

Bash shell di Linux 🐧

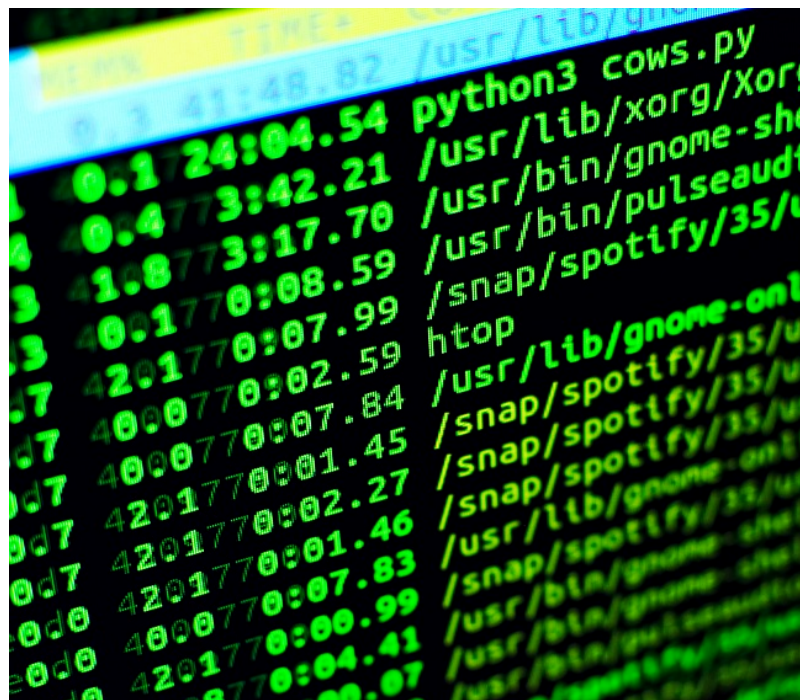
Introduzione

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 quadrate, 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**.

Variabili di sistema:

È presente un altro tipo di variabile, ovvero quelle di **ambiente** o di **sistema**, 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

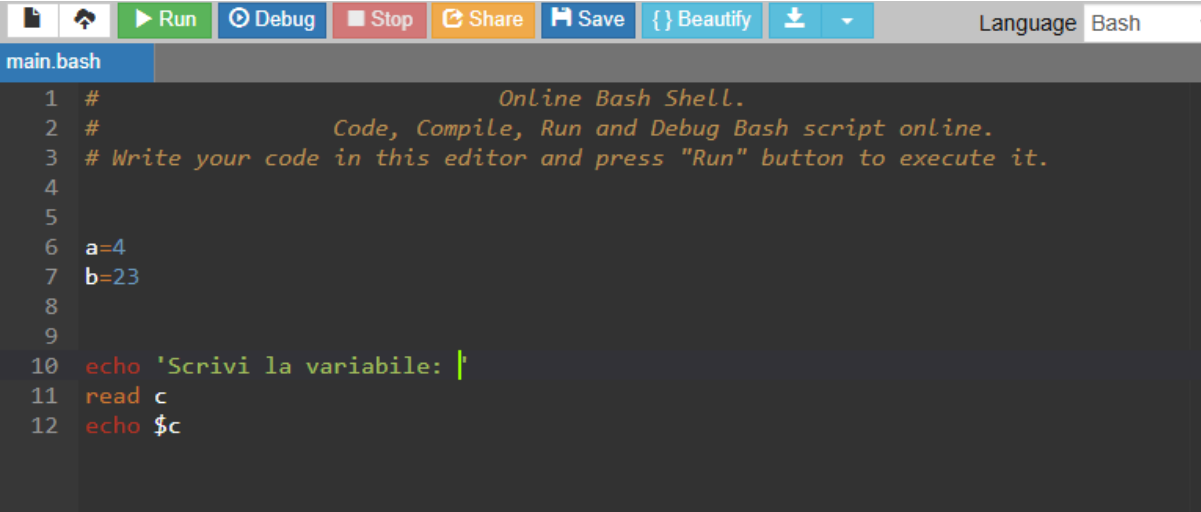
I **parametri** sono particolari variabili con solo la possibilità di leggerle, contenenti un valore.

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:

Per leggere valori di variabili in input viene usato il comando `read`, in questo modo:



The image shows a web-based code editor for Bash. At the top, there is a toolbar with buttons for Run, Debug, Stop, Share, Save, Beautify, and a download icon. The language is set to Bash. The editor contains a script named 'main.bash' with the following content:

```
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 a=4
7 b=23
8
9
10 echo 'Scrivi la variabile: '
11 read c
12 echo $c
```

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

Strutture iterative

Il ciclo for cicla un condizione finché quest'ultima è vera:

```
for i in 1 2 3 4 5
do
    echo "Numero $i"
done
```

Esempi di esercizi

Struttura if

Write your code in this editor and press "Run" button to execute it.

```
echo "Scrivi il tuo voto di TPSIT: "  
read voto  
echo "Il tuo voto è: "$voto  
  
if [[ "${voto}" -le 10 ]]  
then  
    case $voto in  
        10)  
            echo "Hai preso 10, molto bene!";;  
        9)  
            echo "Hai preso 9, bene";;  
        8)  
            echo "Hai preso 8, bene ";;  
        7)  
            echo "Hai preso 7, non male ";;  
        6)  
            echo "Hai preso 6, ok ";;  
        5)  
            echo "Hai preso 5, poteva andare meglio ";;  
        4)  
            echo "Hai preso 4, male! ";;  
        3)  
            echo "Hai preso 3, molto male!";;  
        *)  
            echo "Voto non valido!";;  
    esac  
else echo "voto non valido"  
fi
```

In questo esercizio ho utilizzato una prima struttura if, e successivamente un switch per associare un output diverso per ogni voto.

Generazione Password

Per generare delle Password relativamente complesse e randomiche utilizzeremo varie tecniche:

1. *RANDOM* genera numeri randomici interi da 0 a 32767.
2. Con il comando: *date +%s* la Password2 sarà formata dai secondi trascorsi dall'**epoch**.
3. (*date +%s+%N | sha256sum | head -c10*): utilizzando questo comando, i passaggi per ottenere l'output saranno questi:
 - o *date +%s+%N*: otteniamo i secondi (s) e nanosecondi (N) trascorsi dal **epoch**, per esempio 1709180882+123456789.
 - o attraverso la pipeline passiamo l'output precedente (*date*) al comando successivo.
 - o Con il comando *sha256sum* otteniamo il checksum, ovvero una stringa da 64 caratteri esadecimali, del comando *date* .
 - o attraverso la pipeline passiamo l'output precedente (*sha256sum*) al comando successivo.
 - o infine tramite *head -c10* prendiamo solo i primi 10 caratteri della stringa, creata dal *sha256sum*

```
1 #!/bin/bash
2
3 Password=${RANDOM}    #Con RANDOM vengono generati numeri casuali.
4 echo "PAssword1: $Password"
5
6 Password2=$(date +%s) #In questo modo viene generata la password a partire dalla data e i secondi
7 echo "PAssword2: ${Password2}"
8
9 Password3=$((date +%s%N ) | sha256sum | head -c10) #La password è composta dai primi 10 caratteri
10 echo "PAssword3: ${Password3}"
```

Input del programma

Uscita del programma

PAssword1: 4690
PAssword2: 1732614191
Password3: 3d14b4a6d7

[Execution complete with exit code 0]

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.

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/>