LabSO

Introduzione: bash, make, gcc

Ambiente

- **s.o. Linux (Ubuntu / debian)**
- **Il terminale (CLI): la shell Bash**
- **editor esterno per codice/testi**

Bash – 1

- **Terminale singolo**
- **Terminali multipli**
- **Comandi diretti (clear, ls, df, free)**
- **File e cartelle: navigazione e permessi**
- **comandi multipli (*&&* || *;*)**
- **input/output/piping**

Bash – 2

- **scripting:**
  - **struttura di base: righe e commenti**
  - **hashbang/shebang**
  - **esempio “Hello World”...**

Bash – 3

- **“Hello World”**

(file “hello.sh”)

```
#!/bin/bash
#Script “Hello world”
echo Hello World
```

```
chmod +x hello.sh
```

```
./hello.sh
```

Bash – 4

- **Variabili:**
  - **assegnazione (*VAR=VALORE*)**
  - **utilizzo (*${VAR}*)**
  - **speciali:**
    - **$@, $#, $1 … $n e shift**
    - **altro (PID, vettori, etc.)**

Bash – 5

- **Aritmetica**
  - **espansione aritmetica intera: *$((…))*, esempio con variabili (espansione letterale, di valore)**

```
#!/bin/bash
x=2+3
echo "x=$x"
z1=$(( x*2 ))
z2=$(( $x*2 ))
echo "z1=$z1"
echo "z2=$z2"
```

  - ***bc* per aritmetica avanzata**

```
echo "$(echo '1.12 + 2.02' | bc)"
```

Bash – 6

- **Operatori logici e costrutti condizionali – 1**
  - **variabile *$?* per il codice di ritorno**
  - **testing semplice (*test: …* o *[…]*)**

```
#!/bin/bash
echo "res=$?"
x=3
test $x -eq 3
echo "res=$?"
[ $x -eq 0 ] # notare gli
spazi!
echo "res=$?"
```

  - **! (not) && || -eq -ne -lt -gt -f (file) -d (directory)**

Bash – 7

- **Operatori logici e costrutti condizionali – 2**
  - **[ ] built-in, [[ ]] keywords (es. con < e >)**
  - **If [[…]]; then … fi**
  - **If [[…]]; then … else … fi**
  - **case $var in**
    …|…) … ;;
    ***) … ;;**
    esac

```
#!/bin/bash
echo "Digita una lettera
e premi INVIO"
read char
case $char in
    y) echo "Input: y" ;;
    n) echo "Input: n" ;;
    *) echo "Input: nè y
nè n" ;;
esac
```

Bash – 8

- **Iterazioni (for anche su file, while e until)**

```
#!/bin/bash
for (( i = 1 , j = 1 ; i <= 10 ; i += 1, j *= 2 )); do
        echo "i=$i, j=$j"
done
```

```
#!/bin/bash
echo "Crea un file 'semaforo.txt'..."
# attende che un file sia creato
until [[ -e "semaforo.txt" ]] ; do sleep 3; done
echo "File creato, ora eliminalo..."
# attende che un file sia eliminato
while [[ -e "semaforo.txt" ]] ; do sleep 3; done
echo "File eliminato!"
```

Bash – 9

- **Funzioni**
  - **definizione:**
    ```
    func() {
        ...
    }
    ```
  - **invocazione:**
    ```
    func arg1 … argn
    ```
  - **variabili $1, …, $n e $@**

Bash – 10

- **Subshell: *$(…)*, esempio:**

```
DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
```

- **redirezionamento:**
  n>file n>&m (es. 1>log.txt 2>&1)
- **uscita: exit codice_errore (es. exit 1)**

Bash – 11

- **Esercizio per casa. Creare uno script che accetta come argomenti:**

  `-h` o `--help`: mostra un testo d'aiuto

  `--fibonacci <n>`: calcola l'n-esimo numero di fibonacci

  --output <nome>: stampa l'output su file

  es.: `./fibon.sh --fibonacci 5 --output fibon5.txt`

Make – 1

- **il tool “make” e il file di istruzioni**
- **struttura base: intestazione, tabulazioni e comandi**
- **invocazione make -f <nomefile>**

(file “istruzioni”)

```
comandi:
    echo "Ciao"
```

```
make -f istruzioni
```

Make – 2

- commenti (righe con #)
- regole (nome regola con “:”)
- comandi (righe con tabulazioni)
- esecuzione per righe come sub-shell (es. *cd*)
- regole multiple e regola di default
- “Makefile” di default
- evitare eco dei comandi (es.: @echo, @ls)

Make – 3

- **dependenze:**

  regola: dipendenza-1 … dipendenza-n

  (in generale una dipendenza è un'altra regola o un file di cui si controlla l'esistenza o l'aggiornamento rispetto all'esecuzione precedente)

- **pseudo regola .PHONY: …**
  (esplicitamente definisce un nome come regola e non lo considera un file)

Make – 4

- **definizione di “macro”**

(file “istruzioni”)

```
MACRO1=Pippo
main:
    @echo Hello $(MACRO1)
```

```
make -f prova
```

```
make -f prova MACRO1=Pluto
```

= (ad ogni occorrenza)

(se ad esempio la macro è definita come
MACRO1=$(MACRO2)
sarà elaborata al momento, anche se la seconda MACRO è definita successivamente)

:= (alla definizione)

Make – 5

- **funzioni speciali:**
  - **shell: per “catturare” l’output di un comando ad esempio per impostare una variabile, ad esempio:** `CONTENUTO=$(shell cat dati.txt)`
    attenzione: converte gli “a capo” in spazi ed elimina gli eventuali finali (anche più di uno)
  - **wildcard: per elencare una lista di files, ad esempio:** `LISTASRC=$(wildcard *.c)`
  - **altre… (utilità, manipolazione file o testi, etc.)**

Make – 6

- **esempio con macro**

```
CC        = gcc
CFLAGS    = -Wall -Wextra // extra warnings
foo : foo.o
      $(CC) -o foo foo.o $(CFLAGS)
foo.o : foo.c foo.h
      $(CC) -c foo.c $(CFLAGS)
```

Make – 7

- **Esercizio per casa. Creare un makefile con una regola `help` di default che mostri una nota informativa, una regola `backup` che crei un backup di una cartella appendendo ".bak" al nome e una `restore` che ripristini il contenuto originale. Per definire la cartella sorgente utilizzare una macro apposita.**

gcc - 1

- **compilazione: gli step.**

  `meta-source →source → assembly → binary`

- **gcc è un compilatore multi source/target, nel nostro caso per compilare progetti “C” in: assembly/oggetto/binario**

(esempio sorgente di base: “main.c”)

```
int main() {
    return 0;
}
```

gcc – 2

- esempio esecuzioni:
  - gcc main.c
  - gcc main.c -S
  - gcc main.c -E
  - gcc main.c -c
  - gcc main.c -o main
- riprovare aggiungendo `#include <stdio.h>` come prima riga del sorgente

gcc – 3

- il debugger gdb: cosa è e a cosa serve
- occorre tenere traccia dei simboli in fase di compilazione:
  `gcc -g -o main main.c`
- aprire una sessione: gdb ESEGUIBILE e poi utilizzare la shell interattiva:
  - run [...args…] per eseguire il programma
  - break <function_name, file:linea, …> per impostare un breakpoint
  - step <count> per fare uno (o più) step generici (*)
  - next <count> per fare uno (o più) step principali(*)
  - continue procede finchè possibile (es.: prox. bp)
  - help, print …, set [variable] …, quit,
  - Info, list, backtrace, … comandi di gestione

gcc – 4

- **esempio debugger**

(file “main.c”)

```
#include <stdio.h>
int main() {
    int i=0;
    char *s="Esempio";
    printf("TESTO='%s'\n", s);
    return (i);
}
```

```
gcc -g -o main main.c
gdb main
```

nella shell di debug in sequenza provare:

break main, run, step, print i, set variable i=3, print i, continue, quit

gcc – 5

- **Esercizio per casa. Creare una cartella con un sorgente “main.c” di prova e un makefile di supporto che effettui la compilazione con la regola “build” e cancelli i file generati con la regola “clean”**