

## 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 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 “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, throw, exec, fork, vfork, load, or 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>	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; <b>all-reg</b> 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
<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
<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

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

show values [*n*] show last 10 values [or surrounding *\$n*]  
show conv display all convenience variables

## Symbol Table

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

## GDB Scripts

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

## Signals

handle *signal act* specify GDB actions for *signal*:  
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

## Debugging Targets

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

## Controlling GDB

set *param value* set one of GDB's internal parameters  
show *param* display current setting of parameter

Parameters understood by **set** and **show**:

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

file [*file*] use *file* for both symbols and executable; with no arg, discard both  
core [*file*] read *file* as coredump; or discard  
exec [*file*] use *file* as executable only; or discard  
symbol [*file*] use symbol table from *file*; or discard  
load *file* dynamically link *file* and add its symbols  
add-sym *file addr* read additional symbols from *file*, dynamically loaded at *addr*  
info files display working files and targets in use  
path *dirs* add *dirs* 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

## Source Files

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

## GDB under GNU Emacs

M-x gdb run GDB under Emacs  
C-h m describe GDB mode  
M-s step one line (**step**)  
M-n next line (**next**)  
M-i step one instruction (**stepi**)  
C-c C-f finish current stack frame (**finish**)  
M-c continue (**cont**)  
M-u up *arg* frames (**up**)  
M-d down *arg* frames (**down**)  
C-x & copy number from point, insert at end  
C-x SPC (in source file) set break at point

## GDB License

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

Copyright ©1991,'92,'93,'98,2000 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.