

Introduction to gdb

Hisashi Todd Fujinaka

CS Tutors – Portland State University

May 3, 2005

This document describes why you want to use gdb, and some basic commands to use in gdb to debug your program. It is much easier to use ddd as a graphical front-end to gdb, but there are times when you just want a command-line without all the graphics.

1 What is gdb?

gdb is the GNU Debugger. It is useful for debugging programs written in C/C++ and assembly language. It is less useful when debugging threaded programs.

2 How to use gdb

2.1 Compile your program with the “-g” flag.

The “-g” flag compiles extra debugging information into your program. Without it, you’ll find that the output of gdb is cryptic. There is a drawback: the “-g” flag can make your file much larger.

```
g++ -g myfile.cpp
```

2.2 Start the debugger with your program

```
menkib:~ % gdb a.out
GNU gdb 4.18
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "sparc-sun-solaris2.7"...
(gdb)
```

2.3 Set a breakpoint in your program

A breakpoint tells the debugger where to stop in your program. After stopping, you can examine registers, examine and set variables, run your code one line at a time, etc.

If you don't set a breakpoint, the program can run all the way to the end of the program and out of the debugger.

Breakpoints can be set using **line numbers** or **function names**. One trick is to set the breakpoint at main.

```
(gdb) break main
Breakpoint 1 at 0x105bc: file myfile.c, line 6.
(gdb) run
Starting program: /u/toddf/a.out

Breakpoint 1, main (argv=0x1, argc=-4261756) at myfile.c:6
6          x = *argv[99];
```

2.4 Other breakpoint commands

The breakpoints currently set can be shown using *info break*.

The breakpoints can be modified using *ignore* to temporarily disable a breakpoint, *delete* to remove a breakpoint, or *clear*. (*delete* takes a breakpoint number, and *clear* takes a line number or function name.)

2.5 Setting conditions on breakpoints

Conditions can be set on breakpoints by using the *condition*.

```
(gdb) condition 1 root==NULL
(gdb)
```

2.6 Step through the program

The two commands to step through the program are *step* and *next*. *step* will go through a program one line at a time, or if you give it a number, it will step through that many lines of code.

next will step through code, skipping subroutines. This is often useful when you don't want to step into system subroutines. *next* can also take a number for the number of lines of code to execute.

2.7 Examining variables

It is useful to examine the contents of variable when debugging. One way to do this is to *print* the variables.

print can also take a format so the variable can be displayed as *octal*, *hexadecimal*, *decimal*, *unsigned decimal*, *t(binary)*, *float*, *address*, *instruction*, *char* and *string*.

```
(gdb) print x
$7 = 15 '\017'
(gdb) print /x x
$8 = 0xf
(gdb) print /u x
$9 = 15
(gdb)
```

Another way of showing the variables is to use *display* command that prints the message each time a command is executed until the “undisplay” command is used.

```
(gdb) display x
1: x = 0 '\000'
(gdb)
```

2.8 Setting variables

You can set variables within the code by using the *set* command.

```
(gdb) set x=15
(gdb) print x
$1 = 15 '\017'
```

3 Examining core dumps with gdb

Core dumps contain information about where a program is generating errors. Examining the core dumps pinpoints where the error occurs, and shows the data passed into the functions where the error occurs.

3.1 Compile the program with '-g'

```
g++ -g myfile.cpp
```

3.2 Start gdb with the program AND the core file

```
menkib:~ % gdb main core
GNU gdb 4.18
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "sparc-sun-solaris2.7"...
Core was generated by 'main'.
Program terminated with signal 11, Segmentation Fault.
Reading symbols from /usr/lib/libm.so.1...done.
```

```

Reading symbols from /usr/lib/libc.so.1...done.
Reading symbols from /usr/lib/libdl.so.1...done.
Reading symbols from /usr/platform/SUNW,Ultra-5_10/lib/libc_psr.so.1...done.
#0  Node<int>::next (this=0x0) at main.cc:28
28      Node<T>* next () const { return next_; }
(gdb)

```

3.3 See which function called the function where the crash occurs

Sometimes core dumps occur within system code, or the place where the core dump occurs isn't the place you need to look for errors. You can check the stack frames to see the nesting of the function calls by using the *backtrace* (bt) command.

```

(gdb) bt
#0  Node<int>::next (this=0x0) at main.cc:28
#1  0x267e0 in LinkedList<int>::remove (this=0x3c778,
    item_to_remove=@0xffbef804) at main.cc:77
#2  0x175b4 in main (argc=1, argv=0xffbef894) at main.cc:120

```

3.4 Move around the stack frames to check out variables

Change the stack frame to observe by using the *frame* command and the number of the stack frame from the *backtrace* command (bt).

```

(gdb) frame 2
#2  0x175b4 in main (argc=1, argv=0xffbef894) at main.cc:120
120      list->remove (1);
(gdb) print list
$1 = (LinkedList<int> *) 0x3c778

```

4 More information

More information can be found on the web, in the gdb book in the tutor lab, from the gdb man page, or from the gdb info page (on linux).

<http://www-2.cs.cmu.edu/~gilpin/tutorial/>

Feel free to contact CS tutors at tutors@cs.pdx.edu with any questions or suggestions.