



# > Le variabili di shell

Ultima puntata della panormica sulla Programmazione della shell con GNU/Linux



elle prime due puntate di questo mini-corso abbiamo trattato diversi argomenti che ci offrono solide basi per iniziare a programmare con la shell. In questa terza ed ultima puntata analizzeremo le istruzioni iterative, il calcolo aritmetico e le variabili di shell, argomenti che necessitano della lettura delle prime due puntate per essere compresi.

## >> Variabili di shell

Le **variabili di shell** sono delle variabili speciali che la shell crea e utilizza durante il suo funzionamento. Esse sono molto utili poiché consentono al programmatore di **ottenere alcuni dati sul sistema, sull'utente, sul funzionamento della shell** e così via.

### ECCO ALCUNE VARIABILI DEFINITE DIRETTAMENTE DALLA SHELL:

**PPID** Il PID del processo genitore della shell  
**PWD** Il percorso dell'attuale directory di lavoro  
**OLDPWD** Il percorso della precedente directory di lavoro  
**REPLY** L'output del comando read in assenza di input  
**UID** User ID dell'utente corrente  
**EUID** User ID effettivo dell'utente corrente  
**GROUPS** Array dei GID di cui l'utente è membro  
**BASH** Il nome utilizzato per avviare questa istanza di bash  
**BASH\_VERSION** La versione di questa istanza di bash  
**SHLVL** Variabile incrementata di uno per ogni istanza di bash avviata  
**RANDOM** Un numero intero casuale  
**SECONDS** Numero di secondi trascorsi dalla chiamata della shell  
**LINENO** Numero della linea corrente in uno script  
**HISTCMD** Numero del comando corrente nella storia della shell  
**OPTARG** L'ultimo argomento opzione processato da getopt  
**OPTIND** L'indice del prossimo argomento che getopt processerà  
**HOSTTYPE** Il tipo di macchina su cui bash sta girando  
**OSTYPE** Il sistema operativo su cui bash sta girando  
**MACHTYPE** Architettura e sistema operativo della macchina su cui bash sta girando

### ECCO ALCUNE VARIABILI DEFINIBILI, OLTRE CHE DALLA SHELL, ANCHE DALL'UTENTE:

**IFS** Internal Field Separator; viene usato nella suddivisione in parole e di default corrisponde a "<spazio><tab><nuovalinea>"  
**PATH** Percorso in cui la shell cerca i comandi da eseguire  
**HOME** La home directory dell'utente corrente  
**CDPATH** Il percorso di ricerca per il comando CD  
**ENV** File di configurazione per l'esecuzione di script  
**MAIL** File contenente la posta elettronica dell'utente  
**MAILPATH** La directory dei file di posta elettronica dell'utente  
**MAILCHECK** Intervallo tra un controllo della posta ed un altro  
**PS1** L'invito primario  
**PS2** L'invito secondario  
**PS3** L'invito del costrutto select  
**PS4** Parametro usato durante un trace di esecuzione  
**HISTSIZE** Il numero di comandi da memorizzare nella storia della shell  
**HISTFILE** Il nome del file su cui memorizzare la storia della shell  
**HISTFILESIZE** Il numero di linee massime occupabili nel file della storia della shell  
**TMOUT** Tempo di attesa massimo (timeout)  
**PROMPT\_COMMAND** Comando da eseguire prima di mostrare un nuovo prompt

Per maggiori informazioni sulle variabili di shell vi invito a leggere la pagina **man** di **bash**.

## >> Calcolo aritmetico

Come per ogni linguaggio di programmazione che si rispetti, anche la bash offre degli strumenti per effettuare **operazioni aritmetiche di base**. Ecco l'elenco delle operazioni supportate :

? + Meno e più unari  
 ! ~ Negazione logica e "bit a bit"  
 \* / % Moltiplicazione, divisione, modulo (resto)  
 + ? Addizione, sottrazione  
 << > Shift "bit a bit" a sinistra e a destra  
 <= >= < > Confronti  
 == != Uguaglianza e disuguaglianza  
 & AND "bit a bit"  
 ^ OR esclusivo "bit a bit"  
 | OR "bit a bit"  
 && AND logico  
 || OR logico  
 = \*= /= %= += -= ?= <<= >= &= ^= |= Assegnamento

Le **operazioni** si effettuano all'interno di **doppie parentesi tonde** o tra **parentesi quadre** precedute dal simbolo **\$**.

Ecco uno script che costituisce una calcolatrice di base per numeri interi:

```
#!/bin/bash
# Calcolatrice per numeri interi
echo "Calcolatrice per numeri interi"
echo "Operazioni supportate : + - * /"
echo -n "Primo valore: "
read PRIMO_VALORE
echo -n "Secondo valore: "
read SECONDO_VALORE
echo -n "Operazione: "
read OPERAZIONE
case $OPERAZIONE in
    "+") RISULTATO=$((PRIMO_VALORE+$SECONDO_VALORE))
    ;;
    "-") RISULTATO=$((PRIMO_VALORE-$SECONDO_VALORE))
    ;;
    "*") RISULTATO=$((PRIMO_VALORE*$SECONDO_VALORE))
    ;;
    "/") RISULTATO=$((PRIMO_VALORE/$SECONDO_VALORE))
    ;;
    "&& RESTO=$((PRIMO_VALORE%SECONDO_VALORE)) && echo
    "Resto: $RESTO" ;;
    esac
echo "Risultato: $RISULTATO"
```

## >> Istruzioni iterative

Le **istruzioni iterative** costituiscono un particolare tipo di istruzioni che eseguono ciclicamente delle operazioni. Oltre al costutto select che abbiamo analizzato nella puntata precedente, le istruzioni iterative supportate dalla bash sono **for**, **while** e **until**.

L'istruzione **for** esegue una scansione all'interno di una **lista** e **mostra in sequenza** gli argomenti che gli sono stati forniti. Analizziamo insieme questo esempio:

```
#!/bin/bash
# Riproduttore di file Ogg Vorbis
echo "Riproduttore di file Ogg Vorbis"
if [ $# = 0 ]
then
echo "Devi fornire al programma almeno il nome di un
file Ogg Vorbis"
exit
else
for PARAMETRO
do
ogg123 "$PARAMETRO"
done
echo "Riproduzione completata"
fi
```



MID HACKING

In questo caso l'istruzione **for** è stata utilizzata per creare un piccolo **riproduttore di file Ogg Vorbis** servendosi di ogg123. La sintassi di **for** è molto semplice: si definisce una **variabile** che assumerà ogni volta il valore di uno degli argomenti ottenuti attraverso **in** e una **lista** (se omessi, come in questo caso, si fa riferimento a tutti i parametri posizionali a partire dal primo), mentre le operazioni da eseguire vengono racchiuse tra **do** e **done**.

L'istruzione **while** esegue un'operazione fino a quando una condizione risulta sempre vera. Ecco un esempio:

```
#!/bin/bash
# Calcolo la potenza di un numero intero positivo con
while
echo "Programma per il calcolo delle potenze di numeri
interi"
echo -n "Base: "
read BASE
echo -n "Esponente: "
read ESPONENTE
if [ $BASE = "0" ]
then
RISULTATO="0"
elif [ $ESPONENTE = "0" ]
then
RISULTATO="0"
elif [ $BASE = "1" ]
then
RISULTATO=$BASE
elif [ $ESPONENTE = "1" ]
then
RISULTATO=$BASE
else
RISULTATO=$((BASE*BASE))
ESPONENTE=$((ESPONENTE-2))
while [ $ESPONENTE != 0 ]
do
RISULTATO=$((RISULTATO*BASE))
ESPONENTE=$((ESPONENTE-1))
done
fi
echo "Il risultato è: $RISULTATO"
```

La **condizione** va espressa tra **parentesi quadre**, mentre le istruzioni da eseguire vanno racchiuse tra **do** e **done**.

L'istruzione **until** invece svolge il compito opposto di **while**: esegue delle istruzioni fino a quando la condizione espressa risulta sempre falsa. Ecco la parte di codice relativa al calcolo di potenze con **while** riscritta utilizzando **until**:

```
until [ $ESPONENTE = 0 ]
do
RISULTATO=$((RISULTATO*BASE))
ESPONENTE=$((ESPONENTE-1))
done
```

L'istruzione **until** ha la stessa sintassi di **while**.

## >> Conclusione

Il nostro mini-corso di programmazione della shell con GNU/Linux termina qui. Spero di essere stato abbastanza chiaro nell'esposizione dei concetti e di avervi fornito tutti gli elementi necessari per realizzare solidi script di shell.

Per dubbi, suggerimenti e critiche potete inviarmi una e-mail all'indirizzo specificato in basso.

Per apprendere ulteriori nozioni relative alla programmazione di shell con GNU/Linux potete consultare il box Documentazione presente in quest'articolo.

Vi auguro una felice programmazione con GNU/Linux :) 📧

ptips