Lezione 4: Controllo del flusso e costrutti avanzati

Laboratorio di Elementi di Architettura e Sistemi Operativi

28 Marzo 2012

Parte 1: Soluzione degli esercizi

Statistiche

- Ho ricevuto 21 soluzioni
- Le correzioni non sono ancora pronte, saranno pubblicate quanto prima
- Le soluzioni complete sono disponibili sulla pagina del corso

Esercizio 2

Creare una sottodirectory `bin` all'interno della propria home directory in cui mettere gli script. Fare in modo che gli script contenuti in `bin` possano essere invocati da qualunque directory con il nome del file, senza dover specificare l'intero percorso.

```
$ cd
$ mdkir bin
$ export PATH=$PATH:$HOME/bin
```

- L'ultimo comando *aggiunge* al path la directory `$HOME:/bin`.
- *Il comando `export PATH=$HOME/bin` non è corretto*
- Alla chiusura della shell il `PATH` ritorna al valore di sistema.
- Per rendere il cambiamento permanente si deve aggiungere `export PATH=$PATH:$HOME/bin` al file `.profile`.

Esercizio 8

Creare uno script che accetti due parametri: `estensione` e `stringa`. Lo script deve elencare tutti i file presenti ricorsivamente nella cartella corrente la cui `estensione` è uguale al primo argomento e che contengono almeno un'occorrenza della `stringa` passata come secondo argomento.

```
#!/bin/bash
ext=$1
string=$2
listafile=$(find . -type f -name "*."$ext 2>/dev/null)
grep -l $string $listafile
```

Le soluzioni basate su `ls -R` o `grep -R` *non sono corrette*:

- `ls -R` non ritorna il percorso completo

- `grep -R` si aspetta un elenco di directory e poi cerca ricorsivamente *tra tutti i file*, senza filtrarli per estensione
- ricordate che i metacaratteri vengono espansi dalla shell *prima* di eseguire il comando: `grep -Rlwi $string *.$ext` cerca tra i file con estensione `$ext` nella sola cartella corrente!

Parte 2: controllo del flusso negli script

Strutture condizionali

Il costrutto `if-then-else` esegue comandi diversi a seconda del valore di verità di una o più condizioni:

```
if [ condizione ]
then
  comando1
  comando2
  ...
fi
```

```
if [ condizione1 ]
then
  comando1
  ...
elif [ condizione2 ]
then
  comando2
  ...
else
  comando3
  ...
fi
```

- Le parentesi `[ ]` che racchiudono la condizione sono un'abbreviazione del comando `test`, che può essere usato al loro posto
- *Attenzione:*
  - agli spazi tra le parentesi e l'operatore di confronto
  - al `$` davanti alla variabile che si usa nel test

Esempio:

```
if [ $a = 0 ]; then
  echo $a
fi

if test $a = 0; then
  echo $a
fi
```

Il comando `test`

Sintassi: *`test expression`*
restituisce exit status 0 se `expression` è vera, 1 altrimenti.

Espressioni che controllano se un file possiede certi attributi:

`-e file` vera se `file` esiste;

`-f file` vera se `file` è un file normale;

`-d dir` vera se `dir` è una directory;

`-r file` vera se `file` è leggibile dall'utente;

`-w file` vera se `file` è scrivibile dall'utente;

`-x file` vera se `file` è eseguibile dall'utente;

Espressioni su stringhe:

`-z str` vera se `str` è di lunghezza zero;

`-n str` vera se `str` non è di lunghezza zero;

`str1 = str2` vera se `str1` è uguale a `str2`;

`str1 != str2` vera se `str1` è diversa da `str2`;

Espressioni su valori numerici:

`num1 -eq num2` vera se `num1` è uguale a `num2`;

`num1 -ne num2` vera se `num1` è diverso sa `num2`;

`num1 -lt num2` vera se `num1` è minore di `num2`;

`num1 -gt num2` vera se `num1` è maggiore di `num2`;

`num1 -le num2` vera se `num1` è minore o uguale a `num2`;

`num1 -ge num2` vera se `num1` è maggiore o uguale a `num2`.

Espressioni composte:

`exp1 -a exp2` vera se sono vere sia `exp1` che `exp2`

`exp1 -o exp2` vera se è vera `exp1` o se è vera `exp2`

`!exp` vera se `exp` è falsa

Esempi:

```
if [ n -lt 0 ] ; then
  echo $n " è un numero negativo"
elif [ n -gt 0 ] ; then
  echo $n " è un numero positivo"
else
  echo $n "  è uguale a zero"
fi
```

```
if [ -e "$HOME/.bash_profile" ]; then
  echo "you have a .bash_profile file";
else
  echo "you have no .bash_profile file";
fi
```

```
if [ -e "$HOME/.bash_profile" -o -e "$HOME/.bashrc" ]
then
  echo "you have a bash config file";
else
  echo "you have no bash config file";
fi
```

Selezione per casi

Sintassi:

```
case stringa in
espressione_1)
  comandi_1
  ...
  ;;
espressione_2)
  comandi_2
  ...
  ;;
...
*)
  comandi_default
  ...
  ;;
esac
```

- Vengono eseguiti i comandi `commandi_1`, `commandi_2`, ...a seconda del fatto che string sia uguale a `espressione_1`, `espressione_2`,...
- I comandi `comandi_default` vengono eseguiti soltanto se il valore di `string` non coincide con nessuna delle espressioni.
- Nelle espressioni si possono usare i metacaratteri `*` e `?`.

Esempio:

```
echo "Premere un tasto, quindi premere INVIO."
read Tasto
case $Tasto in
  [a-z])
    echo "Hai premuto una lettera." ;;
  [0-9])
    echo "Hai premuto un numero." ;;
  * )
    echo "Hai premuto un altro tasto." ;;
esac
```

Costrutti iterativi - il ciclo `for`

- Ciclo for

  ```
  for variabile in lista
  do
    comandi
  done
  ```

- `lista` può essere:
  - un elenco di valori
  - una variabile (corrispondente ad una lista di valori)
  - un meta-carattere che può espandersi in una lista di valori
  - l'ouput di un comando (usando l'operatore `$( comando )`
- In assenza della clausola `in`, il for opera su `$@`, cioè la lista degli argomenti

Esempi:

```
for file in *
do
  ls -l  $file
```

```
done

*****************************************

LIMIT=10
# il comando seq genera sequenze di numeri
for a in $(seq $LIMIT)
do
  echo $a
done
```

Costrutti iterativi - il ciclo `while`

- Ciclo while

```
while [ condizione_di_permanenza_ciclo ]
do
  comandi
done
```

- La parte tra `[ ]` indica l'utilizzo del comando `test` (come per if)

Esempio:

```
LIMIT=10
a=1
while [ $a -le $LIMIT ]
do
  echo $a
  let "a+=1"
done
```

Un esempio per mettere tutto assieme

Esercizio. *Scrivere uno script che prende due parametri: una directory e la dimensione di un file in byte. Lo script deve fornire il nome e la dimensione di tutti i file contenuti nella directory parametro ai quali avete accesso e che sono più piccoli della dimensione data. Si controlli che i parametri passati sulla linea di comando siano due e che il primo sia una directory.*

Esempio di soluzione:

```
#!/bin/bash
# prima parte: controllo dei parametri
if [ $# -ne 2 ] ; then
  echo "uso: listfile.sh <directory> <dimensione>"
  exit 1
fi
if [ ! -d "$1" ] ; then
  echo "ERRORE: " $1 " non è una directory!"
  exit 1
fi

# seconda parte: esecuzione del compito
for i in $1/*
do
  if [ -r "$i" -a -f "$i" ]
  then
```

```
    size=$(wc -c < "$i")
    if [ $size -lt $2 ]
    then
      echo $(basename "$i") è di dimensione $size byte
    fi
  fi
done

exit 0
```