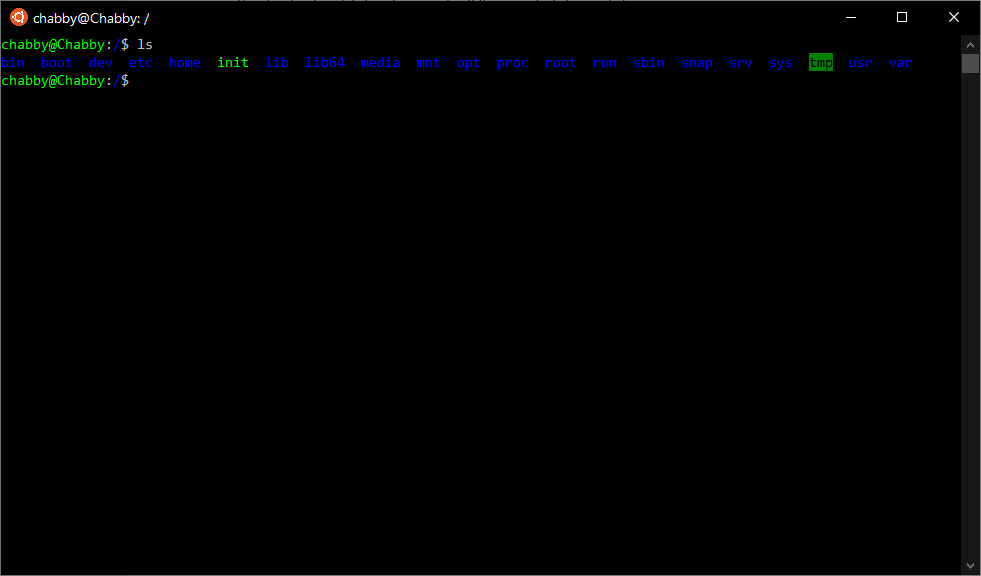
**Laboratorio Sistemi Operativi: Bash**

**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).

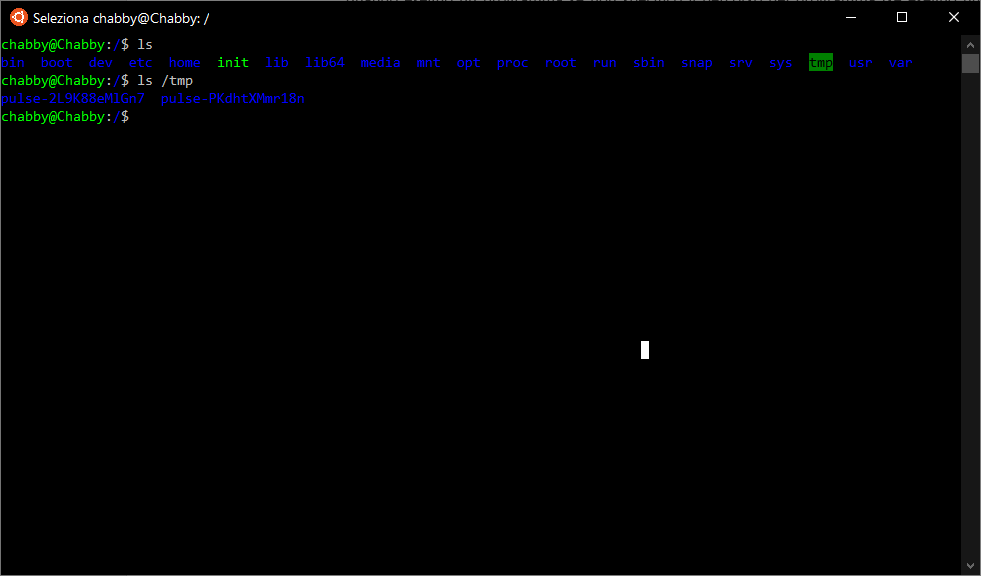
Prompt:



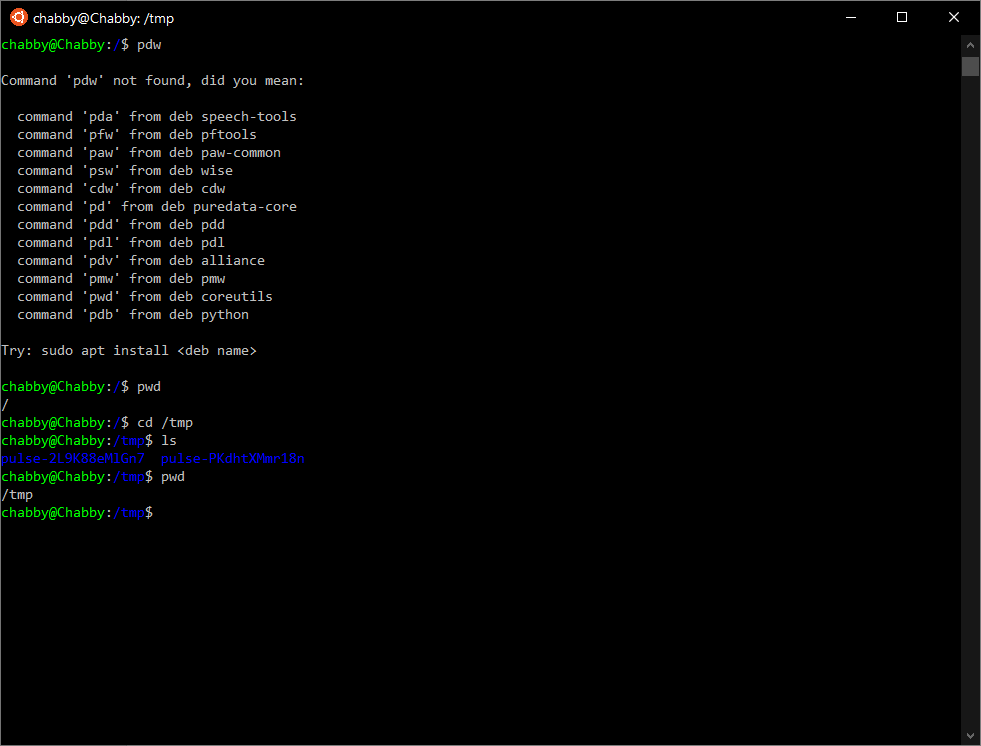
* Il prompt visualizza di default:
  + **utente attivo**: chabby
  + **separatore**: @
  + **nome del host**: Chabby
  + dopo i due punti, specifica la **cartella corrente**:
    - /#: in questo momento mi trovo nella **root delle cartelle del file system**
  + **#:**
    - per terminare la specifica della cartella corrente (separatore)

Comands:

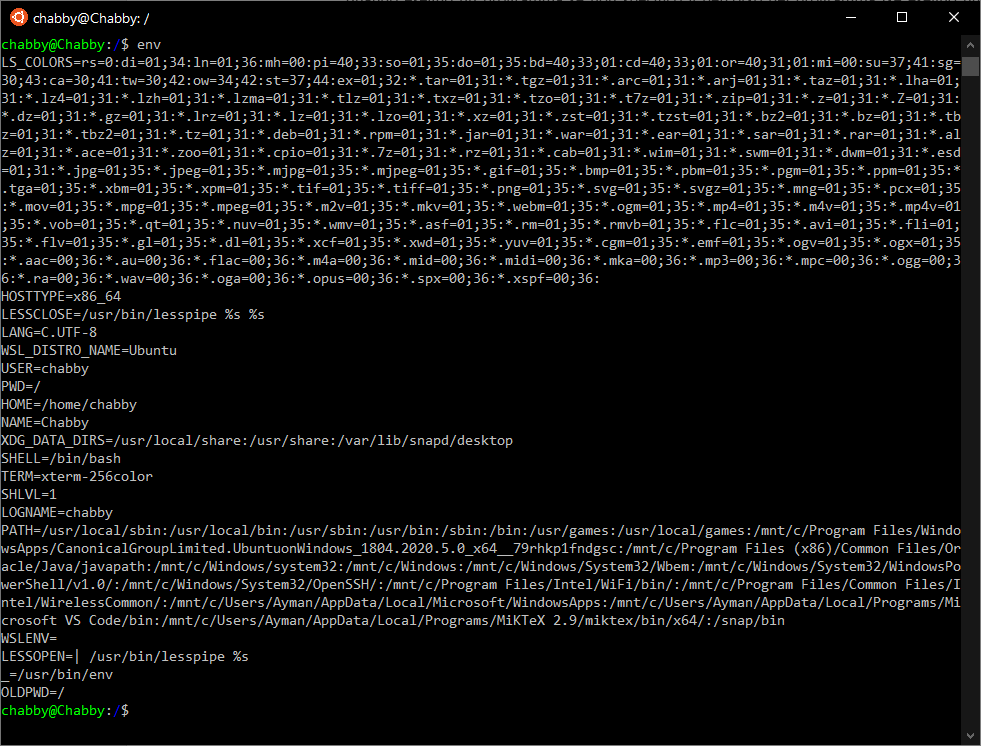
* **ls**:
  + Mostra il contenuto del file system della root directory (senza altre indicazioni)
  + **ls /tmp**:
    - mostra il contenuto della cartella specificata



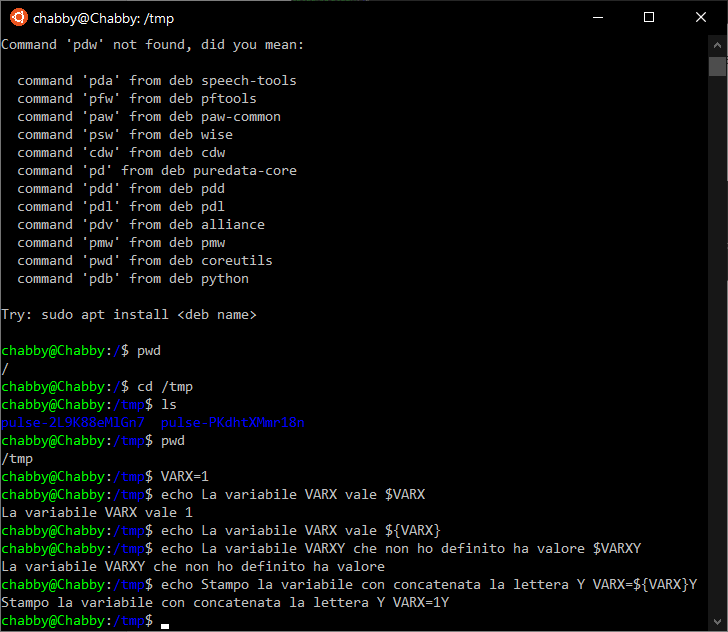
* **pwd**:
  + Mostra il percorso della cartella corrente
* **cd**:
  + Serve per cambiare cartella di lavoro corrente.



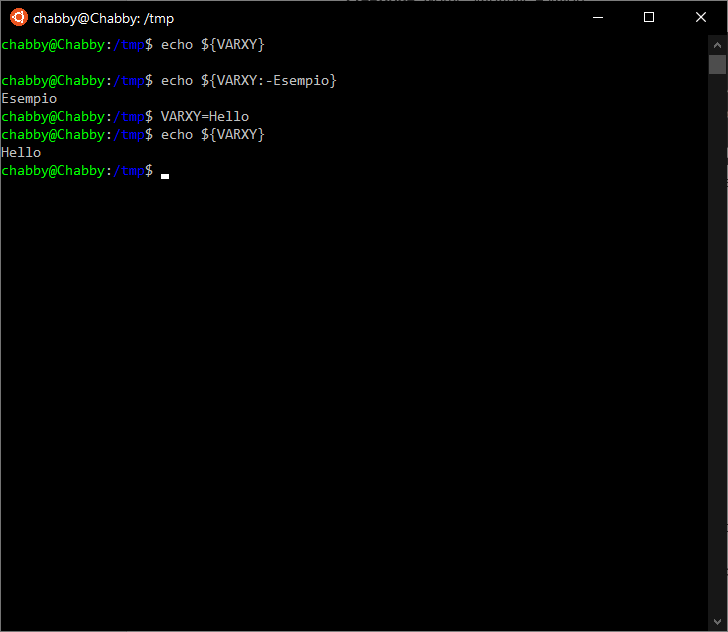
* **echo**:
  + Comando che riproduce a video quello che scrivo.
* **ctrl + R:**
  + **Reverse search**, digitando dei caratteri, viene proposto il primo comando, indietro nella history dei comandi che ho già inserito, che soddisfa la ricerca.
  + Premendo **esc** interrompo la ricerca, premendo **invio** 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:



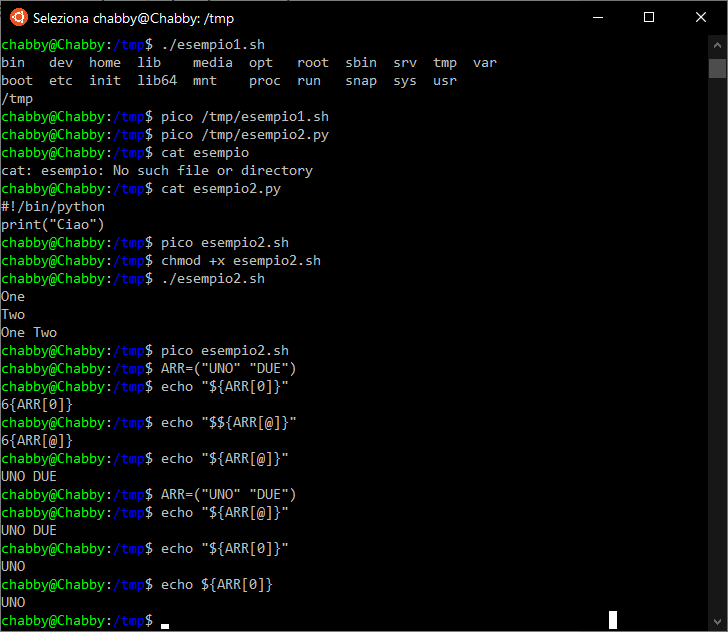
* + La variabile **PATH** contiene una **serie di percorsi assoluti** ed è appunto la lista che **viene utilizzata quando inserisco un comando che non è riconosciuto direttamente** (non è salvato nella lista di comandi interni memorizzati nell’eseguibile bash).
  + Oltre alle variabili d’ambiente, posso 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}



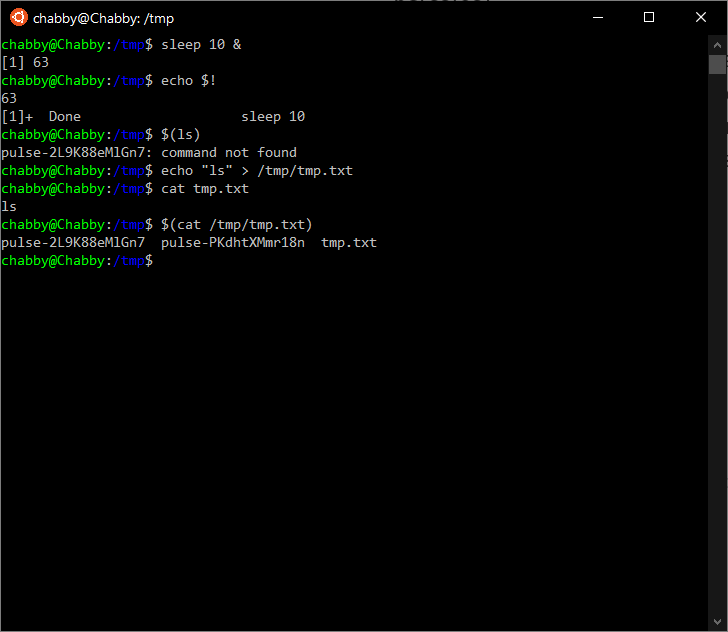
* + - Valore di default:
      * Tramite il comando **${Nome\_Variabile:-Default}**, posso **assegnare un valore di default** alla variabile, in modo da accedere alla variabile indicata utilizzando il valore di default specificato nel caso la variabile stessa fosse vuota:



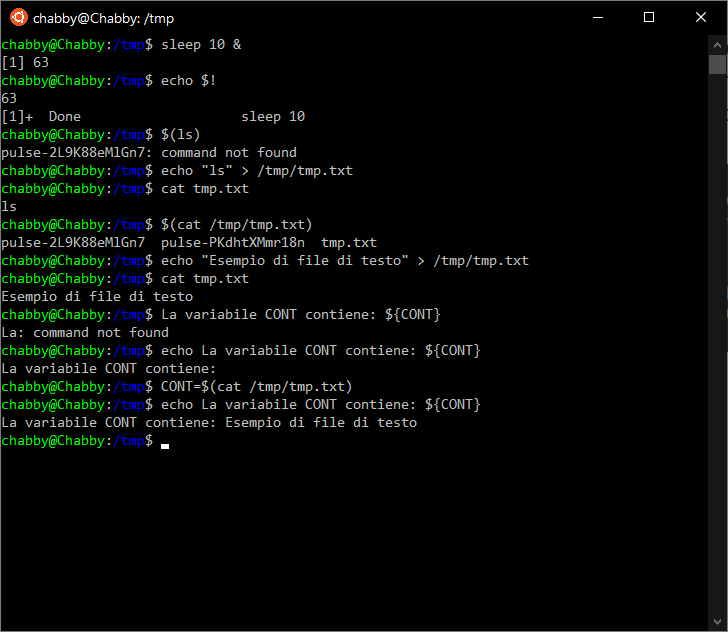
* + **Variabili speciali**:
    - **PATH**:
      * Indica una **lista di percorsi assoluti dove andare a cercare degli eseguibili** se dovessi richiamare un comando esterno (non salvato nel programma bash)
      * Posso cambiare il suo valore mediante il comando:
        + **PATH = /percorso\_nuova\_cartella**
      * 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

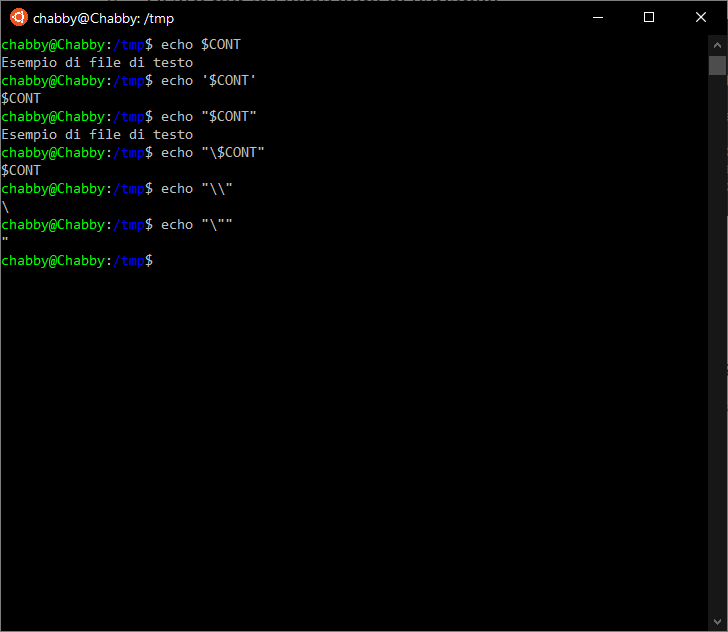


* **$( “comando” )**:
  + Viene **generato un nuovo processo** e viene eseguito il comando specificato, l’output prodotto dal comando viene eseguito nella bash iniziale.
  + Nell’esempio sottostante, il comando $(ls) fa si venga generato un nuovo processo che eseguirà il comando ls e restituirà al processo 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.



* + Usato per assegnazioni di variabili:



**Quoting**:

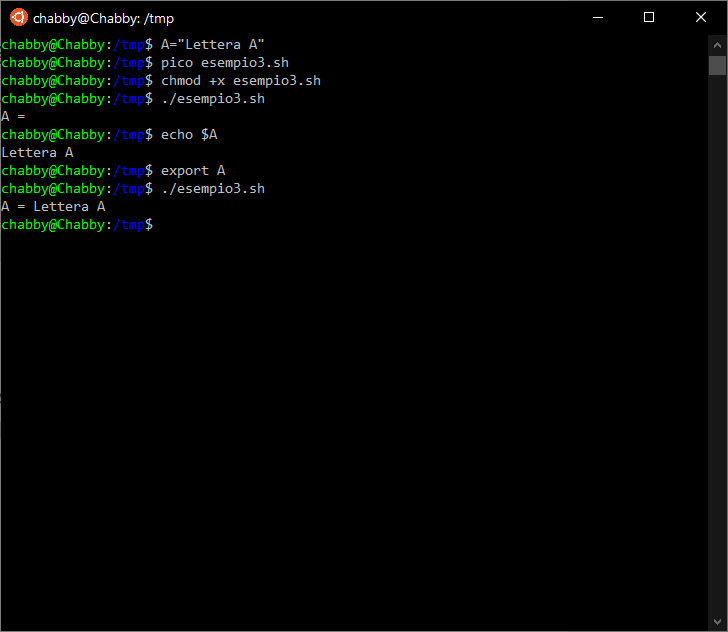
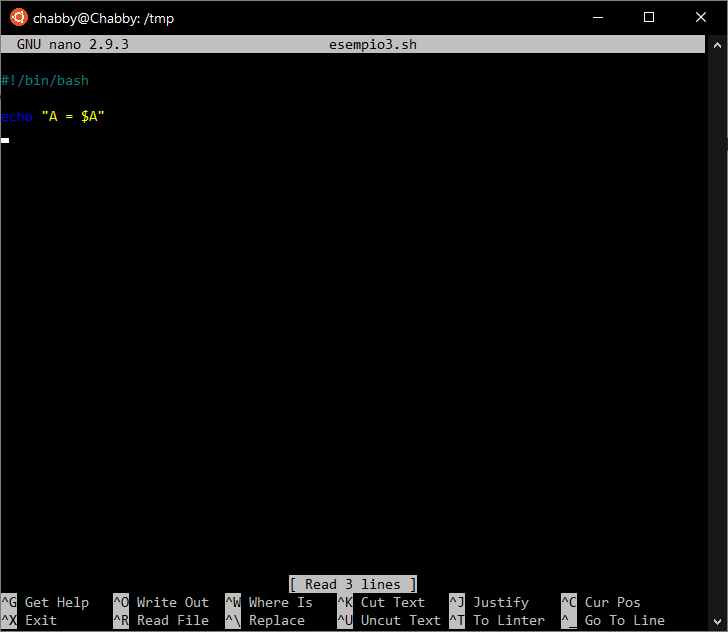
* **echo “testo”,** **‘testo’** oppure **testo** hanno lo stesso effetto.
* **echo ‘$Nome\_variabile’** stampa il nome della variabile e non il suo contenuto.
* Con il carattere **\** di **escaping**, specifico che quello che segue deve essere interpretato come carattere letterale.

Hashbang:

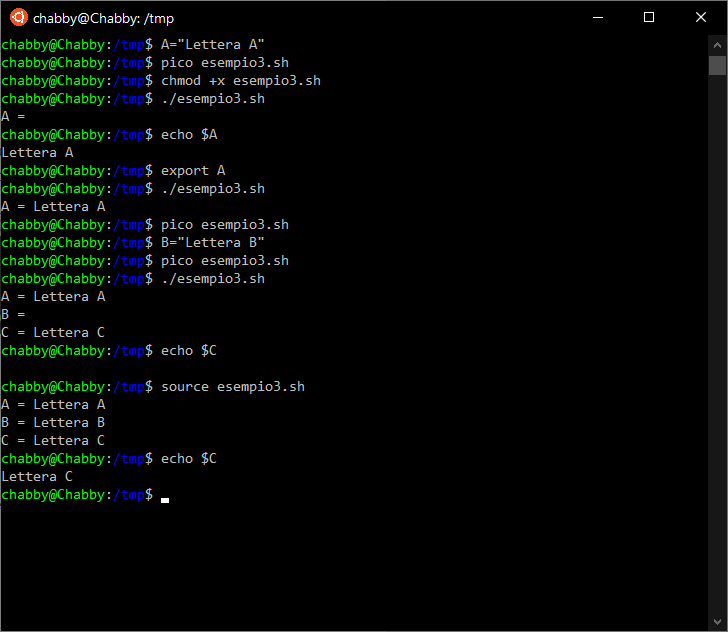
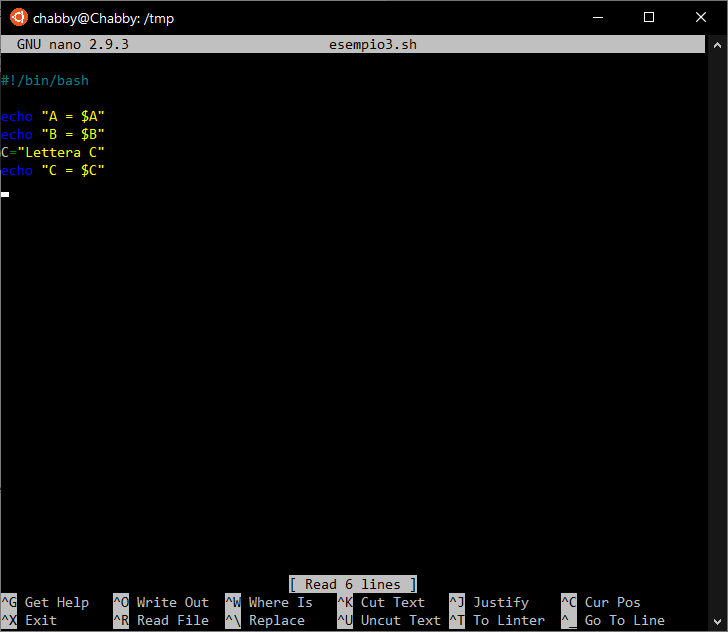
* Uno script è meglio che abbia un **commento** speciale della forma: **#!/percorso\_assoluto\_eseguibile argomento**
* È una direttiva in cui oltre all’argomento (unico) viene passato in input anche il sorgente stesso di dove si trova questo commento.
* **#!/bin/bash:**
  + Cerca nella **cartella** **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**



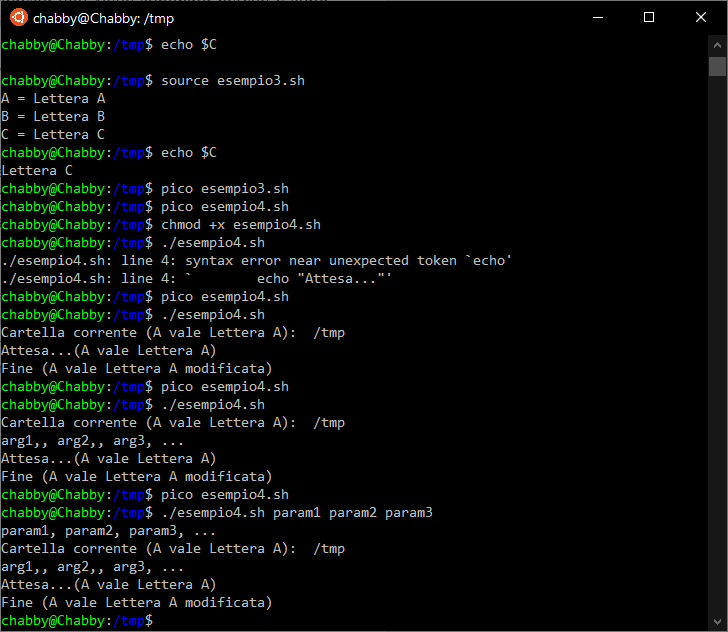
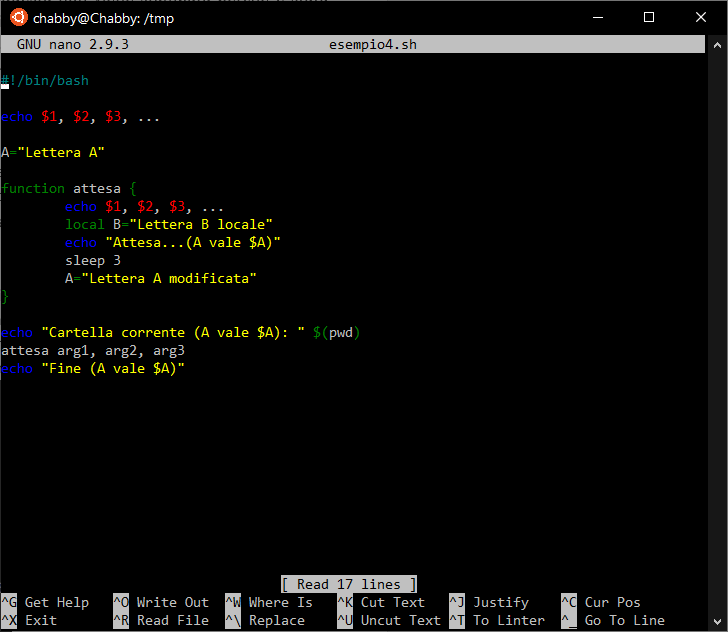
* Dopo aver eseguito il comando export A, la variabile A viene resa accessibile ai processi figli del mio script.



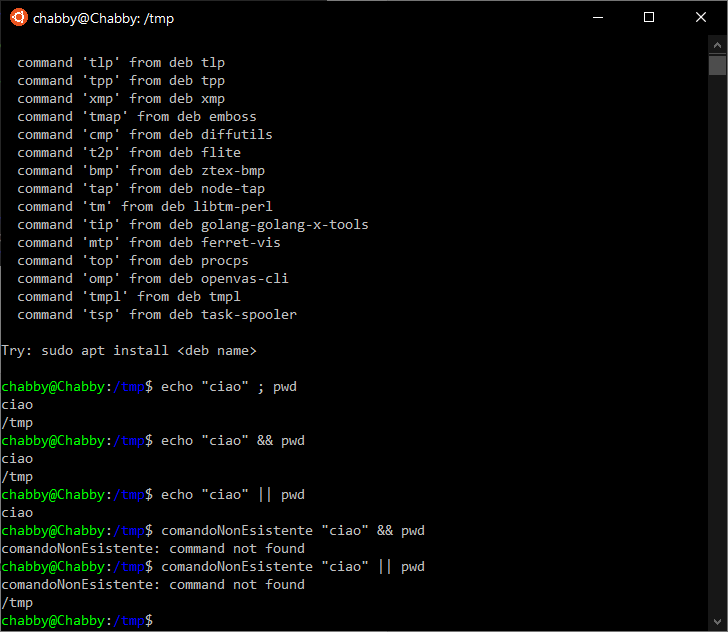
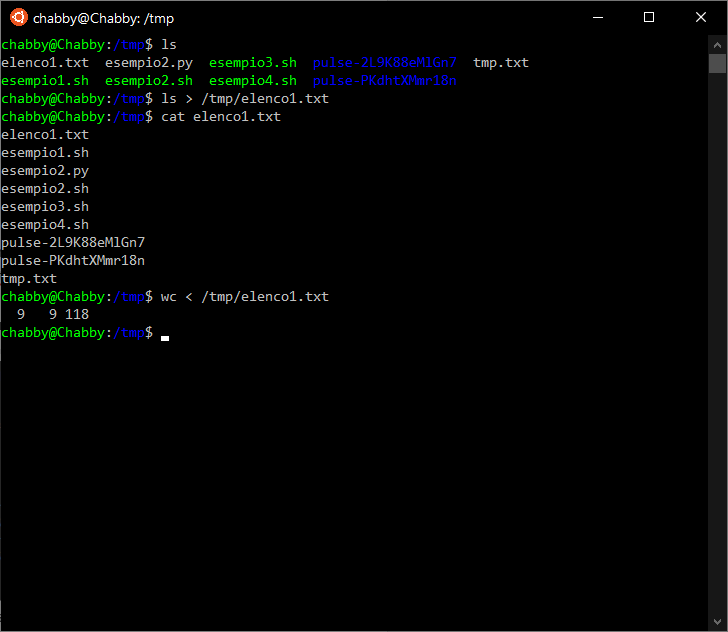
* In questo caso definisco una variabile $B, se provo ad eseguire lo script: la variabile $B non è visibile al processo figlio e nel processo padre se provo a stampare la variabile $C questa non viene stampata perché è stata definita nel processo figlio e quindi è nata ed è stata distrutta assieme al 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:

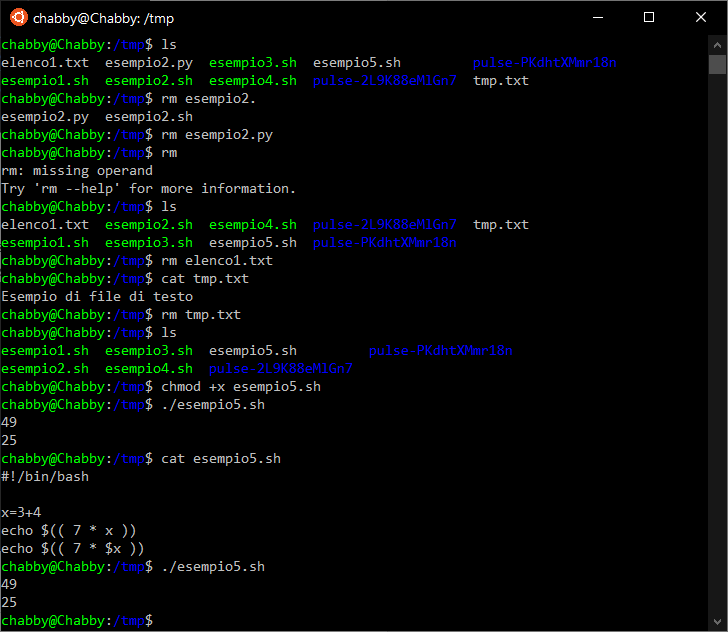
* Un blocco è un comando composto, una lista di comandi concatenati racchiusa tra parentesi graffe (utile ad esempio se voglio lanciare una lista di comandi in background ({comando1 ; comando 2 ; comando 3} &)
* Una funzione è un modo per dare una denominazione ad un comando composto:
  + **function**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

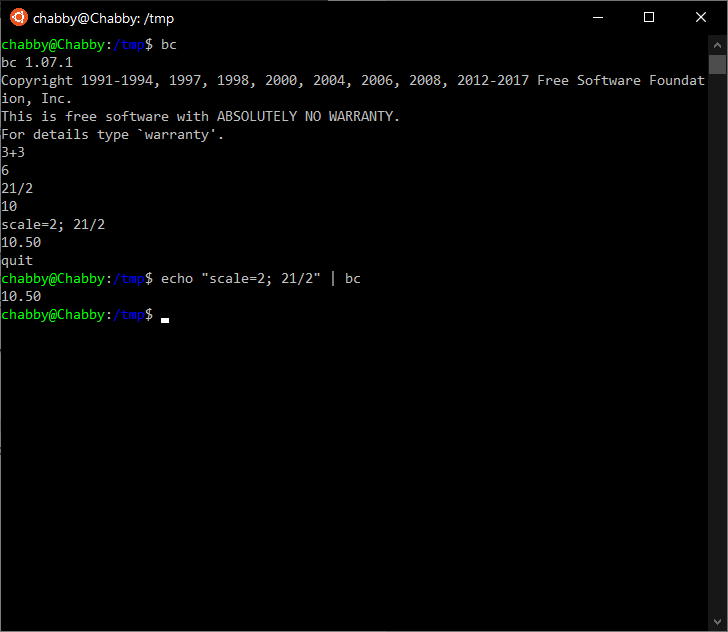
 

Chaining and Redirection:

* 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)
* 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
  + 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.

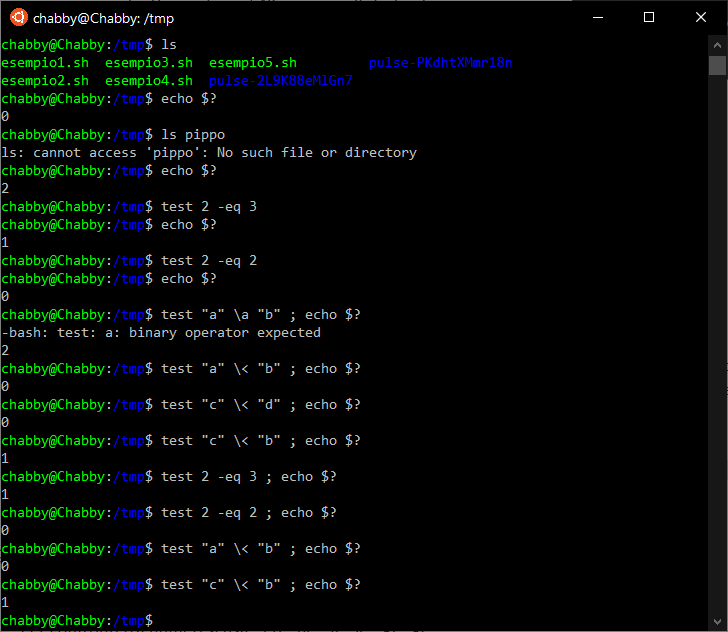
Integer Expression:

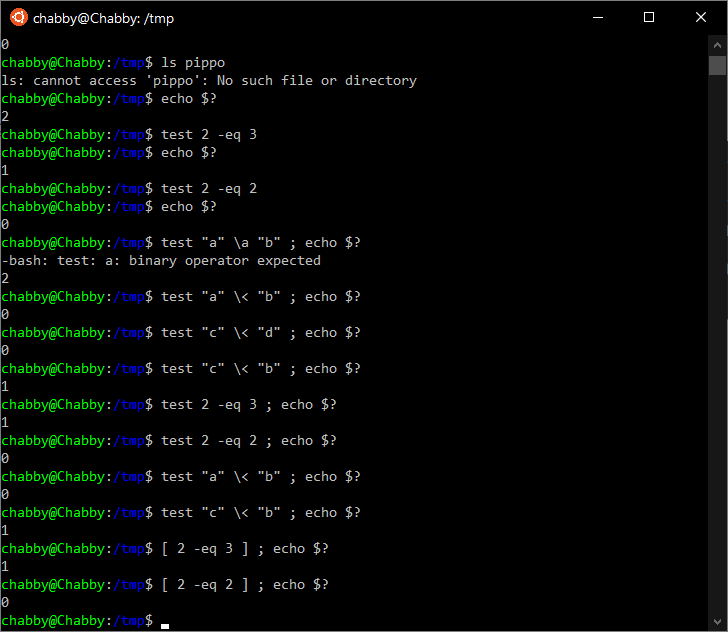
* **$((espressione)):** 
  + Usato per **elaborazione di operazioni matematiche** basilari.
  + Nel primo caso, la variabile x viene interpretata come variabile e viene computata l’operazione 3+4 = 7 prima di moltiplicare la x per 7, ottenendo così 49 come risultato finale.
  + Nel secondo caso, **l’interprete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



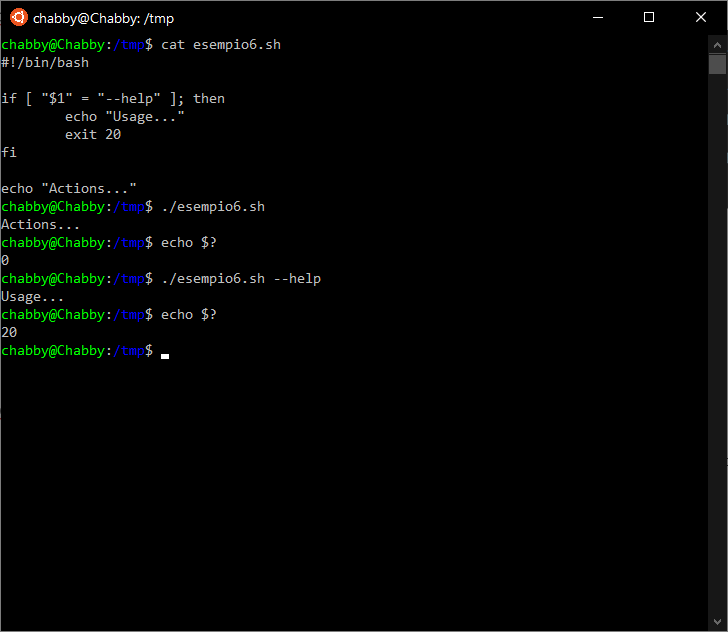
* Il comando echo genera un output e tramite **pipe** (il carattere **|**) questo output viene dato in input al comando successivo, in questo caso è **bc** che elabora e restituisce il risultato, che verrà stampato tramite echo.

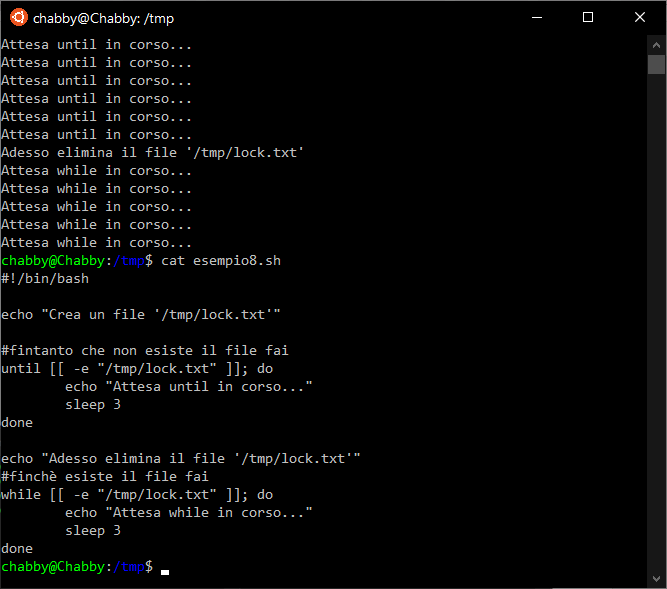
Logic:

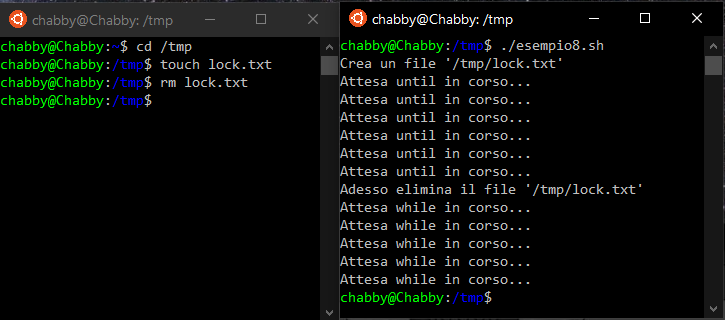
* Per valutare si usa il comando **test** che sfrutta il **codice di ritorno** di un comando per capire se ha avuto successo o meno (se ritorna 0 è andato a buon fine, qualsiasi valore diverso da 0 indica l’insuccesso).
* Per **confronti tra numeri** si usa:
  + **-eq, -ne, -lt, -le, -gt, -ge**
* Per **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:



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





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.

