Laboratorio di Programmazione 1

Docente: dr. Damiano Macedonio

Lezione 2 – 10/10/2012

Lezione 3 – 17/10/2012

Bash: Il Linguaggio di Script

- La shell bash non è solo un interprete di comandi è anche un *linguaggio di script.*
- Attraverso il linguaggio di script è possibile automatizzare una serie di attività complesse che richiederebbero un certo numero di comandi.
- Gli script vengono interpretati (non compilati) ed eseguiti una istruzione alla volta.
- Altri linguaggi di script: Perl, Phyton, PHP, JavaScript, Ruby, ecc...

Bash: Il Linguaggio di Script

- Per scrivere uno script bash si può usare un qualsiasi editor di testo: vi, emacs, gedit, ecc…

- Ogni script bash deve iniziare con la riga:

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

  che indica al sistema operativo di utilizzare l'interprete Bash per eseguire lo script.

- Per poter eseguire uno script si devono avere i permessi di esecuzione (*x*) sul file:

  ```
  $ chmod u+x hello.sh
  $ ls –l hello.sh
  -rwx------ hello.sh
  $ ./hello.sh
  ```

Bash: Il Primo Script

- Script che sposta tutti i files di testo (*.txt*) contenuti nella directory corrente in una nuova directory *texts*, e poi cancella la directory *texts* con tutto il suo contenuto.
  - `mkdir texts`
  - `mv *.txt texts`
  - `rm -r texts`

- Invece di essere costretti a digitare tutti questi comandi in modo interattivo nella shell, scriviamo uno script:

```
#!/bin/bash
# this script deletes all text files
mkdir texts
mv *.txt texts
rm -r texts
echo “Deleted all text files!”
```

Bash: Le Variabili

- In uno script bash si possono utilizzare delle *variabili*.
- Le variabili sono sempre memorizzate come stringhe, ma ci sono delle strutture nel linguaggio che convertono le variabili in numeri per fare i calcoli.
- Le variabili non devono essere dichiarate, per crearle basta as

Senza spazi!

```
#!/bin/bash

str=“hello world!”
echo $str
```

`str` indica il nome e `$str` indica il valore

Bash: Le Virgolette

- Per assegnare ad una variabile un valore contenente degli spazi o caratteri speciali, tale valore deve essere racchiuso tra virgolette (singole o doppie).

- Le *virgolette doppie* (partial quoting) permettono di risolvere le variabili al loro interno: ogni variabile nella stringa viene sostituita dal valore che essa assume in quel momento.

```
var="stringa di test"
newvar="Il valore di var è $var."
echo $newvar
  Il valore di var è stringa di test.
```

- Le *virgolette singole* (full quoting) non permettono la risoluzione delle variabili al loro interno.

```
var="stringa di test"
newvar='Il valore di var è $var.'
echo $newvar
  Il valore di var è $var.
```

Bash: Il Carattere di Escape

- Il carattere di escape \ conserva il valore del carattere letterale che segue:
  - `ls *a`

    `*a` indica tutte le sequenze di caratteri seguite da `a`
  - `ls \*a`

    `\*a` indica la sequenza di caratteri: `*a`

- Utile ad esempio per utilizzare delle virgolette all'interno di una stringa.
  - `Var=“ stringa di \“test\” ”`
  - `echo $var`

    `stringa di “test”`

Bash: Il Comando Read

- Il comando **`read`** consente di leggere l'input da tastiera e memorizzarlo in una variabile.

```
#!/bin/bash
echo -n “Inserire il nome del file da eliminare: ”
read file
rm -i $file
```

- Il comando `read` legge il nome del file da eliminare e lo memorizza nella varibile `file`.

- Il valore della variabile `file` viene usato dall'ultimo comando ($ per estrarre il valore).

Bash: Il Comando Read

- Alcune opzioni interessanti:
  - `read –s`

    Non stampa a video l'input.
  - `read –nN`

    Accetta soltanto *N* caratteri di input.
  - `read –p "message"`

    Visualizza il messaggio *message.*
  - `read –tT`

    Attende l'input per *T* secondi.

- `read –s –n1 -p "Yes (Y) or not (N)?" answer`

  `Yes (Y) or not (N)?` (il valore digitato non viene visualizzato)

Bash: Sostituzione dei Comandi

- Il carattere $ può essere usato anche per sostituire dei comandi con il loro output.

```
$(command)
```

- Esempi:

```
list=$(ls)
echo $list
  hello.sh read.sh

rm $(find -name “*.tmp”)

ls $(pwd)
```

Bash: Operatori Aritmetici

- Classici operatori aritmetici:
  - `+  somma`
  - `-  sottrazione`
  - `*  moltiplicazione`
  - `/  divisione`
  - `** esponenziazione`
  - `%  modulo (ovvero: resto della divisione)`
- Un'espressione aritmetica può essere valutata usando `$[expr]` o `$((expr))`
  - `echo $((123+20))`
    `143`
  - `val=$[123+20];echo $[10*$val]`
    `1430`
- Il costrutto `let` può essere usato per associare un'espressione ad una variabile.
  - `let x=10+2*7`
  - `echo $x`
    `24`

Bash: Esercizio 1

- Creare uno script bash che richiede due numeri ed esegue la loro: somma, sottrazione, moltiplicazione e divisione (quoziente e resto), stampando poi i risultati ottenuti.

- Elementi utili:
  - Comando `echo`
  - Comando `read`
  - Operatori aritmetici
  - Variabili

Bash: Esercizio 1 (Soluzione)

```
#!/bin/bash
echo -n "Digitare il primo numero: "
read x
echo -n "Digitare il secondo numero: "
read y
add=$(($x+$y))
sub=$(($x-$y))
mul=$(($x*$y))
div=$(($x/$y))
mod=$(($x%$y))
echo "Somma: $add"
echo "Differenza: $sub"
echo "Prodotto: $mul"
echo "Quoziente: $div"
echo "Resto: $mod"
```

Bash: Costrutti Condizionali

- Un costrutto condizionale permette di decidere quale azione eseguire, a seconda del valore assunto da un'espressione.

- Il costrutto condizionale fondamentale è chiamato if.

```
if [ expression ];    # Attenzione agli spazi prima di []
then
  statements
elif [ expression ];  # Attenzione agli spazi prima di []
then
  statements
else
  statements
fi
```

- Le parti `elif` (else if) e `else` sono opzionali.

Bash: Espressioni

- Un'espressione [*expression*] può essere:
  - Un confronto fra stringhe.
  - Un confronto fra numeri.
  - Un operatore su file.
  - Un operatore logico.

- Operatori su stringhe:
  - s1 = s2

    Valuta se due stringhe sono uguali.
  - s1 != s2

    Valuta se due stringhe sono diverse.
  - -n s1

    Valuta se la lunghezza di s1 è maggiore di zero.
  - -z s1

    Valuta se la lunghezza di s1 è uguale a zero.

va uno spazio prima e dopo l'operatore binario.

Bash: Espressioni

- Operatori su numeri:
  - n1 –eq n2

    Valuta se due numeri *n1* e *n2* sono uguali.
  - n1 –ge n2

    Valuta se il numero *n1* è maggiore o uguale al numero *n2*.
  - n1 –le n2

    Valuta se il numero *n1* è minore o uguale al numero *n2*.
  - n1 –ne n2

    Valuta se due numeri *n1* e *n2* sono diversi.
  - n1 –gt n2

    Valuta se il numero *n1* è strettamente maggiore di *n2*.
  - n1 –lt n2

    Valuta se il numero *n1* è strettamente minore di *n2*.

Bash: Espressioni

- Operatori su file:
  - `-d` path

    Controlla se il percorso (o file) dato rappresenta una directory.
  - `-f` path

    Controlla se il percorso (o file) dato rappresenta un file ordinario.
  - `-e` file

    Controlla se il nome del file esiste.
  - `-s` file

    Controlla se un file ha lunghezza maggiore di 0.
  - `-r` file

    Controlla se l'utente possiede il permesso di lettura per file (directory).
  - `-w` file

    Controlla se l'utente ha il permesso di scrittura per un file (directory).
  - `-x` file

    Controlla se l'utente ha il permesso di esecuzione per un file (directory).

Bash: Espressioni

- Operatori logici:
  - `!` expr

    Negazione (NOT) un’espressione logica.

  - expr1 `-a` expr2

    Congiunzione (AND) tra due espressioni logiche.

  - expr1 `–o` expr2

    Disgiunzione (OR) tra due espressioni logiche.

Bash: Esercizio 2

- Scrivere uno script bash che richiede il proprio nome utente e lo confronta con il valore della variabile d'ambiente LOGNAME. Se i due valori sono uguali stampa un messaggio di benvenuto, altrimenti stampa un messaggio di errore.

- Elementi utili:
  - Comando `echo`
  - Comando `read`
  - Operatori su stringhe
  - Costrutto condizionale

Bash: Esercizio 2 (Soluzione)

```
#!/bin/bash
echo -n "Inserire lo username: "
read name
if [ $name = $LOGNAME ];
  then
  echo "Bentornato $name."
else
  echo "Utente non riconosciuto."
fi
```

Bash: Esercizio 3

- Scrivere uno script bash che richiede un numero compreso tra 1 e 10. Se il numero inserito è corretto viene stampato il suo doppio, altrimenti viene visualizzato un messaggio di errore.

- Elementi utili:
  - Comando `echo`
  - Comando `read`
  - Operatori aritmetici
  - Operatori su numeri
  - Operatori logici (alt.)
  - Costrutto condizionale

Bash: Esercizio 3 (Soluzione)

```
#!/bin/bash
echo -n "Inserire un numero 1 < x < 10:"
read num
if [ $num -lt 10 ];
  then
  if [ $num -gt 1 ];
    then
    echo "$num*2=$(($num*2))"
  else
    echo "Inserimento sbagliato!"
  fi
else
  echo "Inserimento sbagliato!"
fi
```

Bash: Esercizio 3 (Soluzione Alt)

```
#!/bin/bash
echo -n "Inserire un numero 1 < x < 10:"
read num
if [$num -gt 1 -a $num -lt 10];
then
  echo "$num*2=$(($num*2))"
else
  echo "Inserimento sbagliato!"
fi
```

Bash: Esercizio 4

- Scrivere uno script bash che:
  - Controlla l'esistenza di una directory "backup", se non esiste la crea.
  - Richiede un nome di file all'utente.
  - Se il file esiste ne crea una copia in backup aggiungendo l'estensione .bak, altrimenti visualizza un messaggio di errore.

- Elementi utili:
  - Operatori su file
  - Operatore condizionale
  - Comandi `cp`, `mkdir`, ecc…

Bash: Esercizio 4 (Soluzione)

```
#!/bin/bash
directory=./backup
if [ -d $directory ];
  then
  echo "La directory esiste"
else
  mkdir ./backup
fi
echo "Inserisci il nome di un file"
read file
if [ -f $file ];
  then
  cp $file ./backup/$file.bak
else
  echo "Il file $file non esiste"
fi
```

Bash: Parametri

- È possibile passare degli argomenti ad uno script bash.

- Gli argomenti sono chiamati parametri posizionali, poichè possono essere riferiti attraverso la loro posizione.
  - Il primo argomento corrisponde alla variabile 1, ecc.
  - Il parametro posizionale *n* può essere referenziato come `${n}`, o come `$n` quando *n* consiste in una singola cifra.

- Parametri speciali
  - `$#` è il numero dei parametri passati
  - `$0` ritorna il nome dello script in esecuzione
  - `$*` ritorna una singola stringa contenente tutti i parametri passati allo script

Bash: Esercizio 5

- Scrivere uno script bash analogo all'esercizio 1 (operazioni su numeri) tranne che i due valori vengono passati come parametri dello script.

- Se il numero di parametri passati è diverso da 2 viene visualizzato un errore.

Bash: Esercizio 5 (Soluzione)

```
#!/bin/bash
if [ $# -ne 2 ];
  then
  echo "Specificare due valori"
else
  x=$1
  y=$2
  add=$(($x+$y))
  sub=$(($x-$y))
  mul=$(($x*$y))
  div=$(($x/$y))
  mod=$(($x%$y))
  echo "Somma: $add"
  echo "Differenza: $sub"
  echo "Prodotto: $mul"
  echo "Quoziente: $div"
  echo "Resto: $mod"
fi
```

Bash: Il Costrutto Case

- Il costrutto `case` viene usato per decidere l'azione da eseguire in base ad un determinato valore.
- Può essere utilizzato al posto del costrutto `if` se il numero di condizioni è elevato.
- Il valore usato può essere un'espressione.
- Ciascun insieme di azioni relativo ad un valore deve terminare con un doppio punto e virgola `;;`

```
case $var in
val1)
statements;;
val2)
statements;;
*)
statements;;
esac
```

- Il valore `*)` indica qualsiasi valore diverso dai precedenti.

Bash: Esercizio 6

- Scrivere uno script bash che richiede di digitare un valore compreso tra 1 e 10 ed utilizza il comando case per stampare a parole il valore digitato (es. Se viene immesso 5, “Il valore immesso è cinque”). Se il valore non sta nell’intervallo richiesto viene stampato un messaggio di errore.

- Elementi utili:
  - Costrutto `case`
  - Comando `read`
  - Comando `echo`

Bash: Esercizio 6 (Soluzione)

```
#!/bin/bash
echo -n "Digitare un numero tra 1 e 10: "
read x
case $x in
 1) echo "Il valore di x e' uno.";;
 2) echo "Il valore di x e' due.";;
 3) echo "Il valore di x e' tre.";;
 4) echo "Il valore di x e' quattro.";;
 5) echo "Il valore di x e' cinque.";;
 6) echo "Il valore di x e' sei.";;
 7) echo "Il valore di x e' sette.";;
 8) echo "Il valore di x e' otto.";;
 9) echo "Il valore di x e' nove.";;
 10) echo "Il valore di x e' dieci.";;
 *) echo "Valore fuori dall\'intervallo consentito";;
esac
```

Bash: Costrutto di Iterazione For

- Il costrutto di iterazione (ciclo) `for` è utilizzato quando si vuole ripetere un gruppo di comandi per ogni elemento contenuto in un elenco.

```
for var in lista di valori
  do
    istruzioni
  done
```

- Ad ogni iterazione la variabile `var` assume un valore nella *`lista di valori`*.

```
#!/bin/bash
sum=0
for num in 1 2 3 4 5
Do
   sum=$(($sum+$num))
done
echo $sum
```

Lista implicita

```
#!/bin/bash
list="1 2 3 4 5"
sum=0
for num in $list
Do
   sum=$(($sum+$num))
done
echo $sum
```

Creo una lista

Bash: Costrutto di Iterazione For

- Se la parte `list` viene omessa, a `var` viene assegnato ciascun parametro passato allo script ($1, $2, $3,...).

```
#!/bin/bash
for x
do
 echo “The value of variable x is: $x”
done
```

```
script.sh laboratorio delta
The value of variable x is: laboratorio
The value of variable x is: delta
```

Bash: Esercizio 7

- Scrivere uno script bash che crea una directory, e crea all'interno di essa 5 file il cui nome è file_*x*, dove *x* è un valore progressivo.

- Elementi utili :
  - Comando `mkdir`, `cd`, `touch`
  - Ciclo `for`

Bash: Esercizio 7 (Soluzione)

```
#!/bin/bash
mkdir tmp
cd tmp
lista="1 2 3 4 5"
for x in $lista
  do
  touch file_$x
done
```

Bash: Esercizio 8

- Scrivere uno script bash che conta quanti files sono contenuti nella directory corrente.

- Elementi utili:
  - Ciclo `for`
  - Comando `ls`
  - `Costrutto condizionale`
  - Operatore -f.

Bash: Esercizio 8 (Soluzione)

```
#!/bin/bash

list=$(ls)
count=0
for x in $list
do
  if [ -f $x ];
    then
    count=$(($count + 1))
  fi
done
echo "In questa cartella ci sono $count file"
```

Bash: Esercizio 9

- Scrivere uno script bash che sposta i file passati come argomento su una directory *old* all'interno della vostra home.
  - Verifica che almeno un argomento sia stato fornito, altrimenti stampare un messaggio di errore.
  - Se la directory *old* non esiste, la crea.
    - Usare la variabile d'ambiente $HOME per recurera la vostra home directory.
  - Stampa un messaggio prima di effettuare lo spostamento, contenente la lista dei file che verranno spostati.
  - Sposta i file specificati nella directory *old.*

Bash: Esercizio 9 (Soluzione)

```
#!/bin/bash
if [ $# -eq 0 ]
  then
  echo "Indicare almeno un file da spostare."
fi
directory="$HOME/old"
if [ -d $directory ];
  then
  echo "La directory esiste"
else
  mkdir $directory
fi
echo "File da spostare nella directory old:"
echo $*
for p in $*
  do
  mv $p $directory
done
ls -l $directory
```

Bash: Esercizio 9 (Alternativa)

```
#!/bin/bash

if [ $# -eq 0 ]
  then
  echo inserire almeno un file da spostare
 else
  if [ ! -d $HOME/old ];
    then mkdir $HOME/old
  fi
  echo "sto spostando i seguenti files: $*"
  for x in $*
    do
      mv $x $HOME/old
    done
 fi
```

Bash: Ciclo For Alternativo

- Esiste una forma alternativa di ciclo *for* nella quale si deve specificare un valore iniziale per la variabile, una condizione di uscita ed una espressione di incremento:

```
for (( EXPR1 ; EXPR2 ; EXPR3 ))
do
  istruzioni
done
```

- Alla prima iterazione viene valutata l'espressione EXPR1 che assegna un valore ad una variabile, quindi ad ogni iterazione viene valutata l'espressione EXPR2, il ciclo viene eseguito finché EXPR2 risulta vera. Al termine di ogni iterazione viene valutata anche EXPR3 per modificare il valore della variabile.

Bash: Ciclo For Alternativo

```
#!/bin/bash
echo -n “Inserire un numero: ”
read x
sum=0
for (( i=1 ; $i<=$x ; i=$i+1 ))
  do
     sum =$(($sum+$i))
  done
echo “La somma dei primi $x numeri è: $sum”
```

Se vogliamo la somma dei primi $x numeri dobbiamo mettere <=

Bash: Esercizio 10

- Scrivere uno script bash che crea una directory, richiede un numero *n* all'utente, e crea all'interno della directory creata *n* file il cui nome è file_*x*, dove *x* è un valore progressivo.

- Simile all'esercizio 7, ma usando un ciclo for alternativo.

Bash: Esercizio 10 (Soluzione)

```
#!/bin/bash
mkdir tmp
cd tmp
echo -n "Digitare un numero tra 1 <= x <= 10:"
read x
echo $x
for (( i=1 ; $i<=$x ; i=$i+1 ))
  do
  touch file_$i
done
```

Bash: Il Costrutto While

- Il costrutto *while* permette di creare dei cicli che vengono eseguiti finché una certa condizione specificata è vera.
  - Il ciclo termina appena la condizione diventa falsa.

```
while  [ espressione ]
  do
  istruzioni
  done
```

- Se la condizione non diventa mai falsa, il ciclo non termina.

```
#!/bin/bash
echo –n “Inserire un numero: “; read x
let sum=0; let i=1
while [ $i –le $x ]; do
  let sum=$sum+$i
  let i=$i+1
done
echo “La somma dei primi $x numeri è: $sum”
```

Per mettere più comandi sulla stessa riga, separateli con ;

Bash: Esercizio 11

- Modificare lo script dell’esercizio 10 per utilizzare un ciclo *while* al posto del ciclo for.

Bash: Esercizio 11 (Soluzione)

```
#!/bin/bash
mkdir tmp
cd tmp
echo -n "Digitare un numero tra 1 <= x <= 10:"
read x
i=1
while [ $i -le $x ];
  do
  touch file_$i
  i=$(( $i+1 ))
done
```

Bash: Le Funzioni

- Le funzioni permettono di isolare una parte di un programma, permettendone il riutilizzo.
- Ogni funzione ha un nome che serve per richiarmarla all'interno di uno script.
- Una funzione può restituire un valore attraverso il comando `return`.

```
#!/bin/bash
hello()
{
  echo “Stai eseguendo la funzione hello()”
}

  hello
```

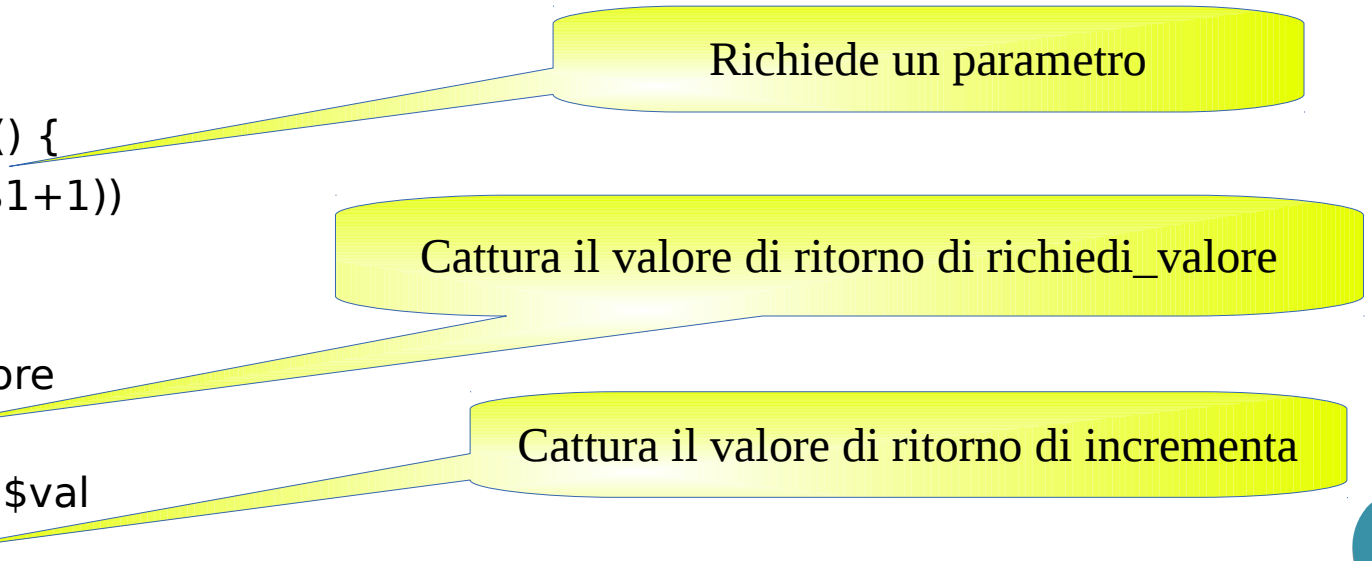
Le Funzioni: un esempio

```
#!/bin/bash

richiedi_valore() {
  echo -n "si inserisca un valore: "
  read x
  return $x
}

Incrementa() {
  return $(($1+1))
}

richiedi_valore
val=$?
incrementa $val
suc=$?
echo "il valore successivo è: $suc"
```



Bash: Esercizio 12

- Scrivere uno script bash che calcola la somma di due numeri passati come parametro. Isolare l'operazione di somma e il controllo sul numero di parametri all'interno di una funzione.

Bash: Esercizio 12 (Soluzione)

```
#!/bin/bash
check() {
if [ $# -ne 2 ];
  then
  return 0
else
  return 1
fi
}

sum() {
  return $(( $1+$2 ))
}

check $*
c=$?
if [ $c -eq 0 ];
  then
  echo "Specificare due valori come parametro."
else
  sum $*
  echo $?
fi
```