

# GUIDA BASH

Realizzata da:

Rusila Marcus, Barbieri Cristian Andrea, Adday Yasser

## Cos'è la Bash Shell?

La BASH SHELL è un interprete di linguaggio di comandi rappresentato con il nome BASH e si trova nella directory /bin. Il suo pathname o anche detto percorso è:

**/bin/bash**

La shell assegna automaticamente una directory sulla quale l'utente possa lavorare detta **Home directory**.

All'accesso la shell legge un file globale con tutti i parametri di sistema e di configurazione dell'ambiente.

Gli script (ovvero un file di testo che contiene una serie di comandi da eseguire in sequenza) di inizializzazione devono essere eseguibili tramite il comando **chmod** e iniziare con il commento:

**#!/bin/bash**

per specificare l'interprete da usare.

Il più utilizzato è il **chmod 755**

## Le variabili nella shell

Le variabili vengono utilizzate dalla shell per memorizzare e modificare stringhe di caratteri o numeri.

**Le variabili possono essere definite dall'utente oppure dal sistema.**

Il comando **set** visualizza le variabili inizializzate e i suoi valori nell'attuale sessione dell'utente nella shell.

Come si assegna una variabile?

**nome\_variabile=valore**

Il suo valore può essere visualizzato tramite il comando **echo** preceduto dal simbolo **\$** che indica alla shell di sostituire il nome della variabile con il suo valore.

## Gli array in bash

In Bash, un array è una struttura dati che consente di memorizzare più valori in una singola variabile. Gli array in Bash possono essere indicizzati numericamente (array indicizzati) o associativi (array associativi), ma qui ci concentreremo sugli array indicizzati, che sono i più comuni.

**Creazione e utilizzo di un array indicizzato:**

Per creare un array, basta assegnare i valori separati da spazi e racchiusi tra parentesi tonde. Ecco un esempio di un array che contiene i nomi di alcune città:

```
# Creazione dell'array
```

```
citta=("Roma" "Milano" "Napoli" "Torino")
```

```
# Accesso agli elementi dell'array
```

```
echo ${citta[0]} # Stampa "Roma"
```

```
echo ${citta[1]} # Stampa "Milano"
```

```
# Modifica di un elemento
```

```
citta[2]="Venezia"
```

```
echo ${citta[2]} # Stampa "Venezia"
```

```
# Stampa di tutti gli elementi dell'array
```

```
echo ${citta[*]} # Stampa "Roma Milano Venezia Torino"
```

## Le primitive di test

Nella bash shell si possono usare delle condizioni che controllano lo stato dei file e delle directory.

La condizione generalmente è indicata con:

**-lettera<nome\_elemento>**

Ecco una lista delle primitive di test

PRIMITIVE DI TEST	
<b>r</b>	Permesso in lettura
<b>w</b>	Permesso in scrittura
<b>x</b>	Permesso in esecuzione
<b>e</b>	Esistenza dell'elemento
<b>O</b>	Proprietà sull'elemento
<b>s</b>	Dimensione maggiore di zero per il file
<b>f</b>	File ordinario
<b>d</b>	Directory

## Operatori di confronto

Quando usiamo delle condizioni può esserci il bisogno di confrontare delle variabili, e per farlo utilizziamo gli **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 (>=)

## Operatori booleani

Gli operatori booleani possono essere utilizzati per costruire delle espressioni condizionali più complesse

OPERATORI BOOLEANI	
<b>! espressione</b>	Rappresenta la negazione logica dell'espressione
<b>espressione -a espressione</b>	Produce un'espressione vera se entrambe le espressioni hanno dato valore logico vero
<b>espressione -o espressione</b>	Produce un'espressione vera se almeno una delle due espressioni ha dato valore logico vero

## La condizione IF

L'istruzione if in bash è una struttura di controllo che consente di eseguire i comandi sulla base del risultato di una condizione. La sua struttura:

```
if [[ condizione ]]; then
    #comando se è vero
else
    #comando se è falso
fi
```

L'**else** presente nella struttura è un “alternativa” ovvero se la condizione non è vera tramite l'**else** eseguirà la condizione falsa.

## Ciclo FOR

Il ciclo **for** consente di eseguire una sequenza di comandi per ogni elemento di una lista specificata.

Struttura base:

```
for variabile in lista; do
    # comando da eseguire
done
```

Esempio pratico:

```
for i in {1..5}; do
    echo "Numero $i"
done
```

## Ciclo WHILE

Il ciclo **while** esegue un blocco di comandi finché una condizione è vera.

Struttura base:

```
while [ condizione ]; do
```

```
# comando da eseguire  
done
```

Esempio pratico:

```
contatore=0  
while [ $contatore -lt 5 ]; do  
    echo "Contatore: $contatore"  
    contatore=$((contatore + 1))  
done
```

## Struttura SWITCH

La struttura **switch**, implementata in bash tramite il comando **case**, consente di eseguire comandi diversi in base al valore di una variabile.

Struttura base:

```
case variabile in
```

```
    valore1)
```

```
        # comando per valore1
```

```
        ;;
```

```
    valore2)
```

```
        # comando per valore2
```

```
        ;;
```

```
    *)
```

```
        # comando di default
```

```
;;  
  
esac
```

Esempio pratico:

```
echo "Inserisci un'opzione (start, stop, restart):"  
read opzione
```

```
case $opzione in  
    start)  
        echo "Avvio del servizio..."  
        ;;  
    stop)  
        echo "Arresto del servizio..."  
        ;;  
    restart)  
        echo "Riavvio del servizio..."  
        ;;  
    *)  
        echo "Opzione non valida."  
        ;;  
esac
```

## Funzioni in Bash

Le funzioni permettono di organizzare il codice in blocchi riutilizzabili.

Sintassi



```
nome_funzione() {  
    # Comandi da eseguire  
    echo "Funzione chiamata con argomento: $1"  
}
```

```
# Chiamata della funzione  
nome_funzione "Test"
```

Esempio pratico (Funzione per sommare due numeri):

```
#!/bin/bash  
# Funzione per sommare due numeri  
  
somma() {  
    risultato=$(( $1 + $2 ))  
    echo "La somma di $1 e $2 è: $risultato"  
}  
  
# Leggere input dall'utente  
echo "Inserisci il primo numero:"  
read num1  
echo "Inserisci il secondo numero:"  
read num2  
  
# Chiamare la funzione  
somma "$num1" "$num2"
```

Esecuzione:

```
bash somma.sh
```

## Redirezione Input/Output

La redirezione permette di leggere o scrivere su file.

Esempi:

# Scrive su file

```
echo "Ciao mondo" > output.txt
```

# Aggiunge a un file

```
echo "Riga aggiuntiva" >> output.txt
```

# Legge da file

```
cat < output.txt
```

## Alcuni esempi di esercizi in Bash

**1:**

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

```
#display if the user is the root user or not
```

```
USER__NAME=$(id -nu)
```

```
USER__ID=$(id -u)
```

```
echo "Your user name ${USER__NAME}"
```

```
echo "Your id name ${USER__ID}"
```

```
#Display if the user is the root or not
```

```
if [[ "${USER_ID}" -eq 0 ]]
```

```
then
```

```
    echo 'You are Root'
```

```
else
```

```
    echo 'You are not Root'
```

```
fi
```

```
#esempio di incremento di una variabile numerica
```

```
A=10
```

```
let SOMMA=(${A}+1)
```

```
echo "dieci più uno fa: ${SOMMA}"
```

```
UID_TO_TEST=0
```

```
#Display if the user is the root or not
```

```
if [[ "${USER_ID}" -ne "${UID_TO_TEST}" ]]
```

```
then
```

```
    echo "Your UID does NOT match ${UID_TO_TEST}"
```

```
    #exit 0
```

```
fi
```

```
NAME_TO_TEST='judge'
```

```
#Display if the user is the root or not
```

```
if [[ "${USER_NAME}" != "${NAME_TO_TEST}" ]]
```

```
then
```

```
    echo "Your username does NOT match ${NAME_TO_TEST}"
```

```
    exit 1
```

```
else
```

```
    echo "Your username does match ${NAME_TO_TEST}"
```

```
fi
```

**2:**

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

```
#
```

```
# This script creates a new user on the local system.
```

```
# You will be prompted to enter the username (login), the  
person name and a password.
```

```
# The username, password, and host for the account will be  
displayed.
```

```
# Make sure the script is being executed with superuser  
privileges.
```

```
if [[ "${UID}" -ne 0 ]]
```

```
then
```

```
    echo 'please run with sudo or as root'
```

```
    exit 1
```

```
fi
```

```
# Get the username (login)
```

```
read -p 'Enter the username to create: ' USER_NAME
```

```
# Get the real name (content for the description field)
```

```
read -p 'Enter the name of the person or application that will  
be using this account: ' COMMENT
```

```
# Get the password
```

```
read -p 'Enter the password to use for the account: '  
PASSWORD
```

```
# Create the account
```

```
useradd -c "${COMMENT}" -m ${USER_NAME}
```

```
# Check to see if the useradd command succeeded
```

```
# We don't want to tell the user that an account was created  
when it hasn't been
```

```
if [[ "${?}" -ne 0 ]]
```

```
then
```

```
    echo 'The account could not be created'
```

```
    exit 1
```

```
fi
```

```
# Set the password
```

```
echo ${PASSWORD} | passwd --stdin ${USER_NAME}
```

```
if [[ "${?}" -ne 0 ]]
```

```
then
```

```
    echo 'The password for the account could not be sent'
```

```
    exit 1
```

```
fi
```

```
# Force password change on first login
```

```
password -e ${USER_NAME}
```

```
# Display the username, password, and the host where the user  
was created
```

```
echo
```

```
echo 'username:'
```

```
echo "${USER_NAME}"
```

```
echo
```

```
echo 'password:'
```

```
echo "${PASSWORD}"
```

```
echo
echo 'host:'
echo "${HOSTNAME}"
exit 0
```

3:

```
#!/bin/bash
# Script per fare il backup di una directory

origine="/home/utente/documenti"
destinazione="/home/utente/backup"

if [ ! -d "$destinazione" ]; then
    mkdir -p "$destinazione"
fi

cp -r "$origine"/* "$destinazione"
echo "Backup completato!"
```

4:

```
#!/bin/bash
#creazione di una password pseudo casuale

#PASSWORD=${RANDOM}
#echo "${PASSWORD}${PASSWORD}${PASSWORD}"
```

```
#PASSWORD=$(date +%s%N)
#echo "${PASSWORD}"
```

```
PASSWORD=$(date +%s%N | sha256sum | head -c10)
#echo "${PASSWORD}"
```

```
S_C1=$(echo '!@$%&^*()_ - += ' | fold -w1 | shuf | head -c1)
S_C2=$(echo '!@$%&^*()_ - += ' | fold -w1 | shuf | head -c1)
echo "${S_C1}${S_C2}${PASSWORD}${S_C2}${S_C1}"
```