Bash Scripting

Ripasso linguaggio

Uno script può ricevere parametri in input. Ad esempio:

```
~$./mio_script.sh 37 121
```

esegue lo scrip `mio_script.sh` passando allo stesso i parametri [ ] e [ ].
All'interno dello script è possibile riferirsi al primo parametro tramite la sintassi [ ] e al secondo tramite la sintassi [ ]. Un eventuale terzo parametro sarebbe riferito tramite [ ] e così via. Si noti che il parametro (implicito) $0 si riferisce al [ ], cioè [ ] nel nostro esempio.

In alternativa, è possibile usare l'istruzione [ ]. Si noti che è possibile usare l'opzione -p per stampare contestualmente un messaggio. Il seguente script legge in input un parametro stringa, lo registra nella variabile str e lo stampa in output:

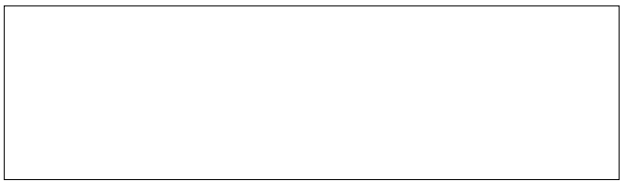
[ ]

È possibile assegnare a una variabile un comando attraverso la sintassi [ ]. Per esempio, [ ] assegna a myvar il comando ls /etc | wc -l.h.
In modo simile è possibile assegnare a una variabile il risultato di un'operazione aritmetica tramite la sintassi con doppia tonda [ ]. Per assegnare a var1 il risultato dell'operazione 3+5, si scrive [ ].

La sintassi dell'istruzione di selezione è la seguente:

[ ]

In caso di assenza del ramo else si ha:



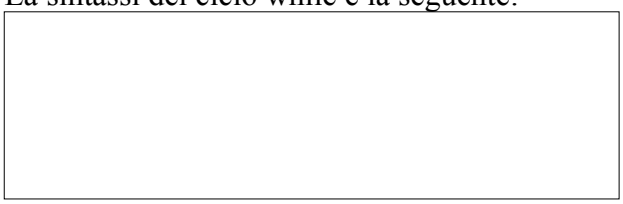
È possibile annidare le istruzioni di selezione o usare la sintassi:

```
  if [ <test> ]
  then
    <commands>
  elif [ <test> ]
  then
    <different commands>
  else
  <other commands>
fi
```

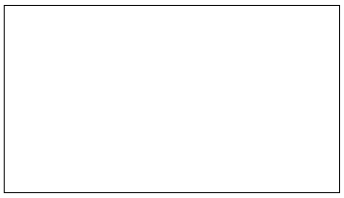
Per quanto riguarda il test della condizione, fare riferimento al comando [ ] di Unix/Linux. Per eseguire un confronto di uguaglianza tra i valori delle variabili intere var0 e var1 si scrive [ ]. Si noti che l'operatore == del comando test effettua il confronto per uguaglianza tra [ ] e non tra variabili intere.
È possibile usare anche gli operatori booleani && (AND) e || (OR). Per testare che i file file0.txt e file1.txt esistano si può scrivere [ ].

In Bash esiste anche il comando test avanzato [ ] che permette di usare gli operatori aritmetici, di confronto e logici in stile C/C++. Il test [$var0 -eq $var1] tra variabili intere si può scrivere più semplicemente [ ].

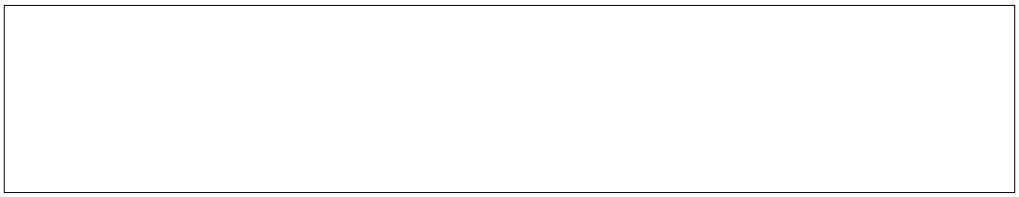
La sintassi del ciclo while è la seguente:



Mentre per il for abbiamo sintassi alternative:

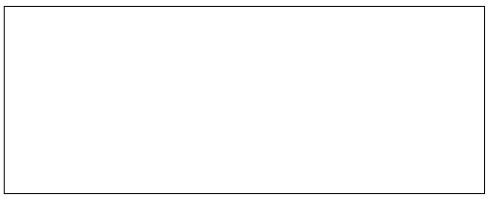


e

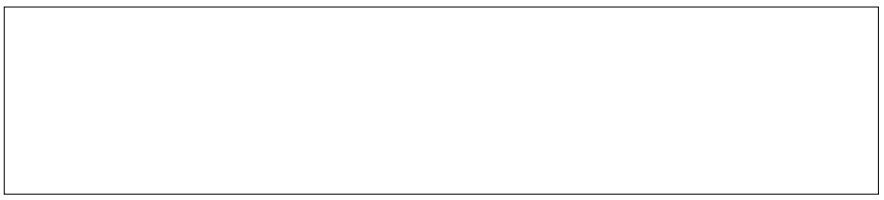


Per esempio, con riferimento alla prima sintassi, possiamo scrivere:

per stampare i numeri da 1 a 5. Lo stesso risultato si ottiene se la lista è definita tramite un intervallo:



Con riferimento alla sintassi stile C/C++ possiamo scrivere, ad esempio:



per stampare i numeri da 0 a 20 a intervalli di 2.

Gli array si possono definire utilizzando la sintassi  , come nel seguente esempio:

v=(2 4 22 55 7 9 3 77)

L'accesso agli elementi dell'array avviene tramite [ ], da [ ] a [ ].
Nell'esempio precedente l'array ha 8 elementi e gli elementi sono v[0], v[1], …, v[7].
È possibile aggiungere un elemento semplicemente scrivendo [ ], così come è possibile eliminare elementi dall'array tramite il comando unset (per esempio, `unset array[1]` elimina l'elemento di indice 1, cioè 4).

La dimensione dell'array è data dalla scrittura [ ].
Tutti gli elementi dell'array sono dati da [ ].

Le funzioni hanno la seguente sintassi:

[ ]

Per esempio, la funzione

```
printarray()
{
  local -n w=${1:-}
  echo "array     : ${w[@]}"
  echo "dimensione: ${#w[@]}"
}
```

riceve un array in input e ne stampa il contenuto e la dimensione.
Se si desidera passare l'array v alla funzione, l'invocazione della stessa sarà:

[ ]

e v diventa il parametro numero 1 per lo script, al quale ci si potrà riferire con la sintassi $1 o [ ] in caso di array, esattamente come avviene quando si invoca uno script. Il meccanismo è lo stesso.

Se v fosse passato come secondo parametro, si potrebbe accedere agli elementi tramite la sintassi [ ]

Nell'esempio si è usato il comando local con l'opzione -n per creare un [ ] all'array passato come argomento. Senza l'opzione -n si sarebbe creata una [ ] dell'array. Senza il comando local si sarebbe creato un riferimeno o una copia globale dell'array.