

**Sistemi Operativi**  
**Unità 2: Utilizzo di Linux**  
**Programmi in Bash**

Martino Trevisan  
Università di Trieste  
Dipartimento di Ingegneria e Architettura

## Argomenti

1. Script Bash
2. Variabili
3. Strutture di controllo
4. Esercizi

# Script Bash

## Script Bash

### Definizione

Con **Bash** si intende il software di Shell che si usa quasi sempre in Linux

Permette di:

- Eseguire comandi (già visto)
- Definire variabili
- Controllare il flusso ( `if` , `while` , ecc...)

E' un linguaggio completo di tutti i costrutti.

Ha una sintassi particolare e problematica

Una lista di comandi può essere racchiusa in un file detto **script**.

# Script Bash

## Esempio

Uno script di esempio nel file `script.sh` :

```
#!/bin/bash  
# primo esempio di script  
echo $RANDOM
```

`#!/bin/bash` Indica che il file è uno script bash

`# primo esempio di script` E' un commento

`echo $RANDOM` Stampa il contenuto della variabile `$RANDOM`

Per eseguire lo script.

```
./script.sh
```

Il file deve avere permessi di lettura ed esecuzione

## Script Bash

### Valore di ritorno

Uno script è una lista di comandi che vengono eseguiti più delle strutture di controllo.

```
ls # Lista i file  
./myprog # Avvia il programma myprog
```

Ogni processo in Linux/POSIX deve fornire un **Valore di Ritorno** al chiamante, ovvero:

- La shell
- Uno script bash
- Un qualsiasi altro processo

## Script Bash

### Valore di ritorno

Il valore di ritorno è un numero intero

- Usato dal chiamante per vedere se c'è stato errore
- Per convenzione **0** se successo,  $\neq 0$  in caso di errore

In uno script bash, si accede al Valore di Ritorno dell'ultimo comando tramite `$?`

## Script Bash

### Parametri di riga di comando

Si possono usare in uno script. Si ottengono con:

- `$1` , `$2` , ...: contenuti dei parametri
- `$0` : nome dello script
- `$#` : numero di argomenti

**Esempio:** il nome dello script è `script.sh` e viene eseguito come `./script.sh ciao`.

```
#!/bin/bash
echo $0 # Stampa "./script.sh"
echo $1 # Stampa "ciao"
```



# Variabili

## Variabili

### Nomi

Combinazione illimitata di lettere, numeri e underscore.

Non possono cominciare con numeri e sono CASE sensitive.

### Lettura di variabili da terminale

Istruzione `read nomevar`

```
read nome
```

### Stampa su schermo

Si utilizza `echo` .

```
echo "Testo su schermo"
```

## Variabili

### Utilizzo di variabili

Si accede con `$nomevar`.

```
read x    # Legge X da terminale
y=$x      # Assegna a y il valore di x
echo $y   # Stampa quanto letto
```

I tipi principali sono:

- **Stringhe:** `a="testo"`
- **Interi:** `a=47`

**Nota:** non inserire spazi prima e dopo `=` durante assegnazione

## Variabili

### Quoting

Le stringhe vanno racchiuse tra " o tra '.

Il carattere \ indica il quoting. Permette di usare nella stringa il carattere di quoting

#### Esempio:

`a="ciao"` indica la stringa `ciao`

`a='ciao a tutti'` indica la stringa `ciao a tutti`

`a="ha detto: \"ciao\""` indica la stringa `ha detto: "ciao"`

## Variabili

## Quoting

### Quoting:

Le stringhe definite con `"` possono contenere delle variabili che vengono valuate.

```
a="test"  
b="this is a $a"  
c='this is a $a'
```

La variabile `b` contiene `this is a test`

La variabile `c` contiene `this is a $a`

## Variabili

**Esempio:** leggere due stringhe da tastiera e stamparle.

```
read a  
read b  
echo $a $b
```

## Variabili

## Operazioni Matematiche

Solo numeri interi con segno

- Se si usano valori floating non segnala errore ma fa i calcoli con numeri interi

Operazioni ammesse: `+` `-` `*` `/` `%` `<<` `>>` `&` `^` (or esc.) `|`

Utilizzo: l'espressione `$(( var1 + var2 ))` restituisce la somma di due variabili

Scrivere `$(( $var1 + $var2 ))` è equivalente

## Variabili

**Esempio:** Si scriva un programma che legge due interi da tastiera e stampa il prodotto

```
#!/bin/bash
read a
read b
c=$(( a * b ))
echo "Il prodotto è $c"
```

### Note:

Osservare la forma `c=$(( a * b ))`

Osservare la concatenazione naturale in `echo "Il prodotto è $c"`



# Strutture di controllo

## Strutture di controllo

### Condizioni

Le condizioni hanno forme

```
if condizione then
    ramo 1
elif condizione2 then
    ramo 2
else
    ramo alternativo
fi
```

Esistono molte sintassi alternative per esprimere le condizioni.  
Ne vedremo una parte.

## Strutture di controllo

### Condizioni tra numeri

Si utilizza la sintassi `(( espressione ))`

Gli operatori di confronto sono i classici: `==` `!=` `<` `>` `<=` `>=`

Esempio:

```
read n1 n2
if (( n1<n2 ))
then
    echo "$n1 minore di $n2"
elif (( n1==n2 )) then
    echo "$n1 uguale a $n2"
else
    echo "$n1 maggiore di $n2"
fi
```

## Strutture di controllo

### Condizioni tra stringhe

Si utilizza la sintassi `[[ espressione]]`

Gli operatori di confronto sono:

- `=` `!=` : uguaglianza o differenza  
- `>` `<` : ordinamento alfabetico
- `-z` : vero se la stringa è vuota (o la variabile non è definita).  
`! -z` è vero se la variabile non è vuota
- E' necessario usare l'operatore `$` e mettere spazi tra operandi

Esempio: `if [[ $a != $b ]]`

Esempio: `if [[ ! -z $var ]]` : vero se `var` esiste e non è vuota

## Strutture di controllo

## Condizioni tra stringhe

### Esempio:

```
#!/bin/bash
read s1
read s2
if [[ $s1 = $s2 ]]
then
    echo "Le stringhe sono uguali"
else
    echo "Le stringhe sono diverse"
fi
```

## Strutture di controllo

### Condizioni su file

E' molto semplice testare se un file esiste, è vuoto o è una cartella

- `-a path` : vero se `path` esiste
- `-f path` : vero se `path` è un file
- `-c path` : vero se `path` è una cartella
- `-s path` : vero se `path` non è vuoto
- `-r path` : vero se posso leggere `path`
- `-w path` : vero se scrivere leggere `path`
- `-x path` : vero se eseguire/attraversare leggere `path`

## Strutture di controllo

**Esempio:** si scriva un programma che legge due path da tastiera. Se sono uguali, controlla che il path corrisponda a una cartella. Se affermativo, stampa il path.

```
#!/bin/bash
echo "Inserisci il primo path:"
read s1
echo "Inserisci il secondo path:"
read s2
if [[ $s1 = $s2 ]]
then
    if [[ -d $s1 ]]
    then
        echo "$s1 è una cartella"
    else
        echo "$s1 non è una cartella"
    fi
else
    echo "Le due stringhe non sono uguali"
fi
```

## Strutture di controllo

### Operatori logici

Si possono creare condizioni composte con gli operatori booleani

- `&&` : and
- `||` : or

Sintassi: `if condizione1 && condizione2`

Esempio: `if (( a>b )) && [[ $c="hello" ]]`



## Strutture di controllo

### Operatori logici

Si possono usare anche per eseguire comandi secondo una logica voluta

- Ogni comando/programma avviato in bash fornisce alla shell/script chiamante un valore di ritorno
- Per convenzione un comando ritorna: **0** se successo, **≠ 0** in caso di errore

#### Conseguenza:

In bash, il valore **0** è interpretato come `true`

Un valore **≠ 0** come `false`

**NOTA:** diverso da altri linguaggio come C o Java!

## Strutture di controllo

### Operatori logici

#### Utilizzo in espressioni di comandi

Esegue comando2 se comando1 non dà errore

```
comando1 && comando2
```

Esegue comando2 se comando1 dà errore

```
comando1 || comando2
```

**Esempio:** eseguo `myprog` solo se la compilazione è andata a buon fine

```
gcc myprog.c -o myprog && ./myprog
```

## Strutture di controllo

### Cicli **for**

#### Versione semplice

```
for n in 1 2 3 4
do
    echo "valore di n = $n"
done
```

```
for nome in mario giuseppe vittorio
do
    echo "$nome"
done
```

# Strutture di controllo

## Cicli **for**

### Versione completa

#### Sintassi:

```
for ((INITIALIZATION; TEST; STEP))  
do  
    [COMMANDS]  
done
```

#### Esempio:

```
for ((i = 0 ; i <= 1000 ; i++))  
do  
    echo "Counter: $i"  
done
```

**Nota:** sintassi molto simile al C. Importante!

## Strutture di controllo

### Cicli **while**

```
n=0
while ((n<4))
do
    ((n=n+1))
    echo $n
done
```

```
n=0
until((n>4))
do
    ((n=n+1))
done
```

# Esercizi

## Esercizi

Si scriva un programma che riceve due argomenti. Se entrambi sono dei file, stampa il contenuto di entrambi

```
#!/bin/bash
if [ "$#" != "2" ]
then
    echo "Servono due argomenti"
else
    if [[ -f $1 ]] && [[ -f $2 ]]
    then
        cat $1
        cat $2
    else
        echo "Non sono due file"
    fi
fi
```

## Esercizi

Si scriva un programma che per ogni file/cartella nella cartella corrente dice se esso è un file o una cartella.

```
#!/bin/bash
for file in *
do
    if [[ -f $file ]]
    then
        echo "$file è un file"
    elif [[ -d $file ]]
    then
        echo "$file è una cartella"
    fi
done
```



## Esercizi

Si scriva un programma che per ogni file/cartella nella cartella corrente dice se esso è un file o una cartella.

```
#!/bin/bash
for file in *
do
    if [[ -f $file ]]
    then
        echo "$file è un file"
    elif [[ -d $file ]]
    then
        echo "$file è una cartella"
    fi
done
```

## Esercizi

Si scriva un programma che riceve un intero come argomento. Se esso è minore di 10, crei i file `0.txt` , ..., `9.txt`

```
#!/bin/bash
if [ "$#" != "1" ]
then
    echo "Serve un argomento"
else
    if (( $1 < 10 ))
    then
        for (( i=0; i<$1 ; i++))
        do
            touch $i.txt
        done
    else
        echo "L'argomento non è minore di 10"
    fi
fi
```