- Script shell -

Script

Uno script di shell BASH è un file di testo che inizia con:

`#!/bin/bash`

e che ha il permesso di esecuzione

Il resto del file contiene comandi di shell

Non c'è differenza tra quello che si può scrivere al prompt e quello che si può scrivere in uno script

Il mio primo script

1. Con un editor di testi (nano, pico, kate, emacs, vi) creare un file con il seguente contenuto:

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

1. Salvarlo col nome “mio_script”
2. Dargli permessi di esecuzione
3. Eseguirlo digitando “./mio_script”

Perché `#!/bin/bash` ?

I primi due caratteri dicono a bash che il file è uno script

Il resto dice a bash qual è l'interprete per questo script

Risultato: viene invocato l'interprete passandogli come argomento il nome dello script

Provare con:

```
#!/bin/echo
```

```
#!/bin/cat
```

Variabili predefinite negli script

- $0 il nome dello script stesso (argv[0])
- $1… $9 parametri da riga di comando (argv[i])
- $# numero di parametri ricevuti (argc)

- $* tutti i parametri in una stringa singola
- $@ tutti i parametri in stringhe separate

- $! process ID (PID) del processo corrente
- $? *exit status* dell'ultimo comando eseguito

Esercizio *eccho*

Scrivere uno script “**eccho**” che prende un argomento e lo stampa due volte

Esempio: ./`eccho prova`

scrive due volte “prova”

Esercizio *bis*

Scrivere uno script “**bis**” che prende un comando come argomento e lo esegue due volte

Esempio: ./`bis ls -l`

esegue due volte “ls -l”

Esercizio *swap*

Realizzare lo script “**`swap`** `file1 file2`”: dati come argomenti due file esistenti, scambia i loro nomi.

L'*exit status*

- Ogni comando restituisce un intero detto *exit status* al chiamante
- In C, è 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

Comando *if*

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

Come test usa l'*exit status* del comando

Per mettere `if` e `then` sulla stessa linea, usare “;”

Espressioni condizionali

Il comando “test exp” valuta exp come espressione condizionale

- cioè, termina con exit status 0 se exp è vera
- “test exp” si può abbreviare “[exp]” (spazi obbligatori)

Operatori ammessi:

- su stringhe: ==, !=, -z
- su interi: -lt, -le, -eq, -ne, -ge, -gt
- operatori unari su nomi di file: -e, -f, -r, -w, -x

Per informazioni: “man test”

Espressioni condizionali

```
if [ -z “$1” ]
then
        echo “Questo script richiede un argomento.”
        exit 1
fi
```

```
if [ $# -lt 4 ]
then
        echo “Questo script richiede 4 argomenti.”
        exit 1
elif [ ! -e “$1” ]
then
        echo “Il file $1 non esiste.”
        exit 1
fi
```

Esercizio *swap* (miglioramento)

Migliorare swap in modo che:

- controlli di aver ricevuto 2 argomenti
- se ha ricevuto meno o più di 2 argomenti, segnali l'errore ed esca
- se ha ricevuto due argomenti, ma almeno uno dei due file non esiste, segnali l'errore ed esca

Sostituzione aritmetica con $((...))

- $((*exp*)) valuta *exp* come espressione aritmetica
- $((*exp*)) viene sostituito dalla shell con il valore di *exp*
- Solo aritmetica su numeri interi

Esempi: sia “a” una variabile con valore 7

espressione	sostituita con	note
$(($a+1))	8	
$((a+1))	8	
$((a++))	7	“a” viene incrementata
$((a*3 > 8))	1	1 equivale a “vero”

Sostituzione aritmetica

Operatori:

- aritmetici: +, -, /, *, %
- elevamento a potenza: **
- bit-a-bit: <<, >>, &, |, ~
- booleani: <, <=, ==, !=, >, >=, &&, ||, !

Come si usa un'espressione aritmetica come espressione condizionale?

Ciclo *while*

```
while comando
do
    sequenza
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)

Ciclo *while*

Esempio:

```
while true
do
     echo “Inserisci il nome di un file \
   da visualizzare (q per uscire):”
     read nome_file
     if [ nome_file == “q” ]
     then
          break
     else
          cat $nome_file
     fi
done
```

Esercizio while -

- Si realizzi uno script "**scriviNumeri.sh**" che scrive a video i numeri da 0 a N: **0,1,2,…….,N-1**

  Il valore di N viene passato allo script da riga di comando.
- Esempio di lancio: $ ./scriviNumeri.sh N

Soluzione

```
#!/bin/bash
COUNTER=0
while [  $COUNTER -lt $1 ];
do
    echo il valore di counter è $COUNTER
     COUNTER=$((COUNTER+1))
done
```

Esercizio - while -

- Si realizzi uno script che chiameremo “**creaFiles.sh**” che genera n file vuoti denominati:

  node1.html, **node2.html**,…**nodeN.html**

  nella directory di lancio. Il valore di N viene passato allo script da riga di comando.

- Esempio di lancio:

  $ ./creaFiles.sh N

Soluzione Esercizio – while -

```
#!/bin/bash
if  test $# -ne 1
then
   echo “Wrong number of parameters $#”
   echo “Usage: $0 param”
fi

i=0
   while [ $i -lt $1 ];
   do
        i=$((i+1))
        touch node$i.html
   done
```

Inizializzazione della variabile di ciclo

Entra nel ciclo fintanto che la variabile i è minore di n (less than)

Creazione file

Ciclo *for*

```
for var in lista valori
do
    sequenza 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

Differenza tra $* e $@

file *prova:*

```
#!/bin/bash
for x in “$*”
do
    echo "ecco $x"
done

echo ‘Ora con $@’

for x in "$@"
do
    echo "ecco $x"
done
```

```
> prova 1 2 3
ecco 1 2 3
Ora con $@
ecco 1
ecco 2
ecco 3
```

Esercizio

- Scrivere uno script shell (shell program) di nome **cercaFileReg** che, nella directory corrente, (di lancio) crea un file di nome **fileReg** contenente l'elenco di tutti i file regolari.

  Nota: (Creare una sottodirectory **bin** all'interno della propria work directory in cui mettere gli script)

  Suggerimenti:

  Usare $(comando) per assegnare il risultato del comando lista

Esempio di lancio dello script:

$ chmod +x cercaFileReg.sh (permessi per esecuzione)

$ ./cercaFileReg.sh

Soluzione Esercizio

Script Shell:

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

La prima linea dello script deve iniziare con #!, che indica al kernel che lo script è direttamente eseguibile, poi nome dell'interprete dei comandi shell (Bourne again shell).

```
for file in $(ls)
do
if [ -f $file ]
then
   echo $file >> fileReg
fi
done
```

fornisce il contenuto della nostra directory

variabile da testare

vero se il file esiste ed è un file regolare

espressione condizionale if

Per effettuare un ciclo tra una lista di valori di tipo stringa si può usare il comando for