# BASH SHELL

La **Bash shell** (Bourne Again Shell) è il **software** che ci consente di eseguire il codice di linux, ed è in grado di interpretare i comandi permettendoci di gestire un intero sistema operativo.

**Bash** è una **"Unix shell"**, vale a dire un'**interfaccia a linea** (o riga) di comando che permette l'interazione con sistemi operativi derivati da Unix; è stata utilizzata su molti altri sistemi, non necessariamente derivati da Unix. È stata creata verso la fine degli anni '80 da Brian Fox.

Lo scopo era quello di fornire un software libero alternativo alla **Bourne shell**, in precedenza una delle principali shell per i sistemi GNU/Linux, incorporando tutte le caratteristiche assieme ad alcune valide aggiunte, come l'aritmetica intera e la valutazione delle espressioni regolari.

Bash può eseguire comandi con una risposta in tempo reale ed eseguire interi script conosciuti come "Bash shell script".

In definitiva, un Bash Shell script è un programma scritto nel linguaggio di programmazione Bash.

I Bash script vengono comunemente usati per molte **attività di amministrazione di sistema**, come l'analisi dei log, il backup dei dischi, e così via. Sono anche molto usati come script di installazione di programmi complessi. Insomma, la conoscenza pratica dello Shell Scripting è essenziale per tutti coloro che desiderano diventare degli amministratori di sistema esperti.

Qui di seguito c'è una tabella con la sintassi dei comandi principali utilizzati nello scripting nella Bash Shell, Bash offre un sacco di controlli e confronti integrati, arrivando abbastanza utili in molte situazioni.

## Caratteri Speciali

Creato da Filippo Sperandio & Stefano Casonato 4^A 2024

| | |
|---|---|
| * | Jolly che rappresenta "qualsiasi stringa" (compreso nulla) |
| ? | Jolly che rappresenta un singolo carattere |
| [ ] | Rappresenta una classe di caratteri (caratteri dentro le parentesi quadre) |
| ~ (tilde) | Rappresenta la home directory dell'utente |
| \| | Pipe, invia l'output di un comando come input di un altro |
| & | Esegue un comando in background |
| ; | Separatore di comandi, esegue più comandi consecutivamente |
| && | segue il comando successivo solo se il primo comando ha successo |
| \ | Carattere di escape, usato per "sfuggire" a un carattere speciale o per trattarlo come normale |
| $ | Rappresenta una variabile o l'output di un comando |

L’utilizzo della variabile speciale “**$0**”, è necessario se vogliamo visualizzare a schermo:

- il nome del file attualmente in esecuzione;
- trovare il “Login Shell” ossia il primo programma eseguito sotto il tuo User ID.

**Sono presenti altri caratteri speciali:**

| | |
|---|---|
| $n | In base al numero scritto dopo il “$”, es. ($1, $2, …) viene stampato il parametro dato al programma. |
| $@ o $* | Rappresenta tutti gli argomenti passati allo script o ad una funzione, ma il funzionamento cambia se vengono utilizzate le virgolette: vengono gestiti in modo sicuro gli argomenti, soprattutto quando contengono spazi. |

| | |
|---|---|
| $# | Espande al numero di parametri posizionali passati (in formato decimale). |
| $? | Espande allo stato di uscita dell'ultimo comando eseguito in primo piano (foreground). |
| $- | Espande alle opzioni correnti della shell, come specificato nel comando set o quelle impostate dalla shell stessa (es. opzione -i per modalità interattiva). |
| $$ | Espande al PID (Process ID) del processo della shell corrente. Se usato in un sottoprocesso (subshell), espande al PID della shell che ha invocato il sottoprocesso. |
| $! | Espande al PID del processo più recente che è stato messo in background. |

## Condizioni

Quando inizi a scrivere e utilizzare le tue condizioni, ci sono alcune regole che dovresti sapere per evitare di ottenere errori difficili da rintracciare. Qui seguiamo tre importanti:

1. **Spazi:** Utilizza spazi adeguati all'interno delle strutture di controllo (come `if`, `for`, `while`) per separare le parole chiave e migliorare la leggibilità.

2. **Una linea per ogni parola chiave:** Le parole chiave come `if`, `then`, `else`, `elif` e `fi` devono sempre iniziare una nuova riga. Questo rende il codice più leggibile e evita errori di sintassi. Se hai bisogno di concatenare più istruzioni su una stessa riga, utilizza il carattere “`;`” per separarle.

3. **Citazione delle variabili:** Quando utilizzi variabili di tipo stringa all'interno di condizioni o comandi, è sempre consigliabile racchiuderle all'interno di apici singoli (') o doppi ("). Questo previene problemi causati da spazi o caratteri speciali all'interno

Per confrontare le variabili vengono utilizzati diversi operatori, questi sono elencati nella tabella:

| Operatore | Descrizione |
|---|---|
| `-eq` | uguale a |
| `-ne` | diverso da |
| `-lt` | minore di |
| `-le` | minore o uguale a |
| `-gt` | maggiore di |
| `-ge` | maggiore o uguale a |

oltre al confronto delle variabili si può creare ed utilizzare anche strutture logiche come if, for, while…

**if:**

```
if [ condizione1 ]; then
    # comandi da eseguire se condizione1 è vera
elif [ condizione2 ]; then
    # comandi da eseguire se condizione2 è vera
else
    # comandi da eseguire se nessuna delle condizioni è vera
fi
```

**for:**

```
for (( i=inizio; i<=fine; i++ )); do
    # comandi
done
```

## while:

```
while [ condizione ]; do
    # comandi
done
```

## switch:

```
case variabile in
    pattern1)
        # comandi per il pattern1
        ;;
    pattern2)
        # comandi per il pattern2
        ;;
    *)
        # comandi per il caso predefinito (opzionale)
        ;;
esac
```

## ECCO ALCUNI ESERCIZI SVOLTI IN CLASSE

```
#!/bin/bash

UID_TO_TEST=0
USER_NAME=$(id -nu)
USER_ID=$(id -u)
x=7

echo "Your user name ${USER_NAME}"
echo "your id name ${USER_ID}"

if [[ "${USER_ID}" -eq 0 ]]
then
        echo 'you ara Root'
else
        echo 'you are not Root'
fi

#--------------------------------------------------------------------------

USER_NAME=$(id -nu)
USER_ID=$(id -u)
```

```
if [[ "${USER_ID}" -ne "${UID_TO_TEST}" ]]
then
	echo "Your UID does not match ${NAME_TO_TEST}"
else
	echo "your username does match ${NAME_TO_TEST}"
fi

#------------------------------------------------------------------------
#esercizio for


for (( x=0; x<="${UID_TO_TEST}"; x++ )); do
	if [[ "${x}" -le 3 ]]
	then
		echo "Studia di piu'"

	elif [[ "${x}" -le 5 ]]
	then
		echo "Ci sei quasi"

	elif [[ "${x}" -le 7 ]]
	then
		echo "Sufficiente"

	elif [[ "${x}" -le 10 ]]
	then
		echo "Bravissimo"
	fi
done
#------------------------------------------------------------------------
#esercizio esempio switch
if [[ "${x}" -le 10 ]]; then
	case "${x}" in
	10)	echo "hai preso 10";;
	9)	echo "hai preso 9";;
	8)	echo "hai preso 8";;
	7)	echo "hai preso 7";;
	6)	echo "hai preso 6";;
	5)	echo "hai preso 5";;
	4)	echo "hai preso 4";;
	3)	echo "hai preso 3";;
	*)	echo "nessun voto";;
	esac
fi
#------------------------------------------------------------------------
#esercizio numero primo
function primeNumber() {
	valore=$1
```

```
        for (( x=2; x< ${valore}; x++ )); do
                if [[ $(( valore%x )) -eq 0 ]]
                then
                echo "${valore} non e' primo"
                return 0
                fi
                echo "${valore} e' primo"
                return ${valore}
                done
}
 primeNumber 8
#-------------------------------------------------------------------------
```