

# GDB QUICK REFERENCE GDB Version 5

## 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 ... &lt;inf&gt;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 “ <code>make</code> ”
<code>shell cmd</code>	execute arbitrary shell command string

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

©1998-2018 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>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 -offset</code>	set breakpoint at address <i>addr</i>
<code>break *addr</code>	set breakpoint at next instruction
<code>break</code>	break conditionally on nonzero <i>expr</i>
<code>break ... if expr</code>	new conditional expression on breakpoint <i>n</i> ; make unconditional if no <i>expr</i>
<code>cond n [expr]</code>	temporary break; disable when reached
<code>tbreak ...</code>	break on all functions matching <i>regex</i> [in <i>file</i> ]
<code>rbreak [file:]regex</code>	set a watchpoint for expression <i>expr</i> 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. [ <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>	select frame number <i>n</i> or frame at address <i>n</i> ; if no <i>n</i> , display current frame
<code>frame [n]</code>	select frame <i>n</i> frames up
<code>up n</code>	select frame <i>n</i> frames down
<code>down n</code>	describe selected frame, or frame at <i>addr</i>
<code>info frame [addr]</code>	arguments of selected frame
<code>info args</code>	local variables of selected frame
<code>info locals</code>	register values [for regs <i>rn</i> ] in selected frame; <code>all-reg</code> includes floating point
<code>info reg [rn]...</code>	
<code>info all-reg [rn]</code>	

## 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>stepti [count]</code>	step by machine instructions rather than source lines
<code>si [count]</code>	
<code>next [count]</code>	execute next line, including any function calls
<code>n [count]</code>	
<code>nexti [count]</code>	next machine instruction rather than source line
<code>ni [count]</code>	
<code>until [location]</code>	run until next instruction (or <i>location</i> )
<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
<code>jump *address</code>	or <i>address</i>
<code>set var=expr</code>	evaluate <i>expr</i> without displaying it; use for altering program variables

## 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 b individual bytes h halfwords (two bytes) w words (four bytes) g giant words (eight bytes)
<code>f</code>	printing format. Any <code>print</code> format, or s null-terminated string i 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

<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

`show values [n]` show last 10 values [or surrounding \$n]  
`show conv` 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 ( <code>auto</code> , <code>c</code> or <code>modula-2</code> )
<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> )
<code>history ...</code>	groups with the following options:
<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
<code>show commands</code>	show last 10 commands
<code>show commands n</code>	show 10 commands around number <i>n</i>
<code>show commands +</code>	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>	read additional symbols from <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 Files

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

## 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 ( <code>step</code> )
<code>M-n</code>	next line ( <code>next</code> )
<code>M-i</code>	step one instruction ( <code>stepi</code> )
<code>C-c C-f</code>	finish current stack frame ( <code>finish</code> )
<code>M-c</code>	continue ( <code>cont</code> )
<code>M-u</code>	up <i>arg</i> frames ( <code>up</code> )
<code>M-d</code>	down <i>arg</i> frames ( <code>down</code> )
<code>C-x &amp;</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 ©1991-2018 Free Software Foundation, Inc. Author: Roland H. Pesch

The author assumes no responsibility for any errors on this card.

This card may be freely distributed under the terms of the GNU General Public License.

Please contribute to development of this card by annotating it. Improvements can be sent to bug-gdb@gnu.org.

GDB itself is free software; you are welcome to distribute copies of it under the terms of the GNU General Public License. There is absolutely no warranty for GDB.