Shell scripting

Shell scripting

- la shell è uno strumento potente per l'esecuzione di comandi arbitrariamente complessi
- ciò nonostante, per automatizzare l'esecuzione di compiti lunghi e ripetitivi, ogni shell fornisce un proprio linguaggio di programmazione *(shell script)* che permette l'esecuzione di programmi *(script)* memorizzati in file eseguibili testuali
- caratteristiche comuni di questi linguaggi:
  - Turing completi
  - approccio procedurale
  - non tipati (l'unico tipo disponibile è la stringa)
  - interpretati
- studieremo il linguaggio della shell Bourne Again (bash)

Script

- uno *script* (non necessariamente di shell) è un programma intelleggibile ad un interprete
  - quando viene richiesta l'esecuzione di un file eseguibile, la shell legge i primi caratteri del file, il file è considerato script se la prima riga ha la forma **`#!path [arg ...]`**
    - “shebang line”
    - per eseguire uno script la shell invoca l'interprete appendendo alla sua lista di parametri posizionali il nome dello script
  - e.g. “cat-script”:

```
#!/bin/cat
contenuto dello script
```

- un *bash script* è un programma interpretabile da bash, e.g.:

```
#!/bin/bash
echo “Hello, world!”
```

Input da stdin

- è possibile leggere da standard input utilizzando il comando interno “read”
- sintassi: *read <nome1> <nome2> ... <nomeN>*
- legge una riga da standard input, la spezza ai caratteri di $IFS, ed associa ogni componente ad una delle variabili specificate
  - nel caso vi siano più componenti che variabili, l'ultima variabile contiene anche le rimanenti
- e.g., script inverti

```
#!/bin/bash
read nome cognome
echo “Dott. $cognome, $nome”
```

Variabili predefinite

- bash offre un vasto insieme di variabili locali e d'ambiente predefinite, ne riportiamo alcune:
- locali
  - $@, $*, $*n*, $# accesso ai parametri a riga di comando
  - $$ PID della shell
  - $- flag associati alla shell corrente
- di ambiente
  - $IFS Internal Field Separator
  - $PS1, $PS2 prompt

Matematica

- è possibile effettuare computazioni aritmetiche in due modi
  - utilizzando l'eseguibile /usr/bin/expr
    - attenzione al quoting!
  - utilizzando l'espansione aritmetica $((...)) della shell
- per effettuare computazioni matematiche in virgola mobile o reali è necessario fare ricorso a programmi esterni (e.g. bc)

Verifica di predicati booleani (1/2)

- il comando interno test permette di valutare predicati e varia il suo valore di ritorno in base alla loro veridicità
- alcuni predicati
  - su file
    - esistenza di un file regolare — test -f *nome*
    - esistenza di una directory — test -d *nome*
    - leggibilità di un file — test -r *nome*
    - “più nuovo di” — test *file1* -nt *file2*
  - su stringhe
    - è la stringa vuota — test -z *stringa*
    - uguaglianza di stringhe — test *stringa1* = *stringa2*

Verifica di predicati booleani (2/2)

- alcuni predicati
  - su numeri interi
    - sintassi generica — test *num1 op num2*
    - op è uno tra: -lt (less than), -le (less or equal), -eq (equal), -ge (greater or equal), -gt (greater than)
  - connettivi logici:
    - and — test *pred1* -a *pred2*
    - or — test *pred1* -o *pred2*
    - not — test ! *pred*
- sintassi concisa di test: [*predicato*]
  - e.g. test $x -ge 10 = [$x -ge 10]

Controllo condizionale

- la bash dispone dell'usuale costrutto if ... then ... else
  - la guardia è un comando, il ramo then viene eseguito se il suo return code è 0, altrimenti viene eseguito il ramo else
- la guardia è tipicamente (ma non necessariamente!) il comando test, nella sua sintassi concisa, e.g.:

```
echo "inserisci numero"
read number
if [ $number -lt 0 ]; then
  echo negative
elif [ $number -eq 0 ]; then
  echo zero
else
  echo positive
fi
```

Pattern matching

- è disponibile un costrutto di pattern matching su stringhe, i pattern sono simili a quelli disponibili per l'espansione delle wildcard, e.g.:

```
case "$1" in
start)
  start-stop-daemon --start ...
  ;;
reload | force-reload)
  ;;
*)
  exit 1
  ;;
esac
```

Cicli limitati

- è disponibile un costrutto di ciclo limitato che permette di iterare il valore di una variabile su un insieme finito di stringhe, e.g.:

```
for dirname in /tmp /var/tmp; do
  rm -f $dirname/*~
done
```

- la semantica usuale del ciclo for delimitato da interi si ottiene utilizzando for in congiunzione con il programma seq e l'espansione dei comandi, e.g.:

```
test -d tests/ && exit 1
mkdir tests/
for i in `seq 1 100`; do
  echo "test_$i" > tests/$i.txt
done
```

Cicli illimitati

- è disponibile un costrutto di ciclo non limitato che permette di ciclare fintanto che l'esecuzione di un comando ha return code 0:

```
while read line; do
  echo $line
done
```

```
x=76
while [ $x -gt 57 ];do
   x=`expr $x - 1`
done
```

- è disponibile anche la versione until che permette di ciclare fintanto che l'esecuzione di un comando ha return code diverso da 0:

```
until false; do
  echo “guru meditation”
done
```

Gestione dei segnali

- è possibile installare presso la shell in esecuzione dei comandi di callback che verranno eseguiti al verificarsi di eventi
  - sono eventi: le ricezioni di segnali e la terminazione della shell
- per installare callback si utilizza il comando interno trap
- e.g.:

```
set -e
tmp=`tempfile`
trap "rm $tmp" EXIT
...
```

Shell function

- è possibile definire funzioni (*shell functions*) utilizzando la sintassi: *nome-funzione* () { *comandi* }
- una volta definite, le funzioni possono essere invocate come normali comandi di shell, passando loro parametri posizionali
  - il loro corpo viene eseguito con un nuovo insieme di parametri posizionali

```
hello ()
{
  echo "Hello, $1 World!"
}
hello "Functions"
```

Shell scripting engineering

- per aumentare la mantenibilità di shell script di dimensioni considerevoli è possibile dividerli in più file
- utilizzando il comando . (punto) o source, è possibile interpretare il contenuto di script esterni
  - è quindi possibile organizzare gli script in librerie di funzioni invocabili esternamente

Debugging

- ad ogni shell è associato un insieme di flag, identificati da una singola lettera, modificabili attraverso il comando interno set con la sintassi
  - set -*flag* attiva un flag
  - set +*flag* disattiva un flag
- due flag sono di particolare utilità per il debugging di script
  - flag “e”, se settato impone alla shell di uscire non appena un comando ritorna un return code diverso da 0
  - flag “x”, se settato la shell stampa su stdout i comandi e i loro argomenti prima di eseguirli

Esercizio

- implementare, utilizzando bash script, una utility **junk**
- synopsis:
  - junk filename ...
  - junk -l
  - junk -p
- quando invocata su una lista di file, sposta questi file in una directory “.junk”, nella home dell'utente (se la directory non esiste, la crea automaticamente)
- quando invocata con -l, mostra il contenuto di .junk/ (segnalando eventualmente che non esiste)
- quando invocata con -p, svuota la directory .junk (chiedendo conferma interattivamente all'utente)