

---

# Makefile - 1

Quando i progetti hanno molti file, l'uso di un makefile facilita la compilazione. Il makefile capisce quale file deve essere ricompilato e fa il link nuovamente solo di questi file.

Esso evita che si riscrivano comandi di compilazione. Evita pure che si facciano compilazioni non necessarie, cioè i file compilati vengono scoperti e confrontati con i sorgenti (quelli da cui dipendono).

Un semplice makefile è composto da una linea di dipendenza, che dice da cosa dipende il comando che si vuol eseguire ('target') ed una linea che dice come ottenere quel comando.

```
target: prerequisiti
        comandi
        ...
        ...
```

Il testo della prima linea deve iniziare nella prima colonna – non accetta spazi all'inizio. La prima linea specifica un 'target' e dopo un ':' i file da cui dipende quel target. (Prerequisiti = file di input necessari per generare quel target).

La seconda linea deve iniziare con un TAB (non la scritta 'TAB') e contiene cosa il programmatore scriverebbe per ottenere quel target.

Esempi di attività che si possono fare sono: generare un eseguibile, fare una installazione, comprimere i file ed inserirli in un archivio, etc.

---

# Makefile - 2

Per eseguire un makefile basta scrivere `make target` se il nome del file è `Makefile`, altrimenti `make -f nomefile target`

Un commento nel file inizia con un `#` e continua fino alla fine della linea.

Es. Creazione di un makefile che svolge le attività:

- creazione di un eseguibile chiamato `stack` a partire dai file sorgenti,
- creazione di un file di archivio `stack.tar.gz` che contiene sorgenti ed eseguibile,
- eliminazione di file eseguibile ed archivio.

Commento:

```
### Il mio primo makefile
```

Target e dipendenze

```
stack: main_stack.c stack.c
```

Comandi per ottenere quel target

```
gcc main_stack.c -o stack
```

```
gzip: main_stack.c stack.c stack
tar -cvf stack.tar main_stack.c stack.c stack
gzip stack.tar
```

clean:

```
rm stack.tar.gz stack
```

---

# Makefile - 3

Un comando di makefile che necessita di una linea lunga può essere spezzato in più linee per mezzo di un backslash '\'

```
rm stack.tar.gz \  
    stack
```

`make` elabora la regola del comando che abbiamo richiesto, ma prima considera i prerequisiti e se necessario elabora prima le regole che permettono di soddisfare i prerequisiti.

In un makefile possiamo definire delle macro (sono simili a variabili che contengono testo o comandi) ed usarle successivamente.

Definizione della macro `oggetti`:

```
oggetti = main_stack.c stack.c stack
```

L'espansione della macro si fa sempre attraverso `$(...)`

```
gzip: $(oggetti)  
    tar -cvf stack.tar $(oggetti)  
    gzip stack.tar
```

In tal modo evitiamo di dover riscrivere la lista dei file più volte.

Un makefile può contenere delle direttive, per includere altri makefile.

```
include filename
```

il programma `make` interrompe la lettura del makefile, per leggere ciascun makefile incluso. Alla fine riprenderà la lettura del makefile iniziale. I file inclusi possono avere delle macro condivise (definite in uno di essi).

---

# Makefile - 4

Possiamo scrivere delle condizioni all'interno di un makefile, per es. per specificare quali librerie includere quando si compila con `gcc`

`ifeq` è la direttiva per iniziare una condizione e verifica se i 2 argomenti, separati da virgola, sono uguali. Le linee seguenti sono elaborate se la condizione è verificata.

```
ifeq ($(CC), gcc)
    $(CC) -o foo $(oggetti) $(lib_gcc)
else
    $(CC) -o foo $(oggetti) $(normal_lib)
endif
```

Si può opzionalmente specificare un `else`

`endif` termina la direttiva di condizione.

La direttiva `ifneq(arg1, arg2)` eseguirà le linee dopo `ifneq` se i due argomenti sono diversi.

La direttiva `ifdef nomevariabile` verifica se la variabile specificata è non vuota ed in caso positivo esegue le linee seguenti.

```
ifdef CC
@echo CC è definita
endif
```

`ifndef` è la direttiva per verificare se una variabile è vuota.

---

# Makefile - 5

Le regole implicite si applicano a tutti i file che hanno una certa estensione.

Sintassi:

```
[s-dir].source-extension.[t-dir][target-extension]:  
    [comando]
```

Per avere il target che l'utente ha inserito, ci vengono in aiuto alcune macro predefinite (la cui espansione non necessita ( )):

`$$` restituisce il target completo (con il percorso)

`$$&` restituisce il nomefile, senza estensione e senza percorso

Esempio:

```
.c:  
    gcc $$ -o $$&
```

quando eseguiamo `make nomefile`, viene eseguito  
`gcc nomefile.c -o nomefile`