

gdb cheat-sheet for reversing

Starting GDB

<code>gdb</code>	start GDB, with no debugging files
<code>gdb program</code>	begin debugging <i>program</i>
<code>gdb --args prg args</code>	begin debugging <i>prg args</i>
<code>gdb program pid</code>	begin debugging running process <i>pid</i>
<code>gdb program core</code>	debug coredump <i>core</i> produced by <i>program</i>

<code>--silent</code>	run silently (AKA: <code>--quiet</code> or <code>-q</code>)
<code>-ix file</code>	Execute command from <i>file</i> before loading the inferior (AKA: <code>--init-command</code>)
<code>-iex cmd</code>	Execute command <i>cmd</i> before loading the inferior (AKA: <code>--init-eval-command</code>)

Stopping GDB

<code>quit</code>	exit GDB; also <code>q</code> or <code>EOF</code> (eg <code>C-d</code>)
<code>INTERRUPT</code>	(eg <code>C-c</code>) terminate current command, or send to running process

Getting Help

<code>help</code>	list classes of commands
<code>help class</code>	one-line descriptions for commands in <i>class</i>
<code>help command</code>	describe <i>command</i>
<code>apropos re</code>	search for the regexp <i>re</i> inside documentation

Executing your Program

<code>r[un] arglist</code>	start your program with <i>arglist</i>
<code>r[un]</code>	start program with current argument list
<code>r[un] ... <inf>outf</code>	start your program with I/O redirected
<code>kill</code>	kill running program
<code>set args arglist</code>	specify <i>arglist</i> for next <code>run</code>
<code>set args</code>	specify empty argument list
<code>show args</code>	display argument list
<code>tty dev</code>	use <i>dev</i> as stdin and stdout for next <code>run</code>

<code>set startup-with-shell [on off]</code>	Use the shell to run the program?
<code>set exec-wrapper w</code>	use the wrapper <i>w</i> to launch programs; e.g.: <code>set exec-wrapper env 'LD.PRELOAD=X.so'</code>

<code>show env</code>	show all environment variables
<code>show env var</code>	show value of environment variable <i>var</i>
<code>set env var string</code>	set environment variable <i>var</i>
<code>unset env var</code>	remove <i>var</i> from environment

<code>set disable-randomization [on off]</code>	disable ASLR?
<code>set follow-fork-mode mode</code>	<i>mode</i> = <code>parent child</code>
<code>set detach-on-fork [on off]</code>	detach one of the processes after a fork?

[] surround optional arguments ... show one or more arguments

Breakpoints and Watchpoints

<code>break [file:]line</code>	set breakpoint at <i>line</i> number [in <i>file</i>]
<code>b [file:]line</code>	eg: <code>break main.c:37</code>
<code>break [file:]func</code>	set breakpoint at <i>func</i> [in <i>file</i>]
<code>break [+ -]offset</code>	set break at <i>offset</i> lines from current stop
<code>break *addr</code>	set breakpoint at address <i>addr</i>
<code>break</code>	set breakpoint at next instruction
<code>break ... if expr</code>	break conditionally on nonzero <i>expr</i>
<code>cond n [expr]</code>	new conditional expression on breakpoint <i>n</i> ; make unconditional if no <i>expr</i>
<code>tbreak ...</code>	temporary break; disable when reached
<code>hbreak ...</code>	as <code>break</code> , but hardware-assisted
<code>rbreak [file:]regex</code>	break on all functions matching <i>regex</i> [in <i>file</i>]
<code>watch expr</code>	set a watchpoint for expression <i>expr</i>
<code>rwatch ...</code>	read watchpoint
<code>awatch ...</code>	read/write (i.e., access) watchpoint
<code>catch event</code>	break at <i>event</i> , which may be <code>catch</code> , <code>throw</code> , <code>exec</code> , <code>fork</code> , <code>vfork</code> , <code>load</code> , or <code>unload</code> .
<code>info break</code>	show defined breakpoints
<code>info watch</code>	show defined watchpoints

<code>clear</code>	delete breakpoints at next instruction
<code>clear [file:]fun</code>	delete breakpoints at entry to <i>fun()</i>
<code>clear [file:]line</code>	delete breakpoints on source line
<code>delete [n]</code>	delete breakpoints [or breakpoint <i>n</i>]
<code>disable [n]</code>	disable breakpoints [or breakpoint <i>n</i>]
<code>enable [n]</code>	enable breakpoints [or breakpoint <i>n</i>]
<code>enable once [n]</code>	enable breakpoints [or breakpoint <i>n</i>]; disable again when reached
<code>enable del [n]</code>	enable breakpoints [or breakpoint <i>n</i>]; delete when reached
<code>ignore n count</code>	ignore breakpoint <i>n</i> , <i>count</i> times

<code>commands n [silent] command-list</code>	execute GDB <i>command-list</i> every time breakpoint <i>n</i> is reached. [silent suppresses default display]
<code>end</code>	end of <i>command-list</i>

<code>save breakpoint [file]</code>	saves breakpoints and their info (can be restored with <code>source</code>)
-------------------------------------	--

Program Stack

<code>backtrace [n]</code>	print trace of all frames in stack; or of <i>n</i> frames—innermost if <i>n</i> >0, outermost if <i>n</i> <0
<code>bt [n]</code>	
<code>frame [n]</code>	select frame number <i>n</i> or frame at address <i>n</i> ; if no <i>n</i> , display current frame
<code>up n</code>	select frame <i>n</i> frames up
<code>down n</code>	select frame <i>n</i> frames down
<code>info frame [addr]</code>	describe selected frame, or frame at <i>addr</i>
<code>info args</code>	arguments of selected frame
<code>info locals</code>	local variables of selected frame
<code>info reg [rn]...</code>	register values [for regs <i>rn</i>] in selected frame;
<code>info all-reg [rn]</code>	<code>all-reg</code> includes floating point

Execution Control

<code>continue [count]</code>	continue running; if <i>count</i> specified, ignore this breakpoint next <i>count</i> times
<code>c [count]</code>	
<code>step [count]</code>	execute until another line reached; repeat <i>count</i> times if specified
<code>s [count]</code>	
<code>s[tep]i [count]</code>	step by machine instructions
<code>next [count]</code>	execute next line, including any function calls
<code>n [count]</code>	
<code>n[ext]i [count]</code>	next machine instruction
<code>until [location]</code>	run until next instruction (or <i>location</i>) or the current stack frame returns
<code>finish</code>	run until selected stack frame returns
<code>return [expr]</code>	pop selected stack frame without executing [setting return value]
<code>signal num</code>	resume execution with signal <i>s</i> (none if 0)
<code>jump line</code>	resume execution at specified <i>line</i> number or
<code>jump *address</code>	<i>address</i>
<code>set var=expr</code>	evaluate <i>expr</i> without displaying it;

Display

<code>print [/f] [expr]</code>	show value of <i>expr</i> [or last value \$] according to format <i>f</i> :
<code>p [/f] [expr]</code>	
<code>x</code>	hexadecimal
<code>d</code>	signed decimal
<code>u</code>	unsigned decimal
<code>o</code>	octal
<code>t</code>	binary
<code>a</code>	address, absolute and relative
<code>c</code>	character
<code>f</code>	floating point
<code>call [/f] expr</code>	like <code>print</code> but does not display <code>void</code>
<code>x [/Nuf] expr</code>	examine memory at address <i>expr</i> ; optional format spec follows slash
<code>N</code>	count of how many units to display
<code>u</code>	unit size; one of
	<code>b</code> individual bytes
	<code>h</code> halfwords (two bytes)
	<code>w</code> words (four bytes)
	<code>g</code> giant words (eight bytes)
<code>f</code>	printing format. Any <code>print</code> format, or
	<code>s</code> null-terminated string
	<code>i</code> machine instructions
<code>disassem [addr]</code>	display memory as machine instructions

Automatic Display

<code>display [/f] expr</code>	show value of <i>expr</i> each time program stops [according to format <i>f</i>]
<code>display</code>	display all enabled expressions on list
<code>undisplay n</code>	remove number(s) <i>n</i> from list of automatically displayed expressions
<code>disable disp n</code>	disable display for expression(s) number <i>n</i>
<code>enable disp n</code>	enable display for expression(s) number <i>n</i>
<code>info display</code>	numbered list of display expressions

Expressions

<i>expr</i>	an expression in C, C++, or Modula-2 (including function calls), or:
<i>addr@len</i>	an array of <i>len</i> elements beginning at <i>addr</i>
<i>file::nm</i>	a variable or function <i>nm</i> defined in <i>file</i>
{ <i>type</i> } <i>addr</i>	read memory at <i>addr</i> as specified <i>type</i>
\$	most recent displayed value
\$ <i>n</i>	<i>n</i> th displayed value
\$\$	displayed value previous to \$
\$\$ <i>n</i>	<i>n</i> th displayed value back from \$
\$_	last address examined with x
\$_	value at address \$_
\$ <i>var</i>	convenience variable; assign any value
show values [<i>n</i>]	show last 10 values [or surrounding \$ <i>n</i>]
show conv	display all convenience variables

Symbol Table

info address <i>s</i>	show where symbol <i>s</i> is stored
info func [<i>regex</i>]	show names, types of defined functions (all, or matching <i>regex</i>)
info var [<i>regex</i>]	show names, types of global variables (all, or matching <i>regex</i>)
whatis [<i>expr</i>]	show data type of <i>expr</i> [or \$] without evaluating; ptype gives more detail
ptype [<i>expr</i>]	
ptype <i>type</i>	describe type, struct, union, or enum

GDB Scripts

source <i>script</i>	read, execute GDB commands from file <i>script</i>
define <i>cmd</i> <i>command-list</i> end	create new GDB command <i>cmd</i> ; execute script defined by <i>command-list</i> end of <i>command-list</i> Whenever you run <i>foo</i> , if user-defined hook-foo exists, it is executed before; if hookpost-foo exists, it is executed after. hook-stop is executed when program execution stops: before BP commands are run, displays are printed, or the stack frame is printed.
document <i>cmd</i> <i>help-text</i> end	create online documentation for new GDB command <i>cmd</i> end of <i>help-text</i>

Checkpoints (only under Linux)

checkpoint	snapshots current execution state; beware: when restored, each checkpoint has a PID different from program's original PID
info checkpoints	list saved checkpoints in the current session
restart <i>id</i>	restore checkpoint <i>id</i> ; beware: breakpoints, gdb variables, etc. are not affected; a checkpoint only restores things that reside in program being debugged, not in debugger
delete checkpoint <i>id</i>	delete the previously-saved checkpoint <i>id</i>

Controlling GDB

set <i>param value</i>	set one of GDB's internal parameters
show <i>param</i>	display current setting of parameter
Parameters understood by set and show :	
complaint <i>limit</i>	number of messages on unusual symbols
confirm <i>on/off</i>	enable or disable cautionary queries
editing <i>on/off</i>	control readline command-line editing
height <i>lpp</i>	number of lines before pause in display
language <i>lang</i>	Language for GDB expressions (auto , c or modula-2)
listsize <i>n</i>	number of lines shown by list
prompt <i>str</i>	use <i>str</i> as GDB prompt
radix <i>base</i>	octal, decimal, or hex number representation
verbose <i>on/off</i>	control messages when loading symbols
width <i>cpl</i>	number of characters before line folded
write <i>on/off</i>	Allow or forbid patching binary, core files (when reopened with exec or core)
history ...	groups with the following options:
h ...	
h exp <i>off/on</i>	disable/enable readline history expansion
h file <i>filename</i>	file for recording GDB command history
h size <i>size</i>	number of commands kept in history list
h save <i>off/on</i>	control use of external file for command history
print ...	groups with the following options:
p ...	
p address <i>on/off</i>	print memory addresses in stacks, values
p array <i>off/on</i>	compact or attractive format for arrays
p demangl <i>on/off</i>	source (demangled) or internal form for C++ symbols
p asm-dem <i>on/off</i>	demangle C++ symbols in machine-instruction output
p elements <i>limit</i>	number of array elements to display
p object <i>on/off</i>	print C++ derived types for objects
p pretty <i>off/on</i>	struct display: compact or indented
p union <i>on/off</i>	display of union members
p vtbl <i>off/on</i>	display of C++ virtual function tables
show commands	show last 10 commands
show commands <i>n</i>	show 10 commands around number <i>n</i>
show commands +	show next 10 commands

Working Files

file [<i>file</i>]	use <i>file</i> for both symbols and executable; with no arg, discard both
core [<i>file</i>]	read <i>file</i> as coredump; or discard
exec [<i>file</i>]	use <i>file</i> as executable only; or discard
symbol [<i>file</i>]	use symbol table from <i>file</i> ; or discard
load <i>file</i>	dynamically link <i>file</i> and add its symbols
add-sym <i>file addr</i>	read additional symbols from <i>file</i> , dynamically loaded at <i>addr</i>
info files	display working files and targets in use
path <i>dirs</i>	add <i>dirs</i> to front of path searched for executable and symbol files
show path	display executable and symbol file path
info share	list names of shared libraries currently loaded

Logging

show logging	show current values
set logging [on off]	enable/disable
set logging file <i>file</i>	default is gdb.txt
set logging overwrite [on off]	append by default
set logging redirect [on off]	redirect only to logfile

Debugging Targets

target <i>type param</i>	connect to machine, process, or file; e.g. target remote sshpass -ppw ssh -T [-p port] [user@]host gdbserver - prog [args]
attach <i>param</i>	connect to another process
detach	release target from GDB control

Shell Commands

cd <i>dir</i>	change working directory to <i>dir</i>
pwd	Print working directory
make ...	call “ make ”
shell <i>cmd</i>	execute shell command <i>cmd</i> (AKA: !)

Signals

handle <i>signal act</i>	specify GDB actions for <i>signal</i> :
print	announce signal
noprint	be silent for signal
stop	halt execution on signal
nostop	do not halt execution
pass	allow your program to handle signal
nopass	do not allow your program to see signal
info signals	show table of signals, GDB action for each

Source Files

dir <i>names</i>	add directory <i>names</i> to front of source path
dir	clear source path
show dir	show current source path
list	show next ten lines of source
list -	show previous ten lines
list <i>lines</i>	display source surrounding <i>lines</i> , specified as: [<i>file</i> :] <i>num</i> [<i>file</i> :] <i>function</i> <i>+off</i> <i>-off</i> <i>*address</i>
list <i>f,l</i>	from line <i>f</i> to line <i>l</i>
info line <i>num</i>	show starting, ending addresses of compiled code for source line <i>num</i>
info source	show name of current source file
info sources	list all source files in use
forw <i>regex</i>	search following source lines for <i>regex</i>
rev <i>regex</i>	search preceding source lines for <i>regex</i>

Text User Interface (TUI)

<code>gdb --tui</code>	start GDB in TUI mode (also <code>C-x C-a</code>)
<code>C-x 1</code>	one window layout
<code>C-x 2</code>	one windows layout
<code>C-o 2</code>	change active window
<code>Left, Up, Right, Down</code>	scroll active window
<code>PgUp, PgDown</code>	scroll active window by page up/down
<code>C-n</code>	walk to previous command
<code>C-p</code>	walk to next command
<code>C-l</code>	refresh the screen
<code>layout NAME</code>	
<code>next</code>	next layout
<code>prev</code>	previous layout
<code>src</code>	source and command windows
<code>asm</code>	assembly and command windows
<code>split</code>	source, assembly, and command windows
<code>regs</code>	display regs and current window
<code>tui reg GROUP</code>	
<code>next</code>	cycle though all available reg. groups
<code>prev</code>	cycle though in reverse order
<code>general</code>	general purpose registers
<code>float</code>	floating point registers
<code>vector</code>	vector registers
<code>all</code>	all registers

GDB under GNU Emacs

<code>M-x gdb</code>	run GDB under Emacs
<code>C-h m</code>	describe GDB mode
<code>M-s</code>	step one line (s tep)
<code>M-n</code>	next line (n ext)
<code>M-i</code>	step one instruction (s tepi)
<code>C-c C-f</code>	finish current stack frame (f inish)
<code>M-c</code>	continue (c ont)
<code>M-u</code>	up <i>arg</i> frames (u p)
<code>M-d</code>	down <i>arg</i> frames (d own)
<code>C-x &</code>	copy number from point, insert at end
<code>C-x SPC</code>	(in source file) set break at point

GDB License

<code>show copying</code>	Display GNU General Public License
<code>show warranty</code>	There is NO WARRANTY for GDB. Display full no-warranty statement.

Copyright ©2017 by zxgio, ©1991-2016 Free Software Foundation, Inc.
Author: Roland H. Pesch

This cheat-sheet may be freely distributed under the terms of the GNU General Public License; the latest version can be found at:

<https://github.com/zxgio/gdb-cheatsheet/>