Utilizzare getopts in Bash per gestire opzioni e argomenti

Quando si creano script Bash complessi, spesso è utile fornire agli utenti la possibilità di passare opzioni e argomenti da riga di comando, come si vede comunemente nei programmi Unix. Per gestire tali opzioni, uno strumento utile è il comando getopts, che permette di analizzare in modo semplice e robusto le opzioni fornite a uno script. In questo articolo, vedremo come funziona getopts e come utilizzarlo per gestire vari tipi di opzioni e argomenti.

`getopts` è un built-in di Bash progettato per analizzare le opzioni di uno script, in modo simile a come i comandi Unix (come `ls` o `cp`) accettano opzioni precedute da trattini (`-` o `--`). Permette di specificare opzioni singole o multiple, opzioni con argomenti e opzioni obbligatorie o facoltative.

A differenza di `getopt` (una versione più vecchia), `getopts` è specifico per shell POSIX, e funziona internamente in Bash in modo più sicuro e portabile.

La sintassi generale di `getopts` è la seguente:

```
getopts "opzioni" variabile
```

- **opzioni**: una stringa che definisce le opzioni supportate dallo script. Le lettere rappresentano le opzioni accettate. Se un'opzione richiede un argomento, la lettera sarà seguita da due punti (`:`).
- **variabile**: la variabile in cui verrà memorizzata l'opzione corrente durante l'analisi.

Consideriamo un esempio semplice in cui il nostro script accetta le opzioni `-a` e `-b`, dove `-b` richiede un argomento:

```
#!/bin/bash

while getopts "ab:" opt; do
  case $opt in
    a)
      echo "Opzione -a attivata"
      ;;
    b)
      echo "Opzione -b con argomento '$OPTARG'"
      ;;
    \?)
      echo "Opzione non valida: -$OPTARG" >&2
      ;;
  esac
done
```

Spiegazione del codice:

1. **"ab:"**: L'opzione `a` non richiede un argomento, mentre `b` richiede un argomento (indicato dal `:` dopo la `b`).
2. **$opt**: Il valore dell'opzione corrente viene memorizzato in questa variabile.
3. **$OPTARG**: Se un'opzione richiede un argomento, il valore dell'argomento viene memorizzato in questa variabile.
4. **while**: Il ciclo `while` esegue l'analisi delle opzioni fino a quando tutte le opzioni fornite sono state processate.
5. **case**: Utilizziamo un costrutto `case` per gestire le diverse opzioni.

In Bash è possibile passare più opzioni insieme, ad esempio `-a -b valore` può essere scritto come `-ab valore`. `getopts` gestisce questo comportamento automaticamente.

È buona norma fornire un'opzione `-h` o `--help` per spiegare il funzionamento dello script. Aggiungiamo questa opzione al nostro esempio:

```
#!/bin/bash

usage() {
  echo "Uso: $0 [-a] [-b argomento] [-h]" 1>&2
  exit 1
}

while getopts "ab:h" opt; do
  case $opt in
    a)
      echo "Opzione -a attivata"
      ;;
    b)
      echo "Opzione -b con argomento '$OPTARG'"
      ;;
    h)
      usage
      ;;
    \?)
      usage
      ;;
  esac
done
```

L'opzione `-h` richiama la funzione `usage` che stampa una breve guida e termina lo script.

Può essere necessario assicurarsi che alcune opzioni siano obbligatorie. Possiamo aggiungere una logica che controlla se l'opzione obbligatoria è stata fornita:

```
#!/bin/bash
```

```
b_flag=false

while getopts "ab:" opt; do
  case $opt in
    a)
      echo "Opzione -a attivata"
      ;;
    b)
      b_flag=true
      echo "Opzione -b con argomento '$OPTARG'"
      ;;
    \?)
      echo "Opzione non valida" >&2
      exit 1
      ;;
  esac
done

# Verifica che l'opzione -b sia stata fornita
if [ "$b_flag" = false ]; then
  echo "L'opzione -b è obbligatoria" >&2
  exit 1
fi
```

Se lo script viene eseguito senza l'opzione `-b`, verrà visualizzato un messaggio di errore.

Conclusione

L'utilizzo di `getopts` in Bash permette di gestire le opzioni di uno script in modo flessibile ed efficiente. Abbiamo visto come analizzare opzioni semplici, opzioni con argomenti, gestire l'opzione di aiuto e verificare la presenza di opzioni obbligatorie. Questi sono solo alcuni dei casi d'uso possibili; la combinazione di `getopts` con altre caratteristiche di Bash permette di creare script robusti e user-friendly.