

Repositories\Laboratorio-Sistemi-Operativi\Shell.md

Gli script shell permettono di creare dei programmi (bash) da eseguire.

Variabili

Le variabili permettono di leggere/salvare dei valori all'interno dello script, non hanno dei tipi. Il loro valore può essere letto ponendo un \$ prima del nome della variabile. L'assegnazione avviene tramite il carattere = (senza spazi).

```
test=100  
echo Il valore è $test
```

Variabili di ambiente

Le variabili di ambiente sono variabili che vengono settate all'avvio del sistema e sono disponibili per tutti i programmi che vengono eseguiti. Esempi di variabili di ambiente sono PATH, HOME, USER, SHELL, PWD, etc.

Variabili speciali

Le variabili speciali sono variabili che vengono settate automaticamente dal sistema operativo e sono disponibili per tutti i programmi che vengono eseguiti.

- **\$0** Nome del programma
- **\$1 - \$9** Argomenti passati al programma
- **\$#** Numero di argomenti passati al programma
- **\$?** Codice di ritorno dell'ultimo comando eseguito
- **\$\$** PID del processo corrente
- **\$RANDOM** Numero casuale che viene generato ad ogni lettura
- **\$LINENO** Numero della linea corrente nel file
- **\$SECONDS** Numero di secondi trascorsi dall'avvio dello script
- **\$*** Tutti gli argomenti passati al programma

Scrittura a stdout

Per scrivere a stdout si può usare il comando echo.

```
echo "Hello World"
```

Lettura da stdin

Per leggere da stdin si può usare il comando `read`. Usando l'opzione `-p` si può specificare un messaggio da mostrare.

```
read -p "Inserisci il tuo nome: " nome
echo "Ciao $nome"
```

Stringhe

Le stringhe possono essere scritte in due modi, in maniera "letterale" e con sostituzione di variabili.

- Usando `' '` il testo all'interno della stringa viene interpretato letteralmente.
- Usando `" "` All'interno del testo vengono sostituite tutte le eventuali variabili presenti con il loro valore.

```
test=100
echo 'Il valore è $test'
echo "Il valore è $test"
```

Command substitution

Il command substitution permette di eseguire un comando e ritornare il suo output. Viene scritto come:

```
$( comando )
```

```
//oppure (sconsigliato)
` comando `
```

```
files=$(ls)
echo $files
echo nesting $(echo $(ls))
```

Arithmetic expansion

L'arithmetic expansion permette di eseguire operazioni aritmetiche e ritornare il risultato. I numeri sono tutti trattati come interi, possono anche essere inserite variabili.

```
$(( operazione ))
```

```
dieci=10
undici=$(( $dieci + 1 ))
echo $undici
```

Export di variabili

Di default le variabili sono locali allo script corrente, per renderle disponibili a tutti i processi figli è necessario esportarle.

```
export variabile=valore
```

```
export test=200
```

let

Il comando let permette di eseguire operazioni aritmetiche su variabili. Permette anche di fare incremento, etc. Possono essere eseguite più operazioni separate da spazi e si possono scrivere all'interno di " "

```
let <codice>
```

```
test=2
dieci=10
let test++
let "test = $test * 2" "test = $test + $dieci"
```

expr

Esegue l'espressione e stampa il risultato, ogni operazione deve essere separata da spazi. expr

```
```bash
expr 5 + 4
expr 5 * 4 //bisogna fare escape del carattere *
```

## doppia parentesi

Permette di eseguire operazioni aritmetiche (come quelle per let) senza valori di ritorno.

```
((codice))
```

```
test=2
dieci=10
((test++))
```

```
((test = test + dieci))
echo $test
```

## command grouping

Codice scritto all'interno di parentesi viene eseguito in un subshell figlia.

```
(codice)

(
 codice
 ...
)

(ls -la)

(
 echo "Hello from the other side"
)
```

Possiamo anche usare le graffe {}, questo verrà eseguito all'interno dello stesso processo

```
{ codice }
{
 codice
 ...
}
```

## array

Gli array sono una collezione di valori. Possono essere definiti in due modi:

- Usando `array=(valore1 valore2 ...)`
- Usando `array[n]=valore`

```
array=(1 2 3 4 5)
echo $array //stampa il primo elemento
echo ${array[0]} //stampa il primo elemento
echo ${array[4]}
```

## Sequenze di comandi

Comandi possono essere eseguiti in sequenza e separati da caratteri speciali che ne cambiano il comportamento. Si possono eseguire anche gruppi di comandi.

- ; Esegue i comandi in sequenza, non termina lo script se uno dei comandi fallisce
- && Esegue i comandi in sequenza, termina lo script se uno dei comandi fallisce
- || Esegue il secondo comando solo se il primo fallisce
- & Esegue il comando in background asincronamente
- | Esegue comandi in sequenza, il risultato del primo comando viene passato al secondo

```
comando1; comando2
comando1 && comando2 && comando3
comando1 || comando2
comando
(
 codice
 ...
) ?
comando1 | comando2
```

## Lunghezza di una variabile

Per ottenere la lunghezza del contenuto di una variabile si può usare il comando #, verrà trattato come stringa e coterà il numero di caratteri

```
a=1000
echo ${#a} //4
b="Hello World"
echo ${#b} //11
```

## test and exit status





Il comando test permette di eseguire condizioni, ritorna 0 se la condizione è vera, 1 se è falsa.

```
test condizione
```

un altro modo per eseguire il test è usare l'operatore [ e ]

```
[condizione]
```

## Condizioni

Le condizioni sono espressioni che possono essere valutate come true o false. Queste sono quelle standard di bash:  Condizioni 1  Condizioni 2  Condizioni 3 Ma esistono anche altre condizioni più "moderne", per poter essere usate bisogna racchiuderle in [[ condizione ]] anzichè [ condizione ]  Condizioni 3

# if

```
if condizione
then
 codice
elif condizione
then
 codice
else
 codice
fi
```

# case

```
case $variabile in
 valore1)
 codice
 ;;
 valore2)
 codice
 ;;
 *)
 codice default
 ;;
esac
```

# while

Ciclo che esegue il codice finchè la condizione è vera

```
while condizione
do
 codice
done
```

# until

Ciclo che esegue il codice finchè la condizione è falsa

```
until condizione
do
 codice
done
```

# for

Ciclo che itera una lista

```
for variabile in lista
do
 codice
done
```

Ex:

```
for file in ./$folder/*
do
 echo $file
done
```

# break & continue

- **break** interrompe il ciclo
- **continue** interrompe l'iterazione corrente e passa alla successiva si può passare un valore numerico per indicare di quanti livelli interrompere il ciclo

```
break numero
continue numero
```

# select

Ciclo che itera una lista all'infinito e permette di scegliere un valore. La variabile indicata sarà quella dove verrà salvato il valore scelto.

```
select variabile in lista
do
 codice
done
```

Ex:

```
select name in "John" "Paul" "George" "Ringo"
do
 echo "Hello $name"
done
```

# function

Per definire una funzione si usa il comando `function` seguito dal nome della funzione e il codice, non ha un valore di ritorno, ma può ritornare uno status code tramite il `return`

```
function nome_funzione {
 codice
 return status_code
}
```

Non potendo avere valore di ritorno, si può o usare una variabile globale, oppure far stampare il risultato della funzione, che verrà poi chiamata tramite `$()`. I parametri della funzione sono salvati in `$1`, `$2`, `$3` (se eseguito tramite `$()`) Ex:

```
function somma {
 echo $(expr $1 + $2)
}
result=$(somma 5 4)
echo $result
```

Le variabili possono essere dichiarate locali o globali, se dichiarate locali non possono essere usate all'esterno della funzione.

```
function somma {
 local a=5
 echo $(expr $a + $2)
}
```

# eval

Esegue il codice passato come parametro

```
eval codice
```

Ex:

```
a="cat file.txt"
eval $a
```

# declare

Permette di dichiarare variabili locali o globali, di impostarne il tipo e altri parametri

- **-a** array
- **-i** intero



- **-r** read only Il contenuto vero e proprio delle variabili rimane sempre una stringa, ma alcuni comandi, sapendo il tipo, possono eseguire operazioni diverse.

declare [parametro] variabile

Ex:

```
declare -a array
declare -i intero
declare -r read_only
array[0]=1
intero=5/2 //senza bisogno di expr o let!
```

## Esempi

Prende tutti i file nella directory che iniziano con "a" o "b" e finiscono con "?", conta all'interno del file quante volte è presente la parola "test", se è maggiore di 2, stampa quanti ne ha trovati, sennò "too few tests"

```
for file in ./[ab]*\?.txt; do
 count=$(egrep -c test $file)
 if [$count -gt 2]
 then
 echo "$file: $count"
 else
 echo "$file: too few tests"
 fi
done
```

Prende il primo parametro passato al comando e calcola il numero di fibonacci su di esso

```
function myfunction {
 if [$1 -gt 1]; then
 ((a=$1-1))
 ((b=$1-2))
 temp1=$(myfunction $a)
 temp2=$(myfunction $b)
 ((res=temp1+temp2))
 else
 if [$1 -eq 1]; then
 res=1
 else
 res=0
 fi
 fi
 echo $res
}
```

```
res=$(myfunction $1)
echo $res
```