

Launching

<code>gdb -p <pid></code>	attach gdb to an existing process
<code>gdb --args <exe_file> [arg1] [arg2] ...</code>	make gdb launch an executable file, passing some arguments to the executable file
<code>gdb --batch -x <command_file> --args <exe_file> [arg1] [arg2] ...</code>	make gdb launch an executable file (passing some arguments to it), then have gdb execute commands from a "command file"

Execution Control

<code>continue</code>	continue execution of the process until a breakpoint (or catchpoint) is hit
<code>next</code>	continue execution of the process until the next line of code
<code>step</code>	continue the execution of the process until the first instruction of the next function (aka "step into")
<code>finish</code>	continue execution of the process until the current function returns

"Ctrl + c" (while the gdb terminal window is focused) to break the process right where it is at.

Viewing Source

<code>layout next</code>	enable tui mode (top part of terminal will show source location of where you're currently broken)
<code>tui disable</code>	disable tui mode
<code>list</code>	print next 10 lines of code
<code>list -</code>	print previous 10 lines of code
<code>list <file>:<function></code>	print 10 lines of code around a specific function in a file

Viewing Variables

<code>print <variable_name></code>	print a variable (must be in scope of course)
<code>backtrace</code>	print callstack
<code>backtrace -n</code>	print top n frames of callstack (n can be any number)
<code>frame n</code>	select frame number n (nth frame from the top). Frame is another word for a particular function in a callstack.
<code>up</code>	select the next frame up
<code>down</code>	select the next frame down
<code>info args</code>	print the arguments passed to the selected frame (function)
<code>info locals</code>	print the local variables defined in the selected frame



By MeLikeyCode

Not published yet.

Last updated 20th April, 2020.

Page 1 of 2.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>

Breakpoints

info breakpoints	print information about breakpoints/catchpoints that you've placed
break <function-name>	place a breakpoint on a particular function. Write the full name of the function, as you would specify it in C++ (e.g. ClassName::function_name). If you have multiple functions with the same name, include argument types to disambiguate.
break <file>:<line-number>	place a breakpoint on a particular line of a particular file
delete <breakpoint-number>	delete (remove) a particular breakpoint. "info breakpoints" will show you the breakpoint numbers for all the breakpoints that you've placed.
disable <breakpoint-number>	disable a particular breakpoint. A disabled breakpoint still shows up when you do "info breakpoints", but it won't be hit until you enable it.
enable <breakpoint-number>	enable a particular breakpoint
break <function> if <condition>	place a conditional breakpoint (i.e. only break if a certain condition is true). The condition is any valid C++ expression that evaluates to true or false. You can use convenience variables as well as any variables in scope for the condition.
condition <breakpoint-number> [condition]	define a condition for an already placed breakpoint. If you leave out the [condition], then you are saying to make the breakpoint unconditional (i.e. remove any existing conditions, if there are any).
commands <breakpoint-number>	specify gdb commands that should automatically run whenever the breakpoint is hit. After you run this command, typing a bunch of newline seperated commands that should be executed when the breakpoint is hit. After you are done entering your commands, type "end".



By **MeLikeyCode**

Not published yet.

Last updated 20th April, 2020.

Page 2 of 2.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>