NAME

gdb − The GNU Debugger

SYNOPSIS

gdb [OPTIONS] [*prog*|*prog procID*|*prog core*]

DESCRIPTION

The purpose of a debugger such as GDB is to allow you to see what is going on ‘‘inside’’ another program while it executes — or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

You can use GDB to debug programs written in C, C++, Fortran and Modula−2.

GDB is invoked with the shell command `gdb`. Once started, it reads commands from the terminal until you tell it to exit with the GDB command `quit` or `exit`. You can get online help from GDB itself by using the command `help`.

You can run `gdb` with no arguments or options; but the most usual way to start GDB is with one argument or two, specifying an executable program as the argument:

```
gdb program
```

You can also start with both an executable program and a core file specified:

```
gdb program core
```

You can, instead, specify a process ID as a second argument or use option `−p`, if you want to debug a running process:

```
gdb program 1234
gdb −p 1234
```

would attach GDB to process `1234`. With option **−p** you can omit the *program* filename.

Here are some of the most frequently needed GDB commands:

break [*file***:][***function***|***line***]**
Set a breakpoint at *function* or *line* (in *file*).

run [*arglist***]**
Start your program (with *arglist*, if specified).

bt Backtrace: display the program stack.

print *expr*
Display the value of an expression.

c Continue running your program (after stopping, e.g. at a breakpoint).

next
Execute next program line (after stopping); step *over* any function calls in the line.

edit [*file***:]***function*
look at the program line where it is presently stopped.

list [*file***:]***function*
type the text of the program in the vicinity of where it is presently stopped.

step
Execute next program line (after stopping); step *into* any function calls in the line.

help [*name***]**
Show information about GDB command *name*, or general information about using GDB.

quit
exit
Exit from GDB.

For full details on GDB, see *Using GDB: A Guide to the GNU Source-Level Debugger*, by Richard M. Stallman and Roland H. Pesch. The same text is available online as the `gdb` entry in the `info` program.

OPTIONS

Any arguments other than options specify an executable file and core file (or process ID); that is, the first argument encountered with no associated option flag is equivalent to a **−−se** option, and the second, if any, is equivalent to a **−c** option if it's the name of a file. Many options have both long and abbreviated forms; both are shown here. The long forms are also recognized if you truncate them, so long as enough of the option is present to be unambiguous.

The abbreviated forms are shown here with **−** and long forms are shown with **−−** to reflect how they are shown in **−−help**. However, GDB recognizes all of the following conventions for most options:

```
−−option=value
−−option value
−option=value
−option value
−−o=value
−−o value
−o=value
−o value
```

All the options and command line arguments you give are processed in sequential order. The order makes a difference when the **−x** option is used.

−−help
−h List all options, with brief explanations.

−−symbols=*file*
−s *file*
Read symbol table from *file*.

−−write
Enable writing into executable and core files.

−−exec=*file*
−e *file*
Use *file* as the executable file to execute when appropriate, and for examining pure data in conjunction with a core dump.

−−se=*file*
Read symbol table from *file* and use it as the executable file.

−−core=*file*
−c *file*
Use *file* as a core dump to examine.

−−command=*file*
−x *file*
Execute GDB commands from *file*.

−−eval−command=*command*
−ex *command*
Execute given GDB *command*.

−−init−eval−command=*command*
−iex
Execute GDB *command* before loading the inferior.

−−directory=*directory*
−d *directory*
Add *directory* to the path to search for source files.

−−nh
Do not execute commands from *˜/.config/gdb/gdbinit*, *˜/.gdbinit*, *˜/.config/gdb/gdbearlyinit*, or *˜/.gdbearlyinit*

−−nx
−n Do not execute commands from any *.gdbinit* or *.gdbearlyinit* initialization files.

−−quiet
−−silent
−q ‘‘Quiet’’. Do not print the introductory and copyright messages. These messages are also suppressed in batch mode.

−−batch
Run in batch mode. Exit with status `0` after processing all the command files specified with **−x** (and *.gdbinit*, if not inhibited). Exit with nonzero status if an error occurs in executing the GDB commands in the command files.

Batch mode may be useful for running GDB as a filter, for example to download and run a program on another computer; in order to make this more useful, the message

```
Program exited normally.
```

(which is ordinarily issued whenever a program running under GDB control terminates) is not issued when running in batch mode.

−−batch−silent
Run in batch mode, just like **−−batch**, but totally silent. All GDB output is supressed (stderr is unaffected). This is much quieter than**−−silent** and w ould be useless for an interactive session.

This is particularly useful when using targets that give **Loading section** messages, for example.

Note that targets that give their output via GDB, as opposed to writing directly to `stdout`, will also be made silent.

−−args *prog* **[***arglist***]**
Change interpretation of command line so that arguments following this option are passed as arguments to the inferior. As an example, take the following command:

```
gdb ./a.out −q
```

It would start GDB with **−q**, not printing the introductory message. On the other hand, using:

```
gdb −−args ./a.out −q
```

starts GDB with the introductory message, and passes the option to the inferior.

−−pid=*pid*
Attach GDB to an already running program, with the PID *pid*.

−−tui
Open the terminal user interface.

−−readnow
Read all symbols from the given symfile on the first access.

−−readnever
Do not read symbol files.

−−dbx
Run in DBX compatibility mode.

−−return−child−result
GDB's exit code will be the same as the child's exit code.

−−configuration
Print details about GDB configuration and then exit.

−−version
Print version information and then exit.

−−cd=*directory*
Run GDB using *directory* as its working directory, instead of the current directory.

−−data−directory=*directory*
−D Run GDB using *directory* as its data directory. The data directory is where GDB searches for its auxiliary files.

−−fullname
−f Emacs sets this option when it runs GDB as a subprocess. It tells GDB to output the full file name and line number in a standard, recognizable fashion each time a stack frame is displayed (which includes each time the program stops). This recognizable format looks like two **\032** characters, followed by the file name, line number and character position separated by colons, and a newline. The Emacs-to-GDB interface program uses the two **\032** characters as a signal to display the source code for the frame.

−b *baudrate*
Set the line speed (baud rate or bits per second) of any serial interface used by GDB for remote debugging.

−l *timeout*
Set timeout, in seconds, for remote debugging.

−−tty=*device*
Run using *device* for your program's standard input and output.

SEE ALSO

The full documentation for GDB is maintained as a Texinfo manual. If the `info` and `gdb` programs and GDB's Texinfo documentation are properly installed at your site, the command

```
info gdb
```

should give you access to the complete manual.

Using GDB: A Guide to the GNU Source-Level Debugger, Richard M. Stallman and Roland H. Pesch, July 1991.

COPYRIGHT