

**Linux:**  
**La shell e i comandi**  
**principali**

**[www.lugcr.it](http://www.lugcr.it)**

# Perché ?

**Perché, nell'era delle interfacce grafiche si deve utilizzare qualcosa di così retrò, complicato e poco intuitivo come la linea di comando, la modalità testo ?**

## **1 : potenza**

I comandi Unix hanno quasi 30 anni di storia alle spalle. Centinaia di persone hanno lavorato per migliorarli - non c'è praticamente nulla che non si possa fare.

Provate ad aprire in Word un file di 100 MB per cercare una parola, poi provate:

**grep parola file**

# Perchè ?

## **2 : semplicità**

Ogni comando esegue operazioni semplici - non dovete combattere per trovare dei menù nascosti chissà dove fra diecimila altri, o delle opzioni di configurazione magari inaccessibili se non cambiando chissà quale file .ini o voce del Registry.

È sufficiente che applichiate una sola regola: leggi il manuale. Basta man comando.

# Perchè ?

## **3 : flessibilità**

Fra i programmi GUI - Graphical User Interface - e i programmi da linea di comando, c'è la stessa differenza che c'è fra un trapano multifunzione da hobbista, e l'attrezzatura di frese, troncatrici, pialle elettriche di un professionista. Sì, probabilmente riuscite a rabberciare qualcosa anche con il primo, ma se volete fare sul serio, velocemente e bene, servono le cose serie.

Avete tanti strumenti e li potete combinare come volete, e fare quello che vi serve - non quello che un programmatore ha un giorno pensato che vi sarebbe servito.

# Perchè ?

## 4 : velocità

se non siete completamente inabili alla tastiera, scrivere pochi caratteri è sicuramente più veloce che trascinare un mouse fra infiniti menu e finestre di dialogo. Ma questo è il meno.

Sono soprattutto le operazioni ripetitive che diventano molto più veloci, tanto più quando comincerete ad apprezzare e imparare la possibilità di programmare la linea di comando.

Per di più, rinunciare all'interfaccia grafica significa che basta poca memoria, poca velocità di CPU. Sotto Linux, un 486 con 8MB può fare un ottimo servizio come server Web, email, file server e un sacco di altre cose. Ma se provate a far partire l'interfaccia grafica, diventerà un sistema inutilizzabile...

# Perchè ?

## **5 : accessibilità**

Provate a eseguire un programma su un PC a 10.000 Km di distanza, attraverso un'Internet affollata come sempre.

O semplicemente ad accedere al vostro account su un server, da un PC di qualcun'altro su cui non avete quel bel programmino di accesso da remoto. Il telnet è (quasi) sempre disponibile (e quando non lo è lui ci sono sostituti assai migliori).

# Perchè ?

## **6 : sicurezza**

Bene, cominciamo così: installate un nuovo programma su Windows.

Riavviate. Crash. E adesso?

Scheda grafica preistorica e sconosciuta - niente driver video. E adesso?

Quando nient'altro funziona, la riga di comando è sempre (beh, quasi sempre) disponibile per risolvere i vostri problemi. E potete far partire il sistema con due floppy.

# Perchè ?

## **7 : disponibilità di programmi**

In effetti per Linux, e credo anche per tutti gli altri Unix, usando l'interfaccia testuale è possibile fare tutto, ma veramente tutto: leggere la posta elettronica e i newsgroups, visualizzare siti web, usare l'ftp, ascoltare cd, gli mp3, giocare, tutto ma veramente tutto.

Inoltre tutti questi programmi hanno uno sviluppo notevole anche adesso, nell'era delle interfacce grafiche e quindi sono spesso allo stato dell'arte.

# Filosofia

## Comandi semplici

molti comandi, molto specializzati

```
[sb@pcsash sb]$ wc /etc/passwd
```

## Pipeline

l'output di un comando può essere utilizzato come input di un altro.

Questo sistema si chiama pipeline: un "tubo" attraverso cui passa un flusso di dati, che viene modificato dai diversi comandi che si trovano lungo il tubo.

```
cat /etc/passwd|grep o|sort
```

# Filosofia

## opzioni omogenee

le opzioni, specificate con `-x` o `--xxxx`, cercano di essere il più possibile omogenee, cioè di avere lo stesso significato, fra i diversi comandi:

```
cat --help
```

```
grep --help
```

# Filosofia

## **file omogenei**

tutti i file sono uguali (molto diverso rispetto a come erano i sistemi precedenti).

Unix, poi, fa un passo oltre: non solo i file sono uguali, ma anche tutte le periferiche sono file, e quindi uguali. Si può scrivere allo stesso modo sullo schermo, su un file, sui settori fisici di un floppy, sul modem o la stampante - potete persino provare a scrivere su uno scanner o sulla scheda audio - ma il fatto che una cosa si possa fare, non significa necessariamente che abbia un senso farla...

*e neanche che non sia rischioso!*

# Login

**Red Hat Linux release 5.1 (Manhattan)**

**Kernel 2.0.35 on an i586**

**login: sb**

**Password: Cappuccino**

**Last login: Sat Nov 21 10:12:48 on tty2**

**Il System Manager ti saluta**

**You have mail.**

**[sb@pcsash sb]\$**

**Questo significa e che vi identificate e autenticate con il sistema:**

chi va là? sono quello che tu chiami sb

parola d'ordine? Cappuccino

# Login

**Il sistema (in realtà un ben preciso programma che si chiama login) controlla che esista un utente con il nome dato (username, non nome effettivo), e che la password corrisponda a quella registrata nel file /etc/passwd**

**A questo punto, avete una "identità": lo username, a cui corrisponde uno userid (o uid) numerico; e appartenente ad alcuni gruppi, a cui corrispondono dei groupid (o gid) numerici.**

# Login

## Esempio :

```
[leandro: ~]$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
leandro:x:1000:100:Leandro su nerone,,,:/opt/leandro:/bin/bash
ospite:x:1002:100:Ospite di Nerone,,,:/home/ospite:/bin/bash
rpc:x:32:32:RPC portmap user::/bin/false
```

```
[leandro: ~]$
```

# Login

```
[sb@pcsash sb]$ whoami
```

```
sb
```

```
[sb@pcsash sb]$ id
```

```
uid=601(sb) gid=601(sb) groups=601(sb),100(users),10(wheel),503(tape)
```

**L'idea di identità sarà forse familiare a chi di voi usa una rete, e certo a chi usa Internet: molto probabilmente il vostro indirizzo di email è un account (con diritti limitati) su un server Unix.**

## Prompt e shell

```
[sb@pcsash sb]$ _
```

Questo è quello che viene chiamato "command line shell prompt", o semplicemente "prompt" o varie altre abbreviazioni. Cioè "sono pronto a eseguire i comandi".

Dopo l'autenticazione, il sistema passa ad eseguire, con la vostra "identità", un programma specificato nel file `/etc/passwd` (per l'esattezza quello indicato dell'ultimo campo, vedi l'esempio). Se il vostro è un "normale" account Unix, questo programma sarà del genere chiamato "shell di comando". In altri casi, potrebbe essere una "shell ristretta", un programma di BBS, o niente del tutto (come nel caso degli account di posta presso un provider, il `/bin/false` dell'esempio).

## Prompt e shell

**Un programma shell è una interfaccia fra voi e gli altri programmi presenti nel sistema. Un esempio di shell che probabilmente tutti conoscete è il command.com del DOS.**

**Sotto Unix, la scelta della shell è una questione strettamente personale, vi sono più shell e potete decidere quale preferite;**

**bash , che per Linux è la shell di default, e probabilmente la più potente e flessibile.**

# La Shell Bash

## Dalla manpage di Bash:

Bash is an sh-compatible command language interpreter that executes commands read from the standard input or from a file. Bash also incorporates useful features from the Korn and C shells (ksh and csh).

Bash is intended to be a conformant implementation of the IEEE POSIX Shell and Tools specification (IEEE Working Group 1003.2).

# Altre Shell

## Altre shell :

Bourne Shell e derivate :

**sh Bourne shell, la prima shell di Unix. In Linux è sostituita da bash. Sui sistemi standard POSIX, è sostituita da ksh.**

**Ash, una shell minima, con molti comandi integrati, occupa poca RAM e poco spazio disco, anche perché ha poche funzioni.**

**ksh - Korn Shell, introduce la history (accesso ai comandi precedenti) e l'editing della linea.**

**Zsh, offre praticamente le stesse funzioni di ksh, e alcune altre cose esoteriche, che però ovviamente trovate anche in bash.**

C Shell.

**csh - C shell, utilizza una sintassi derivata dal linguaggio C. In Linux è sostituita da tcsh.**

**Tcsh, evoluzione della precedente, alla quale aggiunge history e editing di linea.**

# Caratteristiche di Bash

**Vi permette di non specificare l'identificativo completo del programma; la shell, se gli viene richiesto un nome di programma senza specificare la directory, lo cerca nel PATH (che è una variabile d'ambiente).**

**Alias, abbreviazioni definibili per comandi molto usati**

## **Wildcards**

per operare su gruppi di file con nomi simili: [sb@pcsash sb]\$ ls prova\*1998.txt

Come nel caso del DOS, il carattere '\*' viene sostituito da uno o più caratteri mentre il carattere '?' viene sempre sostituito da un solo carattere.

# Caratteristiche di Bash

## **variable expansion (espansione delle variabili)**

```
[sb@pcsash sb]$ Saluto=Ciao
```

```
[sb@pcsash sb]$ echo "$Saluto $USER, sei su $HOSTNAME"
```

```
Ciao sb, sei su pcsash.sash.lan
```

USER e HOSTNAME sono due variabili che vengono normalmente impostate in ogni sistema e quindi sono dette "d'ambiente" (vedi il paragrafo).

È consuetudine, per altro molto seguita, in ambito Unix, di indicare le variabili con lettere maiuscole.

# Caratteristiche di Bash

## **command (backtick) expansion**

lo standard output di un comando può diventare una stringa che fa parte di una linea di comando:

```
[sb@pcsash sb]$ hn=`cat /etc/HOSTNAME`
```

```
[sb@pcsash sb]$ echo $hn
```

```
pcsash.sash.lan
```

## **history expansion**

Permette di richiamare comandi precedenti, si realizza tramite i tasti cursore.

# Caratteristiche di Bash

## **variabili di ambiente**

la shell si ricorda e vi permette di modificare certe impostazioni usate dai programmi. Queste impostazioni sono memorizzate in alcune variabili come \$USER, \$HOME, \$TERM, \$DISPLAY, \$PATH e altre (che per l'appunto cominciano tutte per \$) che possono essere valide per tutti gli utenti (quando impostate in /etc/profile) e/o per un singolo utente, (quando impostate modificando il file ~/.bash\_profile ovviamente quando è in uso Bash).

## **controllo di esecuzione**

processi in foreground e background, identificativo di un processo, blocco di un processo, riavvio di un processo, terminazione di un processo: & , %n, ^C, ^Z, fg, bg, kill

## **Scripting**

i file .bat del DOS, molto più potenti.

# Caratteristiche

## Altre cose...

~ (ALT 126) rappresenta la directory home dell'utente o ~/username: per la home di un'altro utente.

Il prompt della shell può essere diverso, e può contenere diverse informazioni. Nel caso della bash, ma anche per la maggior parte delle altre shell, è configurabile. Dal minimalista '!' a cose tipo '[user@host: directory]\$ '

Solitamente viene impostato per tutti in /etc/profile, tramite il valore della variabile PS1. Ovviamente ogni utente ne può modificare le impostazioni nel proprio ~/.bash\_profile.

Quello che rimane uno standard è che un utente normale ha un prompt che finisce in \$, e il superuser ha un prompt che finisce in #

# Superuser

**Una questione psicoanalitica ovvero: se volete sempre usare il superuser, avete un problema...**

**Abbiamo visto che al login ci viene assegnata una "identità". Abituati ai PC Win95, dove ai diversi login corrispondono, tutt'al più, diverse configurazioni, ci troviamo invece di fronte a un grosso cambiamento: diversi account hanno diversi diritti - per quanto riguarda l'accesso ai file, alle periferiche.**

**Esistono, su tutti i sistemi Unix, delle identità che non appartengono a persone reali, ma a certi "ruoli". Ad esempio i programmi che gestiscono la posta sono eseguiti con una identità speciale, "mail", che gli dà il diritto di scrivere nelle caselle di posta degli utenti.**

# Superuser

**Esiste un account che ha tutti i diritti su tutto, senza alcuna limitazione, ed è l'account del system manager o "superuser": root.**

**Root ha diritto di accesso, in lettura e scrittura, a qualsiasi cosa sul sistema: dalle configurazioni, alle periferiche, la posta di tutti gli utenti, persino ai programmi che sono in esecuzione. Il system manager ha poteri illimitati.**

**Come utenti Linux, probabilmente sarete voi stessi i system manager del vostro PC. Non dimenticatevi mai che il superuser ha possibilità illimitate - e quindi capacità illimitata di causare danni. La prima regola, quindi, è: non fidatevi di voi stessi. Quando avete l'identità di root, pensate sempre due volte al comando che state scrivendo, o preparatevi a sopportarne le conseguenze.**

# Superuser

Da **NON** fare mai (specialmente adesso) !!!!

```
# rm -rf /*
```

è sufficiente a cancellare qualsiasi file sulle vostre partizioni Linux e non... e altri possono facilmente fare polpette di tutto l'harddisk.

# Superuser

**Se avete appena installato Linux, come prima contromisura, createvi un utente "normale" col vostro nome:**

```
# adduser mario
```

**e assegnategli una password:**

```
# passwd mario
```

**e usate sempre questo user. Usate root solo quando non ne potete fare a meno: per configurare il sistema, per installare del nuovo software...**

# L'ABC di UNIX

**Mediante il comando:**

```
$ ls -la
```

**otteniamo tutti i dettagli su tutti i file presenti nella cartella in cui ci troviamo, se eseguiamo questo comando nella nostra home troviamo alcuni file “nascosti” (in Unix ciò significa semplicemente che il nome del file inizia con un '.') che contengono configurazioni riguardanti la nostra utenza e che vengono automaticamente copiati da /etc/skel nella home directory dell'utente quando esso viene creato.**

**Questi sono file di configurazione personali, cioè che possiamo cambiare senza essere superuser.**

**A noi, poi, interesseranno:**

```
.bashrc .bash_profile .bash_logout
```

**che sono i file di configurazione di bash.**

# L'ABC di UNIX

## Proviamo:

```
[sb@pcsash sb]$ cat /boot/System.map
```

```
... etc etc ...
```

```
001b4c20 A _end
```

**cat - conCATenate scrive in uscita, a video in questo caso, il contenuto dei file indicati.**

# L'ABC di UNIX

**Chissà cos'era - era troppo lungo, ed è scorso via troppo velocemente; proviamo a guardarlo con calma:**

```
[sb@pcsash sb]$ less /boot/System.map
```

**less - più di more che è un altro comando per "scorrere lentamente" il contenuto di un file.**

**Possiamo scorrerlo come vogliamo, tornare indietro, persino cercare parole:**

```
/end
```

**vi sono molte altre opzioni, visibili tramite:**

```
h
```

**per terminare:**

```
q
```

# L'ABC di UNIX

**A volte quello che si interessa è solo l'inizio di un file:**

```
[sb@pcsash sb]$ head /System.map
```

**o solo la fine:**

```
[sb@pcsash sb]$ tail /System.map
```

**o seguire cosa viene via via aggiunto ad un file:**

```
[sb@pcsash sb]$ tail -f /var/log/messages
```

# L'ABC di UNIX

**Dopo aver guardato dei file già fatti, proviamo a farne uno noi:**

```
[sb@pcsash sb]$ touch fileprova
[sb@pcsash sb]$ ls -l fileprova
-rw-rw-r-- 1 sb sb 0 Nov 21 17:49 fileprova
[sb@pcsash sb]$ cat fileprova
[sb@pcsash sb]$
```

**è vuoto: proviamo a scriverci qualcosa...**

```
[sb@pcsash sb]$ echo "che bella scritta" fileprova
che bella scritta fileprova
```

**questo comando ha scritto sul terminale. Il secondo argomento non è il nome di un file su cui scrivere, ma viene semplicemente attaccato al primo...**

# L'ABC di UNIX

**Allora, approfittiamo della capacità della shell di redirezionare l'output di un programma:**

```
[sb@pcsash sb]$ echo "che bella scritta" >fileprova
```

```
[sb@pcsash sb]$ cat fileprova
```

```
che bella scritta
```

**aggiungiamo qualcosa:**

```
[sb@pcsash sb]$ echo "costa solo $10" >>fileprova
```

```
[sb@pcsash sb]$ cat fileprova
```

```
che bella scritta
```

```
costa solo 0
```

## L'ABC di UNIX

**Strano? No. Siamo inciampati in una variabile di shell e bash ne ha sostituito il valore nell'espressione (vedi il paragrafo). Ma possiamo impedirglielo, mettendo la stringa fra virgolette singole, che impediscono la "shell expansion", o precedendo il carattere incriminato con un '\' (backslash):**

```
[sb@pcsash sb]$ echo 'costa solo $10 ' >>fileprova
```

```
[sb@pcsash sb]$ cat fileprova
```

```
che bella scritta
```

```
costa solo 0
```

```
costa solo $10
```

# L'ABC di UNIX

**proviamo di nuovo:**

```
[sb@pcsash sb]$ echo "ma che bella scritta !" > a  
bash: !": event not found
```

**Whoops! Cos'è successo? Stavolta il carattere '!' ha chiesto a bash di cercare nella history un comando che cominciasse con i caratteri subito successivi, ma possiamo rimediare, come nel caso di '\$', o con le virgolette singole, o con un backslash davanti al '!'**

# L'ABC di UNIX

**Comunque, a parte questi piccoli inconvenienti, abbiamo visto quello che volevamo: come usare la redirectione dell'output per scrivere in un file. Ci sono altre cose in proposito:**

con `cmd < file` si redireziona l'input

**Esempio :**

```
echo "ls -la" > command
```

```
bash < command
```

**col comando '`2>`' file si redireziona l'error output (stderr, per chi sa il C), che altrimenti viene scritto su video anche se l'output (stdout) è redirezionato.**

**Molto utile quando quello che interessa sono gli errori...**

# L'ABC di UNIX

**Altri esempi di redirectione:**

**Questo comando comporta che l'output di un programma venga scritto su file con eventuale sovrascrittura:**

```
ls -l > ls-l.txt
```

**otterremo un file di nome 'ls-l.txt' che conterrà la lista dei file presenti nella directory di lavoro. Volendo semplicemente “appendere” il risultato al file, che se non esiste verrà creato, basta utilizzare**

```
ls-l >> ls-l.txt
```

**Potete redirezionare lo stdout (1) di un programma sullo stderr (2) o il contrario:**

```
1>&2    2>&1
```

# L'ABC di UNIX

**Un ultimo esempio di redirectione:**

```
rm -f $(find / -name nomefile) &> /dev/null
```

**in questo modo si rimuove 'rm' senza richiesta di conferma '-f' ogni file che corrisponde al risultato '\$' della ricerca 'find / -name nomefile' cioè:**

**trova a partire dalla radice del sistema ogni file che si chiama 'nomefile'**

**e si fa in modo che stdout e stderr vengano lanciati nel... baratro!**

# L'ABC di UNIX

**Per cercare un file o in un file? Vi sono molti modi...**

**Cercare una stringa di caratteri all'interno di uno o più file: grep e simili. Per sfruttare fino in fondo il più potente egrep, è necessario conoscere le regexp - REGular EXPressions.**

**Su Linux: locate. Utilizza un database (solitamente generato e aggiornato periodicamente da updatedb). Veloce ma poco flessibile, non disponibile sugli altri UNIX e si riferisce sempre al database !!!**

## L'ABC di UNIX

**Find** : non utilizza un database, ricerca direttamente nel filesystem. Notevole impegno del sistema dei dischi, tempi più lunghi. Molto flessibile: è possibile eseguire un comando su ciascuno dei file che hanno passato il vaglio del find, tramite l'opzione **-exec** (pericolosa).

Altro comando utile per la ricerca, soprattutto per root, è il **which**, che ci indica a quale file eseguibile corrisponde un certo comando:

```
[sb@pcsash sb]$ which less  
/usr/bin/less
```

utile per essere sicuri che venga, ad esempio, utilizzata la versione giusta di un comando (nel caso ne esista più di una installata, ad esempio in `/usr/bin` e in `/usr/local/bin`)

# L'ABC di UNIX

**Which è utile anche per rintracciare la "provenienza" di un comando: ad esempio con la RedHat,**

```
[sb@pcsash sb]$ rpm -qf `which less`  
less-332-2
```

**è un modo molto veloce di risalire al pacchetto di origine, e quindi alla versione, di qualsiasi comando.**

# L'ABC di UNIX

## **Editare...**

### **vi**

Abbastanza difficile da usare ma estremamente comodo da usare quando dovrete lavorare su terminali remoti oppure in situazioni di emergenza vista la sua ridottissima dimensione.

### **emacs**

Non è propriamente un editor quanto una religione (vedete il newsgroup `alt.religion.emacs`), visto che è l'editor sviluppato inizialmente da Stallman, colui che ha inventato la licenza GPL, alla base di tutto l'impianto "filosofico" del software libero e che è stato sviluppato un linguaggio di programmazione (lisp) con il quale sono stati sviluppate tantissime applicazioni, sia interne ad emacs che esterne.

# L'ABC di UNIX

## **Editare...**

### **jed**

Versione ridotta di emacs.

### **pico**

Sviluppato inizialmente per il lettore di posta elettronica pine, è molto usato anche da solo.

### **joe**

Molto semplice da usare (utilizza gli stessi comandi di wordstar, per chi se li ricordasse) e molto leggero.

# L'ABC di UNIX

## **Automazione:**

**Linux, come d'altronde tutti gli Unix, è stato creato da programmatori pensando soprattutto a sé stessi...**

**...Il che comporta che per questo tipo di piattaforma esiste una pletera di linguaggi di programmazione di tutti i generi, gusti e apparenza.**

**Si capisce come mai praticamente è possibile automatizzare qualunque compito grazie alla programmabilità di programmi "normali" come le shell oppure grazie all'uso di linguaggi appositi.**

**shell script**

**perl**

# L'ABC di UNIX

## **Documentazione:**

**Tutti i comandi ed in genere i programmi di Linux dispongono sempre un notevole corredo di documentazione.**

## **Pagine man:**

Digitando man comando si ottiene un testo (spesso lungo) che spiega dettagliatamente tutte le opzioni del comando. Tutti i comandi e i programmi hanno una loro pagina man, quasi sempre anche tradotta in italiano.

## **Pagine info:**

Digitando info comando si ottiene invece la pagina info del comando. Questo formato era la vecchia versione della documentazione ed ha caratteristiche molto comode come la possibilità di navigare usando link anche ad altre pagine. Questo formato non viene, ormai, molto usato specialmente dai nuovi programmi e spesso le pagine info non si trovano tradotte.

# L'ABC di UNIX

## **Documentazione in formato testo:**

Tutti i programmi e i comandi vengono distribuiti con la documentazione scritta direttamente dagli autori e la si trova nella directory `/usr/doc/nomeprogramma` (oppure nelle nuove distribuzioni `/usr/share/doc/nomeprogramma`). Normalmente questa documentazione è in normale formato testo ma spesso si trovano anche pagine in html e quasi sempre solo in inglese.

## **Spiegazioni dalla riga di comando:**

Digitando comando `--help`, oppure `-h`, oppure `-?`, si ottiene sempre (o quasi) una lista delle opzioni possibili per il tale comando, a volte con una minima spiegazione del significato delle varie opzioni. Molto comodo per ricordarsi al volo quale opzioni abbiamo a disposizione.