Make

Documentazione
https://www.gnu.org/software/make/manual/html_node/

Giulio Salierno

giulio.salierno@unimore.it

gcc & make utility

- La compilazione del codice sorgente può risultare tediosa su progetti contenenti un grande numero di file
  - /home/my_project/main.c
  - /home/my_project/hello_world.c
  - /home/my_project/fattoriale.c
  - /home/my_project/function.h
- La compilazione manuale dei file può essere eseguita attraverso:
  - > gcc main.c hello_world.c fattoriale.c –o hello_world
- Compilare manualmente progetti contenenti centinaia di file sorgenti è dispendioso. La modifica di un singolo file sorgente comporta la ricompilazione dell'intero progetto!

Make Utility

- Assieme ai makefile permettono di compilare e gestire progetti di grandi dimensioni in maniera automatica
  - Per utilizzare il make è necessario creare un makefile all'interno della directory in cui sono presenti i file sorgenti da compilare
  - Mandando in esecuzione l'utility *make,* tale comando cercherà all'interno della directory corrente un file di nome *makefile* (o Makefile) e lo eseguirà
- La struttura base di un *makefile* è composta da:

  target: dependencies

  [tab] system command

  - *target:* rappresenta il nome del file eseguibile da generare o di un azione da intraprendere es. `clean` (Phony target)
  - dependencies: i file che il make utilizzerà per creare il target

Makefile

- system command: comandi eseguiti dalla utility make per la creazione del target
  - Un target può prevedere più comandi, ogni comando è preceduto da un *tab* (\t) iniziale

  - Esempio di un semplice makefile:

```
main: main.c factorial.c hello.c functions.h
        gcc -o main main.c factorial.c hello.c
```

Attenzione al carattere *tab*

Uso del make file:

```
$ make
 gcc -o main main.c factorial.c hello.c
$ make
 make: `main' is up to date
```

N.B. Se non è stato modificato nessun file sorgente il make non avrà nessuna azione da svolgere per il target main e quindi non produrrà un nuovo eseguibile

Makefile – target multipli

- Può essere utile utilizzare target differenti. In questo modo la modifica di un file sorgente non comporta la ricompilazione dell'intero progetto:

```
all: hello
hello: main.o factorial.o hello.o
        gcc main.o factorial.o hello.o -o hello
main.o: main.c
        gcc -c main.c
factorial.o: factorial.c functions.h
        gcc -c factorial.c
hello.o: hello.c functions.h
        gcc -c hello.c
clean:
        rm -rf  *o
```

tab ↗ tab ↗ tab ↗ tab ↗

- all è un default target contiene solo dipendenze ma non comandi
- clean è un phony target

Makefile – variabili

- Nella specifica del makefile è possibile definire della variabili prima della specifica dei target

```
#Utilizzo di variabili predefinite
#la variabile cc specifica il compilatore da utilizzare
CC=gcc
#opzioni utilizzate dal compilatore C
CFLAGS=-Wall
main: main.c factorial.c hello.c
        $(CC) $(CFLAGS) -o hello main.c factorial.c hello.c
```

tab ↗

- *È possibile accedere al valore delle variabili tramite l'operatore $(VAR)*
- *N.B. La sintassi del makefile è differente da quella shell!*

Makefile – regole implicite(1)

- Dati $f_i.c$ file sorgenti con $0 \leq i \leq 99$ si scriva un makefile che li compili con la seguente proprietà:
  - l'i-esimo eseguibile derivato da $f_i.c$ abbia nome f_i
  - È necessario specificare un target per ogni file ? No aumenterebbe la complessità. I makefile supportano i pattern rules:
  - `%` match con ogni stringa non vuota

    Ad esempio '%.c' fa match con tutti i file che terminano con .c
- Possiamo usare i pattern rules per definire una regola implicita
  - %: %.c

    $(CC) $(CFLAGS) –o $@ $<
- Ogni volta che la regola verrà eseguita la variabile $@ conterrà il nome corrente per quel target, $< il nome della prima dipendenza
- Le regole implicite permettono di specificare in maniera sintetica il modo di creare i file target

Makefile – regole implicite(2)

Soluzione:

```
# variabile cc specifica il compilatore da utilizzare
CC=gcc
#parametro utilizzato dal compilatore C
CFLAGS=-Wall
SRC = $(wildcard *.c) # wildcard per recupero file sorgenti
BIN = $(SRC:.c=) #specifica dei nomi dei file di output

all: $(BIN)

%: %.c
	$(CC) $(CFLAGS)  $< -o $@
```

tab ↗

Uso del make file:

```
$ make
gcc    f1.c -o f1
gcc    f2.c -o f2
gcc    f3.c -o f3
…
…
…
gcc    fi.c –o fi
$ make
make: Nothing to be done for `all'
```

N.B. Se non è stato modificato nessun file sorgente il make non avrà nessuna azione da svolgere per il target all e quindi non produrrà un nuovo eseguibile. Attenzione se tale messaggio viene mostrato la prima volta che si sta usando un makefile significa che c'è qualche errore nella specifica

Makefile – Esempio Esame

- Makefile semplice:

```
main: main.c
        gcc –Wall –o main main.c
```

tab ↗

- Makefile con variabili e regole implicite:

```
CC=gcc
CFFLAGS=-Wall
BIN=main
all: $(BIN)
%: %.c
        $(CC) $(CFLAGS) -o $@ $<
```

tab ↗