

2. Linux

Shell Programming

Summary

- Cos'è uno Script
- Shell di esecuzione
- Esecuzione di uno Script
- Variabili locali e d'ambiente
- Read
- Operazioni aritmetiche
- Strutture di controllo (if, case, for, while)

Cos'è uno Script

Uno **script** è una **serie di comandi** di shell scritti in un file di testo.

Sono stati introdotti alcuni **costrutti** di controllo tipici di un vero **linguaggio di programmazione** per rendere gli script di shell più **potenti**.

Perché usare gli script di shell?

- Per **creare nuovi comandi personalizzati**
- Per **risparmiare** tempo (esecuzione di uno script, invece di molti comandi in sequenza)
- Per **automatizzare** dei task che devono essere **eseguiti frequentemente**
- Per **facilitare** i compiti dell'**amministratore di sistema**.

Come scrivere uno script?

Si può utilizzare un qualunque editor di testo (per esempio **nano**).

Scrivere uno Script

Proviamo a scrivere lo script **hello_world.sh**:

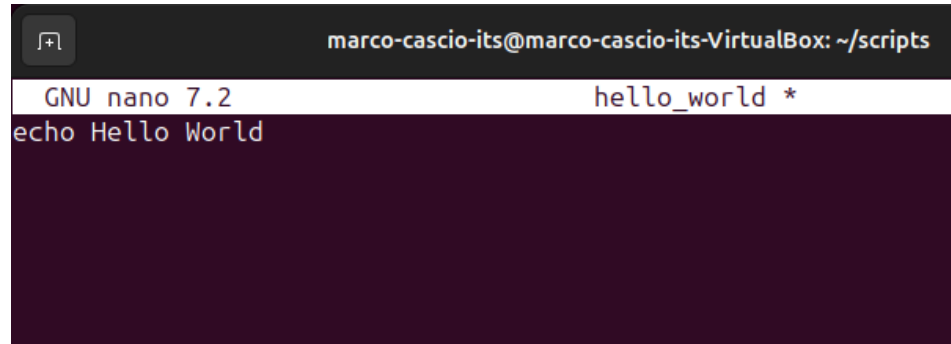
```
$ mkdir scripts
```

```
$ cd ./scripts
```

```
$ nano hello_world.sh
```

- creiamo la directory in cui verranno salvati i nostri script
- spostiamoci nella directory *scripts* appena creata
- apriamo l'editor di testo e creiamo un file *hello_world.sh*

NOTA: l'estensione *.sh* facilmente i file come script di shell



The screenshot shows a terminal window with the title bar 'marco-cascio-its@marco-cascio-its-VirtualBox: ~/scripts'. The terminal content shows the 'GNU nano 7.2' header, the filename 'hello_world *', and the command 'echo Hello World' entered on the first line.

Shell di esecuzione

La **prima riga** di uno script può essere utilizzata per **indicare** quale **shell** interpreta lo script:

`#!/bin/bash` - indica uno script per la shell bash

Per **conoscere** quale **shell** si sta utilizzando, si utilizza il comando:

`$ echo $SHELL`

Commenti/1

Tutte le **righe** che cominciano con il simbolo '#' sono considerati **commenti**, e vengono **ignorati** dall'interprete dello script.

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~/scripts
GNU nano 7.2 hello_world *
#!/bin/bash
echo Hello World

#questo è un commento
```

Commenti/2

Attenzione!!!

#! e # sono **differenti**, in quanto:

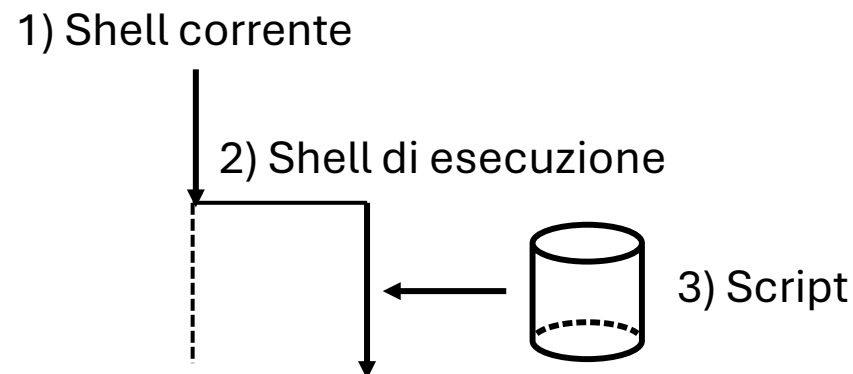
- #! è una **sequenza di caratteri** speciale ed indica al sistema operativo quale **interprete** utilizzare per eseguire lo script e deve essere inserito nella **prima riga** di uno script per poter essere riconosciuto correttamente
- # è un simbolo utilizzato per indicare un **commento** all'interno di uno script. I commenti vengono **ignorati** dall'interprete dello script. Questo simbolo può essere usato in **qualsiasi riga** dello script per aggiungere **notazioni o spiegazioni**.

Esecuzione di uno Script/1

Per **eseguire** uno **script**, è possibile utilizzare:

1. `$ bash < nome_script.sh` - redirectione dello standard di input sullo script
2. `$ bash nome_script.sh`
3. `$./nome_script.sh`

- La shell **ricosce** che il file di input è uno **script** e **determina** quale **shell** deve essere utilizzata per eseguirlo.
- La shell che **esegue** lo script, lo esegue leggendo i **comandi** come se fossero **digitati direttamente** nella shell stessa.

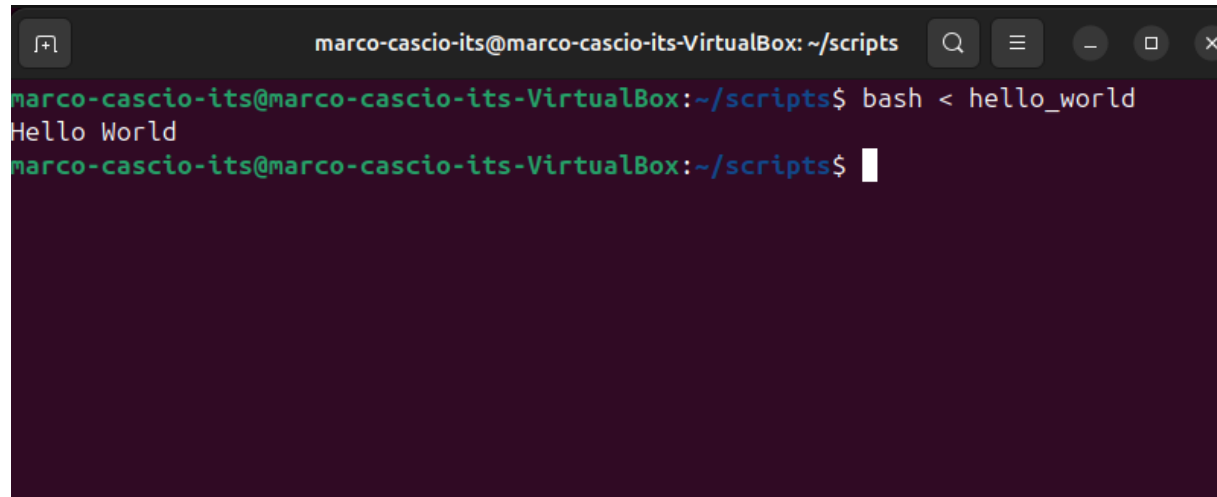


Esecuzione di uno Script/2

Eseguiamo lo script `hello_world.sh`:

Modo 1.

\$ `bash < hello_world.sh`

A terminal window with a dark purple background. The title bar at the top reads "marco-cascio-its@marco-cascio-its-VirtualBox: ~/scripts". The terminal shows the command "bash < hello_world" being entered, followed by the output "Hello World". The prompt "marco-cascio-its@marco-cascio-its-VirtualBox: ~/scripts\$" is visible at the bottom.

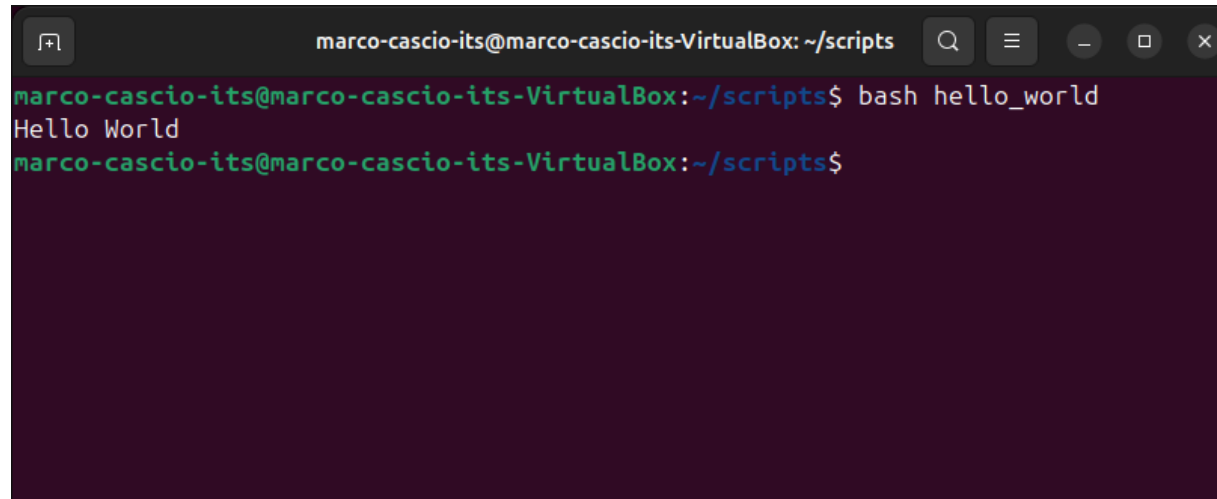
```
marco-cascio-its@marco-cascio-its-VirtualBox: ~/scripts$ bash < hello_world
Hello World
marco-cascio-its@marco-cascio-its-VirtualBox: ~/scripts$
```

Esecuzione di uno Script/3

Eseguiamo lo script `hello_world.sh`:

Modo 2.

\$ `bash hello_world.sh`

A terminal window with a dark purple background. The title bar at the top reads 'marco-cascio-its@marco-cascio-its-VirtualBox: ~/scripts'. The terminal shows the command 'bash hello_world' being entered, followed by the output 'Hello World'. The prompt returns to 'marco-cascio-its@marco-cascio-its-VirtualBox: ~/scripts\$'.

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~/scripts$ bash hello_world
Hello World
marco-cascio-its@marco-cascio-its-VirtualBox: ~/scripts$
```

Esecuzione di uno Script/4

Eseguiamo lo script `hello_world.sh`:

Modo 3.

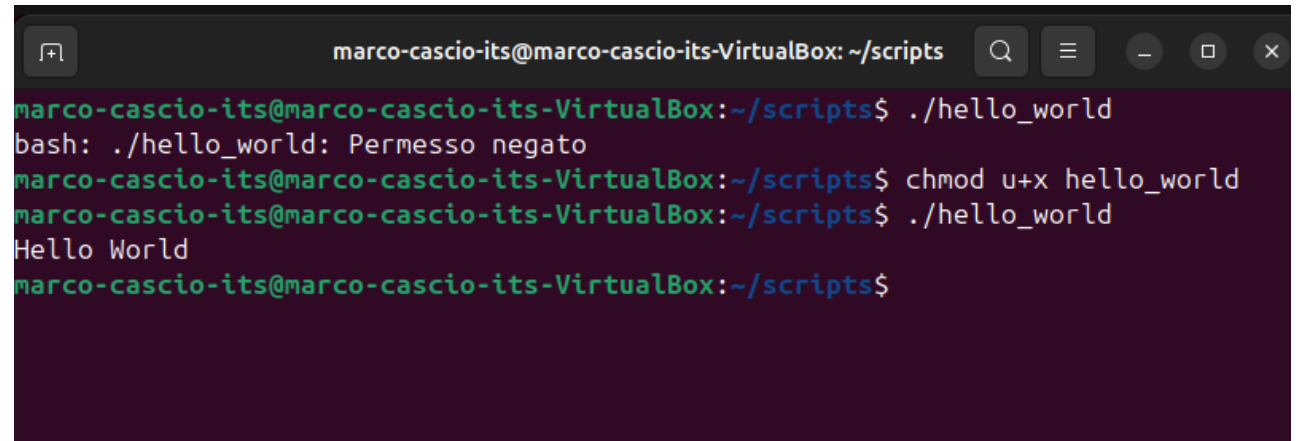
`$./hello_world.sh`

Output: *Permission denied*

- Cambio permessi:

`$ chmod u+x hello_world.sh`

- assegna all'utente il permesso per eseguire lo script `hello_world.sh`



```
marco-cascio-its@marco-cascio-its-VirtualBox: ~/scripts
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ ./hello_world
bash: ./hello_world: Permessi negati
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ chmod u+x hello_world
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ ./hello_world
Hello World
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$
```

Variabili di shell

Una **variabile** è un **nome simbolico** al quale è associato un **valore**.

Le variabili sono utilizzate per **memorizzare** dati che possono essere utilizzati all'interno di **script** e **sessioni di shell**.

La shell permette di definire **due tipi** di **variabili**:

- **variabili locali**
- **variabili di ambiente**

Variabili locali/1

Le **variabili locali** non richiedono alcun comando speciale per essere **create**. Le variabili locali sono semplicemente definite con un'**assegnazione di valore**.

Le **variabili locali** hanno una **validità limitata** all'ambito della shell stessa, in quanto sono **visibili** ed **esistono** solo all'interno e durante l'esecuzione del **contesto** in cui sono state definite, come una **funzione** o uno **script**.

Il **nome** di una **variabile** può contenere **lettere**, **cifre** e **underscore**, ma il primo carattere non può essere un numero.

La **creazione** di una **variabile** e l'**assegnazione** di un **valore** si ottengono con una dichiarazione del tipo:

nome_variabile=valore (senza spazi)

Variabili locali/2

Assegnazione:

- Se **non** viene fornito il **valore** da assegnare, si intende che la **variabile** è uguale ad una **stringa vuota**
- Se la variabile **non esiste**, allora viene **creata** con il **valore specificato**
- Se la variabile **esiste**, il suo **valore precedente** viene **sovrascritto**

Una **variabile locale** può essere **cancellata** con il comando **unset**:

```
$ unset nome_variabile
```

Per **accedere** al **valore** di una **variabile** si utilizza il simbolo **\$**:

```
$ $nome_variabile
```

Variabili locali/3

Esempi:

```
$ a=ciao
```

```
$ echo $a
```

Output: *ciao*

- creo la variabile locale *a* con valore *ciao*
- visualizzo/accedo al valore della variabile *a*

```
$ a=15
```

```
$ echo $a
```

Output: *15*

- sovrascrivo il valore della variabile locale *a* con valore *15*
- visualizzo/accedo al valore della variabile *a*

Variabili locali/4

\$ **b=who**

- definisco la variabile locale *b* con valore il comando *who*

\$ **echo \$b**

- visualizzo/accedo al valore della variabile *a*

Output: *who*

\$ **\$b**

- esegue il comando \$ **who**

Output:

```
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ $b
marco-cascio-its seat0      2024-07-03 11:40 (login screen)
marco-cascio-its tty2      2024-07-03 11:40 (tty2)
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$
```

```
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ who
marco-cascio-its seat0      2024-07-03 11:40 (login screen)
marco-cascio-its tty2      2024-07-03 11:40 (tty2)
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$
```


Variabili locali/5

```
$ unset a
```

```
$ unset b
```

- cancella la variabile a

- cancella la variabile b

```
$ echo $a
```

Output:

```
$ echo $b
```

Output:

Variabili di ambiente/1

Le **variabili di ambiente** sono visibili a **tutti i processi figli** della shell. Questo significa che quando un **processo** viene eseguito dalla shell, **erediterà** le **variabili di ambiente** definite nella **shell genitore**.

Le **variabili di ambiente** persistono e sono **accessibili** ai processi figli finché la **shell genitore** è **attiva** o la **variabile** non viene esplicitamente **eliminata**.

Esiste un insieme di **variabili di ambiente predefinite**, che contengono informazioni utili sull'ambiente di esecuzione. Ad esempio:

- **\$HOME**: path della home directory
- **\$PATH**: path delle directory dove la shell, dopo l'inserimento di un comando, cerca il programma da eseguire.
- **\$MAIL**: path della mailbox dell'utente
- **\$USER**: username dell'utente
- **\$SHELL**: path della shell di login
- **\$TERM**: specifica il tipo di terminale utilizzato

Variabili di ambiente/2

```
$ echo $HOME  
$ echo $MAIL  
$ echo $PATH  
$ echo $TERM  
$ echo $SHELL  
$ echo $USER
```

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ echo $HOME  
/home/marco-cascio-its  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ echo $MAIL  
  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ echo $TERM  
xterm-256color  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ echo $SHELL  
/bin/bash  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ echo $USER  
marco-cascio-its  
marco-cascio-its@marco-cascio-its-VirtualBox:~$
```

Read/1

Il comando **read** legge dallo **standard input** ed assegna alla **prima variabile** specificata la **prima parola** digitata, alla **seconda variabile** specificata la **seconda parola** digitata e così via.

\$ **nano** read_script.sh - crea lo script *read_script.sh*

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~  
GNU nano 7.2 read_script  
#!/bin/bash  
  
echo "Scrivi il tuo nome e cognome:"  
  
#usa il comando read per assegnare alle variabili name e surname i valori di nome e cognome digitati  
read name surname  
  
#saluta, stampando nome e cognome  
echo "Ciao, $name $surname !"
```

Read/2

\$ **bash** read_script.sh

- esegue lo script *read_script.sh*

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ bash read_script  
Scrivi il tuo nome e cognome:  
Marco Cascio  
Ciao, Marco Cascio !  
marco-cascio-its@marco-cascio-its-VirtualBox:~$
```

Operazioni aritmetiche/1

Operatori aritmetici comuni:

+	somma
-	sottrazione
*	prodotto
/	divisione intera
%	resto

Operatori relazionali comuni:

>	maggiore	<	minore
>=	maggiore o uguale	>=	minore o uguale
=	uguale	!=	diverso
&&	AND logico		OR logico

Operazioni aritmetiche/2

Il comando `$ expr <espressione aritmetica>` permette di usare **variabili numeriche** e di **eseguire** con esse semplici **calcoli**.

Esempio:

```
$ expr 1 + 2 (attenzione agli spazi)
```

Output: 3

Esempio:

```
$ a=9 b=3
```

- definisce due variabili numeriche *a* e *b*

Esempio:

```
$ expr $a - $b (attenzione agli spazi)
```

- effettua la sottrazione tra *a* e *b*

Output: 6

Operazioni aritmetiche/3

Le **espressioni aritmetiche** possono essere anche rappresentate come:

- ***`$((<espressione>))`***

Esempio:

```
$ echo $(( $a + $b ))
```

Output: *12*

oppure come:

- ***`$(<espressione>)`***

Esempio:

```
$ echo $[ $a - $b ]
```

Output: *6*

Operazioni aritmetiche/4

Scrivere uno script *multiplier.sh* che consente di moltiplicare due numeri presi in input da tastiera.

Soluzione:

```
$ nano multiplier.sh
```

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~
GNU nano 7.2 multiplier
#!/bin/bash

echo "Digita il primo numero: "
#ricevi il valore del primo numero da tastiera
read num1

echo "Digita il secondo numero: "
#ricevi il valore del secondo numero da tastiera
read num2

#calcola la moltiplicazione tra i due numeri
multiplier=$((num1 * num2))

#visualizza in output il risultato della moltiplicazione
echo "$num1 * $num2 = $multiplier"
```

Operazioni aritmetiche/5

Scrivere uno script *multiplier.sh* che consente di moltiplicare due numeri presi in input da tastiera.

Soluzione:

\$ **bash** multiplier.sh

Output:

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ bash multiplier  
Digita il primo numero:  
10  
Digita il secondo numero:  
30  
10 * 30 = 300  
marco-cascio-its@marco-cascio-its-VirtualBox:~$
```

Strutture di controllo

In ambiente Linux (come anche in programmazione) per compiere una determinata azione si usano delle **strutture di controllo**.

Le principali **strutture di controllo** sono:

- **if**
- **case**
- **for**
- **while**

Struttura if/1

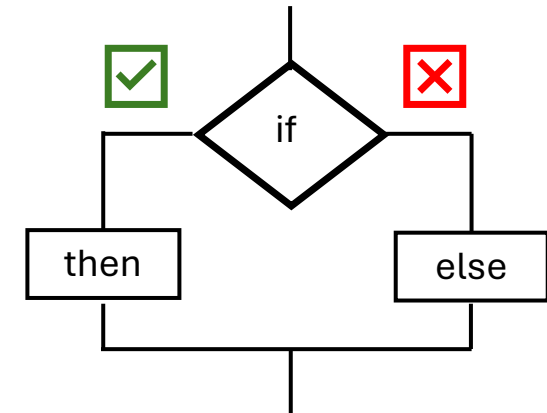
```
if listacomandi1 (oppure condizione)  
then listacomandi2  
else listacomandi3  
fi
```

Se *listacomandi1* ha successo oppure la *condizione* è verificata:

- viene eseguito *listacomandi2*,

altrimenti:

- viene eseguita *listacomandi3* in sequenza.



Struttura if/2

- Operatori matematici

-eq	uguale a
-ne	diverso da
-lt	minore di
-le	minore o uguale a
-gt	maggiore di
-ge	maggiore o uguale a

- Operatori logici

!expr	NOT
expr1 -a expr2	AND
expr1 -o expr2	OR

Struttura if/3

- Operatori su file

-s file	verifica se file ha dimensioni superiori a 0
-f file	verifica se file esiste e non è una directory
-d dir	verifica se dir esiste ed è una directory
-w file	verifica se file è scrivibile
-r file	verifica se file è read-only”
-x file	verifica se file è eseguibile

Struttura if/4

Il comando **test** è utilizzato per verificare se una condizione è **true** o **false** a seconda del tipo di test:

\$ test condizione

Esempio:

\$ test -d scripts

- verifica se *scripts* è una directory

\$ test \$a -le \$b

- verifica se *a* è *minore o uguale* a *b*

Struttura if/5

Esempio 1:

Scrivere uno script *file_dir.sh* che accetti due parametri, ovvero il nome di un file ed il nome di una directory.

Lo script deve verificare se il file e la directory esistono; in caso affermativo, visualizzare un messaggio a riguardo in output.

Soluzione:

\$ nano file_dir.sh

- crea lo script *file_dir.sh*

```
GNU nano 7.2                                     file_dir
#!/bin/bash

#testa se il primo argomento passato come input ($1) è un file
if test -f $1
#se $1 è un file, allora visualizzare un messaggio a riguardo in output
then echo "$1 esiste ed è un file"
else echo "$1 non esiste oppure non è un file"
fi

#testa se il secondo argomento passato come input ($2) è una directory
if test -d $2
then echo "$2 esiste ed è una directory"
else echo "$2 non esiste oppure non è una directory"
fi
```


Struttura if/6

Soluzione

\$ **touch** my_file.txt - crea un file vuoto chiamato *my_file.txt*

\$ **bash** file_dir.sh my_file.txt my_dir - esegue lo script *file_dir.sh* passando come input *my_file.txt* e *my_dir*

Output: *my_file.txt esiste ed è un file*

my_dir non esiste oppure non è una directory

\$ **mkdir** my_dir - crea una directory chiamata

\$ **bash** file_dir.sh my_file.txt my_dir - esegue lo script *file_dir.sh* passando come input *my_file.txt* e *my_dir*

Output: *my_file.txt esiste ed è un file*

my_dir esiste ed è una directory

Struttura if/7

Esempio 2

Scrivere uno script *file_dir.sh* che accetti due parametri, ovvero il nome di un file ed il nome di una directory.

Lo script deve verificare se il file e la directory esistono; in caso affermativo, spostare il file nella directory in questione e visualizzare una scritta con la conferma dell'operazione avvenuta.

Soluzione

\$ nano file_dir.sh

- crea lo script *file_dir.sh*

```
GNU nano 7.2                                     file_dir *
#!/bin/bash

#testa se il primo argomento passato come input ($1) è un file
#e se il secondo argomento passato come input ($2) è una directory
if test -f $1 -a -d $2
#in caso affermativo, sposta il file $1 nella directory $2
then mv $1 $2/$1 ; echo "$1 è stato spostato nella directory $2" ;
else echo "Errore!"
fi
```

Struttura if/8

Soluzione

\$ ls - visualizza i file e le directory nella working directory

\$ **bash** file_dir.sh my_file.txt my_dir - esegue lo script file_dir.sh passando come input *my_file.txt* e *my_dir*

Output:

my_file.txt è stato spostato nella directory my_dir

\$ ls my_dir - visualizza i file nella directory *my_dir*, per verificare lo spostamento

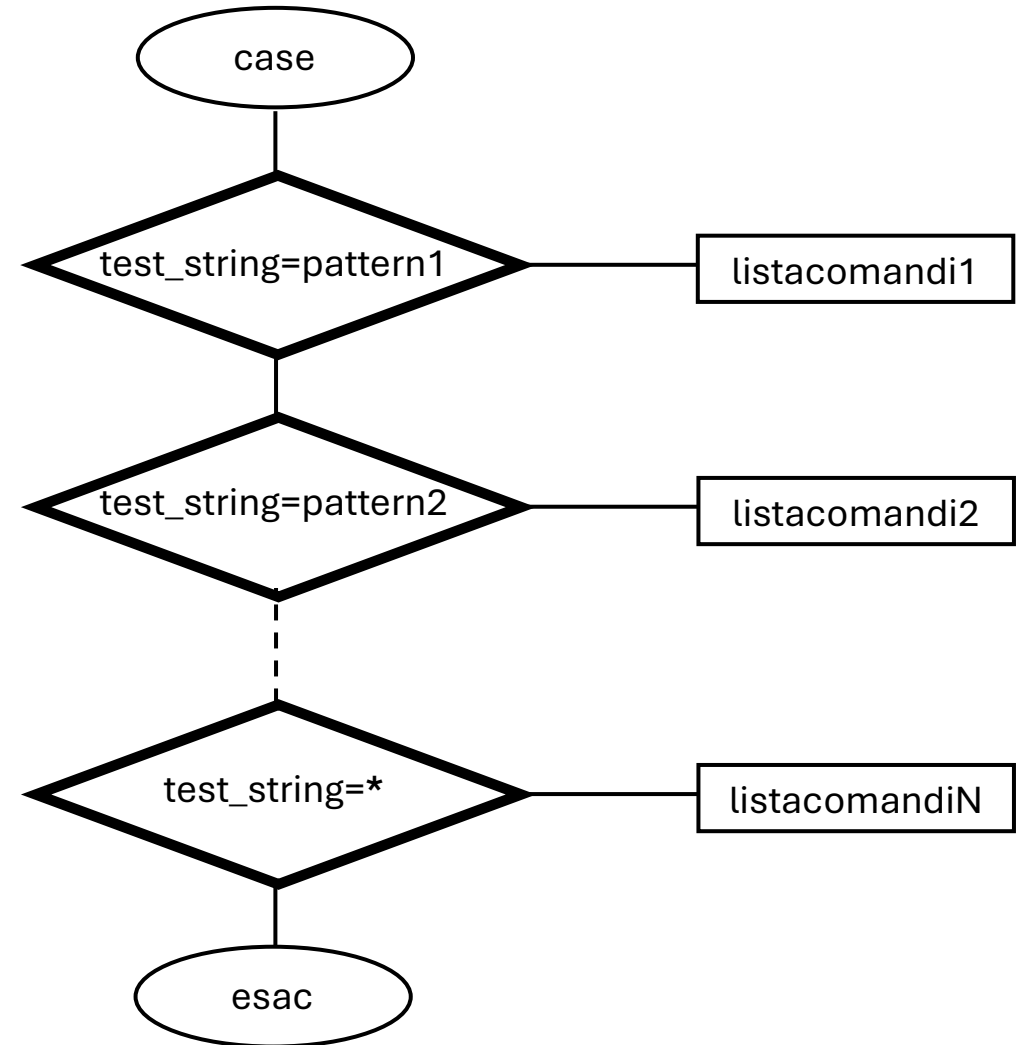
Struttura case/1

```
case test_string in
  pattern1) listacomandi1 ;;
  pattern2) listacomandi2 ;;
  ...
  *) listacomandiN ;;
esac
```

La struttura **case** consente di **effettuare una scelta** nell'**esecuzione** di varie **liste di comandi**.

La **scelta** viene fatta **confrontando** una **stringa** (**test_string**) con una **serie di modelli** (**pattern**) nell'**ordine** in cui esse compaiono.

Se **esiste** una **corrispondenza** con uno dei **pattern**, la relativa **lista di comandi** viene **eseguita**. **Altrimenti**, viene eseguita la **lista di comandi** nel specificata nel **pattern ***).



Struttura case/2

Esempio 1

Scrivere lo script *tasto.sh* che chieda all'utente di premere un tasto e che visualizzi in output un messaggio che specificando il tipo di tasto premuto.

Ad esempio, visualizzare in output "Lettera" se l'utente ha premuto una lettera [a-z], "Numero" se l'utente ha premuto un numero [0-9], "Punteggiatura, spaziatura o altro" se l'utente ha premuto altri tipi di tasti.

Soluzione

\$ nano *tasto.sh* - crea lo script *tasto.sh*

```
GNU nano 7.2                                     tasto
#!/bin/bash

#chiedere all'utente di premere un tasto
echo "Premi un tasto e poi invio: "
#salva nella variabile tasto il tasto premuto dall'utente
read tasto

#struttura case per analizzare il tasto premuto
case $tasto in
[a-z] ) echo Lettera ;; #pattern1
[0-9] ) echo Numero ;; #pattern2
* ) echo Punteggiatura, spaziatura o altro ;; #pattern *)
esac
```

Struttura case/3

Soluzione

\$ **bash** *tasto.sh*

- esegue lo script *tasto.sh*

Output:

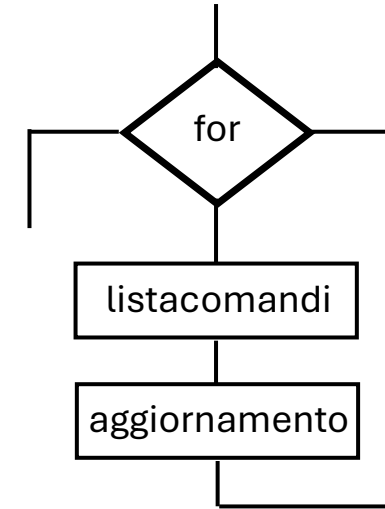
```
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ bash tasto
Premi un tasto e poi invio:
y
Lettera
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ bash tasto
Premi un tasto e poi invio:
9
Numero
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ bash tasto
Premi un tasto e poi invio:
.
Punteggiatura, spaziatura o altro
```

Struttura for/1

```
for i in word1 word2...  
do listacomandi  
done
```

La struttura **for** esegue una **scansione di elementi** ed in corrispondenza di questi **esegue una lista di comandi**.

Alla variabile **i** sono assegnati a turno i valori di **word1**, **word2**, ..., per ogni iterazione del ciclo.



Struttura for/2

Esempio 1

Scrivere lo script *mk_planets.sh*. Questo script deve creare una directory chiamata *planets*. All'interno di tale directory creare 8 file, ognuno dei quali deve essere nominato con il nome di uno dei pianeta del Sistema Solare:

Mercurio, Venere, Terra, Marte, Giove, Saturno, Urano, Nettuno

Soluzione

\$ nano mk_planets.sh

- crea lo script *mk_planets.sh*

```
GNU nano 7.2                                mk_planets
#!/bin/bash

#crea la directory "planets"
mkdir planets

#per ogni pianeta
for planet in mercurio venere terra marte giove saturno urano nettuno
do
    #crea un file con il nome del pianeta nella directory "planets"
    touch planets/$planet
done
```


Struttura for/3

Soluzione

\$ **ls** - visualizza i file e le directory nella working directory

\$ **bash** mk_planets.sh - esegue lo script *mk_planets.sh*

\$ **ls** - visualizza i file nella working directory per verificare che la directory *planets* sia stata create

\$ **ls** planets - visualizza i file in *planet* per verificare le directory con i nomi dei pianeti siano state create

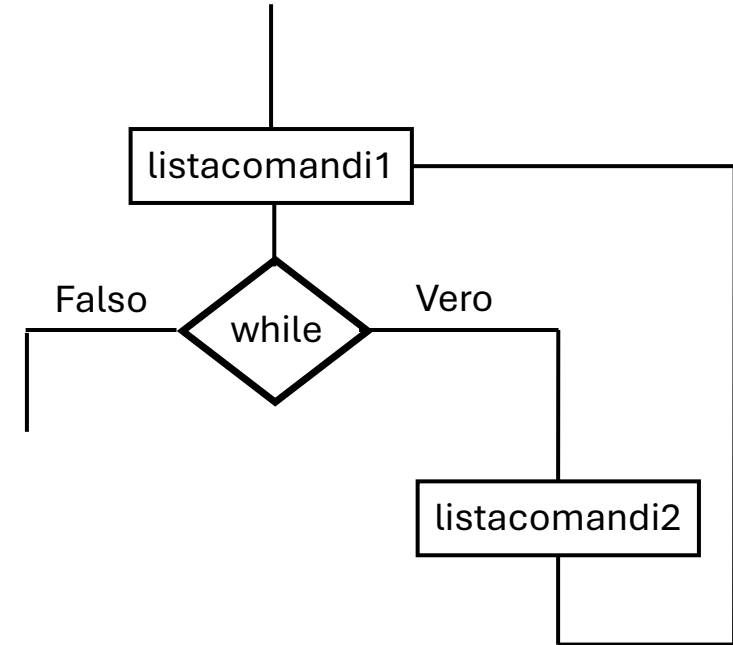
Output:

```
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ bash mk_planets
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ ls
file_dir  hello_world  mk_planets  multiplier  my  my_dir  planets  tast
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ ls planets
giove  marte  mercurio  nettuno  saturno  terra  urano  venere
```

Struttura while/1

```
while listacomandi1 (o condizione)  
do listacomandi2  
done
```

Con la struttura **while**, la *listacomandi2* viene **eseguita** finché la *listacomandi1* (o una **condizione**) risulta essere vera.



Struttura while/2

Esempio 1

Scrivere lo script *numbers_to_file.sh*. Questo script deve contare i numeri da 1 a 30 usando l'istruzione while e se il numero contato è pari, inserire tale numero nel file *pari.txt*.

Soluzione

\$ **nano** numbers_to_file.sh

- crea lo script *numbers_to_file.sh*

Struttura while/3

Soluzione

```
GNU nano 7.2                                     numbers_to_file *
#!/bin/bash

#crea un file chiamato pari
touch pari

#inizializzo il conto, facendo partire la conta da 1
n=1

#struttura while per la conta, finchè n è minore o uguale di 30
while test $n -le 30
do

#mostra in output il numero contato
echo $n

#controlla se il numero contato è pari, ovvero se il calcolo del resto della divisione n/2 è pari a 0
if test [$n % 2] -eq 0
#se il numero contato è pari, inseriscilo nel file pari
then echo $n >> pari
fi

#aggiorna il conto
n=$((n + 1))

done

#notifica all'utente che il conto è finito
echo Conto Finito!
```

Struttura while/3

Soluzione

\$ **bash** numbers_to_file.sh

- esegue lo script *numbers_to_file.sh*

Output:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Conto Finito!
```

Struttura while/4

Soluzione

\$ `cat pari.txt` - legge il contenuto del file *pari.txt*

Output:

```
marco-cascio-its@marco-cascio-its-VirtualBox:~/scripts$ cat pari
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
```