

Bash shell di Linux

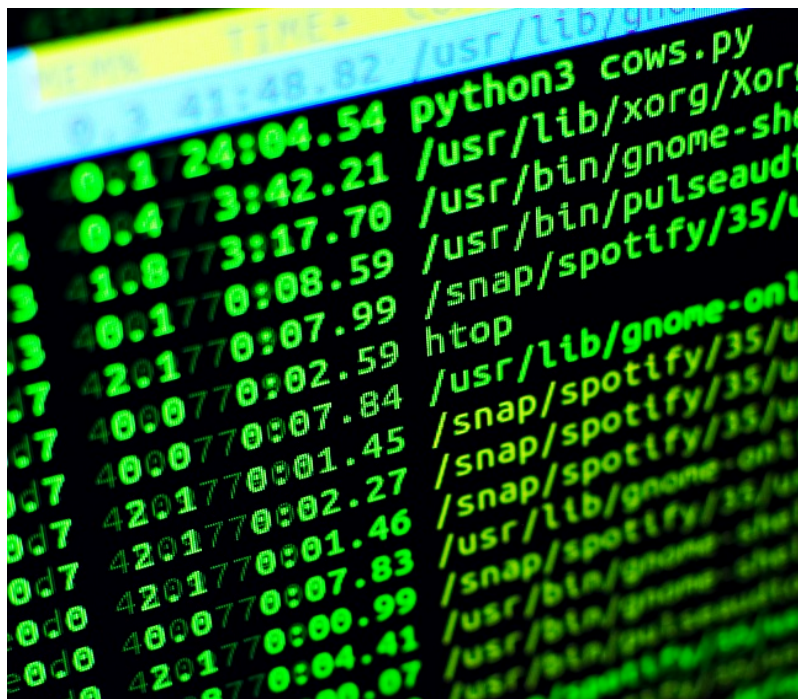
Introduzione:

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 ~]#

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` che indica alla shell il nome dell'interprete di comandi da utilizzare.

Variabili:



```
main.bash
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. Le variabili utente vengono inizializzate in questo modo: `nome_variabile=valore`

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:

Alcune variabili predefinite sono le variabili di sistema che rappresentano i valori di inizializzazione dell'ambiente di lavoro per l'utente.

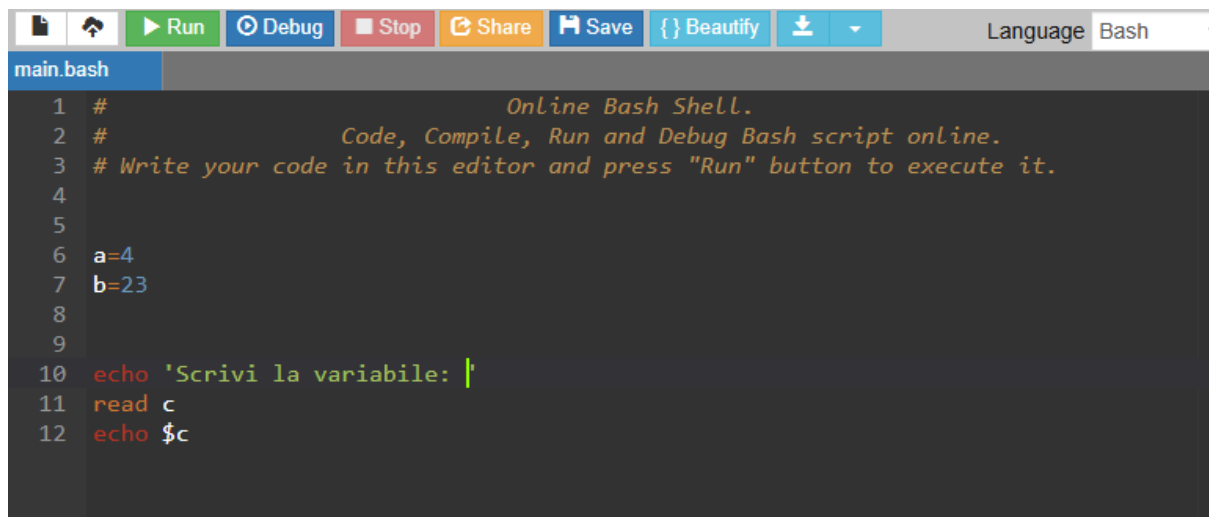
pwd

pat

home

Lettura input:

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



```
main.bash
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
```

Operatori aritmetici:

Per assegnare un valore ad una variabile viene utilizzato il comando `let`, in questo modo:

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

Sono stati introdotti questi operatori per svolgere l'insieme delle operazioni.

OPERATORE	UTILIZZO
+	addizione
-	sottrazione
*	moltiplicazione
/	divisione
**	esponenziale
%	resto

OPERATORI ARITMETICI	
- a	Inverte il segno dell'operando
a + b	Somma i due operandi
a - b	Sottrae il secondo operando dal primo
a * b	Moltiplica i due operandi
a / b	Divide il primo operando per il secondo
a % b	Restituisce il modulo, cioè il resto della divisione tra il primo operando e il secondo
a += b	a = a + b
a -= b	a = a - b
a *= b	a = a * b
a /= b	a = a / b
a %= b	a = a % b

Struttura di selezione:

Viene effettuato attraverso i comandi **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 (>=)

Ciclo for

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 da quando è acceso il computer, a partire dal primo gennaio 1970, una data che spesso viene usata in informatica per misurare il tempo.
3. `(date +%s+%N | sha256sum | head -c10)`, in questo modo a partire dai secondi e nanosecondi trascorsi dal primo gennaio 1970, prendiamo i primi 10 caratteri esadecimali della stringa, formata da 64 caratteri, creata dall'output del comando `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]

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/>