LABORATORIO SISTEMI OPERATIVI: BASH

BASH ENV:

- È un **interprete interattivo** (shell), dove da un terminale si possono inserire dei comandi testuali.
- Con **bash**, si può intendere anche il linguaggio interpretato:
  - Si possono inserire i comandi in forma diretta (**CLI**)
  - O mediante **scripting**, cioè la possibilità di **raccogliere i comandi in un file di testo**, ed eseguirli insieme e in maniera coordinata.

- Lo **scripting**, inteso come una **sequenza di comandi**, ha delle particolarità diverse, in base alla modalità utilizzata:
  - **MODALITÀ CLI**: **singola riga** eseguita mediante il tasto invio.
  - **DOCUMENTO DI SCRIPTING**: **raggruppamento di più comandi** che permette anche l'uso di costrutti più complessi.

- L'**ambiente bash** viene **eseguito** all'interno di un ambiente ben definito, più precisamente, all'interno di un **file system** che contiene:
  - CURRENT WORKING DIRECTORY:
    - **Cartella di riferimento di lavoro**
    - Se eseguo un comando (**ls**), senza specificare a quale zona del file system (in quale cartella) mi riferisco, il comando verrà eseguito nella cartella corrente (se eseguo il comando ls, questo mi mostrerà i file contenuti nella cartella corrente).

  - LISTA DI CARTELLE:
    - Intesa come **percorso assoluto all'interno del file system**, è una **lista** che viene **percorsa quando eseguo un programma se non specifico il percorso del programma** (se eseguo un programma, questo non viene cercato nella cartella corrente, ma viene percorsa questa lista di cartelle).

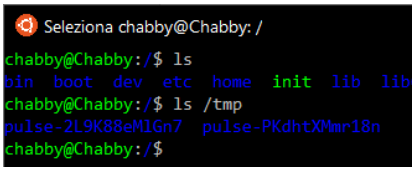
PROMPT:

```
chabby@Chabby:/$ ls
bin  boot  dev  etc  home  init  lib  lib64  media  mnt  opt  proc  root  run  sbin  snap  srv  sys  tmp  usr  var
chabby@Chabby:/$
```

  - Il prompt visualizza di default:
    - **UTENTE ATTIVO**: chabby
    - **SEPARATORE**: @
    - **NOME DEL HOST**: Chabby
    - dopo i due punti, specifica la **cartella corrente**:
      - /#: in questo momento mi trovo nella **root delle cartelle del file system**
    - **#:**
      - per terminare la specifica della cartella corrente (separatore)

COMANDS:

- **ls**:
  - Mostra il contenuto del file system della root directory (senza altre indicazioni)
  - **ls /tmp**:
    - mostra il contenuto della cartella specificata



- **pwd**:
  - Mostra il percorso della cartella corrente

- **cd**:
  - Serve per cambiare cartella di lavoro corrente.

```
chabby@Chabby:/$ pwd
/
chabby@Chabby:/$ cd /tmp
chabby@Chabby:/tmp$ ls
pulse-2L9K88eM1Gn7  pulse-PKdhtXMmr18n
chabby@Chabby:/tmp$ pwd
/tmp
chabby@Chabby:/tmp$
```

- **echo**:
  - Comando che riproduce a video quello che scrivo.

- **ctrl + R:**
  - **Reverse search**, digitando dei caratteri, viene proposto il primo comando, indietro nella history dei comandi che ho già inserito, che soddisfa la ricerca.
  - Premendo **esc** interrompo la ricerca, premendo **invio** eseguo il comando.

- **clear**:
  - Pulisce la finestra del terminale (elimina tutte le scritte).

- **"comando" &**:
  - Se aggiungo **&** dopo un altro comando, quello che succede è che il comando viene eseguito in **background** da un altro processo.
  - Non posso concatenare comandi con il **;** se mando il primo comando in background (non posso fare **comando & ; comando**)

- **cat** "nome file":
  - Comando che dato il nome di un file, mi mostra sul terminale il suo contenuto

- **env**:
  - Comando che mostra l'ambiente di lavoro e le sue variabili:

```
chabby@Chabby:/$ env
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=
30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;
31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:
*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tb
z=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.al
z=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd
=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*
.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35
:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01
;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;
35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35
:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;3
6:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
HOSTTYPE=x86_64
LESSCLOSE=/usr/bin/lesspipe %s %s
LANG=C.UTF-8
WSL_DISTRO_NAME=Ubuntu
USER=chabby
PWD=/
HOME=/home/chabby
NAME=Chabby
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
SHELL=/bin/bash
TERM=xterm-256color
SHLVL=1
LOGNAME=chabby
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/mnt/c/Program Files/Windo
wsApps/CanonicalGroupLimited.UbuntuonWindows_1804.2020.5.0_x64__79rhkp1fndgsc:/mnt/c/Program Files (x86)/Common Files/Or
acle/Java/javapath:/mnt/c/Windows/system32:/mnt/c/Windows:/mnt/c/Windows/System32/Wbem:/mnt/c/Windows/System32/WindowsPo
werShell/v1.0/:/mnt/c/Windows/System32/OpenSSH/:/mnt/c/Program Files/Intel/WiFi/bin/:/mnt/c/Program Files/Common Files/I
ntel/WirelessCommon/:/mnt/c/Users/Ayman/AppData/Local/Microsoft/WindowsApps:/mnt/c/Users/Ayman/AppData/Local/Programs/Mi
crosoft VS Code/bin:/mnt/c/Users/Ayman/AppData/Local/Programs/MiKTeX 2.9/miktex/bin/x64/:/snap/bin
WSLENV=
LESSOPEN=| /usr/bin/lesspipe %s
_=/usr/bin/env
OLDPWD=/
```

- La variabile **PATH** contiene una **serie di percorsi assoluti** ed è appunto la lista che **viene utilizzata quando inserisco un comando che non è riconosciuto direttamente** (non è salvato nella lista di comandi interni memorizzati nell'eseguibile bash).

- Oltre alle variabili d'ambiente, posso creare delle mie variabili:
  - Le variabili hanno come valore di default il niente (null) e hanno **scope globale**.
  - **CREAZIONE**: Nome_variabile = valore
  - **VALORE**: $Nome_variabile oppure ${Nome_variabile}

```
chabby@Chabby:/tmp$ VARX=1
chabby@Chabby:/tmp$ echo La variabile VARX vale $VARX
La variabile VARX vale 1
chabby@Chabby:/tmp$ echo La variabile VARX vale ${VARX}
chabby@Chabby:/tmp$ echo La variabile VARXY che non ho definito ha valore $VARXY
La variabile VARXY che non ho definito ha valore
chabby@Chabby:/tmp$ echo Stampo la variabile con concatenata la lettera Y VARX=${VARX}Y
Stampo la variabile con concatenata la lettera Y VARX=1Y
chabby@Chabby:/tmp$
```

  - Valore di default:
    - Tramite il comando **${Nome_Variabile:-Default}**, posso **assegnare un valore di default** alla variabile, in modo da accedere alla variabile indicata utilizzando il valore di default specificato nel caso la variabile stessa fosse vuota:

```
chabby@Chabby: /tmp
chabby@Chabby:/tmp$ echo ${VARXY}

chabby@Chabby:/tmp$ echo ${VARXY:-Esempio}
Esempio
chabby@Chabby:/tmp$ VARXY=Hello
chabby@Chabby:/tmp$ echo ${VARXY}
Hello
chabby@Chabby:/tmp$
```

- **VARIABILI SPECIALI**:
  - **PATH**:
    - Indica una **lista di percorsi assoluti dove andare a cercare degli eseguibili** se dovessi richiamare un comando esterno (non salvato nel programma bash)
    - Posso cambiare il suo valore mediante il comando:
      - **PATH = /percorso_nuova_cartella**
    - Normalmente **si aggiungono percorsi a quelli preesistenti** mediante il comando:
      - **PATH = $PATH:/percorso_nuova_cartella** (aggiungo in coda il nuovo percorso)

  - **$1, …, $9**: usate all'interno degli script per utilizzare gli eventuali parametri passati allo script.
  - **$@**: contiene l'elenco di tutti gli argomenti che sono eventualmente passati ad uno script.
  - **$?**: contiene il codice di ritorno dell'ultimo comando eseguito .
  - **$#:** contiene il numero di argomenti passati allo script.
  - **$$**: contiene l'identificativo del processo attuale (in esecuzione in questo momento).
  - **$!** : contiene l'identificativo dell'ultimo processo mandato in background.

- **ARRAY**:
  - ARR[@] identifica l'intero array

```
chabby@Chabby:/tmp$ ARR=("UNO" "DUE")
chabby@Chabby:/tmp$ echo "${ARR[@]}"
UNO DUE
chabby@Chabby:/tmp$ echo "${ARR[0]}"
UNO
chabby@Chabby:/tmp$ echo ${ARR[0]}
UNO
chabby@Chabby:/tmp$
```

- **$(“comando”)**:
    - Viene **generato un nuovo processo** e viene eseguito il comando specificato, l’output prodotto dal comando viene eseguito nella bash iniziale.
    - Nell’esempio sottostante, il comando $(ls) fa si venga generato un nuovo processo che eseguirà il comando ls e restituirà al processo padre l’output generato. Il processo padre proverà ad eseguire come comandi l’output generato dal comando ls, fallendo in quanto ls restituisce una lista di file.
    - Nel secondo esempio, viene creato un file che contiene la scritta ls.
    - Con il comando $(cat /tmp/tmp.txt) viene stampato il contento del file e visto che il contenuto è il comando ls, quando sarà ridato al processo padre, questo eseguirà il comando.

```
chabby@Chabby:/tmp$ $(ls)
pulse-2L9K88eMlGn7: command not found
chabby@Chabby:/tmp$ echo "ls" > /tmp/tmp.txt
chabby@Chabby:/tmp$ cat tmp.txt
ls
chabby@Chabby:/tmp$ $(cat /tmp/tmp.txt)
pulse-2L9K88eMlGn7  pulse-PKdhtXMmr18n  tmp.txt
chabby@Chabby:/tmp$
```

    - Usato per assegnazioni di variabili:

```
chabby@Chabby:/tmp$ echo "Esempio di file di testo" > /tmp/tmp.txt
chabby@Chabby:/tmp$ cat tmp.txt
Esempio di file di testo
chabby@Chabby:/tmp$ La variabile CONT contiene: ${CONT}
La: command not found
chabby@Chabby:/tmp$ echo La variabile CONT contiene: ${CONT}
La variabile CONT contiene:
chabby@Chabby:/tmp$ CONT=$(cat /tmp/tmp.txt)
chabby@Chabby:/tmp$ echo La variabile CONT contiene: ${CONT}
La variabile CONT contiene: Esempio di file di testo
chabby@Chabby:/tmp$
```

QUOTING:

- **echo “testo”, ‘testo’** oppure **testo** hanno lo stesso effetto.
- **echo ‘$Nome_variabile’** stampa il nome della variabile e non il suo contenuto.
- Con il carattere **** di **escaping**, specifico che quello che segue deve essere interpretato come carattere letterale.

```
chabby@Chabby:/tmp$ echo $CONT
Esempio di file di testo
chabby@Chabby:/tmp$ echo '$CONT'
$CONT
chabby@Chabby:/tmp$ echo "$CONT"
Esempio di file di testo
chabby@Chabby:/tmp$ echo "\$CONT"
$CONT
chabby@Chabby:/tmp$ echo "\\"
\
chabby@Chabby:/tmp$ echo "\""
"
chabby@Chabby:/tmp$
```

HASHBANG:

- Uno script è meglio che abbia un **commento** speciale della forma: **#!/percorso_assoluto_eseguibile argomento**
- È una direttiva in cui oltre all’argomento (unico) viene passato in input anche il sorgente stesso di dove si trova questo commento.
- **#!/bin/bash:**
    - Cerca nella **cartella bin l’eseguibile** che si chiama **bash** (non passo ulteriori argomenti) e a questo viene passato lo script stesso.
    - Usato per **inter-compatibilità** tra gli script, in questo modo il file stesso sta dichiarando che deve essere eseguito attraverso l’interprete bash (se creo un file python e metto come primo commento **#!/bin/python**, quando vado ad eseguire il file, non è l’interprete bash ad eseguire il codice ma bash passa tutto all’interprete python).
- **#!/usr/bin/env bash**:
    - Stessa filosofia, solo che richiamo il tool env al quale è passato il parametro bash
    - Serve per cercare il percorso esatto dell’interprete bash (è env a scegliere il file, mentre se specifico il percorso assoluto sono io che lo specifico).

EXPORTING & SOURCE:

- Lo **scope delle variabili** tipicamente è **limitato al processo in esecuzione**.
- Se voglio esportare il valore di una variabile, cioè far si che il suo valore possa essere utilizzato dai processi figli, posso usare il comando **export $VAR**

```
chabby@Chabby:/tmp$ A="Lettera A"
chabby@Chabby:/tmp$ pico esempio3.sh
chabby@Chabby:/tmp$ chmod +x esempio3.sh
chabby@Chabby:/tmp$ ./esempio3.sh
A =
chabby@Chabby:/tmp$ echo $A
Lettera A
chabby@Chabby:/tmp$ export A
chabby@Chabby:/tmp$ ./esempio3.sh
A = Lettera A
chabby@Chabby:/tmp$
```

```
  GNU nano 2.9.3

#!/bin/bash

echo "A = $A"
```

- Dopo aver eseguito il comando export A, la variabile A viene resa accessibile ai processi figli del mio script.

```
chabby@Chabby:/tmp$ ./esempio3.sh
A = Lettera A
B =
C = Lettera C
chabby@Chabby:/tmp$ echo $C

chabby@Chabby:/tmp$ source esempio3.sh
A = Lettera A
B = Lettera B
C = Lettera C
chabby@Chabby:/tmp$ echo $C
Lettera C
chabby@Chabby:/tmp$
```

```
#!/bin/bash

echo "A = $A"
echo "B = $B"
C="Lettera C"
echo "C = $C"
```

- In questo caso definisco una variabile $B, se provo ad eseguire lo script: la variabile $B non è visibile al processo figlio e nel processo padre se provo a stampare la variabile $C questa non viene stampata perché è stata definita nel processo figlio e quindi è nata ed è stata distrutta assieme al processo figlio.
- Se eseguo il comando **source**, quello che succede è che lo script viene eseguito nell'ambiente corrente, quindi lo script può accedere al valore della variabile $B e una volta terminato, anche il processo padre conosce il valore della variabile $C.

BLOCKS & FUNCTIONS:

- Un blocco è un comando composto, una lista di comandi concatenati racchiusa tra parentesi graffe (utile ad esempio se voglio lanciare una lista di comandi in background ({comando1 ; comando 2 ; comando 3} &)
- Una funzione è un modo per dare una denominazione ad un comando composto:
  - **function Nome_Funzione {corpo della funzione}**
  - **Nome_Funzione() {corpo della funzione}**

- Per invocarla basta digitare il nome ed eventualmente una lista di argomenti
  - Gli argomenti sono accessibili all'interno della funzione mediante le variabili $1, $2, $3, …
  - Queste variabili sono le stesse che posso usare nello script principale per accedere agli argomenti che posso passare dalla riga di comando quando eseguo lo script

```
chabby@Chabby:/tmp$ ./esempio4.sh
Cartella corrente (A vale Lettera A):  /tmp
Attesa...(A vale Lettera A)
Fine (A vale Lettera A modificata)
chabby@Chabby:/tmp$ pico esempio4.sh
chabby@Chabby:/tmp$ ./esempio4.sh
Cartella corrente (A vale Lettera A):  /tmp
arg1,, arg2,, arg3, ...
Attesa...(A vale Lettera A)
Fine (A vale Lettera A modificata)
chabby@Chabby:/tmp$ pico esempio4.sh
chabby@Chabby:/tmp$ ./esempio4.sh param1 param2 param3
param1, param2, param3, ...
Cartella corrente (A vale Lettera A):  /tmp
arg1,, arg2,, arg3, ...
Attesa...(A vale Lettera A)
Fine (A vale Lettera A modificata)
chabby@Chabby:/tmp$
```

```
#!/bin/bash

echo $1, $2, $3, ...

A="Lettera A"

function attesa {
        echo $1, $2, $3, ...
        local B="Lettera B locale"
        echo "Attesa...(A vale $A)"
        sleep 3
        A="Lettera A modificata"
}

echo "Cartella corrente (A vale $A): " $(pwd)
attesa arg1, arg2, arg3
echo "Fine (A vale $A)"
```

CHAINING AND REDIRECTION:

- Per **concatenare** due o più comandi posso usare:
  - **echo "ciao" ; pwd**
  - **echo "ciao" && pwd**
    - esegue il primo comando e va a buon fine, controlla che anche il secondo comando vada a buon fine, quindi lo esegue (è un and corto-circuitato)
  - **echo "ciao" || pwd**
    - esegue il primo comando, visto che è andato a buon fine, non serve eseguire il secondo (è un or logico basta un vero perché l'espressione sia vera)

```
chabby@Chabby:/tmp$ echo "ciao" ; pwd
ciao
/tmp
chabby@Chabby:/tmp$ echo "ciao" && pwd
ciao
/tmp
chabby@Chabby:/tmp$ echo "ciao" || pwd
ciao
chabby@Chabby:/tmp$ comandoNonEsistente "ciao" && pwd
comandoNonEsistente: command not found
chabby@Chabby:/tmp$ comandoNonEsistente "ciao" || pwd
comandoNonEsistente: command not found
/tmp
chabby@Chabby:/tmp$
```

- Per **reindirizzare** un output posso usare il comando **>**:
  - **comando > /path_File_testo/comando** (se creo anche il file, allora il file andrà a contenere come prima riga il nome stesso del file)
  - **comando >> /path_File_testo:** appendo i dati al file (non sovrascrivo)
  - **comando < /path_File_testo:** prendo i dati da file e li passo al comando

```
chabby@Chabby:/tmp$ ls
elenco1.txt  esempio2.py  esempio3.sh  pulse-2L9K88eM1Gn7  tmp.txt
esempio1.sh  esempio2.sh  esempio4.sh  pulse-PKdhtXMmr18n
chabby@Chabby:/tmp$ ls > /tmp/elenco1.txt
chabby@Chabby:/tmp$ cat elenco1.txt
elenco1.txt
esempio1.sh
esempio2.py
esempio2.sh
esempio3.sh
esempio4.sh
pulse-2L9K88eM1Gn7
pulse-PKdhtXMmr18n
tmp.txt
chabby@Chabby:/tmp$ wc < /tmp/elenco1.txt
  9   9 118
chabby@Chabby:/tmp$
```

  - Casi particolari di reindirizzamento sono:
    - **ls 1 > /dev/null 2>&**1
    - 1 indica che tutto quello che normalmente sarebbe andato a video deve andare sul file speciale **/dev/null** (che reindirizza l'output eliminandolo) mentre la scrittura 2>&1 indica che tutti i messaggi di errore (2) devono andare al canale numero 1 che è il numero del canale normale.

INTEGER EXPRESSION:

- **$((espressione)):**
  - Usato per **elaborazione di operazioni matematiche** basilari.
  - Nel primo caso, la variabile x viene interpretata come variabile e viene computata l'operazione 3+4 = 7 prima di moltiplicare la x per 7, ottenendo così 49 come risultato finale.
  - Nel secondo caso, **l'interprete sostituisce $x con il suo valore letterale**, quindi si ha $((7 * 3 + 4)) che fa 25

```
chabby@Chabby:/tmp$ cat esempio5.sh
#!/bin/bash

x=3+4
echo $(( 7 * x ))
echo $(( 7 * $x ))
chabby@Chabby:/tmp$ ./esempio5.sh
49
25
chabby@Chabby:/tmp$
```

- Quando si vogliono fare **operazioni più complicate** si usa il tool **bc**, se non gli passo input apre una shell interattiva dedicata alle operazioni matematiche, altrimenti elabora l'input che ha in ingresso.

```
chabby@Chabby:/tmp$ bc
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software Foundat
ion, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
3+3
6
21/2
10
scale=2; 21/2
10.50
quit
chabby@Chabby:/tmp$ echo "scale=2; 21/2" | bc
10.50
chabby@Chabby:/tmp$
```

- Il comando echo genera un output e tramite **pipe** (il carattere **|**) questo output viene dato in input al comando successivo, in questo caso è **bc** che elabora e restituisce il risultato, che verrà stampato tramite echo.

LOGIC:

- Per valutare si usa il comando **test** che sfrutta il **codice di ritorno** di un comando per capire se ha avuto successo o meno (se ritorna 0 è andato a buon fine, qualsiasi valore diverso da 0 indica l'insuccesso).
- Per **confronti tra numeri** si usa:
  - **-eq, -ne, -lt, -le, -gt, -ge**
- Per **confronti tra stringhe** si usa:
  - **=, !=, \<, \>**
- Per **verificare** se esiste in un **percorso** dato una cartella o un file:
  - **-d, -e, -f**
- La stessa cosa si può fare con le **parentesi quadrate**, molto utile per i costrutti che valutano espressioni:

```
chabby@Chabby:/tmp$ test 2 -eq 3 ; echo $?
1
chabby@Chabby:/tmp$ test 2 -eq 2 ; echo $?
0
chabby@Chabby:/tmp$ test "a" \< "b" ; echo $?
0
chabby@Chabby:/tmp$ test "c" \< "b" ; echo $?
1
chabby@Chabby:/tmp$
```

```
chabby@Chabby:/tmp$ [ 2 -eq 3 ] ; echo $?
1
chabby@Chabby:/tmp$ [ 2 -eq 2 ] ; echo $?
0
```

CONDITIONAL STRUCTURES:

- **[[espressione]]**:
  - valuta espressione in forma booleana ma con sintassi diverse (ad esempio < > vengono valutati come operatori di disuguaglianza e non di redizionamento)
  - differisce da **[espressione]** solo per il parsing dell'espressione che deve valutare

- **Costrutti**:
  - if then elif else fi
  - case … esac

- **Loop**:
  - for do done
  - while
  - until

```
chabby@Chabby:/tmp$ cat esempio6.sh
#!/bin/bash

if [ "$1" = "--help" ]; then
        echo "Usage..."
        exit 20
fi

echo "Actions..."
chabby@Chabby:/tmp$ ./esempio6.sh
Actions...
chabby@Chabby:/tmp$ echo $?
0
chabby@Chabby:/tmp$ ./esempio6.sh --help
Usage...
chabby@Chabby:/tmp$ echo $?
20
```

```
#!/bin/bash

for i in 1 2 3 4 5; do
        echo "I = $i"
done

echo "-----"

for i in {1..8}; do
        echo "J = $i"
done

echo "-----"

#forma più simile ai costrutti di alto livello
for(( i=1; i <= 10; i++ )); do
        echo "X = $i"
done
```

```
I = 1
I = 2
I = 3
I = 4
I = 5
-----
J = 1
J = 2
J = 3
J = 4
J = 5
J = 6
J = 7
J = 8
-----
X = 1
X = 2
X = 3
X = 4
X = 5
X = 6
X = 7
X = 8
X = 9
X = 10
```

```
chabby@Chabby:/tmp$ cat esempio8.sh
#!/bin/bash

echo "Crea un file '/tmp/lock.txt'"

#fintanto che non esiste il file fai
until [[ -e "/tmp/lock.txt" ]]; do
        echo "Attesa until in corso..."
        sleep 3
done

echo "Adesso elimina il file '/tmp/lock.txt'"
#finchè esiste il file fai
while [[ -e "/tmp/lock.txt" ]]; do
        echo "Attesa while in corso..."
        sleep 3
done
chabby@Chabby:/tmp$
```

```
chabby@Chabby:~$ cd /tmp
chabby@Chabby:/tmp$ touch lock.txt
chabby@Chabby:/tmp$ rm lock.txt
chabby@Chabby:/tmp$
```

```
chabby@Chabby:/tmp$ ./esempio8.sh
Crea un file '/tmp/lock.txt'
Attesa until in corso...
Attesa until in corso...
Attesa until in corso...
Attesa until in corso...
Attesa until in corso...
Attesa until in corso...
Adesso elimina il file '/tmp/lock.txt'
Attesa while in corso...
Attesa while in corso...
Attesa while in corso...
Attesa while in corso...
Attesa while in corso...
chabby@Chabby:/tmp$
```

EXIT CODE:

- Tutti i comandi hanno un **codice d'usc**ita che indica il successo o il fallimento del comando stesso.
- Posso visualizzare questo codice tramite il comando: **echo $?**
- Il codice di ritorno è 0 in caso di successo o maggiore di 0 (fino a 255) in caso di insuccesso.
- **Return** viene usato in una funzione per restituire un codice di ritorno al chiamante, con **exit** si esce dallo script restituendo un codice d'uscita al chiamante.

```
chabby@Chabby:/tmp$ cat esempio10.sh
#!/bin/bash

function prova {
        echo "Prova"
        return 3
}

prova
echo $?
exit 9
chabby@Chabby:/tmp$ ./esempio10.sh
Prova
3
chabby@Chabby:/tmp$ echo $?
9
chabby@Chabby:/tmp$
```