

gdb - The GNU debugger

Introduzione al debugger più famoso nel mondo GNU/Linux

Alessandro Galli

Montebelluna Linux User Group

11 novembre 2007 – Montebelluna



Licenza d'utilizzo

Copyright © 2007, Alessandro Galli.

Questo documento viene rilasciato secondo i termini della licenza Creative Commons (<http://creativecommons.org>).

L'utente è libero di:

distribuire, comunicare al pubblico, rappresentare o esporre in pubblico la presente opera

alle seguenti condizioni:

Attribuzione Deve riconoscere la paternità dell'opera all'autore originario.

Non commerciale Non può utilizzare quest'opera per scopi commerciali.

No opere derivate Non può alterare, trasformare o sviluppare quest'opera.

In occasione di ogni atto di riutilizzo o distribuzione, deve chiarire agli altri i termini della licenza di quest'opera.

Se ottiene il permesso dal titolare del diritto d'autore, è possibile rinunciare a ciascuna di queste condizioni. Le utilizzazioni libere e gli altri diritti non sono in nessun modo limitati da quanto sopra. Questo è un riassunto in lingua corrente dei concetti chiave della licenza completa (codice legale), reperibile sul sito Internet

<http://creativecommons.org/licenses/by-nc-nd/2.0/legalcode>



Sommario

- 1 Intro
- 2 Avviare il programma
 - Preparazione
 - Avviare gdb
 - TUI
 - Avviare il debug
- 3 Bloccare il programma
 - Breakpoint
 - Stepping
- 4 Examine
 - Stack
 - Variables
- 5 Modify
 - Set variables
 - Jump!
 - Others
- 6 Conclusion



Essere un programmatore...

Linus Torvalds: "95% of Programmers consider themselves in the top 5%".

Essere un hacker...

Hackers sees bug. Hackers fixes bug.



Documentazione

La fonte di documentazione ufficiale:

- `info gdb`

Installazione in sistemi debian

Utilizzare il comando:

- `#apt-get install gdb-doc`



Compilare il programma

Come compilare il programma

- Utilizzare l'opzione `-g` per salvare la symbols table nell'eseguibile del programma.
- Utilizzare l'opzione `-O0` per eliminare l'ottimizzazione.

Compilare il programma d'esempio

```
gcc -lc -O0 -g example.c -o example
```



Avviare un programma in debug

Avviare gdb

```
gdb <nome programma>
```

Altre basilari opzioni

- `-d <dir>` aggiunge `<dir>` tra le cartelle in cui vengono cercati i sorgenti.
- `-x <file>` esegue i comandi gdb presenti nel file `<file>`.

Possibilità

E' possibile debuggare...

- un programma già avviato
- un programma in uno stato preciso (usando un core file)
- un programma in un altro computer



Interfaccia friendly

Per agevolare il lavoro è possibile abilitare una interfaccia testuale con il comando:

- `(gdb) set layout src`
- `C-x a`

Comandi TUI

Alcuni comandi in modalità TUI:

- `C-x o` Cambia finestra attiva
- `C-L` Refresh dello screen



Con TUI vi innamorerete di gdb!



Cominciare l'esecuzione

Primi comandi

- (gdb) `set args <args>` Impostare gli argomenti per il programma.
- (gdb) `set environment <envs>` Modificare le variabili d'ambiente.
- (gdb) `tty <tty>` Imposta <tty> come terminale per il programma.
- (gdb) `run` **via!**



Come fermo il programma

Breakpoints

- `break <dove>` ferma l'esecuzione del programma nel punto indicato
- `info break` stampa una tabella dei breakpoints
- `disable <num>` disabilita un breakpoint
- `clear <quale>` cancella un breakpoint

Esempi di breakpoint

- `break file.c:45`
- `break main`
- `break +30`



r 07



L'evoluzione dei breakpoints

Breakpoints condizionali

E' possibile indicare che un break point fermi l'esecuzione del programma solo se si verificano determinate condizioni:

- `break 37 if num > 4` ferma il programma alla riga 37 solo se la variabile `num` è maggiore di 4
- `condition <num>` cancella la condizione del breakpoint numero `<num>`
- `break 37 ignore 3` ferma il programma solo la terza volta che la riga 37 viene eseguita

Comandi associati a breakpoints

E' possibile indicare dei comandi da eseguire quando un breakpoint viene raggiunto:

- `break 37`
- `commands`

Watchpoint

Ferma il programma quando il valore di un'espressione cambia.

Catchpoint

Ferma il programma quando accade un evento.

Alcuni esempi

- (gdb) watch a
- (gdb) watch a+b
- (gdb) catch exec
- (gdb) catch fork





E per ripartire?

Per riprendere l'esecuzione vi sono i comandi:

- (gdb) `continue` continua l'esecuzione
- (gdb) `step 3` esegui tre volte un'istruzione (entra nelle funzioni)
- (gdb) `next` continua l'esecuzione fino alla riga successiva (non entra nelle funzioni)
- (gdb) `finish` continua fino a che la funzione ritorna
- (gdb) `until 34` continua fino alla riga 34 (o altra posizione indicata)



Signals

(gdb) `info signals` per vedere come sono gestiti i segnali da gdb
e (gdb) `handle SIGNUM nostop|stop|print` per modificare il modo in cui sono gestiti.

Threads

(gdb) `break <where> thread 28` imposta il breakpoint in <where> solo per il thread n° 28. Per vedere i thread attivi (gdb) `info threads`.





Dove sono?

Per visualizzare lo stack (e gli stack frames) è possibile usufruire dei seguenti comandi:

- `backtrace` per vedere lo stack di frames (ognuno con il suo numero)
- `backtrace full bt` con variabili locali
- `frame <framenum>` seleziona e stampa il frame numero `<framenum>`



Visualizzare le variabili

Comandi per visualizzare variabili

- (gdb) `print <nome variabile>` stampa una variabile (eventualmente è possibile specificare un formato).
- (gdb) `x /NFU <indirizzo in memoria>` per stampare dall'<indirizzo di memoria>, per N volte, una variabile di lunghezza U (byte, half word, word ...) formattandola secondo F
- (gdb) `display <nome variabile>` stampa una variabile ogni volta che il programma si ferma

```
      F
    B   C
  P T E O
B Z F E D
O F C L T B
```

70
60
50
40
30





Print e set

Print si può usare anche per impostare il valore di una variabile: `(gdb) print a=4` oltre a mostrare il risultato dell'espressione assegna 4 alla variabile `a`.

La stessa cosa può essere fatta con `set variable a=4` senza che il valore dell'espressione venga mostrato.



Saltare in qua e in là nel programma

Per saltare da un punto all'altro del programma

- `jump <linenum>` salta alla riga corrispondente e fa partire il programma, a meno che in quella riga non ci sia un breakpoint.



return

Per far ritornare una funzione è possibile utilizzare il comando `(gdb) return EXPR` che fa ritornare la funzione e come valore di ritorno viene utilizzato il risultato di *EXPR*

Segnali

Per lanciare un segnale al programma usare il comando `(gdb) signal SIGNAL`.

Chiamate di funzioni

Per chiamare una funzione all'interno del vostro programma utilizzare `(gdb) print nomefunzione` per vederne il risultato o `(gdb) call nomefunzione` per non mostrare il valore di ritorno o se la funzione ritorna void.



Bye Bye

Fine

Ringrazio tutti per l'attenzione. Alla prossima!

Autore

- Alessandro Galli
- <http://www.montellug.it>
- <http://krdm.sourceforge.net>
- alessandro.galli@gmail.com





info gdb

Manuale gdb

info gdb

