Le funzioni in Bash

Le funzioni in Bash sono un concetto fondamentale per scrivere script efficienti e modulari. Esse permettono di riutilizzare codice, ridurre la ripetizione e migliorare la manutenibilità degli script. In questo articolo esploreremo cosa sono le funzioni in Bash, come si definiscono, quali sono le loro caratteristiche principali, e come utilizzarle in modo efficace.

Una funzione in Bash è un blocco di codice che può essere definito una sola volta e chiamato più volte durante l'esecuzione di uno script. Le funzioni sono utili per raggruppare comandi che svolgono un compito specifico, riducendo la necessità di ripetere lo stesso codice in più punti.

In Bash, una funzione può essere definita usando la seguente sintassi:

```
function my_func {
    # Comandi
}
```

Oppure, in una forma più compatta:

```
my_func() {
    # Comandi
}
```

Entrambe le sintassi sono equivalenti. Ecco un esempio semplice:

```
greet() {
    echo "Hi, $1!"
}
```

Questa funzione, `greet`, prende un argomento e stampa un messaggio di saluto. `$1` si riferisce al primo argomento passato alla funzione.

Una volta definita, una funzione può essere chiamata semplicemente utilizzando il suo nome:

```
greet "There"
```

L'output sarà:

```
Hi, There!
```

Gli argomenti possono essere passati a una funzione nello stesso modo in cui vengono passati agli script. Gli argomenti sono accessibili all'interno della funzione tramite `$1`, `$2`, `$3`, e così via. Se vuoi accedere a tutti gli argomenti passati, puoi usare `$@` o `$*`.

Esempio:

```
multiply() {
    result=$(($1 * $2))
    echo "The result is: $result"
}
```

```
multiply 3 4
```

Output:

```
The result is: 12
```

In Bash, una funzione può restituire un valore tramite il comando `return`. Tuttavia, il valore restituito può essere solo un numero intero compreso tra 0 e 255. Per restituire valori più complessi (come stringhe), si può usare `echo` o assegnare il risultato a una variabile.

Esempio di ritorno di un codice di stato:

```
check_file() {
    if [ -e "$1" ]; then
        return 0
    else
        return 1
    fi
}

check_file "test.txt"

if [ $? -eq 0 ]; then
    echo "File exists."
else
    echo "File does not exist."
fi
```

Le variabili dichiarate all'interno di una funzione in Bash hanno uno scopo globale per default. Questo significa che se una variabile è modificata

all'interno di una funzione, il cambiamento influenzerà anche il contesto esterno alla funzione. Per evitare questo, si possono usare le variabili locali.

Esempio:

```
my_script() {
    local local_var="Hello"
    echo "$local_var"
}

my_script
```

Qui, `local_var` è visibile solo all'interno della funzione `my_script`.

Vediamo come combinare più funzioni in uno script per svolgere un'attività più complessa. Supponiamo di voler creare uno script che gestisce un semplice sistema di log.

```
#!/bin/bash

log() {
    local level="$1"
    local message="$2"
    local date_time=$(date +"%Y-%m-%d %H:%M:%S")
    echo "[$date_time] [$level] $message"
}

info() {
    log "INFO" "$1"
}

error() {
    log "ERROR" "$1"
}

cmd() {
    local command="$1"
```

```
    if $command; then
        info "$command executed successfully."
    else
        error "$command failed to execute."
    fi
}

# Uso delle funzioni
cmd "ls /non_existing"
cmd "echo 'This works'"
```

Questo script definisce tre funzioni: `log`, `info`, `error` e `cmd`. La funzione `log` centralizza la gestione del formato del messaggio di log. Le funzioni `info` ed `error` sono semplicemente delle scorciatoie per loggare rispettivamente messaggi informativi e di errore. La funzione `cmd` esegue un comando e registra un messaggio di successo o di errore a seconda del risultato.

Conclusione

Le funzioni in Bash sono uno strumento potente per migliorare l'organizzazione e la manutenibilità degli script. Con una corretta comprensione delle loro caratteristiche e un uso appropriato, si possono scrivere script Bash più modulabili, riutilizzabili e facili da gestire. Sebbene le funzioni di Bash siano semplici rispetto a quelle in linguaggi di programmazione più complessi, esse offrono abbastanza flessibilità per coprire una vasta gamma di esigenze di scripting.