Cheatography

Closed-source Debugging with GDB Cheat Sheet

by fristle via cheatography.com/5574/cs/1012/

GDB: Launching

Launching GDB

gdb *programfile*	Start GDB ready to launch and debug *programfile*
gdb --args *program arg1 arg2*	Start GDB as above but supplying command line arguments to the target process.
gdb -p *pid*	Attach GDB to a running target process.

Selecting the Start of Debugging

gdb$ start	Run the debuggee and break at *main()* (if it exists).
gdb$ attach *pid*	Attach GDB to a running target process.

Adding a shim

gdb$ set exec-wrapper env 'LD_PRELOAD=*libfoo.so*'	The dynamic library file *libfoo.so* will be loaded into the address space of the debuggee.

Logging

gdb$ set logging file *filename*	The default logfile is gdb.txt but you can use this to change it.
gdb$ set logging overwrite off	The default is on, which overwrites the existing log file.
gdb$ set logging on	Turns on logging.
gdb$ echo *comment\n*	With logging on, this will add a comment to the logfile.

GDB: Environment

Controlling the environment

gdb$ show env	Display the debuggee's current environment variables.
gdb$ set env *varname* =*value*	Set an environment variable.
gdb$ unset env *varname*	Delete an environment variable.
gdb$ show args	Display the command-line arguments of the debuggee process.
gdb$ set args *arg1 arg2*	Set the command-line arguments to the debuggee process.
gdb$ shell *command*	Run shell commands (useful commands may include "ps -e", etc.)
gdb$ pwd | cd	These two commands can can show or change the working directory of GDB (useful for logging, etc.).

GDB: Execution

Displaying the Call Stack

gdb$ bt	Show the list of stack frames (BackTrace).
gdb$ bt full	Show the list of stack frames with the local variables of each.
gdb$ info frame	Show saved stack pointer, call address, etc. for the selected stack frame.
gdb$ frame *number*	Select stack frame number *number* (and crashed GDB 6.3.50 on OS X).

Controlling Execution

GDB: Execution (cont)

si [*count*]	Step-into (one or *count* instruction forward).
ni [*count*]	Step-over (one or *count* instruction, stepping over function calls).
return [*value*]	Immediately return from the current function, optionally setting the return value.
finish	Stop after finishing execution of the current function.
continue	Any time GDB is stopped, this will continue normal execution.

GDB: Memory

Memory Images

gdb *program* -c *memorydumpfile*	Debug *program* using a memory dump file, *imagefile*.
gdb$ generate-core-file	(not in Mac OS X) Dump the debuggee process memory to disk.

Reading Disassembly and Memory

gdb$ set disassembly -flavor intel	Use the modern syntax for x86-64 assembly. This is not the default.
gdb$ set disassemble -next-line on	Disassemble the next instruction every time GDB stops. You want to turn this on.
gdb$ x/4i 0x00001234	Disassemble (eXamine) the first 4 instructions at address 0x00001234.

By **fristle**
cheatography.com/fristle/

Published 2nd May, 2013.
Last updated 1st June, 2014.

GDB: Memory (cont)

gdb$ x/32i $rip	Disassemble the first 32 instructions starting at the current instruction ($RIP on x86-64).
gdb$ x/32i $rip-16	Same command, but attempting to disassemble both forward and backward from the current instruction.
gdb$ info address *symbolname*	Display the address in memory of a given symbol, specified by name.
gdb$ info symbol 0x00001234	Displays the symbol name (if any), executable segment, and executable module associated with the given address.
gdb$ x/1s 0x00001234	Display one null-terminated string at address 0x00001234.
gdb$ x/8xb 0x00001234	Display 8 heXadecimal Bytes of memory starting at address 0x00001234.
gdb$ info registers	Display the value of the regular CPU registers.

GDB: Memory (cont)

gdb$ info all-registers	Display the value of all CPU registers including floating-point and vector registers. Does not include special Machine Specific Registers (MSRs).
gdb$ find *start_address*, *distance*, *value* [, *another_value*, ...]	(not in Mac OS X) Search memory for a value, given a starting point and a search distance/offset.
gdb$ info shared	Display info about all of the executable modules of the debuggee (name, load address, file path, etc.).
gdb$ info functions	Display all of the function symbols available and their associated addresses.
gdb$ info variables	Display all of the variable symbols available and their associated addresses.

GDB: Breakpoints

Managing Breakpoints

gdb$ set breakpoint pending on	Bypasses the warning about breakpoints in modules that aren't loaded yet.

GDB: Breakpoints (cont)

gdb$ break *function*	Sets a breakpoint at *function* ***if*** ("pending" off) or ***when*** ("pending on") a symbol by that name exists.
gdb$ break *0x00001234	Sets a breakpoint at address 0x00001234.
gdb$ break *0x00001234 if somesymbol==somevalue*	This is an example of the conditional breakpoint syntax.
gdb$ catch syscall *name*	Stop when the syscall *name* is called. Omit *name* to stop on every syscall. Instead of name, you can also specify a syscall by number.
gdb$ catch load	(not in Mac OS X) Stop when the debuggee loads any dynamic library. Also: catch unload.
gdb$ info break	List all breakpoints and watchpoints.
gdb$ clear [*breakpointid*]	Deletes one or all existing breakpoints. This is a typically ambiguous command exemplifying the need for this cheat sheet.
gdb$ disable [*breakpointid*]	Disables one or all breakpoints.

Managing Watchpoints (Data Breakpoints)

By **fristle**
cheatography.com/fristle/

Published 2nd May, 2013.
Last updated 1st June, 2014.

Cheatography

Closed-source Debugging with GDB Cheat Sheet

by fristle via cheatography.com/5574/cs/1012/

GDB: Breakpoints (cont)

watch *0x12345678 [mask 0xffffff00]	Stop on any **change** to the 24 most significant bits of a 32-bit value at address 0x12345678.
awatch *0x12345678	Like watch, but also stops on **any** write or read accesses to the given address.
rwatch *0x12345678	Like watch, but only stops on read accesses.

GDB: Concurrency

Multithreaded Debugging

gdb$ info threads	List the threads of the target process.
gdb$ thread *threadID*	Attach GDB to the thread *threadID*.
gdb$ set non-stop on	Only the debugged thread is halted in GDB, the rest continue to run non-stop (unless they are blocking on the thread being debugged).
gdb$ set scheduler-locking on	Only the debugged thread will run when the debuggee is resumed.
gdb$ set scheduler-locking step	Only the debugged thread will step when being step-debugged.
gdb$ show scheduler-locking	Display the current setting value.

Multiprocess Debugging

GDB: Concurrency (cont)

gdb$ set follow-fork-mode child	GDB will detach at a fork() and attach to the new process.
gdb$ set follow-fork-mode parent	(Default) GDB will not detach at a fork().
gdb$ show follow-fork-mode	Display the current setting value.
gdb$ set follow-exec-mode new	GDB will detach at an exec() and attach to the new process.
gdb$ set follow-exec-mode same	(Default) GDB will not detach at an exec().
gdb$ show follow-exec-mode	Display the current setting value.
gdb$ set detach-on-fork off	GDB will not detach at a fork() and will **also** attach to the child process (both will be debugged).
gdb$ show detach-on-fork	Display the current setting value.
gdb$ info inferiors	List all processes under GDB's control. (On Mac OS X: info files)

GDB: Advanced

Anti-Anti Debugging

gdb$ handle *signal [keywords...]*	(Untested) might bypass exception-based anti-debugging
gdb$ catch syscall ptrace	(Untested) Use this breakpoint to return 0 (set $rax = 0; continue), should bypass ptrace() checking by the debuggee.

By **fristle**
cheatography.com/fristle/

Published 2nd May, 2013.
Last updated 1st June, 2014.