*Corso di Laurea in Informatica A.A. 2023-2024*

# Laboratorio di Sistemi Operativi

Alessandra Rossi

# 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

# 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”

# Perchè # ! /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

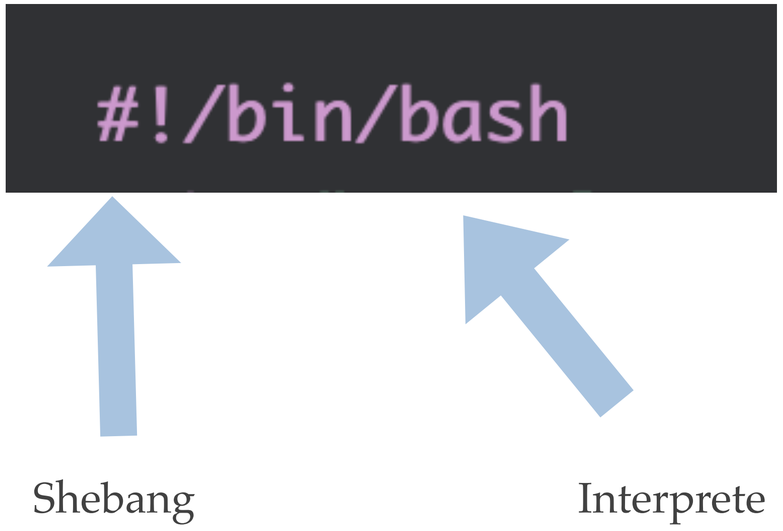
Provare con:

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

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

# Script

La Bash è la shell (ovvero l'interfaccia testuale) più diffusa e utilizzata in ambiente Linux




```
./hello_world.sh
```

```
bash hello_world.sh
```

# Variabili definite

- $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

# Variabili definite

- **$0** - The name of the Bash script.
- **$1 - $9** - The first 9 arguments to the Bash script. (As mentioned above.)
- **$#** - How many arguments were passed to the Bash script.
- **$@** - All the arguments supplied to the Bash script.
- **$?** - The exit status of the most recently run process.
- **$$** - The process ID of the current script.
- **$USER** - The username of the user running the script.
- **$HOSTNAME** - The hostname of the machine the script is running on.
- **$SECONDS** - The number of seconds since the script was started.
- **$RANDOM** - Returns a different random number each time is it referred to.
- **$LINENO** - Returns the current line number in the Bash script.

# Standards I/O & Error

- **STDIN** - /proc/<processID>/fd/0
- **STDOUT** - /proc/<processID>/fd/1
- **STDERR** - /proc/<processID>/fd/2

- **STDIN** - /dev/stdin or /proc/self/fd/0
- **STDOUT** - /dev/stdout or /proc/self/fd/1
- **STDERR** - /dev/stderr or /proc/self/fd/2

# Esercizio

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

Esempio: ./`eccho prova`

scrive due volte “prova”

# Esercizio

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

Esempio: ./`bis ls -l`

esegue due volte “ls -l”

# Exit

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

# Il comando if

Come test usa l'*exit status* del comando

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

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

# 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”

# Esempio

```
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
```

# Esempio

```
if [ <some test> ]
then
    <commands>
fi
```

```
if [ <some test> ]
then
    <commands>
elif [ <some test> ]
then
    <different commands>
else
    <other commands>
fi
```

```
#!/bin/bash
# Basic if statement

if [ $1 -gt 100 ]
then
   echo Hey that\'s a large number.
   pwd
fi

date
```

# Sostituzione

$((*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?

# Sostituzione aritmetica

| Operator | Operation |
|---|---|
| +, -, *, / | addition, subtraction, multiply, divide |
| var++ | Increase the variable var by 1 |
| var-- | Decrease the variable var by 1 |
| % | Modulus (Return the remainder after division) |

**let expression**
Make a variable equal to an expression.

**expr expression**
print out the result of the expression.

**$(( expression ))**
Return the result of the expression.

**${#var}**
Return the length of the variable var.

```
let a=5+4
echo $a # 9

let "a = 5 + 4"
echo $a # 9

let a++
echo $a # 10

let "a = 4 * 5"
echo $a # 20

let "a = $1 + 30"
echo $a # 30 + first command line argument
```

# Ciclo While

## Ciclo *while*

```
while comando
do
    sequenza
comandi
done
```

**Esempio:**

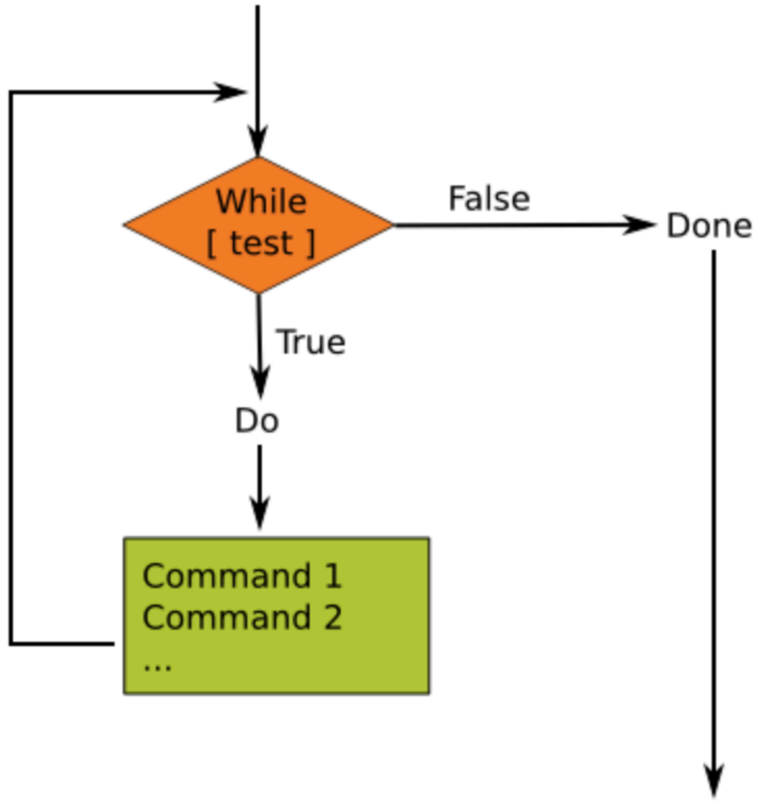
```
i=0
while [ $i -lt 10 ]
do
     i=$(( $i+1 ))
done
```

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

# Esempio

```
while [ <some test> ]
do
    <commands>
done
```

```
until [ <some test> ]
do
    <commands>
done
```



# Esempio

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

break

cat $nome_file
```

# Esercizio

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

# Esercizio: Soluzione

**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
```

# Ciclo For

```
for var in <list>
do
    <commands>
done
```

```
#!/bin/bash
# Make a backup set of files

for value in $1/*
do
   used=$( df $1 | tail -1 | awk '{ print $5 }' | sed 's/%//' )
   if [ $used -gt 90 ]
   then
      echo Low disk space 1>&2
      break
   fi
   cp $value $1/backup/
done
```

```
#!/bin/bash
# Make a backup set of files

for value in $1/*
do
   if [ ! -r $value ]
   then
      echo $value not readable 1>&2
      continue
   fi
   cp $value $1/backup/
done
```

# Il 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.

```
> cut 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

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

**Soluzione**

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

# Read more

https://diraimondo.dmi.unict.it/wp-content/uploads/classes/so/mirror-stuff/abs-guide.pdf

# THANK YOU FOR YOUR ATTENTION