

gdb – The GNU Debugger

Kurt Schmidt

Dept. of Computer Science, Drexel University

December 4, 2017

Intro

The GNU Debugger

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

- A debugger is closely tied to the compiler
- gdb is the command-line debugger for all GNU compilers
 - Language is irrelevant
 - Back end of the compiler is the same (for a given platform)
 - An executable is just a program; it's not a “C program”, nor a “FORTRAN program”, etc.

Invocation

Debugging a Program

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

- First, use the `-g` option, compile your program with extra (debuggin) information

```
$ gcc -g source files... -o prog
```

- Then, load the executable into the debugger:

```
$ gdb prog
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
...
(gdb) _
```

Commands

Using GDB

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

- GDB is very powerful
 - Attach to a running process
 - Examine a corefile
 - Debug multi-threaded programs
- Lots of commands
 - Don't be intimidated
 - I don't know many of them
 - Just knowing some of the basics will get you far

Getting Help

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

- GDB commands are divided into categories
- Type `help` to see these categories:

```
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without ...
user-defined -- User-defined commands
```


Getting Help – Listing a Class

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

- To see commands in a category (class):

```
(gdb) help running
Running the program.

List of commands:

continue -- Continue program being debugged
finish -- Execute until selected stack frame returns
jump -- Continue program being debugged at specified ...
kill -- Kill execution of program being debugged
next -- Step program
run -- Start debugged program
start -- Run the debugged program until the beginning ...
step -- Step program until it reaches a different source line
```

- I've only listed some of the handier commands

Getting Help on a Command

- Use `help cmd` for help on that command:

```
(gdb) help break
Set breakpoint at specified location.
break [PROBE_MODIFIER] [LOCATION] [thread THREADNUM] [if CONDITION]
PROBE_MODIFIER shall be present if the command is to be placed in a
probe point. Accepted values are '-probe' (for a generic,
automatically guessed probe type), '-probe-stap' (for a SystemTap
probe) or '-probe-dtrace' (for a DTrace probe).
LOCATION may be a linespec, address, or explicit location as
described below.
```

With no `LOCATION`, uses current execution address of the selected stack frame. This is useful for breaking on return to a stack frame.

`THREADNUM` is the number from "info threads".
`CONDITION` is a boolean expression.

...

Some Essential Commands

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

Note, many of the commands can be abbreviated.

<code>break b [<i>location</i>]</code>	Set breakpoint
<code>kill</code>	Kill running process
<code>run [<i>arglist</i>]</code>	Run your program
<code>print p [<i>expr</i>]</code>	Print <i>expr</i>
<code>step s</code>	Next line, stepping into functions
<code>next n</code>	Next line, stepping over functions
<code>continue c</code>	Continue to next break
<code>quit q</code>	Exit GDB

Running Your Program

`gdb` – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

<code>set args <i>args</i></code>	Set command-line arguments
<code>set env <i>var val</i></code>	Set environment <i>var</i> to <i>val</i> (for next run)
<code>show args</code>	Show command-line args
<code>show env [<i>var</i>]</code>	Show environment variables [or <i>var</i>]
<code>run [<i>args</i>]</code>	Run your program [with <i>args</i>]
<code>start [<i>args</i>]</code>	Run your program until beginning of main procedure
<code>kill</code>	Kill running process

Looking at Your Code

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

`list` or `l`

- `list`
- `list line_no`
- `list beg,end`
- `list file:line_no`
- `list func_name`

Setting Breakpoints

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

- A place (and/or condition) where execution pauses, waits for a user command
- Can break conditionally at a function or a line number

- `break func_name`
- `break line_no`
- `break file:line_no`
- `break ... if cond`

<code>info break</code>	<code>show breakpoints</code>
<code>delete [n]</code>	<code>delete breakpoints [breakpoint n]</code>
<code>disable [n]</code>	<code>disable breakpoints [breakpoint n]</code>
<code>enable [n]</code>	<code>enable breakpoints [breakpoint n]</code>

Execution Control

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

<code>step s</code>	Next line, stepping into functions
<code>next n</code>	Next line, stepping over functions
<code>continue c</code>	Continue to next break
<code>until loc</code>	Run until <i>loc</i> ; same args as <code>break</code>
<code>finish</code>	Run until frame returns
<code>return [expr]</code>	Pop frame w/out executing [using <i>expr</i>] as return value

Examining Data

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

```
print p [/f] expr
```

Prints *expr*. *f* is a format character

```
display [/f] expr
```

Prints *expr* each time execution
pauses

```
info display
```

Lists displayed expressions

```
undisplay n
```

Removes *n* from display list

The Call Stack

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

<code>backtrace</code> or <code>bt</code>	Print trace of all frames in stack
<code>frame [n]</code>	Select current frame [frame # <i>n</i>]
<code>info frame</code>	Information on selected frame
<code>info args</code>	Arguments of selected frame
<code>info locals</code>	Local variables of selected frame

Some Trickier (but Useful) Commands

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

`set var VAR=expr` Actually modify variables in the program being debugged

- Assignment operator from the language (e.g., :=)
- Keyword `var` is optional
- Useful when symbol name clashes with a GDB command

`jump line` Resume execution at *line*

`jump *address` Resume execution at *address*

Corefiles

Examining Corefiles

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

- A corefile is a snapshot of a process (image) in memory, when it died
- To allow corefiles on Linux (Bash)

```
$ ulimit -c unlimited
```

- Upon a crash, find the corefile, `core`
- Load the executable, along with the corefile, into the debugger

```
$ gdb prog -c core
```

- Examine the program:

```
(gdb) bt
```

- Note, `prog` needn't have been compiled with debug information

Summary

More Power

gdb – The
GNU
Debugger

Kurt Schmidt

Intro

Invocation

Commands

Getting Help

Essentials

Running

Listing

Breakpoints

Execution

Data

Call Stack

Trickier

Corefiles

Summary

- Only common commands (and uses) are shown here
- There is more functionality available
 - You can catch events and signals
 - Debuggers handle multi-threaded programs
 - Look at machine instructions
- Get comfortable with basic commands
 - This much will prove quite useful
- As you need more, explore