

Il gcc in breve

Autore: Matteo Lucarelli
ultima versione su: www.matteolucarelli.net
Documento in costruzione

Il presente documento vuole essere un breve prontuario degli usi del gcc (GNU compiler collection). Illustra quindi solo alcune delle innumerevoli possibilità del noto compilatore, ed in particolare solo i parametri di uso più comune nella programmazione C e C++.

Per una trattazione completa si veda la [documentazione ufficiale GNU](#).

NOTA: il gcc ammette moltissime opzioni, anche multicarattere, quindi le stesse non vanno mai raggruppate (cioè "-d -D" non è uguale a "-dD")

Alcuni esempi d'uso

Compilazione di un singolo file durante lo sviluppo (debug):

```
gcc -g -Wall main.c -o nomefileoggetto.o
```

Compilazione definitiva di un eseguibile statico (includendo ad esempio anche la libm):

```
gcc -O3 -lm -static -o nomeprogramma main.c file1.c file2.c ...filen.c
```

Produzione di una libreria condivisa (il sorgente deve ovviamente essere scritto rispettando le necessarie convenzioni):

```
gcc -shared -o libnomelibreria.so file1.c file2.c ...filen.c
```

Flag generici

Il gcc è una suite di compilatori. I linguaggi supportati sono: c, c++, objective-c, objective-c++, assembler, ada, f77, f95, java.

Per specificare il linguaggio sorgente, che normalmente viene dedotto dall'estensione del file, si utilizza il flag:

```
-x linguaggio
```

In assenza di specifica il comando compila, assembla e linka il sorgente in oggetto. Per effettuare delle operazioni parziali si utilizzano i flag:

```
-c : compila e/o assembla ma non linka  
-S : compila ma non assembla
```

In alcuni casi è comodo poter definire delle macro esternamente al codice (ad esempio nel makefile), utilizzabili ad esempio come flag, all'interno del codice:

```
-D name           : predefinisce la macro name (valore di default è 1)  
-D name=definition : predefinisce la macro name  
-include file     : come se si aggiungesse "#include file"  
                  come prima riga della compilazione
```

I path di ricerca dei file accessori sono normalmente definiti da alcune variabili d'ambiente (si veda nel seguito). Per modificarli:

-Idir : aggiunge dir ai path di ricerca degli header file
-Ldir : aggiunge dir ai path di ricerca delle librerie

Controllo dell'output

Per definire il nome del file generato:

-o file : imposta il file di output

Il file oggetto può includere o meno diverse informazioni per la successiva analisi tramite altri strumenti. I flag più comuni sono:

-g -g1 -g3 : include le informazioni necessarie al debugging.
in particolare:
 -g1 produce informazioni minime
 -g2 massime
-ggdb : include le informazioni di debugging specifiche per il gdb
-p -pg : include le informazioni necessarie al profiling (con prof o gprof)

Il livello di ottimizzazione del binario può essere definito tra 0, nessuna ottimizzazione-default, e 3, massima ottimizzazione.

-On : utilizza il livello n di ottimizzazione
-Os : minimizza la dimensione del binario.

Inoltre, per produrre librerie condivise:

-shared : produce un oggetto linkabile dinamicamente

E' inoltre possibile effettuare delle cross-compilazioni, ovvero generare del codice binario per una architettura differente da quella di sviluppo. Naturalmente in tal caso sarà necessario disporre di tutta la toolchain necessaria (header, librerie, ecc.).

-b architettura : specifica l'architettura target del binario prodotto

Linking

La convenzione per i nomi delle librerie richiede che il file contenente la libreria si chiami "liblibreria.a". In tal caso:

-llibreria : linka la libreria specificata.

Mentre per linkare staticamente:

-static : evita il linking dinamico

Output di console

Durante la compilazione si possono produrre due tipi di avvertimenti:

- **warning** : avvertimenti, che non impediscono il compimento delle operazioni in corso
- **error** : errori, che sono invece bloccanti

Le relative impostazioni di default possono essere modificate tramite i flag:

-Wall : stampa tutti gli warning considerati utili ed evitabili
-Werror : trasforma tutti gli warning in errori
-w : sopprime tutti gli warning
-pedantic : warning pignolo relativamente agli standard ISO del linguaggio

E' possibile inoltre analizzare il lavoro del preprocessore:

-E : mostra il risultato del preprocessing (con -P senza linemarkers)
-dM -dD -dI : genera una lista di #define per tutte le macro definite nel codice. In particolare:
M: tutte
D: esclude le predefinite
I: anche gli include

Per usi particolari (generazioni di script o di makefile) sono inoltre disponibili le seguenti opzioni:

-### : stampa i comandi necessari e le impostazioni utilizzate
-M|-MM -MF file : crea una regola per makefile, in particolare:
-MM non lista tra le dipendenze gli header di sistema
MF (opzionale) specifica il file di output
-MD|-MMD : come -M -MF oppure -MM -MF

Infine:

-v : genera un output verboso

Variabili d'ambiente

Il gcc utilizza alcune variabili d'ambiente, le più comuni sono:

CPATH : percorsi di ricerca degli header file
LIBRARY_PATH : percorsi di ricerca delle librerie
COMPILER_PATH : percorsi di ricerca dei subprogrammi
utilizzati dal gcc (linker, assembler, ecc.)
LANG ,
LC_CTYPE ,
LC_MESSAGES ,
LC_ALL : impostazioni locali, prevalentemente relative al linguaggio