La **Parameter Expansion**

La ***parameter expansion*** è un'occorrenza particolare della *variable expansion* utilizzata per valutare **il numero degli argomenti** passati ad un comando/script e **effettuare operazioni** con questi ultimi.

Quando viene eseguito uno script o un comando vi sono delle **variabili d'ambiente** che vengono a crearsi:

- `$#` contiene **il numero di argomenti (aggiuntivi) passati allo script**
- `$*` contiene **gli argomenti passati** concatenati in un'unica stringa.
- `$@` contiene **gli argomenti passati**, ciascuno ***quotato e tenuto separato da*** `""`
  - Vediamo in seguito la vera e propria differenza
- `$n`, con `n` un numero, contiene l'`n`-esimo argomento. `$0` contiene **il nome del processo** avviato.
  I parametri non possono essere modificati una volta che vengono passati a un programma - sono inoltre considerati i parametri **dopo la valutazione di eventuali metacaratteri**.

```
#!/bin/bash
echo "Ho passato $# argomenti."
echo "Gli argomenti sono $*."

# Eseguo lo script
./contaargs.sh ciao 123
Ho passato 2 argomenti.
Gli argomenti sono ciao 123

./contaargs.sh c{an,iald,ald}e
Ho passato 3 argomenti.
Gli argomenti sono cane cialde calde # Metacaratteri valutati prima del
passaggio dei parametri.
```

La differenza principale tra `$*` e `$@` sta nel loro uso quando quotate con `""`.

- `"$*"` restituisce tutti gli argomenti quotati assieme tra `""`.
- `"$@"` restituisce tutti gli argomenti, **ciascuno** quotato tra `""`. Ciò permette di non spezzare gli argomenti che hanno degli spazi al loro interno.

```
$* == $@ -> $1 $2 $3 ...

"$*" == "$1 $2 $3 ..."
"$@" == "$1" "$2" "$3" ...
```

Valutazioni aritmetiche intere

E' possibile far valutare alla *shell* operazioni aritmetiche che utilizzano **numeri interi**; questo avviene tramite l'operatore `(( ))`.

- Usato normalmente (senza `$`) esso racchiude una riga di comando semplice, valutando un espressione e eventualmente effettuando assegnamenti.
- Usato con `$` viene utilizzato per racchiudere **una parte di una riga di comando**, facendo in modo che il suo contenuto venga valutato prima della sua esecuzione.

```
NUM=1
NUM=${NUM}+3 # NUM=1+3 -> l'operazione viene vista come concatenazione di stringhe.
echo ${NUM} # Output: 1+3 (as string).

NUM=1
((NUM=${NUM}+3)) # NUM=4 -> l'operazione viene valutata ARITMETICAMENTE
echo ${NUM} # Output: 4

NUM=1
((NOMEFILE=pippo${NUM}+3)) # NUM=1, NOMEFILE=3 -> Non viene valutato aritmeticamente pippo1 in quanto non è dichiarata; 3 invece viene valutato.
echo ${NOMEFILE} # Output: 3

NUM=1
NOMEFILE=pippo$((${NUM}+3)) # NUM=1, NOMEFILE=pippo4 -> Il pezzo tra $(()) viene valutato aritmeticamente, poiché è possibile.
echo ${NOMEFILE} # Output: pippo4

A=1
$((B=${A}+1)) # Risultato: 2 command not found. L'operazione infatti viene valutata come l'espansione della variabile B, che ha valore 2 dopo la valutazione aritmetica presente.
```

Le valutazioni aritmetiche possono contenere:

- Operatori aritmetici: `+, *, /, -, %`.
- Assegnamenti di variabile (che avvengono se possibile)

- Parentesi tonde: in tal caso esse hanno la precedenza di operazione.
  Le valutazioni aritmetiche sono calcolate tramite una **subshell** e poi il risultato viene rimesso nella shell padre.

Variable expansion diretta e indiretta

Come abbiamo già visto, è possibile utilizzare il carattere `$` per espandere una variabile, sostituendo il blocco con il contenuto di essa. Circondando con le parentesi graffe `{}` si isola il nome della variabile, permettendone l'uso in contesti anche più complessi.

```
A=1
B=2

echo ${A}${B} # Output: 12
echo $A$B # Output: 12, ma meno leggibile.
```

Questo tipo di espansione si chiama **variabile expansion diretta**.
Esiste anche un tipo di **variable expansion indiretta**, specificata tramite `${!NAME}`; essa viene sostituita dal valore di una variabile che ha per valore un'altra variabile.

```
A="Ciao :)"
B=A # B ha come valore A

echo ${!B} # Output: Ciao :) -> Il valore di B viene trattato come la
variabile A stessa a causa del costrutto ${!}
echo ${!A} # Output: nulla -> A non contiene dentro di se il nome di una
variabile valida.
```