FREE eBook

LEARNING GDB

Free unaffiliated eBook created from
Stack Overflow contributors.

#gdb

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gdb

It is an unofficial and free GDB ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official GDB.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with GDB

Remarks

GDB (GNU Project debugger) is a command line base debugger that is good at analyzing running and cored programs. According to the user manual GDB supports C, C++, D, Go, Objective-C, Fortran, Java, OpenCL C, Pascal, Rust, assembly, Modula-2, and Ada.

GDB has the same feature set as most debuggers but is different from most that I have used in that is all based on typing commands instead of clicking on GUI elements. Some of these features include:

- Setting break points
- Printing the value of variables.
- Setting the value of variables to test things out.
- Viewing the stack

Examples

What is GDB?

GDB, short for GNU Debugger, is the most popular debugger for UNIX systems to debug C and C++ programs. GNU Debugger, which is also called gdb, is the most popular debugger for UNIX systems to debug C and C++ programs.

GNU Debugger helps you in getting information about the following:

- If a core dump happened, then what statement or expression did the program crash on?
- If an error occurs while executing a function, what line of the program contains the call to that function, and what are the parameters?
- What are the values of program variables at a particular point during execution of the program?
- What is the result of a particular expression in a program?

Starting GDB

To start GDB, in the terminal,

```
gdb <executable name>
```

For the above example with a program named main, the command becomes

```
gdb main
```

Setting Breakpoints

You'll probably want you program to stop at some point so that you can review the condition of your program. The line at which you want the program to temporarily stop is called the breakpoint.

```
break <source code line number>
```

Running your program

To run your program, the command is, as you guessed,

```
run
```

Opening a core

```
gdb -c coreFile pathToExecutable
```

Working with a Core File

Created this really bad program

```
 #include <stdio.h>
 #include <ctype.h>

// forward declarations


void bad_function()
{

   int *test = 5;

   free(test);

}


int main(int argc, char *argv[])
{
   bad_function();
   return 0;
}


gcc -g ex1.c

./a.out   //or whatever gcc creates
Segmentation fault (core dumped)

gdb -c core a.out

Core was generated by `./a.out'.
```

Program terminated with signal SIGSEGV, Segmentation fault. #0 __GI___libc_free (mem=0x5) at malloc.c:2929 2929 malloc.c: No such file or directory.

```
(gdb) where
```

#0 __GI___libc_free (mem=0x5) at malloc.c:2929 #1 0x0000000000400549 in bad_function () at ex1.c:12 #2 0x0000000000400564 in main (argc=1, argv=0x7fffb825bd68) at ex1.c:19

Since I compiled with -g you can see that calling where tells me that it didn't like the code on line 12 of bad_function()

Then I can examine the test variable that I tried to free

```
(gdb) up
```

#1 0x0000000000400549 in bad_function () at ex1.c:12 12 free(test);

```
(gdb) print test
```

$1 = (int *) 0x5

```
(gdb) print *test
```

Cannot access memory at address 0x5

In this case the bug is pretty obvious I tried to free a pointer that was just assigned the address 5 which wasn't created by malloc so free has no idea what to do with it.

Read Getting started with GDB online: https://riptutorial.com/gdb/topic/4964/getting-started-with-gdb

Credits

S. No	Chapters	Contributors
1	Getting started with GDB	Community, PSN, Travis