

- 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”

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 $\$((\dots))$

- $\$((exp))$ 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	
$\$((a++))$ incrementata	7	“a” viene
$\$((a*3 > 8))$	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 -

- ④ 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.
- ④ 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
```

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
```

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”.

Case

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
```

- UNTIL-

- Ciclo **until** esegue la lista di comandi finchè la condizione è falsa

```
until condition;  
do  
    comandi  
done
```

- Alcuni test relativi alle proprietà dei file :
 - **-e** file esiste
 - **-d** file directory
 - **-f** il file esiste ed è regolare

Esercizio n° 12 - until -

- Ⓞ Si realizzi uno script che chiameremo "scriviNumeri.sh" che scrive a video i numeri da 20 a 10: **20,19,18,.....,10**
- Ⓞ Esempio di lancio: \$./scriviNumeri.sh

Soluzione

```
#!/bin/bash
COUNTER=20
until [ $COUNTER -lt 10 ];
do
    echo COUNTER: $COUNTER
    COUNTER=$((COUNTER-1))
done
```