

# Script Bash

# Strutture di controllo

**if (condition) then-body**  
**[else else-body]**

**while (condition)**  
**body**

**do**  
**body**

**while (condition)**

**for (initialization; condition;**  
**increment)**  
**body**

**switch (expression) {**  
**case value or regular**  
**expression:**  
**case-body**  
**default:**  
**default-body**

# Exit status

- Ogni comando/programma restituisce un intero detto exit status al chiamante
- in C, e' l'intero restituito dalla funzione main
- Di norma,
  - 0 = terminazione regolare
  - diverso da zero = terminazione irregolare
- La variabile di shell “?” contiene l'exit status dell'ultimo comando eseguito
  - Provare a eseguire un comando qualsiasi, e poi “echo \$?”

# Operatori su comandi

- `cmd1; cmd2`
  - esegue `cmd1` seguito da `cmd2`
- `cmd1 && cmd2`
  - esegue `cmd1`; poi, esegue `cmd2` se `cmd1` è terminato con successo (`exit(cmd1) = 0`)
- `cmd1 || cmd2`
  - esegue `cmd1`; poi, esegue `cmd2` se `cmd1` è terminato con errore (`exit(cmd1) != 0`)
- in tutti e tre i casi, l'exit status complessivo è quello dell'ultimo comando eseguito

# Condizionale

```
if comando  
then  
    lista comandi  
[elif comando  
    lista comandi]  
[else  
    lista comandi]  
fi
```

- come test si usa l'exit status di un comando
- per mettere if e then sulla stessa linea, usare “;”

# Espressioni Condizionali

- il costrutto [ exp ] valuta exp come espressione condizionale (abbrevia **test** exp )
  - cioè, termina con exit status 0 se exp è vera
  - gli spazi dopo [ e prima di ] sono obbligatori
- operatori ammessi:
  - su stringhe: ==, !=, -z (lunghezza nulla)
  - operatori binari su interi: -lt, -le, -eq, -ne, -ge, -gt
  - operatori unari su file: -e, -f, -r, -w, -x
- per informazioni on-line: “man test”

# Espressioni Condizionali

```
if [ -z "$1" ] % se lunghezza nulla di $1
then
    echo "Questo script richiede un argomento."
    exit 1
fi
```

```
if [ $# -lt 4 ] % se num. argomenti < 4
then
    echo "Questo script richiede 4 argomenti."
    exit 1
elif [ ! -e "$1" ] % se non esiste $1
then
    echo "Il file $1 non esiste."
    exit 1
fi
```

# Espressioni Aritmetiche `$( ( ... ) )`

- `$( ( exp ) )` (spazi obbligatori) valuta `exp` come espressione aritmetica
- `$( ( exp ) )` viene sostituito dalla shell con il valore di `exp`
- Esempi: sia “a” una variabile con valore 7

espressione	sostituita con	note
<code>\$( ( \$a+1 ) )</code>	8	
<code>\$( ( \$a++ ) )</code>	8	“a” viene incrementata
<code>\$( ( \$a*3 &gt; 8 ) )</code>	1	1 equivale a “vero”

# ciclo *while*

```
while comando  
do  
    lista comandi  
done
```

Esempio:

```
i=0  
while [ $i -lt 10 ]  
do  
    i=$(( i+1 ))  
done
```

ripete la lista di comandi fintantoché  
il comando viene eseguito con successo  
(come in C)

# Esempio

Script “init” che genera 30file vuoti  
denominati:node1.html, node2.html, ...,node30.html

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

```
i=0
```

```
while [ $i -lt 30 ]
```

```
do
```

```
    i=$(( i+1 ))
```

```
    touch node$i.html
```

```
done
```

# Ciclo *for*

```
for var in lista valori  
do  
    lista comandi  
done
```

- *lista valori* è come una lista di argomenti passata a un comando
- Esempi:
  - for a in 1 2 3
  - for a in \$(ls)
  - for a in “uno” “due” “tre” (diverso da for a in “uno due tre”)
  - for a in “\$@” (diverso da for a in “\$”)
  - for a in \*.txt

# Esempio

Creare uno script “addhead” che in ogni file nodeXXX.html inserisca la stringa

```
</TITLE></HEAD>
```

```
<BODY><H1>nodoxxx.html</H1><BR>
```

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

```
for i in `ls | grep html`
```

```
do
```

```
    echo "<head><title>${i}</title></head><body><h1>${i}</h1></body>" > ${i}
```

```
done
```

# Costrutto case

**case** stringa in

stringa caso 1) lista di comandi 1 ;;

stringa caso 2) lista di comandi 2 ;;

...

**esac**

Se stringa è uguale a stringa caso 1, allora viene eseguita lista di comandi 1 ed esce dal costrutto; altrimenti lista di comandi 1 non viene eseguita, e passa ad elaborare in modo analogo il caso successivo.

Poiché \* rappresenta una stringa qualunque, essa può essere utilizzata per rappresentare “tutti gli altri casi”.

# Esempio

**case** \$word in

hello) echo English ;;

howdy) echo American ;;

gday) echo Australian ;;

bonjour) echo French ;;

"guten tag") echo German ;;

\*) echo Unknown Language: \$word ;;

**esac**

# Script interattivi

- E' possibile creare degli script interattivi grazie all'uso del comando **read**
- Attende l'inserimento di una linea di caratteri da parte dell'utente e assegna la stringa corrispondente ad una variabile di shell.

```
> cat pappagallo
#!/bin/bash
echo "Dimmi qualcosa:"
read cosa
echo "Ti faccio l'eco: $cosa"
```

```
> ./pappagallo
Dimmi qualcosa:
qualcosa
Ti faccio l'eco: qualcosa
```

# Funzioni

- Le **funzioni di shell**, come gli script, sono costituite da sequenze di istruzioni scritte nel linguaggio della shell utilizzata.
- Se uno script viene eseguito richiamando il file dove è memorizzato, l'esecuzione di funzione richiede una **chiamata alla funzione** tramite il nome assegnatole.
- Una funzione può essere definita nello stesso file relativo allo script che la richiama, oppure in un file separato.

# Esempio

```
#!/bin/bash
conferma() { # inizio del corpo della funzione
    echo "Sei sicuro? (S)i/(N)o? [S] "
    read answer
    case $answer in
    s|S|"")
        return 0
        ;;
    n|N)
        return 1
        ;;
    *)
        conferma # richiede in caso di risposta sbagliata
        ;;
    esac
}
# fine del corpo della funzione
if conferma then
    echo "L'utente ha detto SI"
else
    echo "L'utente ha detto NO"
fi
```

```
> ./conferma.sh
Sei sicuro? (S)i/(N)o? [S]
F
Sei sicuro? (S)i/(N)o? [S]
s
L'utente ha detto SI
```

# Esempio

```
#!/bin/sh
```

```
factorial()
```

```
{
```

```
  if [ $1 -gt 1 ]; then
```

```
    i=$(( $1 - 1 ))
```

```
    j=`factorial $i`
```

```
    k=`expr $1 \* $j`
```

```
    echo $k
```

```
  else
```

```
    echo 1
```

```
  fi
```

```
}
```

```
while true
```

```
do
```

```
  echo "Enter a number:"
```

```
  read x
```

```
  factorial $x
```

```
done
```

# Riferimenti

- [Bash Guide for Beginners](#)