

# Bash shell di Linux

## Sommario

1. [Introduzione](#)
2. [Script Bash](#)
3. [Variabili](#)
4. [Operazioni Aritmetiche](#)
5. [Costrutti di Controllo](#)
6. [Esercizi](#)
7. [Glossario](#)
8. [Documentazione](#)

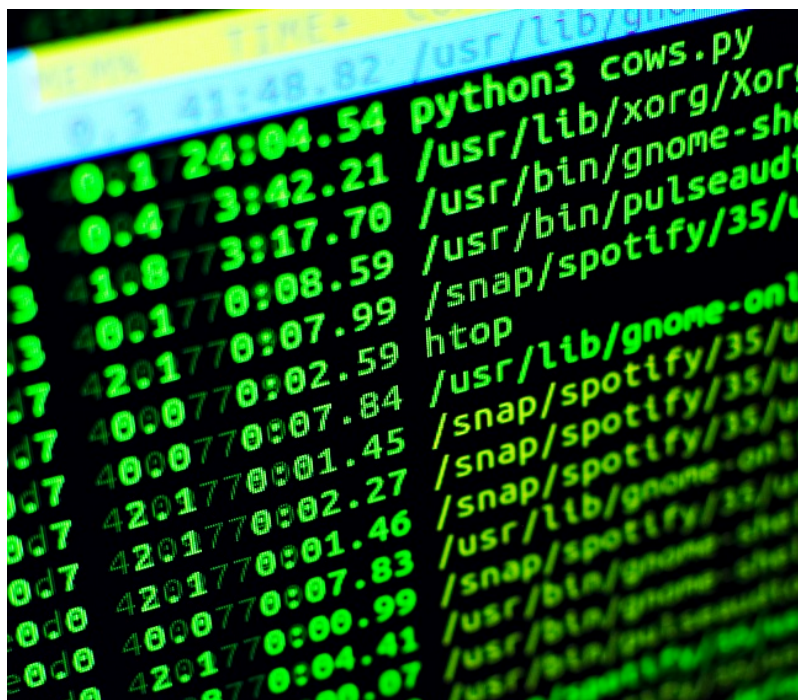
# Introduzione

La Bash shell di Linux, è l'**interfaccia** che permette all'utente di interagire tramite linea di comando con il SO Linux. Venne ideata e progettata dal britannico **Brian Fox** nel 1989, in sostituzione alla tradizionale shell Unix e alla Bourne shell. L'intenzione dell'informatico erano non solo di creare una nuova e migliore shell, ma anche completamente **Open Source**, a differenza della Bourne shell che era limitata dalla sua licenza proprietaria.

La novità della Bash shell fu un **progresso nel linguaggio di scripting**, aggiungendo nuove funzionalità moderne tra cui:

- Gestione array.
- Scripting più avanzato: con l'aggiunta di costrutti logici ( &&, ||, regex) e operatori di assegnazione ( +=, ++).
- Completamento automatico, facilitando la stesura del codice.
- Cronologia dei comandi.
- Maggiore compatibilità, in particolare con lo standard POSIX.

Queste ottimizzazioni portarono la Bash shell ad essere una svolta nel mondo informatico, e ad essere usata fino ai giorni nostri.



Una volta che la shell viene lanciata, esegue uno script di partenza, chiamato `.bashrc` o `.bash_profile` situato nella directory home. Il prompt della shell, ovvero quando la shell è pronta a ricevere le istruzioni, è rappresentato dal simbolo del dollaro: `$`. Se invece eseguiamo la shell come root, o superuser, verrà visualizzato il carattere: `#`.

In questo modo: [username@host ~]\$

[root@host ~]#



← Tux è l'iconica mascotte che rappresenta Linux.

## Script Bash

Uno script bash è una serie di comandi scritti in un file, che vengono letti ed eseguiti dal programma bash, riga per riga. La prima riga dello script deve essere obbligatoriamente **#!/bin/bash**, formata quindi dalla [shebang](#), seguita dall'interprete di comandi da utilizzare, il [pathname assoluto](#). Per convenzione la loro estensione è .sh

Inoltre possiamo assegnare dei permessi di esecuzione ai script, ad ogni singolo utente, il permesso di esecuzione è rappresentato dal carattere "x".

## Eseguire uno script da terminale

Per eseguire uno script da terminale Ubuntu è necessario creare il file "*nome.sh*", usare il comando *which bash* e copiare il percorso del file, successivamente aprire il file con un editor di testo, scrivere il commento con il percorso copiato precedentemente e l'operazione che vogliamo eseguire, assegnare i permessi di esecuzione con il comando *chmod u+x nome\_file.sh*, e infine eseguirlo con il comando *bash nome\_file.sh*

## Debug di uno Script

Gli errori in cui l'utente può incorrere sono gli stessi di qualsiasi altro linguaggio di programmazione, ovvero: errori di sintassi, errori di runtime, errori logici. Per il debug sono utili le seguenti opzioni durante l'esecuzione:

1. **-n** : Non esegue direttamente lo script, ma verifica gli errori di sintassi.
2. **-u** : Durante l'esecuzione visualizza un messaggio di errore se si tenta di utilizzare una variabile non dichiarata.
3. **-v** : Esegue e visualizza ogni riga di codice.
4. **-x** : Visualizza il valore di ogni variabile durante l'esecuzione.

Per implementare queste opzioni è sufficiente aggiungerle nel comando di esecuzione dello script, in questo modo: *bash -n script.sh*

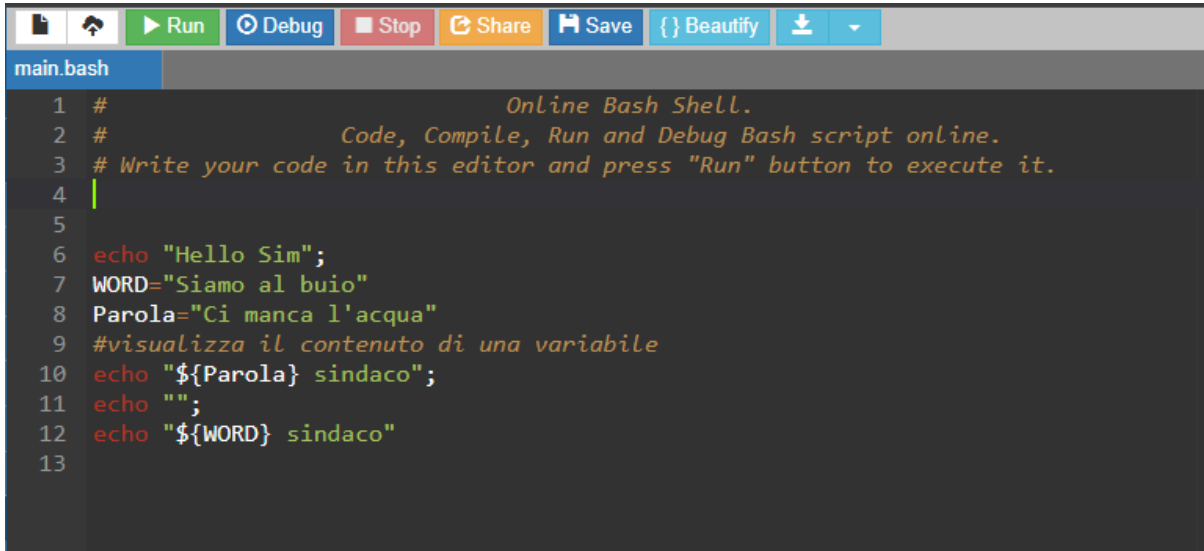
## Commenti dello Script

È possibile aggiungere dei commenti che l'interprete non prende in considerazione, di conseguenza possiamo scrivere righe all'interno del codice precedute dal carattere **#**. il commento termina alla fine della riga. *echo "Tux"           # Questo è un commento.*

## Caratteri Speciali

- **\*** : Indica una qualsiasi stringa.
- **?** : Indica un singolo carattere.
- **[]** :
- **~** : Indica la directory home dell'utente.
- **|** : Carattere di Pipeline, l'output di un comando è l'input del comando successivo.
- **&** :
- **;** : Separatore di comandi, permette di inserire più comandi nella stessa riga.
- **&&** :
- **\** :
- **\$** : Indica una variabile.
- **#** : La riga viene considerata come un commento

# Variabili



```
1 # Online Bash Shell.
2 # Code, Compile, Run and Debug Bash script online.
3 # Write your code in this editor and press "Run" button to execute it.
4
5
6 echo "Hello Sim";
7 WORD="Siamo al buio"
8 Parola="Ci manca l'acqua"
9 #visualizza il contenuto di una variabile
10 echo "${Parola} sindaco";
11 echo "";
12 echo "${WORD} sindaco"
13
```

Le variabili vengono usate per memorizzare stringhe di caratteri e numeri, e possono essere inizializzate dall'utente o di sistema. Sono locali alla shell in uso, e non possono essere viste da altri comandi o applicazioni. Le variabili utente vengono inizializzate in questo modo:

```
nome_variabile=valore
```

Da notare l'assenza di spazi prima e dopo l'uguale, in caso contrario verrebbe visualizzato un errore.

Per visualizzare il contenuto di una variabile utilizziamo il comando:

```
echo "${nome_variabile}";
```

Il carattere **\$** serve a identificare che si tratta di una variabile.

L'inizializzazione degli array avviene attraverso l'uso di parentesi quadre, che indicano la posizione dell'elemento. **numeri[0]=22**

```
nome_array[n]=22
```

Con il seguente comando possiamo visualizzare ogni elemento dell'array: `echo ${numeri[*]};`

Inoltre possiamo modificare la dimensione dell'array, aggiungendo nuovi elementi, e può essere eliminato l'intero array, o un singolo elemento (specificando la posizione), tramite il comando **unset**.

Esercizio esplicativo su variabili e array: [link](#)

## Variabili di sistema:

È presente un altro tipo di variabile, ovvero quelle di **ambiente**, di **sistema** o *environment variable*, contenenti informazioni relative agli account del computer e all'ambiente di lavoro. Non è necessario dichiararle o inizializzarle, ma sono già gestite dalla shell e un qualsiasi programma che viene mandato in stato d'esecuzione, le eredita. A questo processo viene passata una copia delle variabili d'ambiente, con la possibilità di leggerle e processarle.

Alcune variabili di sistema sono:

- **HOME**: La home-directory dell'utente.
- **PATH**: elenco di directory in cui il sistema cerca gli eseguibili dei comandi digitati nel terminale.
- **PS1**: Il prompt della shell.
- **PSW**: Il percorso della directory dove lo script viene eseguito.
- **SHELL**: La shell usata dall'utente.
- **TERM**: Il terminale usato dall'utente.
- **USERNAME**: Il nome dell'utente al momento del login.

Con il comando `env` vengono visualizzate tutte le variabili d'ambiente.

## Parametri posizionali e speciali

I **parametri** sono particolari variabili con solo la possibilità di leggerle, contenenti un valore. I parametri si dividono in **posizionali**, definite da cifre numeriche che indicano la posizione degli argomenti passati come parametri allo script; in questo modo `$1`, `$2`... In caso la posizione dovesse superare il singolo carattere, la cifra deve essere racchiusa tra parentesi graffe: `${13}`. L'altra tipologia di parametri sono quelli **speciali**, tra cui:

- **\$0**: Indica il nome del programma.
- **\$?**: L'output dell'ultimo comando eseguito.
- **\$#**: Il numero di parametri posizionali.
- **\$\***: Gli argomenti su linea di comando, separati da uno spazio.
- **@**: Gli argomenti su linea di comando, separati singolarmente.
- **-**: Le opzioni fornite al comando.
- **\$\$**: Numero ID del processo corrente.
- **\$\_**: Numero ID dell'ultimo processo avviato.
- **\$\_**: L'ultimo argomento del comando precedente.

## Lettura input

Bash mette a disposizione un semplice comando per la lettura dei dati in input da parte di un utente. Questo comando è il *read* che consente l'interruzione dell'esecuzione dello script, in attesa della risposta dell'utente, conclusa dal testo enter. Una volta convalidato l'input, ogni parola viene memorizzata nella variabile scelta. Oltre alla semplice lettura di un valore, possono essere aggiunte delle opzioni, come nell'esempio: [Esercizio di Lettura Input](#)

Le opzioni sono:

1. *-p* Consente la visualizzazione di un messaggio di richiesta.
2. *-n* Specificando un numero, *read* limiterà l'input a quella quantità di caratteri, terminando automaticamente la lettura al raggiungimento del limite.
3. *-s* Nasconderà i caratteri digitati nel terminale, durante la lettura dell'input.
4. *-r* Il comando *read* non terrà conto di eventuali caratteri speciali, come il backslash (\).

# Operazioni aritmetiche

## Comando let

Per memorizzare un risultato in una variabile viene utilizzato il comando `let`.

```
#!/bin/bash
lato_quadrato=5
let perimetro=lato_quadrato*4    #Calcolo perimetro quadrato
echo $perimetro
```

## Operatori aritmetici

Con la Bash shell sono stati introdotti i seguenti **operatori aritmetici**, comuni a moltissimi altri linguaggi di programmazione ad alto livello. Questo ovviamente ha favorito un linguaggio di script più ampio e semplice.

- **+**: Addizione.
- **-**: Sottrazione.
- **\***: Moltiplicazione.
- **/**: Divisione.
- **\*\***: Esponenziale.
- **%**: Resto.

Come nella maggioranza degli altri linguaggi di programmazione, l'uso degli operatori aritmetici è molto semplice, i due operandi vengono divisi dall'operatore, il quale rappresenta l'operazione che vogliamo utilizzare. Possiamo inoltre modificare il segno di una variabile, precedendo un meno ( **-** ) alla variabile.

## Operatori di assegnazione composta

Sempre per avere un linguaggio di script più sviluppato, Brian Fox ha introdotto nella sintassi anche **operatori di assegnazione**, comuni a molti altri linguaggi di programmazione odierni. Generalmente l'operatore che precede il simbolo dell'uguale ( **=** ), specifica l'operazione da svolgere tra il primo operando ( **a** ) e il secondo ( **b** ). Successivamente il risultato viene assegnato al primo operando, in questo modo:

```
a += b ----> a = a + b
a *= b ----> a = a * b
```



# Costrutti di Controllo

## Strutture condizionali:

Viene realizzato attraverso le parole chiave **if**, **then**, **else**, **fi**. Dove **if** indica la prima condizione, **then** il blocco di comandi da eseguire, **else** la condizione opposta, e **fi** la fine della struttura.

```
if [ condizione ]
then
    blocco_comandi_di_then
else
    blocco_comandi_di_else
fi
```

## Operatori di confronto:

OPERATORI DI CONFRONTO	
<b>a -eq b</b>	Vale vero se gli operandi sono uguali (=)
<b>a -ne b</b>	Vale vero se gli operandi sono diversi (!=)
<b>a -lt b</b>	Vale vero se il primo operando è minore del secondo (<)
<b>a -le b</b>	Vale vero se il primo operando è minore o uguale al secondo (<=)
<b>a -gt b</b>	Vale vero se il primo operando è maggiore del secondo (>)
<b>a -ge b</b>	Vale vero se il primo operando è maggiore o uguale al secondo (>=)

## Strutture iterative

La Bash mette a disposizione l'uso di **loops**, ovvero strutture che permettono l'esecuzione di una sezione di codice più volte, a seconda della condizione stabilita. Per stabilire quale sia la sezione di codice da ripetere, è stato deciso di delimitarla dalle parole chiave do e done. I possibili loops sono:

1. **while** (Il ciclo **while** ripete un blocco di codice finché una certa condizione è vera. Dopo ogni esecuzione, la condizione viene controllata di nuovo e, se è ancora vera, il ciclo continua; altrimenti, si interrompe. È logico quindi che all'interno del blocco di codice la condizione possa essere variata, altrimenti risulterebbe ciclare all'infinito, causando un errore.)

2. **until** (Il ciclo until è considerato l'opposto del while, ovvero se la condizione è falsa il blocco di codice viene eseguito, verificando ad ogni interazione la condizione; in caso contrario si interrompe.)
3. **for** (

## Glossario

**Epoch:** è il punto di riferimento per moltissimi sistemi Unix, e indica il tempo trascorso da una data arbitraria, il **1 gennaio 1970**. Viene misurata in secondi interi, ed è stata scelta

**Shebang:** Indica i primi due caratteri rappresentativi dello script: **#!**

**Pathname assoluto:** La posizione dell'interprete dei comandi da utilizzare. `/bin/bash` è il pathname assoluto dell'interprete Bash, mentre `/usr/bin/python3` è il pathname assoluto dell'interprete Python.

**Environment:** È un principio di Linux, e rappresenta l'insieme dei comandi associati all'utente una volta effettuato il login. È costituito dalle variabili d'ambiente, dall'utente dalle impostazioni del sistema e dalla shell.

# Documentazione

Storia Bash shell:

<https://managedserver.it/guida-alle-shell-linux-da-bash-a-bourne-passando-per-zsh-ed-altre/>

Schema riassuntivo:

<http://www.scienze.unitn.it/~fiorella/AppuntiLinux/al-22.html>

Articolo di CodeCamp con esempi di script:

<https://www.freecodecamp.org/italian/news/shell-scripting-per-principianti/>

Documentazione Bash:

<https://rocky-linux.github.io/documentation/LearningBashWithRocky.it.pdf>