

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}addr</i>	read memory at <i>addr</i> as specified <i>type</i>
<i>\$</i>	most recent displayed value
<i>\$n</i>	<i>nth</i> displayed value
<i>\$\$</i>	displayed value previous to <i>\$</i>
<i>\$\$n</i>	<i>nth</i> displayed value back from <i>\$</i>
<i>\$_</i>	last address examined with <i>x</i>
<i>\$_</i>	value at address <i>\$_</i>
<i>\$var</i>	convenience variable; assign any value
<i>show values [n]</i>	show last 10 values [or surrounding <i>\$n</i>]
<i>show conv</i>	display all convenience variables

Symbol Table

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

GDB Scripts

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

Checkpoints (only under Linux)

<i>checkpoint</i>	snapshots current execution state; beware: when restored, each checkpoint has a PID different from program's original PID
<i>info checkpoints</i> <i>restart id</i>	list saved checkpoints in the current session 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
<i>delete checkpoint id</i>	delete the previously-saved checkpoint <i>id</i>

Controlling GDB

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

Working Files

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

Logging

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

Debugging Targets

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

Shell Commands

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

Signals

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

Source Files

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