Lezione 11: Makefile

Laboratorio di Elementi di Architettura e Sistemi Operativi

23 Maggio 2012

Compilazione di progetti software complessi

Il `Makefile`: cos'è?

- Nella lezione scorsa abbiamo visto come suddividere il codice in più parti
  - *Molti file sorgenti ⇒ compilazione snervante!*
- Per automatizzare la compilazione dei sorgenti l'ambiente UNIX mette a disposizione il comando `make`.

Il `Makefile`
In ogni directory che contiene sorgenti è necessario creare un file di testo, chiamato `Makefile`, che contiene le istruzioni per la compilazione.

Come avviare la compilazione
Una volta creato il `Makefile` è sufficiente lanciare il comando `make` (o `make obiettivo`) per avviare la compilazione.

Struttura del `Makefile`

Sintassi delle regole

```
# commento

obiettivo: prerequisiti
  <tab>  comando
  <tab>  comando
  <tab>  ...
```

Spiegazione

`obiettivo`
: Il file da creare

`prerequisiti`
: I file che servono per crearlo

`comando`
: I comandi (si noti il *`tab`* obbligatorio) per creare l'obiettivo.

Cosa fa il `make`?
Controlla se qualcuno frai prerequisiti è stato modificato più recentemente dell'obiettivo. In caso affermativo esegue i comandi.

Un esempio banale

`Makefile`

```
grosso.txt: piccolo1.txt piccolo2.txt
        cat piccolo1.txt piccolo2.txt > grosso.txt
```

- Il comando `make grosso.txt` controlla che il file `grosso.txt` non esista oppure sia più vecchio dei file `piccolo1.txt` e `piccolo2.txt`.
- Quindi esegue il comando `cat` che crea il file `grosso.txt`.
- Nel caso l'obiettivo sia già aggiornato notifica la cosa con la seguente stringa:

```
make: 'grosso.txt' is up to date.
```

Un esempio più complicato

Makefile

```
program: main.o lib.o
        gcc -o program main.o lib.o

main.o: main.c lib.h
        gcc -c main.c

lib.o: lib.c lib.h
        gcc -c lib.c

clean:
        rm *.o program
```

- Il comando `make` (senza parametri) esegue il primo target del `Makefile`.
- `make clean` equivale ad eseguire `rm *.o program`

Variabili

- Nei `Makefile` è possibile utilizzare delle variabili per evitare di riscrivere più volte gli stessi comandi.

Makefile

```
CFLAGS = -Wall
CC = gcc

test.o : test.c
        $(CC) $(CFLAGS) -c test.c
saluti.o : saluti.c
        $(CC) $(CFLAGS) -c saluti.c
```

- Le variabili possono apparire in qualsiasi punto del `Makefile`, obiettivi e dipendenze compresi
- Si possono usare anche le variabili d'ambiente (come `$(HOME)` o `$(SHELL)`).

Regole implicite e variabili automatiche

- Spesso si deve eseguire lo stesso comando su più file.

Makefile

```
%.o : %.c
        gcc -c -o $@ $<
```

- Regole implicite:
  - La regola trasforma qualunque file `.c` in un file `.o`
- Variabili automatiche:

  `$@` Bersaglio

  `$<` Primo prerequisito

  `$^` Tutti i prerequisiti

Un esempio completo

```
CC = gcc
OBJECTS = main.o lib.o
EXEC = program

$(EXEC): $(OBJECTS)
        gcc -o $@ $^

%.o : %.c
        gcc -c -o $@ $<

clean:
        rm $(OBJECTS) $(EXEC)

lib.o: lib.h
main.o: lib.h
```