

## Essential Commands

<code>gdb program [core]</code>	debug <i>program</i> [using <code>coredump core</code> ]
<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 <code>coredump core</code> 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 <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>

## 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 "make"
<code>shell cmd</code>	execute arbitrary shell command string

[ ] 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 -offset</code>	
<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>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 [silent]</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>

## 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>	

## 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>stepi [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 or <i>address</i>
<code>jump *address</code>	
<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
<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

