

# Bash Scripting

---

Università di Modena e Reggio Emilia

*Prof. Nicola Bicocchi ([nicola.bicocchi@unimore.it](mailto:nicola.bicocchi@unimore.it))*



# Bash Script

---

- Editare lo script. La prima linea (shebang) specifica l'interprete da utilizzare per i comandi successivi (`#!/bin/bash`). Si tratta di un linguaggio interpretato (non compilato)! Tutte le altre linee che iniziano con # sono commenti nel codice.

```
$ vim script.sh
```

- Rendere lo script eseguibile

```
$ chmod u+x script.sh # oppure  
$ chmod 755 script.sh
```

- Eseguire lo script

```
$ ./script.sh
```



# Bash Script

---

```
$ vim script.sh
#!/bin/bash
echo Total number of inputs: $#
echo First input: "$1"
echo Second input: "$2"
exit 0

$ chmod 755 script.sh

$ ./script.sh AAPL GOOGL MSFT
Total number of inputs: 3
First input: AAPL
Second input: GOOGL
```



# Bash Script

---

`$./script.sh oppure $script.sh ?`

- Se non specifichiamo un percorso - ma solo un nome - Bash cerca un programma eseguibile nell'elenco di cartelle rappresentato dalla variabile PATH. Se \$PATH non contiene la cartella punto ( . ), i programmi - anche se si trovano nella cartella corrente - non vengono trovati. Due opzioni:
  - Invocazione tramite percorso esplicito (da preferire)
  - `$./script.sh`
  - Modifica alla variabile PATH (uso didattico)
  - `$ export PATH=$PATH:.`
  - `$ script.sh`

# Variabili speciali

---

- All'interno di uno script Bash è possibile accedere ad un gruppo di variabili speciali che semplificano lo sviluppo
- **\$0** Il nome dello script in esecuzione
- **\$1, \$2, .... \$n** n-esimo parametro passato da linea di comando
- **\$\*** tutti i parametri passati a linea di comando
- **\$@** come **\$\*** ma in forma di lista
- **\$#** numero di parametri da linea di comando
- **\$\$** PID della shell
- **\$?** valore di ritorno dell'ultimo comando (exit)
- **Shift** elimina il primo parametro dalla lista **\$1...\$n**, tutti gli altri scorrono indietro di una posizione

# exit

---

- **exit** termina l'esecuzione di una shell (e di conseguenza anche di uno script) e ritorna al chiamante un valore [0, 255]

```
$ bash      # (avvio sotto-shell)
$ exit 17
$ ctrl-d    # (terminazione sotto-shell)
$ echo $?
17
$
```

# expr

---

- **expr** è utilizzato per eseguire operazioni matematiche. Può essere utile negli script in abbinamento a costrutti iterativi.
  - Op. aritmetiche: +,-,\* ,/,%
  - Op. di confronto: <, <=, ==, !=, >=, >
  - Op. logiche: &, |

```
$ expr 2 \* 6
```

```
12
```

```
$ A=12
```

```
$ A=$(expr $A - 1)
```

```
$ echo $A
```

```
$ 11
```



# Costrutti di controllo

---

# test

---

- **test** è un comando per eseguire verifiche di varia natura su stringhe, numeri e file. Un controllo avvenuto con successo produce il valore di ritorno 0, altrimenti 1.

```
$ test 5 -gt 3; echo $? # 0
$ test 5 -gt 3; echo $? # 1
$ test "nicola" == "nicola"; echo $? # 0
$ test "nicola" == "mario"; echo $? # 1
$ test -f /etc/passwd; echo $? # 0
$ test -d /etc/passwd; echo $? # 1
```

## test (strings)

---

- `string1 = string2`      True if strings are identical
- `string1 == string2`      True if strings are identical
- `string1 != string2`      True if strings are not identical
- `string`              Return 0 exit status (=true) if string is not null
- `-n string`              Return 0 exit status (=true) if string is not null
- `-z string`              Return 0 exit status (=true) if string is null

# test (numbers)

---

- int1 –eq int2      Test identity
- int1 –ne int2      Test inequality
- int1 –lt int2      Less than
- int1 –gt int2      Greater than
- int1 –le int2      Less than or equal
- int1 –ge int2      Greater than or equal

# test (files)

---

- -d file      Test if file is a directory
- -f file      Test if file is not a directory
- -s file      Test if the file has non zero length
- -r file      Test if the file is readable
- -w file      Test if the file is writable
- -x file      Test if the file is executable
- -o file      Test if the file is owned by the user
- -e file      Test if the file exists
- -z file      Test if the file has zero length

## test (logic)

---

- -o logical or
- -a logical and
- ! logical not

# if

---

```
if test condizione; then
    comando
else
    comando
fi
```

```
if [ condizione ]; then
    comando
else
    comando
fi
```

```
# Esempio
if [ -f passwd -a -r passwd ];
then
    echo "passwd file leggibile!"
fi
```

```
# Esempio
if [ $# -ne 3 ]; then
    echo "numero parametri errato!"
fi
```

```
# Esempio
if [ ! $# -eq 3 ]; then
    echo "numero parametri errato!"
fi
```

# if

---

- Non è possibile effettuare pattern matching all'interno di un costrutto if-test. Per ovviare al problema, si utilizza il costrutto switch-case che abbina il pattern matching alla possibilità di eseguire più confronti in modo sintetico (evitando else if).

```
# Esempio
if [ "$1" == "n?co*" ]; then
    echo "success"
fi

if [ "$1" != "[0-9]*" ]; then
    echo "success"
fi
```

# case

---

```
case EXPRESSION in
    PATTERN_1)
        STATEMENTS
    ;;
    PATTERN_2)
        STATEMENTS
    ;;
    PATTERN_N)
        STATEMENTS
    ;;
*)
    STATEMENTS
;;
esac
```

```
if [ $# -ne 1 ]; then
    echo "usage: $0 arg"
    exit 1
fi

case "$1" in
/*) echo "Absolute filename"
;;
/*/*) echo "Relative filename"
;;
*) echo "Simple, relative
filename"
;;
esac
exit 0
```



# Costrutti iterativi

---

# for

---

```
for arg in list; do  
    comando/i  
    ...  
done
```

```
# Esempio: tabellina del 5  
for i in 1 2 3 4 5; do  
    echo "5 * $i = $(expr 5 \* $i )"  
done
```

```
# Esempio: mostra i nomi file in home directory  
for fname in "$HOME"/*; do  
    echo "$fname"  
done
```



# while

---

```
while comando_esegue_con_successo; do  
    comando/i  
    ...  
done
```

```
# Esempio  
i=10  
while [ "$i" -gt 0 ]; do  
    echo $i  
    i=$(expr $i - 1)  
done
```



# Struttura script

---

- Nello sviluppo di script è buona norma (*best practice*) aderire ad un canovaccio noto e consolidato
- Definizione interprete (shebang)
- Definizione variabili globali
- Definizione funzioni
- Controllo parametri
- Corpo principale
- Terminazione

```
#!/bin/bash

USAGE="usage: $0 dirname"

if [ $# -ne 1 ]; then
    echo "$USAGE"
    exit 1
fi

if [ ! -d "$1" -o ! -x "$1" ]; then
    echo "$USAGE"
    exit 1
fi

F=0; D=0
for fname in "$1"/*; do
    if [ -f "$fname" ]; then
        F=$(expr $F + 1)
    fi
    if [ -d "$fname" ]; then
        D=$(expr $D + 1)
    fi
done

echo "#files=$F, #directories=$D"
exit 0
```

# Struttura script (regole base)

---

- Trattandosi di un linguaggio antico, l'indentazione è ancora facoltativa (in Python, recente, è obbligatoria!). **Indentazione è comunque di fondamentale importanza!**
- Variabili globali allo script sono maiuscole (ad es. USAGE="\$0 usage: ...")
- **Controllo dei parametri avviene in via negativa.** Si controllano le condizioni di fallimento e, se verificate, si termina lo script ritornando un codice errore (exit 1). Questa pratica evita indentazione eccessiva
- I valori di uscita (exit) utilizzano valori diversi per distinguere successo (exit 0) da fallimento (exit 1). Per differenziare fra diversi tipi di fallimento di utilizzano numeri positivi (exit 2)
- I filesystem moderni supportano la presenza di spazi. Tutte le variabili fuori dal controllo del programmatore (ad es. nomi di file) vanno espansse fra virgolette (echo "\$fname")

# Funzioni

---



# Comandi Principali

---



# Comandi Principali

---

## Gestione Filesystem Management

- Gestione processi
- Gestione rete
- Gestione sistema
- Gestione utente
- Programmazione
- Editing di testi
- Varie

- \$ **man** cmd
- \$ **info** cmd
- \$ **whatis** cmd
- \$ **cmd** --help

# Filesystem

---

- ls
- pwd
- cp
- mv
- ln
- rm
- mkdir
- rmdir
- locate
- which
- cat
- touch

# Filesystem

---

- chmod
- chown
- find
- cmp
- diff

# Processi

---

- ps
- top
- kill
- fg
- bg
- &

# Flussi dati

---

- grep
- sort
- head
- tail
- cut
- wc
- uniq
- sed
- awk

# Varie

---

- echo
- expr

# Compressione

---

- tar
- gzip
- bzip2