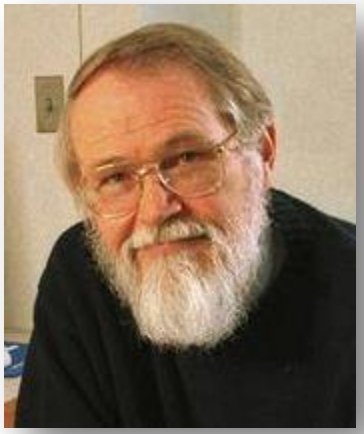
CC BY NC SA

Linux Base

# Scripting
# Esecuzione condizionale
# Costrutti iterativi
# Variabili (quoting, espansione)

*"... a strange language, shaped as much by history as by design"*

— Kernigham & Pike



# Il primo script

```
File Edit Options Buffers Tools Sh-Script Help
echo "Il mio primo script"
echo -n "Ora corrente:"
date
ls /
echo "finito."
```

# Esecuzione

```
giovanni@moth:~/gx$ cat commands.sh
echo "Il mio primo script"
echo -n "Ora corrente:"
date
ls /
echo "finito."
giovanni@moth:~/gx$ . commands.sh
Il mio primo script
Ora corrente:Tue Aug 26 15:02:38 CEST 2014
bin   dev  home        initrd.img.old  lost+found  mnt  proc  run   selinux  sys  usr  vmlinuz
boot  etc  initrd.img  lib             media       opt  root  sbin  srv      tmp  var  vmlinuz.old
finito.
```

# Variabili

```
# la variabile FOO è «locale»
FOO=23

# la variabile FOO è esportata negli ambienti
#+ dei processi figli
export FOO=10

# rimuove l'ambiguità (se serve)
echo ${FOO}BAR
```

# Variabili speciali

**`$RANDOM`**

- un numero casuale

# Variabili speciali

**$BASH**

**$BASHPID**

**$BASHOPTS**

**$BASH_VERSION**

…

– informazioni sulla bash

# Variabili speciali

**`$USER`**

**`$GROUPS`**

- informazioni sull'utente

# $PATH

- La shell cerca i comandi all'interno del **$PATH**
- La cartella corrente «.» non fa parte del path
  - e non è prudente aggiungerla
- È possibile aggiungere le proprie cartelle al **$PATH**
  - nel dubbio: aggiungere sempre alla fine

```
PATH=$PATH:$HOME/scripts
echo $PATH
```

# Argomenti e Variabili speciali

**`$*`**

**`$@`**

- gli argomenti dello script (tutti insieme)
- **`$*`** è equivalente a **`$@`**
- **`"$*"`** non è equivalente a **`"$@"`**

**`$#`**

- numero di argomenti

# Argomenti

**`$0`**

- il nome dello script

**`$1` … `$n`**

- il primo, …, *n*-esimo argomento dello script
- è necessario usare **`${10}`** se n ≥ 10

**`shift`**

- cancella **`$1`** e sposta ogni **`${n}`** in **`${n-1}`**

sha bang

# Il primo script (v2)

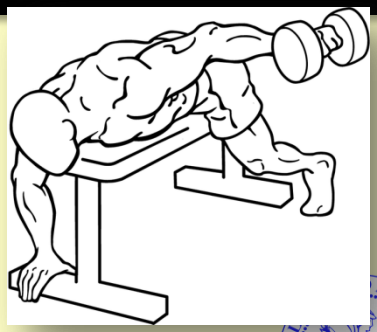
```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

#============================================================================
# Il mio primo script
#============================================================================

echo -n "Ciao $1, ora corrente: "
date
ls /etc
echo "finito."

exit 0
```

```
giovanni@moth:~/gx$ ls -l commands.sh
-rw-rw-r-- 1 giovanni giovanni 259 Aug 26 16:34 commands.sh
giovanni@moth:~/gx$ chmod +x commands.sh
giovanni@moth:~/gx$ ls -l commands.sh
-rwxrwxr-x 1 giovanni giovanni 259 Aug 26 16:34 commands.sh
giovanni@moth:~/gx$ ./commands.sh
Ciao , ora corrente: Tue Aug 26 16:34:28 CEST 2014
bin   dev  home        initrd.img.old  lost+found  mnt  proc  run   selinux  sys  usr  vmlinuz
boot  etc  initrd.img  lib             media       opt  root  sbin  srv      tmp  var  vmlinuz.old
finito.
```

# Esercizio

- Scrivere lo script «**hello**» che stampa a video il nome dell'utente
- Creare la cartella **etc** nella **home**, aggiungerla al **PATH**, ed eseguire lo script

```
$ hello
Ciao giovanni
```



# Informazioni sul processo

**`$$`**

- il PID corrente

**`$UID / $GID`**

- l'UID / il GID corrente

# Test

**$?**

- il valore di ritorno (exit code) dell'ultimo comando
- Convenzione
  - zero: tutto ok
  - maggiore di zero: problemi

```
giovanni@moth:~/gx$ ls /
bin   dev  home        initrd.img.old  lost+found  mnt  proc  run   selinux  sys  usr  vmlinuz
boot  etc  initrd.img  lib             media       opt  root  sbin  srv      tmp  var  vmlinuz.old
giovanni@moth:~/gx$ echo $?
0
giovanni@moth:~/gx$ ls /la_dir_non_esiste
ls: cannot access /la_dir_non_esiste: No such file or directory
giovanni@moth:~/gx$ echo $?
2
giovanni@moth:~/gx$ _
```

# Test

- Il comando «**test**» valuta l'espressione specificata e ne restituisce il valore (0 per vero, 1 falso)
- «**[ expr ]**» è equivalente a «**test expr**»
  - «**[**» è un comando builtin
  - ma esiste anche il file eseguibile **/usr/bin/[**

```
giovanni@moth:~/gx$ test 3 = 3; echo $?
0
giovanni@moth:~/gx$ test 3 = 5; echo $?
1
giovanni@moth:~/gx$ [ 2 = 2 ]; echo $?
0
```

- Attenzione: «**3 = 3**» è un confronto fra parole, non numeri

# Controlli principali

`-f foo` foo esiste ed è un file

`-d foo` foo esiste ed è una cartella

…

Usare «**`man test`**» per l'elenco completo dei controlli

# Controlli principali

| | |
|---|---|
| `s1 = s2` | la stringa s1 è uguale a s2 |
| `s1 != s2` | la stringa s1 è diversa da s2 |

# Controlli principali

| | |
|---|---|
| `n1 -eq n2` | il numero n1 è uguale a n2 |
| `n1 -ne n2` | il numero n1 è diverso da n2 |
| `n1 -ge n2` | il numero n1 è maggiore o uguale a n2 |
| `n1 -gt n2` | il numero n1 è maggiore di n2 |
| `n1 -le n2` | il numero n1 è minore o uguale a n2 |
| `n1 -lt n2` | il numero n1 è minore di n2 |

# [ *exp* ] vs. [[ *exp* ]]

- **[[ *exp* ]]** valuta l'espressione come **[ *exp* ]**
- [[ … ]] è una keyword della shell, non è un comando (interno o esterno)
  - maggiore versatilità
  - test più potenti
  - non *del tutto* standard

# Esecuzione condizionale

```
if espressione
then
   lista comandi
elif espressione
   lista comandi
…
elif espressione
   lista comandi
else
   lista comandi
fi
```

# Esecuzione condizionale

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

if [[ $UID -ne 0 ]]; then
    echo "Questo script deve essere eseguito con i privilegi di root"
    exit 1
fi

echo "Oh yeah"
exit 0
```

```
giovanni@moth:~/gx$ ./root.sh
Questo script deve essere eseguito con i privilegi di root
giovanni@moth:~/gx$ sudo ./root.sh
Oh yeah
giovanni@moth:~/gx$
```

# Hacks

- Il comando viene eseguito se *expr* è vera

```
expr && comando
```

- Il comando viene eseguito se *expr* è falsa

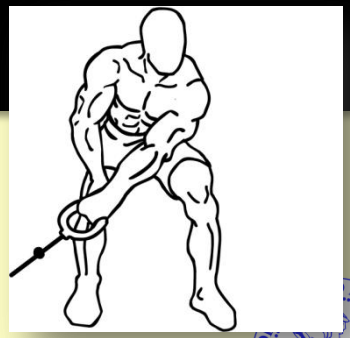
```
expr || comando
```

```
giovanni@moth:~/gx$ [[ -d $HOME/zap ]] && cd $HOME/zap
giovanni@moth:~/gx$ [[ -d $HOME/zap ]] || echo $HOME/zap non esiste
/home/giovanni/zap non esiste
```

# Esercizio

- Scrivere uno script per controllare che tutti gli argomenti siano tutti file esistenti ed eseguibili
- In caso negativo, stampare l'elenco dei parametri non corretti spigando le ragioni

```
$ ./script.sh foo script.sh bar.txt
foo: file non trovato
bar.txt: file non eseguibile
```



# Alternative

```
case parola in
   pat1) comandi ;;
   pat2) comandi ;;
   …
   *) comandi default ;;
esac
```

# Alternative

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

case $1 in
    carot*) echo verdura ;;
    mela) echo frutta ;;
    pera) echo frutta ;;
    *) echo altro ;;
esac

exit 0
```

```
giovanni@moth:~/gx$ ./tmp.sh mela
frutta
giovanni@moth:~/gx$ ./tmp.sh carotina
verdura
giovanni@moth:~/gx$ ./tmp.sh pollo
altro
```

# Espansione intera

- L'espressione dentro (( … )) viene considerata una «*espressione intera*» e valutata di conseguenza

```
(( A = 12 + 45 ))
B=$(( A + 5 ))

if (( A >= 23 )); then
     echo YES
fi
```

- in (( … )) si possono usare: == != > >= < <= …

# Costrutti iterativi

```
while espressione
do
   lista comandi
done
```

```
until espressione
do
   lista comandi
done
```

# Costrutti iterativi

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

t=10

echo -n "COUNTDOWN:"
while (( t >= 0 )); do
    echo -n " $t"
    (( t = t -1 ))
done
echo " -- PARTENZA"

exit 0
```

```
giovanni@moth:~/gx$ ./tmp.sh
COUNTDOWN: 10 9 8 7 6 5 4 3 2 1 0 -- PARTENZA
```

# break/continue

```
while espressione
do
   lista comandi
   expr1 && break
   lista comandi
   expr2 || continue
   lista comandi
done
```

# Costrutti iterativi

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

# Costrutti iterativi

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

n=1
for arg in $*; do
    echo "$n: $arg"
    (( ++n ))
done

exit 0
```

```
giovanni@moth:~/gx$ ./tmp.sh /???
1: /bin
2: /dev
3: /etc
4: /lib
5: /mnt
6: /opt
7: /run
8: /srv
9: /sys
10: /tmp
11: /usr
12: /var
```

# printf

- Per chi ha familiarità con il C, la bash mette a disposizione la **printf**

```
printf "Formattato: %3d\n" 23
```

- Può essere utilizzata per definire variabili

```
printf -v FOO "Formattato: %3d\n" 23
```

# Command substitution

- L'output di un comando può essere utilizzato come fosse una stringa (ad esempio assegnato ad una variabile)

```
DIR=$( ls / )
echo $DIR
```

```
MSG=`echo "La vecchia Bourne shell"`
echo $MSG
```

# Esercizio

- Scrivere lo script **`bar.sh`** che conta il numero di righe dei file passati come argomento e stampa una barra composta da ***hash*** '#', una ogni 10 righe
- Problemi
  - come contare le linee di un file ed assegnare il valore ad una variabile?

# Quoting

- Il tasto racchiuso fra apici doppi viene visto come un singolo argomento, le variabili sono espanse
- Il tasto racchiuso fra apici singoli viene visto come un singolo argomento, le variabili non sono espanse
- Il tasto backslash «\» (escape) cambia il significato del carattere che segue

# Quoting

```
giovanni@moth:~/gx$ ./args.sh "Al Di Meola" 'Stanley Clarke' Jean-Luc\ Ponty
Arg 0: {{Al Di Meola}}
Arg 1: {{Stanley Clarke}}
Arg 2: {{Jean-Luc Ponty}}
giovanni@moth:~/gx$ GUITAR="Al Di Meola"
giovanni@moth:~/gx$ BASS="Stanley Clarke"
giovanni@moth:~/gx$ VIOLIN="Jean-Luc Ponty"
giovanni@moth:~/gx$ ./args.sh "$GUITAR" '$BASS' $VIOLIN
Arg 0: {{Al Di Meola}}
Arg 1: {{$BASS}}
Arg 2: {{Jean-Luc}}
Arg 3: {{Ponty}}
```

# `$*` vs. `$@`

- Se non racchiusi fra doppi apici
  - equivalenti
- Se racchiusi fra doppi apici
  - `"$*"` è espanso in una unica stringa
  - `"$@"` è espanso nei singoli argomenti
- Suggerimento: usate sempre `$@`
- NB: ci sono altre differenze (vedi `$IFS`)

# $* vs. $@

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

#------------------------------------------------------------------------
# programma dimostrativo per esaminare gli argomenti della commandline
#------------------------------------------------------------------------

echo "Dimostrazione di \$*"
argc=0
for f in $*; do
    echo "Arg $argc: {{$f}}"
    (( ++argc ))
done

echo ""; echo "Dimostrazione di \"\$*\""
argc=0
for f in "$*"; do
    echo "Arg $argc: {{$f}}"
    (( ++argc ))
done

echo ""; echo "Dimostrazione di \"\$@\""
argc=0
for f in "$@"; do
    echo "Arg $argc: {{$f}}"
    (( ++argc ))
done
```

# $* vs. $@

```
giovanni@moth:~/gx$ ./args.sh "Al Di Meola" 'Stanley Clarke' Jean-Luc\ Ponty
Dimostrazione di $*
Arg 0: {{Al}}
Arg 1: {{Di}}
Arg 2: {{Meola}}
Arg 3: {{Stanley}}
Arg 4: {{Clarke}}
Arg 5: {{Jean-Luc}}
Arg 6: {{Ponty}}

Dimostrazione di "$*"
Arg 0: {{Al Di Meola Stanley Clarke Jean-Luc Ponty}}

Dimostrazione di "$@"
Arg 0: {{Al Di Meola}}
Arg 1: {{Stanley Clarke}}
Arg 2: {{Jean-Luc Ponty}}
```

# Espansioni

```
giovanni@moth:~/gx$ ./args.sh {uno,due,tre}
Arg 0: {{uno}}
Arg 1: {{due}}
Arg 2: {{tre}}
giovanni@moth:~/gx$ ./args.sh {1..3}
Arg 0: {{1}}
Arg 1: {{2}}
Arg 2: {{3}}
giovanni@moth:~/gx$ ./args.sh {a..c}
Arg 0: {{a}}
Arg 1: {{b}}
Arg 2: {{c}}
giovanni@moth:~/gx$ ./args.sh {1..3}{a..c}
Arg 0: {{1a}}
Arg 1: {{1b}}
Arg 2: {{1c}}
Arg 3: {{2a}}
Arg 4: {{2b}}
Arg 5: {{2c}}
Arg 6: {{3a}}
Arg 7: {{3b}}
Arg 8: {{3c}}
```

# Esercizio

- Scrivere lo script **`args.sh`** utilizzato negli esempi

# Valori di default delle variabili

**`${VAR-valore}`**

- usa valore se non definita

**`${VAR:-valore}`**

- # usa valore se non definita o vuota

# Valori di *default* delle variabili

```
giovanni@moth:~/gx$ FOO=23
giovanni@moth:~/gx$ BAR=
giovanni@moth:~/gx$ unset BAZ
giovanni@moth:~/gx$ ./args.sh $FOO $BAR $BAZ
Arg 0: {{23}}
giovanni@moth:~/gx$ ./args.sh ${FOO-1} ${BAR-2} ${BAZ-3}
Arg 0: {{23}}
Arg 1: {{3}}
giovanni@moth:~/gx$ ./args.sh ${FOO:-1} ${BAR:-2} ${BAZ:-3}
Arg 0: {{23}}
Arg 1: {{2}}
Arg 2: {{3}}
```

# Default + Assegnazione

**`${VAR=valore}`**

- sostituisce ed espande se non definita

**`${VAR:=valore}`**

- sostituisce ed espande se non definita o vuota

# Valori alternativi delle variabili

**`${VAR+valore}`**

- usa valore se definita (anche se vuota)

**`${VAR:+valore}`**

- usa valore se definita e non vuota

# Valori alternativi delle variabili

```
giovanni@moth:~/gx$ FOO=23
giovanni@moth:~/gx$ BAR=
giovanni@moth:~/gx$ unset BAZ
giovanni@moth:~/gx$ ./args.sh $FOO $BAR $BAZ
Arg 0: {{23}}
giovanni@moth:~/gx$ ./args.sh ${FOO+1} ${BAR+2} ${BAZ+3}
Arg 0: {{1}}
Arg 1: {{2}}
giovanni@moth:~/gx$ ./args.sh ${FOO:+1} ${BAR:+2} ${BAZ:+3}
Arg 0: {{1}}
```

# Messaggi di errore

**`${VAR?messaggio}`**

- mostra un messaggio di errore ed interrompe lo script (restituisce 1) se la variabile non è definita

**`${VAR:?valore}`**

- mostra un messaggio di errore ed interrompe lo script (restituisce 1) se la variabile non è definita o è vuota

# Pattern

**`${var%PATTERN}`**

– rimuove il più piccolo *pattern* dalla fine

**`${var%%PATTERN}`**

– rimuove il più lungo *pattern* dalla fine

```
giovanni@moth:~$ FOO=Mammamia
giovanni@moth:~$ echo ${FOO%a*}
Mammami
giovanni@moth:~$ echo ${FOO%%a*}
M
```

# Pattern

**`${var#PATTERN}`**

– rimuove il più piccolo *pattern* dall'inizio

**`${var##PATTERN}`**

– rimuove il più lungo *pattern* dall'inizio

```
giovanni@moth:~$ FOO=Mammamia
giovanni@moth:~$ echo ${FOO#*m}
mamia
giovanni@moth:~$ echo ${FOO##*m}
ia
```

# Pattern

**`${var//tro/sos}`**

- sostituisce tutte le occorrenze di «tro» con «sos»

```
giovanni@moth:~$ FOO=Mammamia
giovanni@moth:~$ echo ${FOO//a/XXX}
MXXXmmXXXmiXXX
```

# Pattern

**`${var:start[:num]}`**

- restituisce una sottostringa

```
giovanni@moth:~$ FOO=Mammamia
giovanni@moth:~$ echo ${FOO:1:4}
amma
giovanni@moth:~$ echo ${FOO: -3}
mia
```

# Riferimento indiretto

**`${!VAR}`**

- il valore della variabile il cui nome è memorizza nella variabile VAR

```
giovanni@moth:~$ BAR=Gargle
giovanni@moth:~$ FOO=BAR
giovanni@moth:~$ echo $FOO
BAR
giovanni@moth:~$ echo ${!FOO}
Gargle
```

# Array

```
# inizializzazione
ARRAY=(23 10)

# elementi dell' array
ARRAY[0]=23
ARRAY[1]=10
echo ${ARRAY[1]}
```

# Array

```
# array completo
echo ${ARRAY[*]}
echo ${ARRAY[@]}

# numero elementi
echo ${#ARRAY[@]}
```

# Array

- Nessun indice è equivalente all'indice 0
  - **`$ARRAY`** equivale a **`${ARRAY[0]}`**

# Array (* vs. @)

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

ARRAY=("Al Di Meola" "Stanley Clarke" "Jan-Luc Ponty")

for a in ${ARRAY[*]}; do
    echo "$a"
done

echo "---"
for a in ${ARRAY[@]}; do
    echo "$a"
done

echo "---"
for a in "${ARRAY[*]}"; do
    echo "$a"
done

echo "---"
for a in "${ARRAY[@]}"; do
    echo "$a"
done
```

# Array (* vs. @)

```
giovanni@moth:~/gx$ ./rite_of_strings.sh
Al
Di
Meola
Stanley
Clarke
Jan-Luc
Ponty
---
Al
Di
Meola
Stanley
Clarke
Jan-Luc
Ponty
---
Al Di Meola Stanley Clarke Jan-Luc Ponty
---
Al Di Meola
Stanley Clarke
Jan-Luc Ponty
```

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

file=()
linee=()
step=1

t=0
for f in "$@"; do
    if [[ -r "$f" && -f "$f" ]]; then
        num=$( cat "$f" | wc -l )
        file[$t]="$f"
        linee[$t]=$num
        if (( step < num / 50 )); then
            (( step = num / 50 ))
        fi
        (( ++t ))
    fi
done

t=0
while (( t < ${#file[@]} )); do
    printf "%-25s (%4d): " $( basename "${file[$t]}" ) ${linee[$t]}
    h=0
    while (( h * step < linee[t] )); do
        echo -n "#"
        (( ++h ))
    done
    echo ""
    (( ++t ))
done

exit 0
```

# Input

**`read var1 [… varN]`**

- legge dallo stdin ed assegna le variabili
- comando interno
- alla prima variabile viene assegnata la prima parola, alla seconda la seconda, …, all'ultima il resto della riga

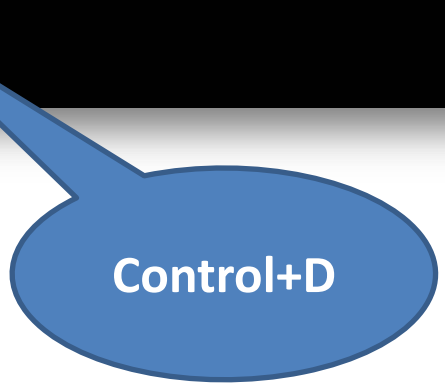
```
giovanni@moth:~$ read foo bar
uno due tre quattro
giovanni@moth:~$ echo $foo
uno
giovanni@moth:~$ echo $bar
due tre quattro
```

# Input

**`read var1 [… varN]`**

- è possibile usare la redirezione
- read è un *espressione vera* se ha assegnato le variabili

```
giovanni@moth:~$ read foo || echo Non hai scritto nulla
Qualcosa!
giovanni@moth:~$ read foo || echo Non hai scritto nulla
Non hai scritto nulla
giovanni@moth:~$
```



# Lettura da file

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

FILE=${1?Un argomento è necessario}

echo "Contenuto di $FILE"
num=0
while read linea; do
    printf "%03d:: %s\n" $num "$linea"
    (( ++num ))
done < "$FILE"

exit 0
```

# Esecuzione automatica

**`~/.bash_profile`**

- Eseguito dalla **bash** quando l'utente si collega alla macchina la prima volta (login)

**`~/.profile`**

- Eseguito dalla **sh** quando l'utente si collega alla macchina la prima volta (login)
- Eseguito da **bash** solo se **`.bash_profile`** manca

**`~/.bashrc`**

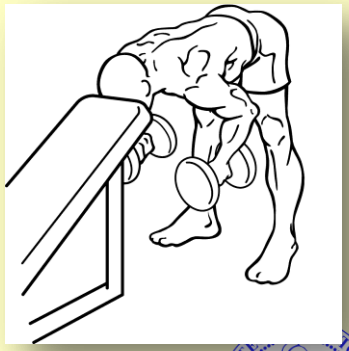
- Eseguito quando l'utente apre un terminale

# Esercizio

- Modificare **.bash_profile** e **.bashrc** facendo stampare due messaggi diversi
- Fare in modo che **.bash_profile** esegua sempre anche **.bashrc**
- Suggerimenti

  `[[ -f .bashrc ]] && source .bashrc`

  usare «`bash -l`» per lanciare una bash di login, «`ps --forest`» per verificare



These slides are licensed under a **Creative Commons**

**Attribution**
**Non Commercial**
**Share Alike**
**4.0 International**

To view a copy of this license, visit
http://creativecommons.org/licenses/by-nc-sa/4.0/

Versione in Italiano:

https://creativecommons.org/licenses/by-nc-sa/4.0/deed.it