

GDB QUICK REFERENCE GDB Version 4

Essential Commands

<code>gdb program [core]</code>	debug <i>program</i> [using coredump <i>core</i>]
<code>b [file:]function</code>	set breakpoint at <i>function</i> [in <i>file</i>]
<code>run [arglist]</code>	start your program [with <i>arglist</i>]
<code>bt</code>	backtrace: display program stack
<code>p expr</code>	display the value of an expression
<code>c</code>	continue running your program
<code>n</code>	next line, stepping over function calls
<code>s</code>	next line, stepping into function calls

Starting GDB

<code>gdb</code>	start GDB, with no debugging files
<code>gdb program</code>	begin debugging <i>program</i>
<code>gdb program core</code>	debug coredump <i>core</i> produced by <i>program</i>
<code>gdb --help</code>	describe command line options

Stopping GDB

<code>quit</code>	exit GDB; also <code>q</code> or EOF (eg C-d)
<code>INTERRUPT</code>	(eg C-c) 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>

Executing your Program

<code>run arglist</code>	start your program with <i>arglist</i>
<code>run</code>	start your program with current argument list
<code>run ... <inf>>outf</code>	start your program with input, output redirected
<code>kill</code>	kill running program
<code>tty dev</code>	use <i>dev</i> as stdin and stdout for next <code>run</code>
<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>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

Shell Commands

<code>cd dir</code>	change working directory to <i>dir</i>
<code>pwd</code>	Print working directory
<code>make ...</code>	call “make”
<code>shell cmd</code>	execute arbitrary shell command string

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

©1998 Free Software Foundation, Inc. Permissions on back

Breakpoints and Watchpoints

<code>break [file:]line</code>	set breakpoint at <i>line</i> number [in <i>file</i>] eg: <code>break main.c:37</code>
<code>b [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 -offset</code>	set breakpoint at address <i>addr</i>
<code>break *addr</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>rbreak regex</code>	break on all functions matching <i>regex</i>
<code>watch expr</code>	set a watchpoint for expression <i>expr</i>
<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</code> [silent] command-list	execute GDB <i>command-list</i> every time breakpoint <i>n</i> is reached. [<i>silent</i> suppresses default display]
<code>end</code>	end of <i>command-list</i>

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>all-reg</code> includes floating point
<code>info all-reg [rn]</code>	

<code>Execution</code>	
<code>continue [count]</code>	
<code>c [count]</code>	
<code>step [count]</code>	
<code>s [count]</code>	
<code>stepi [count]</code>	
<code>si [count]</code>	
<code>next [count]</code>	
<code>n [count]</code>	
<code>nexti [count]</code>	
<code>ni [count]</code>	
<code>until [location]</code>	
<code>finish</code>	
<code>return [expr]</code>	
<code>signal num</code>	
<code>jump line</code>	
<code>jump *address</code>	
<code>set var=expr</code>	

<code>Display</code>	
<code>print [/f] [expr]</code>	
<code>p [/f] [expr]</code>	
<code>x</code>	
<code>d</code>	
<code>u</code>	
<code>o</code>	
<code>t</code>	
<code>a</code>	
<code>c</code>	
<code>f</code>	
<code>call [/f] exp</code>	
<code>x [/Nuf] exp</code>	

<code>N</code>	
<code>u</code>	
	<code>f</code>
	<code>disassem [ad]</code>

<code>Automatic</code>	
<code>display [/f]</code>	
<code>display</code>	
<code>undisplay</code>	
<code>info display</code>	

Expressions

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

Symbol Table

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

GDB Scripts

<code>source script</code>	read, execute GDB commands from file <i>script</i>
<code>define cmd command-list</code>	create new GDB command <i>cmd</i> ; execute script defined by <i>command-list</i>
<code>end</code>	end of <i>command-list</i>
<code>document cmd help-text</code>	create online documentation for new GDB command <i>cmd</i>
<code>end</code>	end of <i>help-text</i>

Signals

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

Debugging Targets

<code>target type param</code>	connect to target machine, process, or file
<code>help target</code>	display available targets
<code>attach param</code>	connect to another process
<code>detach</code>	release target from GDB control

Controlling GDB

<code>set param value</code>	set one of GDB's internal parameters
<code>show param</code>	display current setting of parameter
Parameters understood by <code>set</code> and <code>show</code> :	
<code>complaint limit</code>	number of messages on unusual symbols
<code>confirm on/off</code>	enable or disable cautionary queries
<code>editing on/off</code>	control <code>readline</code> command-line editing
<code>height lpp</code>	number of lines before pause in display
<code>language lang</code>	Language for GDB expressions (auto, c or modula-2)
<code>listsize n</code>	number of lines shown by <code>list</code>
<code>prompt str</code>	use <i>str</i> as GDB prompt
<code>radix base</code>	octal, decimal, or hex number representation
<code>verbose on/off</code>	control messages when loading symbols
<code>width cpl</code>	number of characters before line folded
<code>write on/off</code>	Allow or forbid patching binary, core files (when reopened with <code>exec</code> or <code>core</code>)

groups with the following options:

<code>history ...</code>	
<code>h ...</code>	
<code>h exp off/on</code>	disable/enable <code>readline</code> history expansion
<code>h file filename</code>	file for recording GDB command history
<code>h size size</code>	number of commands kept in history list
<code>h save off/on</code>	control use of external file for command history
<code>print ...</code>	groups with the following options:
<code>p ...</code>	
<code>p address on/off</code>	print memory addresses in stacks, values
<code>p array off/on</code>	compact or attractive format for arrays
<code>p demangl on/off</code>	source (demangled) or internal form for C++ symbols
<code>p asm-dem on/off</code>	demangle C++ symbols in machine-instruction output
<code>p elements limit</code>	number of array elements to display
<code>p object on/off</code>	print C++ derived types for objects
<code>p pretty off/on</code>	struct display: compact or indented
<code>p union on/off</code>	display of union members
<code>p vtbl off/on</code>	display of C++ virtual function tables

`show commands` show last 10 commands
`show commands n` show 10 commands around number *n*
`show commands +` show next 10 commands

Working Files

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

Source File

<code>dir names</code>	dir
<code>show dir</code>	show dir
<code>list</code>	list
<code>list -</code>	list -
<code>list lines</code>	list lines
<code>[file:]num</code>	[file:]num
<code>[file:]function</code>	[file:]function
<code>+off</code>	+off
<code>-off</code>	-off
<code>*address</code>	*address
<code>list f,l</code>	list f,l
<code>info line num</code>	info line num

GDB und

<code>M-x gdb</code>	M-x
<code>C-h m</code>	C-h
<code>M-s</code>	M-s
<code>M-n</code>	M-n
<code>M-i</code>	M-i
<code>C-c C-f</code>	C-c
<code>M-c</code>	M-c
<code>M-u</code>	M-u
<code>M-d</code>	M-d
<code>C-x &</code>	C-x
<code>C-x SPC</code>	SPC

GDB Lice

show copying
show warrant

Copyright ©

The author a

This card may
General Public
development of

GDB itself is f
it under the te
absolutely no v