cosiddetta *home*
  - *shell* Shell utilizzata di default, solitamente Bash, cioè */bin/bash.*

COMANDI

- **who**

  Mostra gli utenti correntemente loggati sul sistema.

  SINTASSI
  - **who** [-alqur]

  FLAG
  - **-a** Mostra le informazioni estese
  - **-l** Mostra il login dei processi di sistema
  - **-q** Mostra tutti gli utenti loggati e il loro numero complessivo
  - **-u** Lista di utenti loggati
  - **-r** Mostra il runlevel corrente.

- **whoami**

  Mostra l'user ID dell'utente correntemente loggato, equivale a **id -un**.

- **id**

  Mostra l'identità degli utenti del sistema.

  SINTASSI
  - **id** [-gGnu]

  FLAG
  - **-g** Mostra solo l'ID del gruppo principale dell'utente loggato
  - **-G** Mostra tutti gli ID dei gruppi
  - **-n** Usata con i flag -u, -g o -G mostra il nome anziché l'ID numerico
  - **-u** Mostra solo l'ID dell'utente correntemente loggato.

- **last**

  Lista gli ultimi login/logout degli utenti sul sistema e gli ultimi riavvi del sistema, interpellando il file */var/log/wtmp,* non leggibile diversamente. Mostra utente, console di login, display, kernel, data di login e tempo di permanenza.

  SINTASSI
  - **last** [-n *num*] [-adx] *stringa*

    Stringa può essere un nome di un utente del sistema oppure *reboot*, che mostra solo data e

  FLAG
  - **-n** *num* Mostra le ultime *num* righe dell'output
  - **-a** Mostra anche l'hostname
  - **-d** Per i login non locali mostra anche l'ip
  - **-x** Mostra anche gli spegnimenti del sistema e i cambiamenti di runlevel.

- **su**

  (substitute user)
  Cambia ID utente, o diventa superuser se *nome_utente* non è specificato. L'opzione **-** (meno) serve per fornire un ambiente simile a quelli che l'utente troverebbe se effettuasse il login direttamente.
  Se si è superuser si diventa utenti normali senza inserire la password.

  SINTASSI
  - **su** [-] [*nome_utente*]

- **sudo**

  Esegue un comando come un altro utente, ossia con i permessi di quest'ultimo.
  I comandi che ogni utente può eseguire anteponendo il comando sudo si trovano nel file ***/etc/sudoers***, che configura quindi "chi" può fare "che cosa" e con quali permessi. Per l'utilizzo del comando sudo viene chiesta la password dell'utente loggato che di default viene conservata per 5 minuti. Essendo il file di configurazione */etc/sudoers* critico per la sicurezza del sistema, si modifica con l'apposito editor **sudoedit**.

  SINTASSI
  - **sudo** [-H *HOME*] [-u *username*|*#uid*]

  FLAG
  - **-H** *HOME* Setta contestualmente la variabile d'ambiente *HOME* in quella specificata, poiché sudo di default non la modifica
  - **-u** *username*|*#uid* Esegue un comando come utente *username* o come utente che ha l'UID specificato.

- **date**

  Mostra o modifica la data e l'ora di sistema, eseguito senza flag mostra ora e data corrente.

  SINTASSI
  - **date** [-d *stringa*] [-r *file*] [-s *stringa*] [-u oppure –utc]
    Il formato di stringa è MMDDhhmmYYYY: MeseGiornoOreMinutiAnno

  FLAG
  - **-d** *stringa* Mostra data e ora descritti da *stringa*, non quelli attuali
  - **-r** *file* Mostra data e ora dell'ultima modifica di *file*
  - **-u** Mostra data e ora nel formato *Coordinated Universal Time.*

  ESEMPI
  1. **date** 122513002009
     Setta la data a 12 dicembre del 2009, alle ore 13:00.
  2. **date**

  OUTPUT
  - Wed Dec 17 18:18:12 CET 2008

- **passwd**

  Cambia la password utente.
  All'utente viene prima chiesta la password attuale, se presente. Essa viene cifrata e confrontata con quella memorizzata.
  Dopo che la password è stata inserita vengono controllati i parametri dell'invecchiamento della password per verificare che essa possa essere inserita in quel momento. Se è tutto ok viene chiesta la nuova password e ne viene misurata la complessità.

  Poiché non accetta password non sufficientemente complesse è bene che la password abbia uno o più dei seguenti insiemi:
  - Lettere minuscole
  - Lettere maiuscole
  - Numeri da 0 a 9
  - Segni di punteggiatura

  Le informazioni sull'invecchiamento delle password possono essere modificate solo dal superuser che può anche bloccare un account.

  SINTASSI
  - **passwd** [-luS] *login*
  - **passwd** [-g] [-r] *gruppo*
  - **passwd** [-x *max*] [-n *min*] [-w *warn*] [-i *inact*] [-e] *login*

  FLAG
  - **-l** Disabilita l'account impostando la password ad un valore che non corrisponde a nessuna possibile password cifrata
  - **-u** Abilita l'account impostando al suo valore precedente
  - **-S** Ispeziona l'account mostrando 7 campi:
    - Nome dell'utente
    - Se è bloccato (L)
    - Non ha password (NP)
    - Password valida (P)
    - Data ultima modifica della password
    - Età (in giorni) minima e massima della password
    - Periodo (in giorni) di avviso e di inattività.
  - **-g** Cambia la password per il gruppo specificato, l'utente deve essere l'amministratore del gruppo o il superuser
  - **-r** Usata insieme a **-g** elimina la password attuale dal gruppo, permettendo l'accesso al gruppo a tutti i membri
  - **-x** *max* Imposta a *max* il numero massimo di giorni per il quale la password è valida, dopo il quale ne viene richiesta la modifica
  - **-n** *min* Setta a *min* il numero minimo di giorni prima che la password possa essere modificata
  - **-w** *warn* Imposta il numero di giorni nei i quali l'utente verrà avvisato della prossima scadenza della propria password, l'avviso parte *warn* giorni prima della

scadenza

- **-i** *inact* Setta a *inact* il numero di giorni massimi di inattività, passati *inact* giorni di inattività (nessun login) l'utente non potrà più accedere al sistema
- **-e** Fa scadere subito una password, obbligando l'utente ad inserirne una nuova al successivo login.

- **useradd**

Crea un nuovo utente o aggiorna le informazioni predefinite per i nuovi utenti. Quest'ultimo utilizzo del comando si ottiene anteponendo **-D** ai flag.
Normalmente questo comando crea un nuovo account utente usando i valori specificati nella riga di comando ed i valori predefiniti dal sistema. Il nuovo account utente verrà aggiunto ai file di sistema processati, sarà creata l'home-directory e verranno copiati i file iniziali a seconda delle opzioni predefinite o della riga di comando.
Se non è specificata alcuna opzione *useradd* mostra i valori predefiniti correnti. In alcune distribuzioni è presente lo script interattivo **adduser**.

SINTASSI

- **useradd** [-m] [-d *home_dir*] [-e *data_scadenza*] [-f *giorni_inattività*] [-g *gruppo_iniziale*] [-G *gruppo1*[*gruppo2*,...]] [-p *password*] [-s *shell*] [-u *uid*] *login*
- **useradd -D** [-g *gruppo_predef*] [-b *home_predef*] [-e *data_scad_predef*] [-f *giorni_inatt_predef*] [-s *shell_predef*]

FLAG

- **-d** *home_dir* Setta la *home-directory* utente, se non specificato la home-directory è definita appendendo il nome di login a *home_predefinita*. Settare non vuol dire creare, ma è un attributo dell'utente
- ***-m*** Crea anche la *home-directory* utente
- **-e** *data_scadenza* La data dev'essere nel formato MM/GG/AA
- **-f** *giorni_inattività* Numero di giorni dopo la scadenza della password prima che l'account verrà permanentemente disabilitato, altri valori possibili:
  - 0 Disabilita l'account non appena la password è scaduta
  - -1 Il default, non disabilita mai l'account.
- **-g** *gruppo_iniziale* Nome o numero del gruppo iniziale di login dell'utente, detto nome o numero di gruppo deve essere esistente; di default è 1
- **-G** *gruppo1,gruppo2* Lista di gruppi supplementari di cui l'utente è altre sì membro. I nomi devono essere separati da una virgola senza spazi bianchi intermedi, come con il flag -g essi devono tutti esistere. Di default l'utente appartiene

solo al gruppo iniziale.

- **-p** *password* Password criptata. Di default la password è disabilitata.
- **-s** *shell* Nome della shell di login dell'utente, normalmente si lascia vuoto questo campo per utilizzare la shell predefinita del sistema
- **-u** *uid* Valore numerico dell'ID utente. Questo valore deve essere univoco e non negativo. Generalmente si usa un valore di ID superiore sia a 99 che ad ogni altro utente, poiché i valori tra 0 e 99 sono riservati agli account di sistema.

*FLAG anteponendo **-D***:

- **-b** *home_predef* Percorso predefinito per la *home-directory* del nuovo utente, il nome di login verrà aggiunto alla fine di *home_predefinita*
- **-e** *data_scad_predef* Data in cui l'account utente verrà disabilitato
- **-f** *giorni_inatt_predef* Numero di giorni dopo la scadenza della password prima che l'account venga disabilitato
- **-g** *gruppo_predef* Nome o numero del gruppo iniziale di login dell'utente, detto nome o numero di gruppo deve essere esistente
- **-s** *shell* Nome della shell di login dell'utente, il programma nominato verrà usato per tutti gli account dei futuri utenti.

- **userdel**

Rimuove l'account di un utente e i file relativi.
Questo comando elimina un account del sistema, rimuovendo tutte le voci che si riferiscono a *login*, che deve esistere e non deve essere attualmente connesso. Per eliminare un account in quest'ultima ipotesi bisogna terminare ogni processo in esecuzione che appartenga a detto account.

SINTASSI

- **userdel** [-r] *login*

FLAG

- **-r** Verranno rimossi i file della home-directory utente, i file collocati in altri file system dovranno essere cercati e rimossi a mano.

VALORI RESTITUITI:

- 0 Successo
- 6 L'utente specificato non esiste
- 8 Utente correntemente loggato.

- **chsh**

  (change shell)
  Cambia la shell di login dell'utente, in altre parole determina il comando iniziale eseguito quando un utente fa il login. Un utente qualsiasi può cambiare la shell di login solo per il proprio account, mentre il superuser può cambiare la shell di login per ogni account.

  SINTASSI
  - **chsh** [-s *shell_di_login*] *utente*

- **chfn**

  (change finger)
  Cambiare le informazioni personali dell'utente quali il nome completo, il numero dell'ufficio, il telefono dell'ufficio e il numero telefonico di casa dell'utente. Queste informazioni sono generalmente mostrate dal comando **finger** o programmi simili. Un utente qualsiasi può modificare solo i propri dati, mentre l'amministratore può cambiare i dati di ogni account. Inoltre, soltanto all'amministratore è permesso usare l'opzione -o per cambiare le parti non definite del campo GECOS.

  SINTASSI
  - **chfn** [-f *nome_completo*] [-r *numero_stanza*] [-w *tel_lavoro*] [-h *tel_casa*] [-o *altro*] *utente*

- **chage**

  (change age)
  Cambia le informazioni sulla scadenza della password. Il comando modifica il numero minimo di giorni tra i cambi di password e la data dell'ultimo cambio. Queste informazioni sono usate dal sistema per determinare quando un utente deve cambiare la propria password. L'uso del comando *chage* è permesso solo al superuser, tranne per l'opzione -l, che può essere usata da un utente non privilegiato per conoscere la scadenza della propria password o dell'account.

  SINTASSI
  - **chage** [-m *gg_min*] [-M *gg_max*] [-d *ultimo_giorno*] [-I *inattività*] [-E *data_scadenza*] [-W *gg_avviso*] *utente*
  - ***chage*** -l *utente*

- **groupadd**

  Crea un nuovo gruppo usando i valori specificati nella riga di comando ed i valori predefiniti dal sistema, il nuovo gruppo verrà quindi aggiunto ai file di sistema necessari.

  SINTASSI
  - **groupadd** [-g *GID* [-o]] *gruppo*

  FLAG
  - -**g** *GID* Specifica il valore numerico dell'identificatore del gruppo (GID), esso deve essere univoco a meno che non si usi, come in *useradd* l'opzione -o. Di default *GID* è il valore minimo superiore sia a 999 che a qualunque altro gruppo, poiché i valori tra 0 e 999 sono riservati per account di sistema.

- **groupdel**

  Modifica i file di account del sistema, rimuovendo tutte le voci che si riferiscono a *gruppo*, che deve esistere.
  Non si può rimuovere un gruppo che sia gruppo primario di un utente senza aver prima rimosso quest'ultimo.

  SINTASSI
  - **groupdel** *gruppo*

  VALORI RESTITUITI
  - 0 Successo
  - 6 Il gruppo specificato non esiste
  - 8 Il gruppo è primario per un utente esistente.

- **groups**

  Mostra i nomi dei gruppi a cui appartiene l'utente correntemente loggato o di quello specificato.
  Per ciascuno di tali gruppi viene mostrato il nome, se il gruppo non ha una voce corrispondente nel file */etc/group* esso viene mostrato in forma numerica.

  SINTASSI
  - **groups** [*utente*]

- **newgrp**

  Effettua l'accesso ad un nuovo gruppo, cioè permette di cambiare il proprio ID di gruppo durante una sessione di login.
  Se viene specificato il flag trattino (**-**) l'ambiente dell'utente viene reinizializzato come se l'utente stesse effettuando nuovamente il login.

  SINTASSI
  - **newgrp** [-] [*gruppo*]

- **sg**

  (switch group)
  Il comando funziona in maniera analoga a *newgrp*, ma accetta un comando che viene quindi eseguito con la shell */bin/sh* e con l'ID gruppo specificato. Il flag trattino (-) ha lo stesso effetto di quella di *newgrp*.

  SINTASSI
  - **sg** [-] *gruppo* [-c *comando*]

ALIAS

- **alias**

  Il comando interno **alias** permette di definire gli alias, cioè dei sostituti ai comandi. Quando deve eseguire un qualunque comando, la shell cerca prima all'interno dell'elenco degli alias e, se lo trova li, lo sostituisce con la stringa associata.
  Il comando interno **unalias** permette di rimuovere la definizione di un alias o di tutti gli alias.
  La sostituzione non avviene se il comando in questione o la prima parola dell'alias è tra virgolette.
  Data la sintassi del comando alias, il nome dell'alias non può contenere il simbolo uguale (=).
  Quando la shell incontra un comando che in realtà è un alias questo viene automaticamente espanso sostituendo al nome dell'alias la stringa precedentemente associata. Dato che gli alias vengono espansi solo quando la shell funziona in modalità interattiva, questi non sono disponibili negli script.
  Il contenuto di un alias può fare riferimento ad un altro alias e così di seguito, questo ciclo si ferma quando non ci sono più corrispondenze con nuovi alias, in modo da evitare una ricorsione infinita.

  SINTASSI
  - **alias** [nome_alias[=stringa]]
  - **unalias** nome_alias
  - **unalias** -a

  ESEMPI
  1. **alias** rm=“rm -i”
     Una volta definito questo alias, ogni volta che si userà il comando *rm* verrà utilizzato il flag *-i*, cioè se ne chiederà ogni volta conferma
  2. **alias** lsa=“ls -a”
     Si definisce un nuovo comando *lsa*, che quando eseguito mostrerà anche i file nascosti, poiché *lsa* verrà espanso con “ls -a”
  3. **unalias** rm
     Viene eliminata la definizione di alias associata ad rm
  4. **unalias** -a
     Vengono eliminate tutte le definizioni di alias.

VARIABILI D'AMBIENTE

TEORIA

In Linux ogni processo ha un *ambiente* (environment) ad esso associato, quindi anche la shell ne ha uno. Un ambiente è un insieme di *variabili di ambiente*.
Una variabile di ambiente ha un nome fisso che la identifica e un valore modificabile che contiene informazioni usate da una o più applicazioni.
Tutte queste variabili sono utili ai programmi e al sistema per vari compiti, tra i quali identificare il percorso di directory, file eseguibili e di configurazione ecc.
Le variabili d'ambiente di un processo sono altre sì note anche ai suoi processi figli.

Per inizializzare una variabile (es. **A**) la sintassi è semplice: **A**=34.
Non ci devono essere spazi prima e dopo il simbolo di uguale (=), mentre il valore è una stringa, che può anche essere vuota.
Ma non deve contenere spazi o punteggiatura, sono ammessi solo il trattino basso underscore (_), la virgola (,) e i due punti (:).

Per inserire spazi e punteggiatura estesa la stringa deve essere racchiusa da virgolette singole (') o doppie (“) ad esempio: ***B***=”frase con spazi! e punteggiatura;”
Per mostrare ed in generale utilizzare una variabile essa dev'essere preceduta dal simbolo del dollaro ($).
Per convenzione le variabili vengono indicate con lettere maiuscole. Es: **echo** $B

Bisogna fare attenzione alla differenza tra variabili della shell e variabili d'ambiente. In entrambi i casi le variabili sono utilizzabili dalla shell ma solo le variabili d'ambiente saranno *esportate* automaticamente ai processi figli, che quindi ne potranno usufruire.

L'assegnazione o la creazione di una variabile ha validità solo nell'ambito della shell stessa, i programmi avviati dalla shell non risentono di queste variazioni.
Per poter ovviare a questo comportamento di default, occorre esportare una o più variabili della shell nelle variabili d'ambiente utilizzando il comando **export**.

Per mostrare le variabili d'ambiente, cioè mostrare l'environment, c'è il comando **printenv**, mentre il comando **set**, lanciato senza parametri, mostra tutte le variabili, comprese quelle della shell e quindi anche create manualmente.
Una volta creata una variabile la si può rimuovere dandola come argomento al comando **unset**.

Le variabili d'ambiente più significative:

- **PATH** Percorso di ricerca eseguibili
- **HOME** Directory home dell'utente loggato
- **PAGER** Percorso del programma paginatore di default
- **USER** Nome dell'utente loggato
- **SHELL** Percorso dell'eseguibile della shell di default
- **PWD** Percorso della directory di lavoro corrente
- **OLDPWD** Percorso della directory precedentemente visitata
- **SHLVL** Numero di livello di annidamento della shell.

Le variabili della shell più significative:

- **PS1** Espressione che indica i campi e i colori del prompt dei comandi
- **HOSTNAME** Nome del sistema
- **UID** User Id numerico dell'utente loggato
- **GROUPS** Vettore contenente i numeri di ID dei gruppi di cui l'utente è membro
- **SECONDS** Numero di secondi trascorsi dall'avvio della shell
- **HOSTTYPE** Nome del tipo di calcolatore
- **OSTYPE** Nome del sistema operativo
- **BASH_VERSION** Numero di versione della shell
- **PPID** Process ID del processo genitore della shell attuale
- **HISTSIZE** Numero di comandi da conservare nel file storico dei comandi
- **HISTFILE** Percorso del file storico dei comandi inseriti

Quando si digita un comando nella shell, essa deve trovare il programma sul filesystem prima di eseguirlo.
Se la shell dovesse cercarlo in tutto il filesystem, sarebbe molto lenta; invece, cerca in una lista di directory contenuta nella variabile di ambiente **PATH**.

Il valore della variabile PATH è una lista di directory separate dal carattere ":" Questa lista di directory costituisce il *percorso di ricerca* della shell; quando si introduce un comando, la shell guarderà ognuna di esse alla ricerca del programma che si è chiesto di avviare, o di cui deve completare il nome.
Potrebbe essere necessario modificare la variabile PATH se si installano programmi in posizioni non standard.

Ad esempio un PATH predefinito sui sistemi Debian potrebbe essere */usr/local/bin:/usr/bin:/bin.* Questo valore è definito nel file */etc/profile* e viene applicato a tutti gli utenti. Si può cambiare facilmente il valore, proprio come si può modificare ogni altra variabile d'ambiente.
(Tutti i file di configurazione di Linux sono testuali, in ASCII puro)

Notare che la ricerca dell'eseguibile si ferma alla prima occorrenza trovata, quindi con il path descritto in precedenza se lo stesso comando fosse sia in */usr/local/bin* che in */bin*, non verrebbe mai utilizzato quello presente in */bin* a meno che, ovviamente non si specifichi il percorso completo dell'eseguibile.
Per vedere da quale percorso verrà utilizzato un eseguibile, cioè un comando si può darlo come argomento a **type**.

Il percorso della home-directory dell'utente, definito dalla variabile **HOME**, si può anche utilizzare nei percorsi con il simbolo tilde (~).
Nella shell tale simbolo è di norma accessibile, con la tastiera italiana, premendo i tasti **ALTGR**+**ì** (in ambiente X-window) oppure **ALTGR**+**0** (nella shell testuale).

COMANDI

- **printenv**

  (print environment)
  Mostra tutte le variabili di ambiente, o quelle specificate. Per mostrare una variabile con questo comando non bisogna farla precedere dal simbolo del dollaro ($) ma bisogna indicarne semplicemente il nome, sempre facendo attenzione alle maiuscole.

  SINTASSI
  - **printenv** [*variabile1 variabile2* ...]

  ESEMPIO
  1. **printenv** SHELL EDITOR LANG

  OUTPUT
  - /bin/bash
  - /bin/nano
  - it_IT

- **set**

  Utilizzato senza argomenti mostra tutte le variabili della shell e d'ambiente. Inoltre permette, con gli opportuni flag, di impostare l'esportazione e altri parametri.

- **unset**

  Rimuove la definizione della variabile indicata come argomento.

  SINTASSI
  - **unset** *variabile1* [*variabile2* ...]

- **type**

  Scrive una descrizione del comando, o lista di comandi indicatogli come argomento, in particolar modo fornisce l'informazione del percorso completo dell'eseguibile che sarà utilizzato e degli eventuali alias presenti al posto del comando.

  SINTASSI
  - **type** *nomecomando*

  ESEMPIO
  1. **type** bash lsl type

  OUTPUT
  - bash is /bin/bash
  - lsl is aliased to `ls -la'
  - type is a shell builtin

- **export**

  Esporta una variabile come variabile d'ambiente.

  SINTASSI
  - **export** *nome*[=valore]

  ESEMPIO
  1. **export** PATH=$PATH:/nuova/directory/da/inserire/nel/path
     aggiunge una nuova directory alla variabile PATH

- **chroot**

  Esegue un comando (se specificato) o una shell con una directory root diversa da quella di sistema.

  SINTASSI
  - **chroot** *nuova_dirroot* [*comando*]

GESTIONE FILE SYSTEM

INTRODUZIONE

Per poter accedere ad un'unità periferica di memorizzazione, occorre che il filesystem di questa sia quindi *montato* in quello globale che ha in **/** la directory radice o root-directory.

Per montaggio si intende l'operazione con la quale un filesystem di una qualsiasi periferica a blocchi viene “innestato” nell'albero globale delle directory. In altre parole per poter accedere ad una partizione, cioè ad un filesystem, sia locale che remoto, bisogna prima *agganciare* il file system a cui si vuole accedere ad un nodo della directory radice.
Anche la partizione principale del sistema, in fase di avvio, viene montata appunto nella directory radice “**/**”.

Di norma le distribuzioni montano temporaneamente le unità esterne nelle apposite subdirectory */mnt* o */media*.

MBR – MASTER BOOT RECORD

Sono i primi 512 byte di un unità di memorizzazione di massa, come un hard disk, pendrive USB etc.
Questo particolare settore è molto importante nell'avvio del sistema (vedesi sezione Avvio del sistema -> Boot Loader) e contiene una informazione preziosissima per l'utilizzo dell'intera unità la tabella delle partizioni: la **Partition table**.

Ecco quali sono le informazioni memorizzate al suo interno:

Byte	Descrizione
446	Sezione del codice eseguibile
4	Firma identificativa opzionale del disco
2	Normalmente nulli.
64	Tabella delle partizioni – Partition table
2	Firma identificativa dell'MBR

I punti interessanti sono i primi 446 byte, che contengono il codice eseguibile all'avvio del sistema (trattato nell'omonima sezione in seguito), e sopratutto i 64 byte della partition table, esaminata nel prossimo paragrafo.

PARTIZIONAMENTO

Prima di entrare nel dettaglio di come vengono gestiti i filesystem in ambito Linux è necessario comprendere il meccanismo mediante il quale i dischi vengono divisi in partizioni.

Una partizione è una zona contigua di un unità di memorizzazione di massa, come un hard disk, una scheda di memoria (es. Secure Digital, Memory Stick), una pen-drive USB etc. Per poter utilizzare ognuna di queste unità, siano esse realizzate con supporti magnetici o altro, deve necessariamente essere definita almeno una partizione, e questo non solamente in ambito Linux.

In ogni hard disk (PATA o SATA) possono essere definite al massimo quattro partizioni **primarie**, definite proprio nella partition table. Limitate a quattro proprio dalla dimensione ristretta di quest'ultima, poiché per ogni partizione bisogna conservare una serie di informazioni come: cilindro/settore iniziale e finale (nel caso di un hard disk), tipo di partizione, eventuali flag, etc. Queste partizioni primarie sono quelle da cui il BIOS può caricare il sistema operativo in fase di boot.

Per ovviare a questo problema si sono introdotte le partizioni **estese.** Queste sono delle particolari partizioni primarie che servono per contenere al loro interno diverse partizioni **logiche**, il settore di boot della partizione estesa contiene appunto le informazioni sulle partizioni logiche contenute, che invece hanno normalmente il proprio settore di boot vuoto.
Le partizioni logiche sono normali partizioni utilizzabili dal sistema operativo come quelle primarie, e non hanno differenze prestazionali con esse.

Le partizioni primarie vengono numerate da Linux con numeri da 1 a 4, mentre le logiche da 5 in su, anche se nel disco ci sono meno di 4 partizioni primarie. La partizione estesa è anch'essa numerata e le partizioni logiche in essa contenute hanno una numerazione crescente, partendo appunto da quella estesa in cui si trovano.

Ovviamente in ognuna di queste partizioni, primarie o logiche che siano, bisogna prima applicare un filesystem per poterle utilizzare. Infatti una cosa è definire una partizione indicando quindi la dimensione, collocazione nel disco, etc.. Altra cosa è poi, successivamente, applicarvi un filesystem per poter permettere al sistema operativo di usare quella porzione di disco.

NOMENCLATURA DEI DEVICE

Ciascun filesystem, ad esempio un hard disk, una pen-drive USB o una generica partizione formattata, è visto dal sistema operativo come un dispositivo (*device*) a blocchi.
Tutte le periferiche, o meglio, tutti i file associati alle periferiche risedono nella directory */dev*. E' qui che ci sono i device a cui possiamo associare una directory, cioè che possiamo montare in una directory.

I nomi dei file associati ai dispositivi di memorizzazione di massa seguono alcune regole: le prime due lettere del nome identificano il tipo di device e contestualmente il cavo o il canale con il quale è connesso al sistema, la terza indica generalmente il numero di device rispetto al canale. La quarta lettera indica il numero di partizione presente sul device.
A meno che nel kernel non sia specificata in fase di configurazione l'utilizzo della libreria *libsata*, gli hard disk, cosi come le periferiche ottiche come i lettori CD/DVD-ROM assumo nomenclatura diversa in base a se utilizzano un collegamento PATA (o EIDE) oppure il più recente SATA (versione 1 o 2).
I primi sanno nominati con file del tipo “hd”, del tipo /dev/**hd**xy, i secondi, come anche le periferiche USB e le schede di memoria lette con lettori che utilizzano collegamento USB, saranno accessibili da file “sd”, del tipo /dev/**sd**XY. Con X, canale utilizzato, che ha valore letterario (a, b, c..) e Y, numero partizione, numerico (1,2,3..).
Esempi:

- /dev/hda Hard disk o unità ottica sul primo canale PATA (EIDE), slot master
- /dev/hdb Hard disk o unità ottica sul primo canale PATA (EIDE), slot slave
- /dev/hdc Hard disk o unità ottica sul secondo canale PATA (EIDE), slot master
- /dev/hda3 Terza partizione dell'hard disk *hda*, le unità ottiche non hanno partizioni
- /dev/sda Primo device seriale: hard disk SATA, unità ottica o pendrive USB
- /dev/sda1 Prima partizione di *sda*, la classica per una pendrive USB
- /dev/sdb2 Seconda partizione del device *sdb*
- /dev/cdrom Link simbolico automatico all'unità ottica del lettore cdrom
- /dev/cdrw Link simbolico automatico all'unità ottica del masterizzatore cdrom
- /dev/dvd Link simbolico automatico all'unità ottica del lettore dvdrom
- /dev/dvdrw Link simbolico aut. all'unità ottica del masterizzatore dvdrom

L'operazione di montaggio di un device viene effettuata attraverso il comando **mount** (trattato nella successiva sezione dei comandi), fornendo come argomenti almeno il percorso del device a blocchi e la directory a cui associarlo.
Si può anche specificare il tipo di filesystem della partizione e le opzioni di montaggio. Se non viene specificato il filesystem, il sistema tenterà di riconoscerlo automaticamente.
L'operazione inversa, cioè “rompere” l'associazione del device a blocchi con la directory si effettua con il comando **umount** (trattato più avanti nei comandi).

MTAB ED FSTAB

I file del sistema che intervengono nel montaggio dei device sono:

- **/etc/mtab**
- **/etc/fstab**

In */etc/mtab* c'è l'elenco di tutti i *mount* finora effettuati dal sistema, il comando mount quindi scrive una riga in questo file. Qui sono specificate anche le opzioni con le quali sono stati montati i vari device.

Il file di configurazione dei file system presenti nel sistema è */etc/fstab*. In questo file sono elencati tutti i mount-point statici del sistema che vengono montati automaticamente durante la fase di avvio, a meno che nel mount-point non sia specificata l'opzione *noauto*.
Di default gli utenti non possono utilizzare il comando mount, a meno che non sia specificato in */etc/fstab* od utilizzino il comando **sudo**.

Questo file fondamentale contiene una serie di righe il cui formato è il seguente:

- *device /mount-point tipo_fs opzioni dump_freq passno*
  - device — Nome del dispositivo, che deve esistere
  - /mount-point — Percorso completo della directory, che deve esistere sulla quale montare il file system
  - tipo_fs — Il tipo di file system del device, da passare a **mount**
  - opzioni — Le opzioni di montaggio di mount, separate da una virgola
  - dump_freq — Viene usato dal programma dump per determinare quali file system richiedono un dump. Se non si specifica nulla viene assunto il valore zero
  - passno — Determina l'ordine secondo il quale i filesystem vengono controllati all'avvio del sistema. I filesystem che devono saltare il controllo devono avere questo campo impostato a zero. Il file system root, che deve essere controllato prima di qualsiasi altra cosa, deve avere *passno* settato a 1. Gli altri file system devono avere valore maggiore di 1. Se hanno lo stesso valore il tool **fsck** tenterà di controllarli in parallelo.

COMANDI

- **mount**

  Monta un file system.
  Quando è presente un mount-point in */etc/fstab* si può montare un device fornendo al comando **mount** anche solo il nome del device o il nome della directory.
  Per mostrare i dispositivi attualmente montati si può o eseguire il comando **mount** senza parametri o consultare il file */etc/mtab.*

  SINTASSI
  - **mount** [-a] [-o *opzioni*] [-t *tipo_fs*] *device directory*

  FLAG
  - **-a** Monta tutti i file system presenti in */etc/fstab*
  - **-o** *opzioni* Monta il file system con delle opzioni, le più frequenti:
    - rw Per i file system leggibili e scrivibili
    - ro Quando si vuole che il file system sia accessibile in sola lettura
    - noauto Per evitare che il file system venga automaticamente montato durante la sequenza di avvio, opposto al flag *auto*
    - dev Interpreta i device speciali a blocchi o a caratteri sul filesystem, l'opposto è *nodev*
    - exec Consente l'esecuzione dei file binari dal file system, il contrario è il flag *noexec*
    - atime Aggiorna la data di accesso agli inode per ogni accesso a file, il flag opposto è *noatime*
    - suid Consente l'effetto del bit di identificazione utente o di identificazione gruppo
    - sync Cerca di fare tutte le operazioni di I/O in maniera sincrona, il flag contraria è *async*
    - user Consente agli utenti ordinari di effettuare il montaggio, opposta al flag *nouser.*
      Implica i flag: noexec, nosuid, nodev
    - users Consente a tutti gli utenti di montare e smontare il filesystem.
      Implica i flag: noexec, nosuid, nodev
    - default Usa i flag di default: rw, suid, dev, exec, auto, nouser, async.

- **-t** *tipo_fs* Questa voce può essere anche *auto*, cioè si tenta il riconoscimento automatico del tipo di filesystem. I più usati sono:
  - ext2 Uno dei file system standard di Linux
  - ext3 Versione dell'ext2 con journalig, il default per quasi tutte le distribuzioni
  - reiserfs File system avanzato con journaling
  - iso9660 File system standard dei cd e dvd rom
  - nfs Network file system, per la condivisione delle directory in rete
  - vfat Per Windows® FAT16 e FAT32
  - ntfs Per Windows® NTFS.

ESEMPI

1. **mount** -t iso9660 */dev/cdrom /mnt/cdrom*
   Monta il cdrom nella directory */mnt/cdrom*
2. **mount** -t auto -o rw */dev/sdb /media/pendrive*
   Monta una pendrive usb in modalità lettura e scrittura, con il riconoscimento automatico del file system, nella directory */media/pendrive.*

- **umount**

Smonta un file system. Si può specificare la directory in cui è montato il file system, o il device di appartenenza oppure entrambi.

SINTASSI

- **umount** [-adnr] [-o *opzioni*] [-t *tipo_fs*] *directory device*

FLAG

- **-a** Smonta tutti i file system elencati in */etc/mtab*
- **-d** Se la device montata era una loop-device, si occupa anche di liberare la risorsa.
- **-n** Smonta il device senza scrivere nel file */etc/mtab*
- **-r** Nel caso lo smontaggio fallisse rimonta il file system in modalità *readonly* (*ro)*
- **-o** *opzioni* Smonta tutti i file system che hanno le *opzioni* indicate in */etc/mtab*
- **-t** *tipo_fs* Smonta tutti i file system che hanno il file system, indicati da *tipo_fs* separati da un virgola, cercando in */etc/mtab.*

ESEMPIO

1. **umount** */mnt/dvd*
   Smonta la directory /mnt/dvd, rimuovendo nell'albero generale l'associazione con il device a blocchi associato.

- **df**

Mostra la quantità di spazio usato e disponibile sui file system. Se usato senza argomenti mostra la quantità di spazio usato e libero su tutti i file system correntemente montati. Altrimenti il comando mostra le informazioni dei file system che contengono i file dati come argomento.

SINTASSI

- **df** [-il] [-hHkm] [-tx *tipo_fs*] [*lista_file*]

FLAG

- **-i** Da informazioni sull'uso degli inode invece che dei blocchi
- **-l** Limita il risultato ai soli file system locali
- **-h** Aggiunge a ciascuna dimensione un suffisso, come M per megabyte *binario*
- **-H** Come -h ma usa le unità ufficiali del SI (Sistema Internazionale), cioè con potenze di 1000 anziché 1024, un Mbyte è quindi considerato di 1000^2 anziché 1024^2
- **-k** Dimensioni in blocchi da 1024 byte
- **-m** Dimensioni in blocchi da 1024^2 byte
- **-tx** *tipo_fs* Con -t limita l'elenco mostrato a file system di tipo *tipo_fs*, con il flag -x esclude quelli di tipo *tipo_fs.*

ESEMPI

1. **df** */etc/resolv.conf* -k
   Mostra la quantità di spazio usato e libero sul file system che contiene */etc/resolv.conf* mostrando le dimensioni in blocchi da 1024 byte (*-k*)
2. **df** -x iso9660 -l
   Mostra la quantità di spazio usato e libero di tutti i file system locali (*-l*) correntemente montati ad eccezione dei cdrom, ossia quelli con file system di tipo iso9660 (*-x iso9660*).

- **du**

Mostra la quantità usata di spazio su disco usato dai file specificati e da ciascuna directory nelle gerarchie con base nei file specificati. Con “spazio di disco usato” si intende lo spazio usato dall'intera gerarchia di file al di sotto del file specificato.
Senza argomenti, il comando mostra lo spazio di disco utilizzato dalla directory corrente.

SINTASSI

- **du** [-acDlLsSx] [-hHbkm] [--block-size=*n*] [--maxdepth=*n*] [*lista_file*]

FLAG

- **-a** Mostra lo spazio occupato da ogni file, non solo dalle directory
- **-b** Stampa le dimensioni in byte invece che in Kbyte
- **--block-size**=*n* Stampa le dimensioni in blocchi da *n* byte
- **-c** Scrive un totale di tutti gli argomenti dopo che

questi sono stati processati

- **-D** De-referenzia i link simbolici in *lista_file* non interviene sugli altri link simbolici
- **-h** Aggiunge a ciascuna dimensione un suffisso, come M per megabyte binario
- **-H** Come -h ma usa le unità ufficiali del SI (Sistema Internazionale), cioè con potenze di 1000 anziché 1024, un Mbyte è quindi di 1000^2 anziché 1024^2
- **-k** Mostra le dimensioni in blocchi da 1024 byte
- **-l** Conta i link, ossia conta la dimensione di tutti i file anche se essi sono stati già contati tramite un loro hard link
- **-L** Deferenzia i link simbolici, cioè mostra lo spazio su disco usato dal file o directory a cui punta il link, anziché lo spazio usato dal link
- **-m** Mostra le dimensioni in blochhi da 1024^2 byte
- **--maxdepth**=*n* Stampa il totale di una directory, o di un file con l'opzione -a, solo se questa è n o meno livelli al di sotto dell'argomento dato in *lista_file*
- **-s** Mostra per ciascun argomento in *lista_file* solo il totale
- **-S** Conta separatamente la dimensione di ciascuna directory, escludendo le dimensioni delle subdirectory
- **-x** Salta le directory che si trovano in un file system diverso da quello in cui si trovano gli argomenti in *lista_file*

ESEMPI

1. **du** */home/utente*
   Mostra la dimensione occupata dalla directory */home/utente* e ricorsivamente da ciascuna delle sue subdirectory
2. **du** */home/utente* -S
   Mostra la dimensione occupata dalla sola directory */home/utente*, subdirectory esclusa.

- **cfdisk**
  Crea, modifica ed elimina le partizioni, utilizzando nella shell una interfaccia grafica minimale basata su *curses.*

- **mkfs**
  Costruisce un filesystem nella partizione specificata con l'opzione **-t**. Si possono utilizzare anche i comandi **mkfs.*tipo_fs***, (es. *mkfs.ext2)*. Le opzioni riguardo la creazione sono equivalenti e ne esistono diverse.

- **mkswap**
  Crea un'area di swap nella partizione specificata o nel file specificato. Esistono diverse opzioni di creazione.

- **swapon/swapoff**
  I due comandi attivano o disattivano l'area di swap specificata.

AVVIO DEL SISTEMA

BIOS

Il sistema operativo viene lanciato in esecuzione dal bootstrap, cioè la procedura di avvio del sistema. Nel seguito è presa a riferimento l'architettura PC x86, ma molti dei concetti illustrati sono più generali.

Non appena viene data tensione alla scheda madre (*motherboard*) del calcolatore viene avviato il programma **BIOS** (*Basic Input Output System*), cioè il sistema di base per l'Input/Output, che identifica una serie di routine, scritte per lo più in linguaggio macchina x86.
Queste routine in parte vengono eseguite all'avvio del sistema ed in parte rimangono a disposizione del sistema operativo, che può richiamarle quando è necessario per mezzo di quelle che vengono chiamate *BIOS Call.*
Esse svolgono funzioni di collegamento tra il software e l'hardware. Per questo, pur essendo pochissimi i produttori di BIOS, ciascuna versione deve essere scritta specificamente, o comunque adattata, per l'hardware sul quale è destinata a girare.

Il BIOS risiede in una EPROM (*Erasable Programmable Ready-Only Memory*) ed una volta avviato controlla la presenza di RAM, periferiche IDE, tastiera ecc., si accerta del funzionamento dell'hardware, cercando di determinarne il tipo.
Una volta compiute tutte queste operazioni il BIOS carica il primo settore del primo cilindro del *dispositivo di boot* selezionato ed inizia ad eseguire le istruzioni in esso contenute, cedendo quindi il controllo del sistema.
Il dispositivo *di boot*, di default il primo hard disk del primo canale IDE, si determina dal menù del BIOS, accessibile nei primi istanti di avvio del calcolatore.

BOOT LOADER

Il BIOS cede il controllo al primo settore del primo cilindro (grande 512 byte) del dispositivo di boot, tipicamente un hard disk.
Questo particolare settore viene chiamato **Master Boot Record** (vedere il paragrafo *Gestione dei filesystem – MBR*).
Nei primi 512 byte del dispositivo di boot c'è un piccolo programma che si occupa di caricare il kernel in memoria ed eseguirlo: il **Boot Loader**.
Il compito del boot loader è caricare il kernel del sistema operativo in memoria ed eseguirlo, cedendogli quindi il controllo.

Una configurazione alternativa, tipica dei sistemi PC con Linux, prevede un bootstrap a due fasi: nell'MBR è installato un **boot manager**, che si occupa di mansioni molto elementari (che possono quindi essere codificate in poco spazio) cioè ricercare la partizione di boot sul dispositivo di boot, cercare e caricare in memoria il boot loader (che in questo caso è un programma eseguibile più complesso ed esteso) ed infine passare a quest'ultimo il controllo.
La seconda fase quindi inizia quando viene caricato l'eseguibile del boot loader, che è il vero e proprio gestore d'avvio del sistema operativo.

Esso è molto più sofisticato del primo stadio e si occupa principalmente, ma non solo, di scegliere quale kernel caricare in memoria per poi cedergli il controllo. Un singolo boot loader è in grado di caricare molti tipi di kernel e quindi anche diversi sistemi operativi presenti in partizioni differenti. Naturalmente può essere caricato un solo kernel per volta.
In alcuni sistemi il boot loader avvia direttamente il sistema operativo, la sua presenza risulta quindi *impercettibile.* In Linux invece il boot loader, salvo specifiche configurazioni, mostra un menù di scelta, dove è possibile selezionare sia il sistema operativo da avviare sia il kernel da utilizzare, nei sistemi per cui si dispone di più kernel.
Ricapitolando nell'MBR ci sono le istruzioni per caricare ed eseguire il boot loader, questo una volta avviato carica in memoria ed esegue il kernel del sistema operativo selezionato dall'utente.

I boot loader di Linux maggiormente utilizzati sono due: Grub e LiLo. Ormai però si utilizza quasi esclusivamente il primo.
Grub (*GRand Unified Bootloader*) è più moderno ed evoluto di Lilo (*Linux Loader*). Anche se la schermata di scelta iniziale può essere simile i due boot loader sono molto diversi tra loro. La prima macro differenza consiste nel fatto che Lilo, contrariamente a Grub, non è installato sul dispositivo di boot (hard disk, floppy etc.), ma è contenuto totalmente nell'MBR, quindi le due fasi descritte in precedenza sono in realtà unificate in unica fase, la prima.
In secondo luogo Grub, grazie al fatto che è costituito da programma *esterno*, offre più possibilità, come quelle di avviare il sistema utilizzando volumi cifrati, usare un'immagine di sfondo, password personalizzabili per l'avvio dei vari kernel ecc.

All'avvio Lilo ha a sua disposizione solo i driver del BIOS per accedere ai dischi; perciò, in presenza di BIOS molto vecchi, Lilo potrebbe avviare solo i kernel memorizzati nei primi due dischi e, al loro interno, nelle sole partizioni che non eccedono la soglia del cilindro numero 1023. I BIOS più recenti, invece, hanno (teoricamente) a disposizione tutti i cilindri per ogni disco.
Lilo è stato da sempre considerato il boot loader predefinito per le distribuzioni Linux, tuttavia negli ultimi tempi, grazie all'aumentare della popolarità di Linux, alla maggiore flessibilità e sicurezza di Grub, sta lentamente diventando una seconda scelta, lasciando spazio a quest'ultimo.

In Linux le immagini del kernel sono dei file compressi, normalmente col nome *vmlinuz* o *bzimage* eventualmente seguiti dal numero di release del kernel. Quindi avere più kernel disponibili significa semplicemente avere più file compressi contenenti le relative immagini, posizionati nella partizione di boot, o in generale ovunque siano raggiungibili dal boot loader.
Il kernel di GNU/Linux, cioè Linux, è in grado di accettare parametri sulla riga di comando che lo lancia in esecuzione. Il numero di parametri che è in grado di riconoscere è piuttosto cospicuo, per maggiorni informazioni si può digitare *man 7 bootparam* o mostrare il file */usr/src/linux/Documentation/kernel-parameters.txt*, presente nei sorgenti del kernel (*usr/src/linux*).

L'elenco dei parametri passati all'avvio del kernel è contenuto nel file */proc/cmdline*. I parametri non riconosciuti dal kernel sono considerati variabili d'ambiente se sono del tipo *nome=valore*, altrimenti vengono considerati parametri da passare al primo processo avviato: **Init**. Una volta avviato, dunque, il kernel si decomprime, esamina ed elabora i parametri passati ed avvia */sbin/init.*

INIT

Init è il primo *processo* (task) avviato sul sistema, l'unico ad essere avviato direttamente dal kernel e pertanto il suo PID è 1. Questo processo è il padre di tutti i processi presenti su un sistema GNU/Linux, in quanto in tutti i sistemi Unix-like, ogni processo deve essere *figlio* di qualcun altro. Init rimane attivo per tutto il funzionamento del sistema, ed ha il compito di inizializzare e gestire il sistema ad un livello più alto di quello del kernel, per permetterne l'utilizzo da parte degli utenti.

Per raggiungere questo scopo utilizza le direttive contenute nel file */etc/inittab*, che specificano le operazioni che devono essere compiute da *init.*
La sintassi di */etc/inittab* è la seguente:

- *id*:*runlevel*:*action*:*command*
  - *id* Identificatore univoco della riga
  - *runlevel* Elenco dei runlevel per i quali la riga dev'essere considerata
  - *action* Direttiva che indica ad init l'azione da intraprendere
  - *command* Nome del file eseguibile associato all'azione.

La riga del file contenente la direttiva *sysinit* viene considerata indipendentemente dal runlevel corrente e prima delle altre. Generalmente questa direttiva viene associata ad un file eseguibile che si occupa di eseguire le operazioni preliminari, di configurazione del sistema come l'inizializzazione dell'hardware riconosciuto, l'impostazione dell'orologio di sistema con la data e l'ora fornite dal BIOS, l'impostazione della tastiera, il controllo dell'integrità dei filesystem (in caso esso si accorga che il precedente spegnimento non sia stato effettuato in maniera corretta), il *mounting* delle partizioni indicate nel file */etc/fstab*.
Successivamente vengono avviati i servizi per l'utilizzo del sistema, i daemon.

RUNLEVEL

Linux può funzionare in modalità diverse, ovvero la sua esecuzione dipende dal valore di un parametro importantissimo detto *runlevel*, per mezzo del quale *init* cambia la modalità di funzionamento del sistema.

I valori di runlevel utilizzati di default sono:

- 0 Halt, spegnimento del sistema
- 1 Single user
- 3 Full multi user
- 5 Full multi user + sessione grafica
- 6 Reboot, riavvio del sistema

La direttiva *initdefault in /etc/inittab* indica il runlevel di default di avvio del sistema.
I comandi per visualizzare e cambiare runlevel sono rispettivamente runlevel e init**it**. *Runlevel* restituisce il valore del runlevel corrente, mentre digitando *init* e il numero di runlevel si manda il sistema in esecuzione in quel runlevel.
Sia nell'avvio che durante l'esecuzione il kernel fornisce informazioni (sempre accessibili) sul funzionamento. Queste sono visualizzabili col comando **dmesg**.

Infine per lo spegnimento del sistema:

- **shutdown**

  Spegne il sistema in maniera sicura, tutti gli utenti loggati ricevono un messaggio quando viene eseguito questo comando. È possibile spegnere o riavviare il sistema immediatamente, dopo un intervallo di tempo, oppure ad un orario prefissato mediante l'argomento *time*.

  SINTASSI

  - **shutdown** [-krhfF] [-t *sec*] *time* [*messaggio_avviso*]

  FLAG

  - **-k** Non spegne realmente il sistema ma manda il messaggio di avviso a tutti gli utenti connessi
  - **-r** Riavvia il sistema dopo lo spegnimento, equivalente a **init 6**
  - **-h** Spegne il sistema, equivalente a **init 0**
  - **-f** Salta il controllo dei filesystem al riavvio
  - **-F** Forza il controllo dei filesystem al riavvio
  - **-c** Annulla un comando di shutdown precedentemente avviato. Non accetta l'argomento time
  - **-t** *sec* Spegne il sistema dopo *sec* secondi
  - +X Spegne il sistema dopo *X* minuti.

  ESEMPI

  1. **shutdown** -h now
     Spegne il sistema subito, *now* è un alias per “+0”
  2. **shutdown** +60 &
     Spegne il sistema tra 60 minuti. Conviene mettere il comando in background perché non restituisce l'output alla shell
  3. **Shutdown** 22:30 &
     Spegne il sistema alle ore 22:30.

BIBLIOGRAFIA

- Pagine di manuale online (man) dei rispettivi comandi
- *Linux – Introduzione* di G. Piscitelli e M. Ruta
- *Lucidi Lezione* di G. Piscitelli del corso di Sistemi Operativi
- *Processi vs. Threads in ambiente Linux* di T. Di Noia
- Appunti presi durante le lezioni di Sistemi Operativi dell'AA 06/07 di G. Piscitelli
- Appunti presi durante il laboratorio di GNU/Linux dell'AA 06/07 di M. Ruta
- *La shell bash* di S. Coppi, M. Fago, G. Gallo, sisinflab.poliba.it/piscitelli/
- *Indroduzione a Unix*, www.zia.ms.it/hp/scolombo/unix.htm
- Documentazione di Gentoo GNU/Linux, www.gentoo.org/doc/it
- Linux Magazine, Edizioni Master, www.edmaster.it
- *Appunti di informatica libera* di Daniele Giacobini, a2.pluto.it
- Leoncavallo s.p.a Corso base di GNU/Linux, www.leoncavallo.org/spip/article.php3?id_article=471
- Tuxation, www.tuxation.com/mbr-tricks-with-linux.html
- Introduzione a Unix, www.zia.ms.it/hp/scolombo/unix.htm
- Learning the shell, www.pluto.it/files/ildp/traduzioni/learnCLI/learnCLI.html
- Corso Linux di Michele Sciabarrà, www.corsolinux.it
- Wikipedia, www.wikipedia.org
- Html.it su linux.html.it
- Google su www.google.it/linux

CONTATTI

Critiche, segnalazioni di errori o imperfezioni, proposte di miglioramento e commenti sono sempre ben accetti.
Se ritieni questo documento utile non esitare a farcelo sapere!
Per contattarci basta mandare una mail (no spam) a info@desfa.org.
Tutte le versioni di questo testo sono disponibili nell'area download di Desfa.org.

RINGRAZIAMENTI

La versione 2.0 di questo testo è stata possibile grazie al lavoro di revisione dell'**Ing. Michele Ruta** del Dipartimento di Elettrotecnica ed Elettronica del Politecnico di Bari.
Ringraziamo la paziente A. Lassandro (Fuffy) per l'iniziale impaginazione minuziosa dal testo grezzo.
Un grazie va a Tarabaz, graphical designer di Sabayon-mania (http://sabayon-mania.forumattivo.com) per la realizzazione della copertina della versione 1.0.
Un particolare ringraziamento è per Omar, admin di Sabayon-mania per l'aiuto nella diffusione di questo testo.
Infine siamo grati a chiunque ha contribuito e contribuirà a migliorare questo lavoro mediante segnalazioni o proposte di miglioramento.