

**1° Corso Introduttivo a GNU/Linux**  
**Seconda lezione**

26 Maggio 2001

LUG Roma3

# Shell BASH, Principi, Esempi, Esercitazione

26 Maggio 2001

LUG Roma3 Luigi Gangitano - Gianluca Granero

Riferimenti bibliografici:

*Appunti Linux*

<http://www.pluto.linux.it/ildp/AppuntiLinux>

*MAN di BASH*

## Scopo della “lezione” di oggi

L’interazione con GNU/Linux attraverso un terminale (o console)

- Principi di funzionamento
- Come interagire con la shell “bash”
- Rudimenti di scripting

## **Cos'è un terminale**

Il terminale, in qualunque forma esso sia (console, terminale remoto, applicazione a finestra all'interno di X) è il mezzo normale di comunicazione tra l'utente e il sistema.

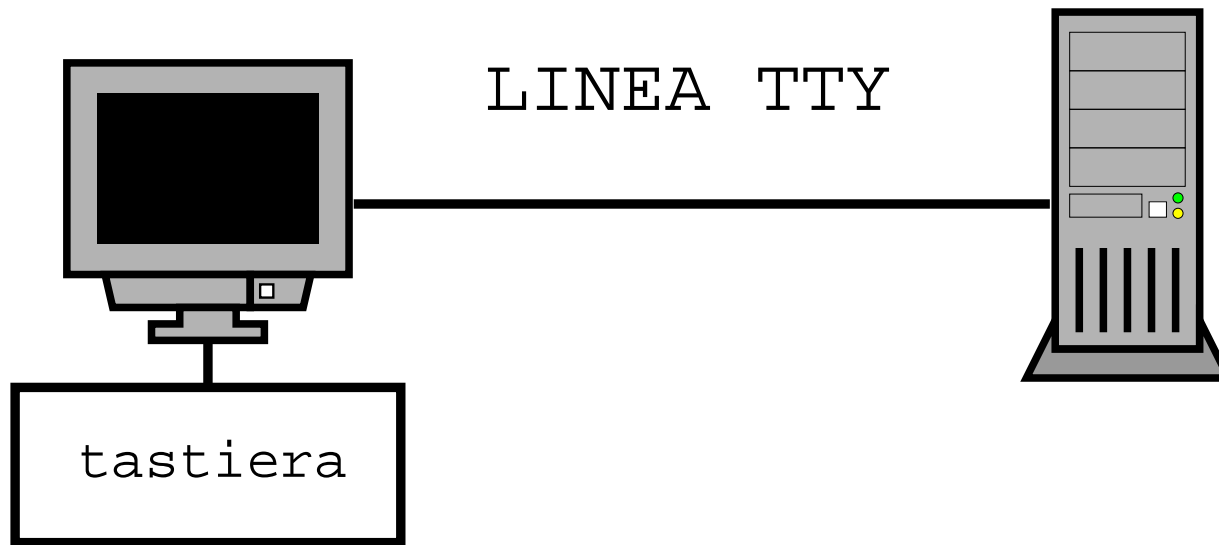


Figure 1: Schema terminale-sistema

Senza il terminale non ci sarebbe alcuna possibilità di avviare nuovi processi e di conseguenza, nemmeno di poter compiere alcuna attività.

## **La shell - DEFINIZIONE**

Una shell è qualsiasi programma in grado di consentire all'utente di interagire con il sistema. Può trattarsi di qualcosa di molto semplice come una riga attraverso cui è possibile digitare dei comandi, oppure un menù di comandi già pronti, o un sistema grafico a icone, o qualunque altra cosa possa svolgere questo compito.

La shell ha questo nome (conchiglia) perché di fatto è la superficie con cui l'utente entra in contatto quando vuole interagire con il sistema: la shell che "racchiude" il kernel.

## Shell a caratteri

Nei sistemi Unix si usano ancora shell a riga di comando, ma queste, anche se povere esteticamente, sono comunque molto potenti e difficilmente sostituibili.

Il prompt del DOS è un esempio di shell a riga di comando (scomoda e poco espressiva)

## Shell Unix

La tipica shell di un sistema Unix è l'*interprete* di un linguaggio di programmazione orientato all'avvio e al controllo di altri programmi.

Questo interprete è in grado di:

- eseguire i comandi impartiti da un utente attraverso una riga di comando in modo interattivo
- eseguire un file script, scritto nel linguaggio della shell.

## login - invocazione della shell

Il meccanismo che porta alla creazione di una shell segue il percorso:

**init - getty - login - shell**

```
giangy@homer.giangy.net -
File Sessions Options Help
[giangy@homer giangy]# pstree |less
init--atd
├─bdflush
├─crond
├─cupsd
├─httpd---11*[httpd]
├─identd---identd---3*[identd]
├─imwheel
├─kpm-idled
├─10*[kdeinit]
├─kdeinit---cat
├─kdeinit--kopuload
├─2*[kdeinit]
├─2*[kdeinit--bash]
├─kdeinit--bash--less
├─pstree
├─5*[mingetty]
└─login---bash
```

## Una shell per utente

il file `/etc/passwd` contiene (tra le altre cose) l'indicazione di quale sia la shell dell'utente:

..

```
giangy:x:501:501:giangy:/home/giangy:/bin/bash
```

..

## Tipi di shell

**Per l'utente:** bash, tcsh, ksh, zsh ...

**per scripting:** sh (più leggera)

**emergenza:** sash (Stand Alone SHell)

**noshell:** per non dare accesso al sistema

**a fantasia** ...

## Shell BASH

Bash è la shell standard di GNU/Linux e di molti altri sistemi Unix.

Bash (Bourne Again SHell) è compatibile con la shell Bourne (sh).

Incorpora alcune funzionalità della shell Korn (ksh) e della shell C (csh).

Bash è progettata per essere aderente alle specifiche POSIX.2.

## Prompt della shell

Quando una shell attende ed esegue i comandi in maniera interattiva. La disponibilità a ricevere comandi viene evidenziata da un messaggio di invito o *prompt*.

Prompt composto da simboli e informazioni utili all'utente per tenere d'occhio il contesto in cui sta operando.

## Esempio prompt

```
[giangy@homer giangy]$ pwd
/home/giangy
[giangy@homer giangy]$ cd /tmp
[giangy@homer /tmp]$ echo $PS1
[\u@\h \W]\$
[giangy@homer /tmp]$ whoami
giangy
[giangy@homer /tmp]$ hostname
homer.giangy.net
[giangy@homer /tmp]$ pwd
/tmp
```

## Storico dei comandi

Bash mantiene uno storico dei comandi utilizzati dall'utente.

Lo storico **NON** si interrompe alla fine di una sessione o con un reboot (è su file).

Freccette (sù, giù) per sfogliarlo

CTRL-R+[qualche carattere] per cercare all'interno dello storico

CTRL-R cerca ancora

## Altri tasti utili

Andare all'inizio della riga - CTRL-A

Andare alla fine della riga - CTRL-E

CANC - CTRL-D

(spesso CANC è mappato male)

## Alias

Bash permette la definizione di nuovi comandi in forma di alias di altri comandi (o serie di comandi).

alias - elenca gli alias esistenti

alias nome="comando con tutti i parametri"

alias dir="ls -color "

## Alias

Bash permette la definizione di nuovi comandi in forma di alias di altri comandi (o serie di comandi).

alias - elenca gli alias esistenti

alias nome="comando con tutti i parametri"

alias dir="ls -color "

## Serie di comandi

È possibile dare più comandi da eseguire in serie separando gli elementi della lista con dei ;

Esempio

```
$ mv pippo.ps pluto.ps;mv pluto.ps pippo.ps
```

Equivale a:

```
$ mv pippo.ps pluto.ps
```

```
$ mv pluto.ps pippo.ps
```

## Ambiente

Ogni programma in funzione nel sistema ha un proprio **ambiente** definito in base a delle **variabili di ambiente**.

## Variabili d'Ambiente

Le **variabili d'ambiente** sono un mezzo elementare e pratico di configurazione del sistema: i programmi, a seconda dei loro compiti e del loro contesto, leggono alcune variabili, e in base al contenuto si comportano.

Esempio:

la variabile PATH

## PATH

La variabile PATH indica tutte le directory dove la shell deve andare a guardare per trovare dei file eseguibili (i programmi per capirci)

È una variabile a tutti gli effetti:

```
$ echo $PATH
/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/usr
/X11R6/bin:/usr/games:/home/giangy/bin:/usr/sbin
:/usr/X11R6/bin:/usr/games
```

Negli anni '70 l'hanno introdotta con Unix

Negli anni '80 l'ha usata il dos (essendo un fratello povero di Unix)

Windows la mantiene anche nelle sue ultime versioni un po' camuffata ma c'e' sempre (anche in 2000)

## La directory corrente

La directory corrente normalmente NON viene inclusa nella variabile PATH  
Non è molto sicuro, e non dovrebbe mai essere “prima” di quelle canoniche.  
Si può aggiungere in fondo a PATH, o ovviare così:

```
./nome_file_eseguibile
```

## Ambiente 2

L'ambiente consegnato a ogni programma che viene messo in esecuzione, è controllato dalla shell che può assegnare ambienti diversi a programmi diversi.

La shell può creare, modificare e leggere queste variabili; ciò è utile per la realizzare file script.

## Gestione Variabili in BASH

### Variabili di shell

Una variabile è definita quando contiene un valore, anche se vuoto.

L'assegnazione di un valore si ottiene con una dichiarazione del tipo:

```
nome_di_variabile=[valore]
```

### Esempio:

```
[giangy@homer giangy]$ pluto=cane_di_topolino  
[giangy@homer giangy]$ echo $pluto  
cane_di_topolino
```

## Esportazione Variabili

Le variabili di shell hanno validità limitata all'ambito della shell stessa.

I comandi interni alla shell stessa sono al corrente di queste variazioni mentre i programmi che vengono avviati non ne risentono.

Per i programmi esterni le variabili devono essere **esportate**. L'esportazione delle variabili si ottiene con il comando interno `export`.

Esempi

```
$ PIPPO="ciao"  
$ export PIPPO
```

Crea la variabile PIPPO e quindi la esporta.

```
$ export PIPPO="ciao"
```

In un colpo solo.

## Variabili Ricapitolando

Creazione variabili e esportazione

```
$ PIPPO="ciao"  
$ export PIPPO
```

Per mostrare il valore di una variabile

```
$ echo $PIPP0
```

Per vedere tutto l'ambiente corrente

```
$ set
```

## Espansione/Sostituzione

Come già si intuiva da alcuni comandi che abbiamo visto la shell, prima di lanciare i programmi e/o comandi fa delle modifiche alla stringa che noi gli passiamo, meccanismo noto come **espansione**

## Passi dell'espansione

1. parentesi graffe;
2. tilde ;
3. parametri e variabili;
4. comandi;
5. aritmetica (da sinistra a destra);
6. suddivisione delle parole;
7. percorso o pathname.

## 2 - Espansione della tilde

Se una parola inizia con il simbolo tilde (~) interpreta quello che segue, fino alla prima barra obliqua (/), come un nome-utente.

E sostituisce questa prima parte con il nome della directory personale dell'utente stesso.

~ da sola sottointende l'utente attuale

Esempi

```
$ cd ~
```

```
$ cd ~tizio/public_html
```

## 3 - di parametri e variabili

Già visto

sostituzione del parametro o della variabile con il suo contenuto.

Esempio

```
[giangy@homer giangy]$ pippo=PIPPO  
[giangy@homer giangy]$ echo $pippo  
PIPPO
```

## 4 - Sostituzione comandi

La sostituzione dei comandi consente di utilizzare quanto emesso attraverso lo standard output da un comando. Ci sono due forme possibili:

`$( comando )`

`` comando ``

Apici “al contrario” `altgr+apice`

Per ora Standard Output = testo riportato come risultato dal comando

## 7 - Espansione di percorso

I simboli \*, ? e [ vengono interpretati ed espansi in serie di stringhe (che indicano percorsi)

Vediamo solo il più importante: \*

Corrisponde ad una qualsiasi stringa

quindi se siamo in una directory con questi file:

```
paperino paperone paperoga minnie topolino
```

```
rm -f pape* -> rm -f paperino paperone paperoga
```

## Caratteri di protezione

A volte magari abbiamo bisogno di scrivere effettivamente caratteri come:

\$ \* { [ . . . }

Esistono 3 modalità:

1. Escape
2. Apici singoli
3. Apici doppi

## Escape

La barra obliqua inversa (\) rappresenta il carattere di escape. Serve per preservare il significato letterale del carattere successivo, cioè evitare che venga interpretato diversamente da quello che è veramente. Esempio

```
ls -l \$fff
```

cerca di leggere il file che si chiama proprio \$fff

Usato da solo (quindi con uno spazio dopo) serve per continuare una linea (nonostante si vada a capo) Esempio

```
$ ls -l \  
supercalifragilischepiralidoso
```

viene interpretato come un unico comando su una linea

## Apici Singoli

Racchiudendo dei caratteri tra apici semplici (') si mantiene il valore letterale di questi caratteri.

Un apice singolo non può essere contenuto in una stringa del genere.

Esempio

```
$ echo '$0 $1 \ ... restano "inalterati".'
```

```
$0 $1 \ ... restano "inalterati".
```

## Apici Doppi

Racchiudendo una serie di caratteri tra una coppia di apici doppi si mantiene il valore letterale di questi caratteri, a eccezione di \$, ' e \. Esempio

```
$ echo "Il parametro \$0 contiene: \"\$0\""  
Il parametro $0 contiene: "-bash"
```

## Flussi

Un accenno rapidissimo a come funzionano i flussi di caratteri.  
Esistono 3 flussi di caratteri che ogni programma lanciato da una shell (quindi tutti) deve gestire:

1. Standard Input
2. Standard Output
3. Standard Error

## Standard Output

Il più facile da capire: **il risultato del programma**

Per l'esattezza i messaggi che il programma invia.

Esempio

```
$ cal
      May 2001
Su Mo Tu We Th Fr Sa
      1  2  3  4  5
...
```

## **Standard Error**

Come Standard Output solo per gli errori che il programma vuole segnalare

## **Standard Input**

Il programma riceve come input un flusso di caratteri  
Un po' come fosse un file

## Redirezione

Possiamo reindirizzare standard input e output verso dei file con:

< e >

Esempio

```
$ ls -lR > lista_file
```

## grep

Comando utilissimo.

seleziona le righe di un file che contengono la stringa cercata

Esempio

file personaggi che contiene

paperino

topolino

paperone

```
$ grep pape personaggi
```

```
paperino
```

```
paperone
```

## Pipe

Quelle che vediamo sono solo le pipe anonime.

Si usano con il carattere |.

Servono per “concatenare” i comandi.

L’output di uno viene usato come input dell’altro

Esempio

```
$ ls -lR | grep pippo
```

fa un ls ricorsivo e riporta solo le righe che contengono la stringa pippo

## sed

Comando utile ma molto complesso, ne vediamo solo un uso: sostituzione di stringhe

Riga per riga di un file (o dello standard output) sostituisce una stringa con un'altra.

### Esempio

abbiamo il file prova.html e vogliamo cambiare alcuni link senza editare il file:

```
$ cat prova.html | \  
    sed 's/<a href="immagini/<a href="images/' > \  
    prova-modificato.html
```

## Un editor universale: vi

Per poter creare degli script è necessario avere un editor di testo. Linux dispone di molti editor, ciascuno con le sue peculiarità, ma uno solo lo troverete in tutti i sistemi Unix, e molto spesso è l'unico che avrete a disposizione.

Benvenuti nel magico mondo di *vi*.

## Esecuzione

Il comando base per accedere a *vi* è, ovviamente, **vi**, ma nelle distribuzioni più recenti è presente una versione *enhanced*, con molte funzionalità in più, che prende il nome di **vim** (*Vi Improved*).

## Descrizione dell'ambiente

Eseguendo il comando **vi** vi troverete davanti all'ambiente di editing, che appare inizialmente piuttosto scarno. L'area di lavoro è completamente dedicata al testo da elaborare, ad esclusione dell'ultima riga in basso che mantiene, di volta in volta, indicazioni sullo stato in cui si trova l'editor.

## Ambienti

**vi** infatti è stato progettato in modo da distinguere anche concettualmente l'immissione del testo dall'interpretazione di comandi che operano sul testo inserito.

In qualsiasi momento, per poter inserire un comando è sufficiente tornare alla *modalità comandi* premendo il tasto **Esc**.

Inoltre il tasto **Esc** consente di annullare un comando non completato.

Premerlo più volte di seguito non ha effetti collaterali.

## Apertura di un file

L'esecuzione del comando **vi** senza argomenti, crea un nuovo buffer vuoto. Se invece vogliamo editare un file esistente possiamo invocare **vi** seguito dal nome del file che voglia editare.

```
vi nomefile
```

In alternativa, una volta avviato **vi**, è possibile aprire un file con il comando

```
:e
```

```
:e nomefile
```

## Inserimento del testo

Premendo il tasto **i** oppure **Ins** si passa alla modalità di inserimento del testo, in cui è possibile digitare del testo normalmente.

Solitamente questa modalità è identificata dalla presenza della stringa – *INSERT* – nella barra di stato.

```
Testo inserito dopo aver premuti il tasto [i]
```

```
~
```

```
~
```

```
~
```

```
~
```

```
-- INSERT --
```

È da notare che il carattere ~ (tilde) identifica le linee vuote.

## Modifica del testo

La pressione del tasto **r** attiva la modalità modifica del testo in cui il testo inserito sovrascrive il testo esistente.

L'attivazione di tale modalità è identificata dalla presenza della stringa – *REPLACE* – nella barra di stato.

Lo stesso risultato si ottiene premendo due volte il tasto **Ins**.

## Cancellazione del testo

Per cancellare del testo è possibile utilizzare il comando **x** che cancella il carattere identificato dal cursore, oppure il comando **dd** preceduto da un numero *n* il quale, appunto, cancella *n* righe a partire da quella occupata dal cursore.

```
Testo inserito nelle slide precedenti  
continua sulla seconda riga  
e sulla terza  
~
```

L'esecuzione del comando **2 dd** quando il cursore si trova sulla prima riga cancellerà le prime due righe e lascerà la terza.

## Copia ed incolla

Il comando **yy**, eventualmente preceduto da un numero, copia in un buffer *n* righe a partire da quella in cui si trova il cursore.

La funzione “Taglia” in realtà l’abbiamo già vista, dato che viene svolta dal comando **dd** che non si limita a cancellare una riga ma la copia in un buffer.

Per incollare si usano i comandi **p** oppure **P**. Il primo incolla a partire dalla riga seguente a quella in cui si trova il cursore, il secondo a partire da quella in cui si trova il cursore.

## Ricerche nel testo

Per cercare una stringa all'interno di un file di testo si usa il comando `/`. Tra l'altro questo comando svolge la stessa funzione in molte situazioni (all'interno di `man`, `less`, `more`, ecc.).

Dopo la pressione del tasto `/` è possibile digitare la stringa da cercare che apparirà nella barra di stato.

Per ripetere l'ultima ricerca effettuata si usa il tasto **n**.

## Salvataggio e chiusura

Per salvare il testo editato, si usa il comando **:w**, mentre per uscire dal programma si usa il comando **:q**.

I due comandi possono esseri combinati (**:wq**) per salvare ed uscire nello stesso momento.

Inoltre e' possibile forzare l'uscita senza salvare (con conseguente perdita delle modifiche), con il comando **:q!**.

## Riepilogo 1

- **vi** *nomefile*, apre il file in vi
- **Esc** torna alla modalità comandi
- **:e** (*edit*) apre un file
- **i** o **Ins** inserisce del testo
- **r** o due volte *Ins* sovrascrive il testo
- **[n] dd** taglia *n* righe
- **[n] yy** copia *n* righe
- **p** incolla dopo la riga corrente
- **P** incolla a partire dalla riga corrente

## Riepilogo 2

- / cerca all'interno del testo
- n ripete l'ultima ricerca
- :w salva
- :q chiude
- :q! chiude senza salvare

## Un esempio di script

Vogliamo ora proporvi un piccolo esempio di script BASH.

Lo script che stiamo per scrivere ha il compito di modificare una TAG HTML in tutti i file con estensione *.html* all'interno della directory corrente. La stringa originale e quella nuova vengono memorizzate in due variabili, in modo da rendere lo script facilmente adattabile alle esigenze del momento.

## Un esempio di script

Innanzitutto inseriamo l'identificatore del comando che deve essere utilizzato per interpretare lo script:

```
#!/bin/bash
```

## Un esempio di script

Quindi inizializziamo due variabili **ORIGINALE** e **NUOVO** con il testo da cercare e da sostituire.

```
ORIGINALE='<a href="immagini'
```

```
NUOVO='<a href="images'
```

## Un esempio di script

A questo punto possiamo inizializzare il ciclo for che scandisce i file con estensione *.html*.

```
for i in `ls *.html`; do
```

Notate l'uso degli apici inversi ( ` ` ) per espandere il risultato del comando **ls**

## Un esempio di script

All'interno del ciclo facciamo una copia di back-up del file e poi lo passiamo a **sed** perchè effettui la sostituzione.

```
cp $i $i.old ;  
cat $i.old | \  
sed "s/$ORIGINALE/$NUOVO/" > \  
$i;
```

## Un esempio di script

Infine chiudiamo il ciclo *for*.

done

A questo punto **chmod 755 *nomefile*** trasforma il file in uno script eseguibile. Et voilà!

## Un esempio di script

```
#!/bin/bash
ORIGINALE='<a href="immagini'
NUOVO='<a href="images'

for i in `ls *.html`; do
cp $i $i.old ;
cat $i.old | \
sed "s/$ORIGINALE/$NUOVO/" > \
$i;
done
```