

GNU DE BUGGER

©1998-2005 Mitch Richling
Last Updated 2005-09-12

Getting Started

- On the command line: `gdb [options] [prog [core | PID]]`
- `-s symfile` The symbol file to use
- `-e prog` The program to debug
- `-c core` The core file to use
- `-cd=working_dir` The working directory for gdb
- `-tty=dev` TTY for I/O of debugged program
- `-command=file` Run gdb commands in file
- `-n` Don't run `.gdbinit`
- `-q` Quiet mode
- After startup:
 - `attach PID` connect to another process
 - `detach` release target from GDB control

Stopping GDB

- `quit` Exit GDB (C-d works too)
- `C-c` Terminate current command, or send to running process

Working Files

- `file [file]` use `file` for both symbols and executable
- `core-file [file]` read `file` as core dump
- `exec-file [file]` use `file` as executable only
- `symbol-file [file]` use symbol table from `file`
- `info files` display working files and targets in use
- `path dirs` push `dirs` onto executable and symbol search path
- `show path` display executable and symbol file path
- `info sharedlibrary` list names of shared libraries currently loaded
- `directory [names]` push `names` onto source path, or clear it
- `show directories` show current source path
- `info source` show name of current source file
- `info sources` list all source files in use

Listing Source & Code

- `list [bp-spec]` display source surrounding `bp-spec`
- `list` show current/next lines of source
- `list < + | - >` show lines after / before
- `list f, t` from line `f` to line `t`
- `info line [bp-spec]` show addresses for compiled code for source line
- Sets default address for `x/i`.
- search following source lines for `regex`
- `forward-search regex` search following source lines for `regex`
- `reverse-search regex` search preceding source lines for `regex`
- `disassemble [[addr1]` display memory as machine instructions
- `[addr2]]` Function around PC, function around `addr1`, or between `addr1` and `addr2`

Executing Program

- `run [args] [<inf] [>outf]` Run it
- `kill` kill it
- Before 'run', some things can be set:
 - `tty dev` use `dev` as stdin and stdout
 - `set args [arglist]` Set/Clear arg list
 - `show args` Display argument list
 - `set env var string` Set environment variable `var`
 - `unset env var` Remove `var` from environment
 - `show env [var]` Show value of one, or all, environment var(s)

Execution Control

- `continue [count]` continue running past breakpoint
- `advance bp-spec` run up to `bp-spec`
- `step [count]` execute until another line reached
- `stepi [count]` step by machine instructions
- `next [count]` execute next line, including any functions
- `nexti [count]` next machine instruction
- `until [bp-spec]` run till src line after current or `bp-spec`
- `finish` run until selected stack frame returns
- `return [expr]` pop stack frame with function return `expr`
- `signal num` resume execution with signal `num`
- `thread thrd` switch to thread number `thrd`
- `thread apply thrd cmd` apply `cmd` to `thrd` ('all' for all threads)
- `jump line` resume execution at specified `line` number
- `jump *address` resume execution at specified `address`
- `set var=expr` Set `var` by evaluating `expr`

Signals

- `info signals` show table of signals & GDB actions
- `handle signal act` Set signal actions for `signal`. `act` may be:
 - `[no]print` Announce (or not) signal
 - `[no]stop` Stop (or not) program execution
 - `[no]pass` Pass (or not) signal to program

Program Stack

- `backtrace [n]` print trace of all frames in stack
- `frame [n]` innermost `n` frames if `n>0` & outermost if `n<0` select frame number `n` current frame
- `up n` select frame `n` frames up
- `down n` select frame `n` frames down
- `info frame [addr]` describe selected frame, or frame at `addr`
- `info args` arguments of selected frame
- `info locals` local variables of selected frame

Breakpoints and Watchpoints

- `break [bp-spec] [thread thrd]` set breakpoint. if no `bp-spec` then set at next instruction break only when `expr!=0`
- `break bp-spec [thread thrd] if expr`
- `bp-spec` may be one of:
 - `[file:]line line-number` `[file:]func function`
 - `*addr address` `±offset ±offset lines from stop`
- `tbreak ...` Same as `break`, but disable when reached (temporary)
- `rbreak regex` break on all functions matching `regex`
- `watch expr` set a watchpoint for expression `expr`
- `condition n [expr]` new conditional expression on breakpoint `n` make unconditional if no `expr`
- `info breakpoints` show defined breakpoints
- `info watchpoints` show defined watchpoints
- `clear` delete breakpoints at next instruction
- `clear [file:]fun` delete breakpoints at entry to `fun()`
- `clear [file:]line` delete breakpoints on source line
- `delete [n]` delete breakpoints
- `disable [n]` disable breakpoints
- `enable [n]` enable breakpoints
- `enable once [n]` enable breakpoints; disable again when reached
- `enable del [n]` enable breakpoints; delete when reached
- `ignore n count` ignore breakpoint `n`, `count` times
- `commands n [silent]` execute GDB `command-list` every time breakpoint `n` is reached.
- `command-list` end of `command-list`
- `end`

Expressions

- `expr` an expression in C, C++ (including function calls)
- `*addr@len` an array of `len` elements beginning at `addr`
- `[file:]name` a variable or function name
- `{type}addr` read memory at `addr` as specified `type`
- `$` most recently displayed value
- `$n` `n`th displayed value
- `$$` displayed value previous to `$`
- `$$n` `n`th displayed value back from `$`
- `$_` last address examined with `x`
- `$__` value at address `$_`
- `set $var = expr` Assign a value to a convenience variable
- `$var++` Increment a convenience variable
- `show values n` show last 10 values or surrounding `$n`
- `show conv` display all convenience variables
- `$_exitcode` Exit status of the program
- `$bpnum` Most recently set breakpoint
- `$cdir` Compilation dir for current source file
- `$cwd` Current working directory
- `$fp` Frame pointer register
- `$pc` Program counter register
- `$ps` Processor status register
- `$sp` Stack pointer register
- `whatis expr` Data type of `expr`

Symbol Table

- `info address s` show where symbol `s` is stored
 - `info symbol addr` show what symbol is at `addr`
 - `info functions [regex]` show names, types of functions
 - `info variables [regex]` show names, types of global variables
 - `info types [regex]` show type names
 - `whatis [expr]` show data type of `expr`
 - `p[ty] [expr]` like `whatis`, but more info.
 - `p[ty] type` describe type, struct, union, or enum
- NOTE: See 'Program Stack' for info on local variables

Display

- `display [/f] expr` print `expr` each time program stops
- `display` display all enabled expressions on list
- `undisplay n` remove number specified display elements
- `delete display n` remove number specified display elements
- `disable display n` disable display for expression(s) number `n`
- `enable display n` enable display for expression(s) number `n`
- `info display` numbered list of display expressions

Printing

- `inspect [/f] [expr]` Same as `print`
 - `print [/f] [expr]` Show value of `expr`
- The format specifier, `/f`, can be:
- `x` hexadecimal
 - `d` signed decimal
 - `u` unsigned decimal
 - `o` octal
 - `a` address (abs or relative)
 - `t` binary
 - `c` character
 - `f` floating point
 - `s` string
- `x [N/f] expr` examine memory at address `expr`
- `N` is count of how many units to display
- `u` is the unit and may be one of:
- `b` bytes
 - `w` words (4 bytes)
 - `h` half words (2 bytes)
 - `g` giant words (8 bytes)
- `f` may be any print format or
- `s` null-terminated string
 - `i` machine instructions
- `printf "fmt_str" [,expr...]` Print using `printf` format string

Running Program Info

- `info pid` Process ID of the program
- `info program` Execution status of the program
- `info all-registers` List of all registers and their contents
- `info float` Print the status of the floating point unit
- `info registers` List of integer registers and their contents
- `info vector` Print the status of the vector unit

Online Help

- `apropos string` Search the online help for `string`
- `help string` Get help for `string`.

GDB 2 DBX

GDB	DBX
<code>break line fnc *adr</code>	stop at <code>line</code> <code>func</code> <code>addr</code>
<code>break ... if expr</code>	stop ... -if <code>expr</code>
<code>cond n</code>	stop ... -if <code>expr</code>
<code>tbreak</code>	stop ... -temp
<code>watch expr</code>	stop <code>expr</code> [slow]
<code>watch var</code>	stop modify <code>&var</code> [fast]
<code>catch x</code>	intercept <code>x</code>
<code>info watch</code>	status
<code>info break</code>	status
<code>clear [fun]</code>	delete [n]
<code>delete</code>	delete all
<code>disable</code>	handler -disable all
<code>disable n</code>	handler -disable n
<code>enable</code>	handler -enable all
<code>enable n</code>	handler -enable n
<code>ignore n cnt</code>	handler -count n <code>cnt</code>
<code>backtrace n</code>	where n
<code>frame n</code>	frame n
<code>info reg reg</code>	print <code>\$reg</code>
<code>finish</code>	step up
<code>signal num</code>	cont sig <code>num</code>
<code>jump line</code>	cont at <code>line</code>
<code>set var=expr</code>	assign <code>var=expr</code>
<code>x/<fmt> addr</code>	<code>x addr/fmt</code>
<code>disassem addr</code>	<code>dis addr</code>
<code>info func regex</code>	<code>funcs regex</code>
<code>ptype type</code>	<code>whatis -t type</code>
<code>define cmd</code>	<code>function cmd</code>
<code>handle sig</code>	stop sig <code>sig</code>
<code>info signals</code>	status; catch
<code>attach pid</code>	debug - <code>pid</code>
<code>core file</code>	debug <code>a.out file</code>
<code>dir name</code>	pathmap <code>name</code>
<code>info line n</code>	listi n
<code>info source</code>	file
<code>forw/rev regex</code>	search/bsearch <code>regex</code>

GDB Parameters

- `set param value`
- `showparam`
- Possible parameters (default listed first for on/off-type parameters):
 - `complaint limit` number of messages on unusual symbols
 - `confirm on/off` enable or disable cautionary queries
 - `height lpp` number of lines before pause in display
 - `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
 - `print address on/off` print memory addresses in stacks, values
 - `print array off/on` compact or attractive format for arrays
 - `print demangl on/off` C++ syms in source (demangled) or internal form
 - `print asm-demangle on/off` demangle C++ symbols in machine-instruction output
 - `print elements limit` number of array elements to display
 - `print object on/off` print C++ derived types for objects
 - `print pretty off/on` struct display: compact or indented
 - `print union on/off` display of union members
 - `print vtbl off/on` display of C++ virtual function tables

Extending & Customizing

- Controlled output commands
 - `echo text` Print the given text. C-style escape sequences are supported.
 - `output expression` Print the given expression.
 - `output/fmt expression` Print the given expression using the given print format.
 - `printf string expressions` Print the given expressions using the given printf format string.
- `define cmdName ... end` Define a new command. Arguments are accessed through `$arg0 ... $arg9`.
 - If `cmdName` is "hook-foo" where "foo" is a GDB built-in command, then the command will be executed just before "foo".
 - If `cmdName` is "hookpost-foo" where "foo" is a GDB built-in command, then the command will be executed just after "foo".
 - If `cmdName` is "stop-foo", then run command at program stops.
- Control structures allowed in user defined commands:
 - `if expr ... [else ...] end` Executes statements only if the expression is true. Otherwise executes the else clause if it exists.
 - `while expression ... end` Executes body while expression is true
 - `document cmdName ... end` Document user defined command and make documentation available from online help. The command must already be defined. read and execute commands found in file
- `source filename`
- Startup sequence:
 - Read `~/.gdbinit`
 - Process command line arguments,
 - Read init file in working directory
 - Read command files given in `-x` arguments

C++ Stuff

- Most containers (and strings)
 - `p aContainer.empty()`
 - `p aContainer.size()`
- Printing Container Contents
 - C++ Strings: `p aString.c_str()`
 - GCC Vectors (GCC): `p (*aVector._M_impl._M_start)@(aVector.size())`