# GDB to LLDB command map

Below is a table of GDB commands with the LLDB counterparts. The built in GDB-compatibility aliases in LLDB are also listed. The full lldb command names are often long, but any unique short form can be used. Instead of “**breakpoint set**”, “**br se**” is also acceptable.

## Execution Commands

| GDB | LLDB |
|---|---|
| Launch a process no arguments. | |
| `(gdb) run`<br>`(gdb) r` | `(lldb) process launch`<br>`(lldb) run`<br>`(lldb) r` |
| Launch a process with arguments `<args>`. | |
| `(gdb) run <args>`<br>`(gdb) r <args>` | `(lldb) process launch -- <args>`<br>`(lldb) run <args>`<br>`(lldb) r <args>` |
| Launch a process for with arguments **`a.out 1 2 3`** without having to supply the args every time. | |
| `% gdb --args a.out 1 2 3`<br>`(gdb) run`<br>`...`<br>`(gdb) run`<br>`...` | `% lldb -- a.out 1 2 3`<br>`(lldb) run`<br>`...`<br>`(lldb) run`<br>`...` |
| Or: | |
| `(gdb) set args 1 2 3`<br>`(gdb) run`<br>`...`<br>`(gdb) run` | `(lldb) settings set target.run-args 1 2 3`<br>`(lldb) run`<br>`...`<br>`(lldb) run` |

```
...
```

```
...
```

Launch a process with arguments in new terminal window (macOS only).

```
(lldb) process launch --tty -- <args>
(lldb) pro la -t -- <args>
```

Launch a process with arguments in existing terminal /dev/ttys006 (macOS only).

```
(lldb) process launch --tty=/dev/ttys006 -- <args>
(lldb) pro la -t/dev/ttys006 -- <args>
```

Set environment variables for process before launching.

```
(gdb) set env DEBUG 1
```

```
(lldb) settings set target.env-vars DEBUG=1
(lldb) set se target.env-vars DEBUG=1
(lldb) env DEBUG=1
```

Unset environment variables for process before launching.

```
(gdb) unset env DEBUG
```

```
(lldb) settings remove target.env-vars DEBUG
(lldb) set rem target.env-vars DEBUG
```

Show the arguments that will be or were passed to the program when run.

```
(gdb) show args
Argument list to give program being debugged when it is started is "1 2 3".
```

```
(lldb) settings show target.run-args
target.run-args (array of strings) =
[0]: "1"
[1]: "2"
[2]: "3"
```

Set environment variables for process and launch process in one command.

```
(lldb) process launch -v DEBUG=1
```

Attach to a process with process ID 123.

```
(gdb) attach 123
```

```
(lldb) process attach --pid 123
(lldb) attach -p 123
```

Attach to a process named "a.out".

```
(gdb) attach a.out
```

```
(lldb) process attach --name a.out
```

| | |
|---|---|
| | **(lldb)** pro at -n a.out |

Wait for a process named "a.out" to launch and attach.

| | |
|---|---|
| **(gdb)** attach -waitfor a.out | **(lldb)** process attach --name a.out --waitfor<br>**(lldb)** pro at -n a.out -w |

Attach to a remote gdb protocol server running on system "eorgadd", port 8000.

| | |
|---|---|
| **(gdb)** target remote eorgadd:8000 | **(lldb)** gdb-remote eorgadd:8000 |

Attach to a remote gdb protocol server running on the local system, port 8000.

| | |
|---|---|
| **(gdb)** target remote localhost:8000 | **(lldb)** gdb-remote 8000 |

Attach to a Darwin kernel in kdp mode on system "eorgadd".

| | |
|---|---|
| **(gdb)** kdp-reattach eorgadd | **(lldb)** kdp-remote eorgadd |

Do a source level single step in the currently selected thread.

| | |
|---|---|
| **(gdb)** step<br>**(gdb)** s | **(lldb)** thread step-in<br>**(lldb)** step<br>**(lldb)** s |

Do a source level single step over in the currently selected thread.

| | |
|---|---|
| **(gdb)** next<br>**(gdb)** n | **(lldb)** thread step-over<br>**(lldb)** next<br>**(lldb)** n |

Do an instruction level single step in the currently selected thread.

| | |
|---|---|
| **(gdb)** stepi<br>**(gdb)** si | **(lldb)** thread step-inst<br>**(lldb)** si |

Do an instruction level single step over in the currently selected thread.

| | |
|---|---|
| **(gdb)** nexti<br>**(gdb)** ni | **(lldb)** thread step-inst-over<br>**(lldb)** ni |

Step out of the currently selected frame.

| | |
|---|---|
| **(gdb)** finish | **(lldb)** thread step-out<br>**(lldb)** finish |

Return immediately from the currently selected frame, with an optional return value.

| | |
|---|---|
| **(gdb)** return &lt;RETURN EXPRESSION&gt; | **(lldb)** thread return &lt;RETURN EXPRESSION&gt; |

Backtrace and disassemble every time you stop.

```
(lldb) target stop-hook add
Enter your stop hook
command(s). Type 'DONE' to
end.
> bt
> disassemble --pc
> DONE
Stop hook #1 added.
```

Run until we hit line **12** or control leaves the current function.

| GDB | LLDB |
|---|---|
| `(gdb) until 12` | `(lldb) thread until 12` |

## Breakpoint Commands

| GDB | LLDB |
|---|---|

Set a breakpoint at all functions named **main**.

| GDB | LLDB |
|---|---|
| `(gdb) break main` | `(lldb) breakpoint set --name main`<br>`(lldb) br s -n main`<br>`(lldb) b main` |

Set a breakpoint in file **test.c** at line **12**.

| GDB | LLDB |
|---|---|
| `(gdb) break test.c:12` | `(lldb) breakpoint set --file test.c --line 12`<br>`(lldb) br s -f test.c -l 12`<br>`(lldb) b test.c:12` |

Set a breakpoint at all C++ methods whose basename is **main**.

| GDB | LLDB |
|---|---|
| `(gdb) break main`<br>*(Hope that there are no C functions named **main**).* | `(lldb) breakpoint set --method main`<br>`(lldb) br s -M main` |

Set a breakpoint at and object C function: **-[NSString stringWithFormat:]**.

| GDB | LLDB |
|---|---|
| `(gdb) break -[NSString stringWithFormat:]` | `(lldb) breakpoint set --name "-[NSString stringWithFormat:]"`<br>`(lldb) b -[NSString stringWithFormat:]` |

Set a breakpoint at all Objective-C methods whose selector is **count**.

| GDB | LLDB |
|---|---|
| `(gdb) break count`<br>*(Hope that there are no C or C++ functions named **count**).* | `(lldb) breakpoint set --selector count`<br>`(lldb) br s -S count` |

Set a breakpoint by regular expression on function name.

| | |
|---|---|
| **(gdb)** `rbreak regular-expression` | **(lldb)** `breakpoint set --func-regex regular-expression`<br>**(lldb)** `br s -r regular-expression` |

Ensure that breakpoints by file and line work for #included .c/.cpp/.m files.

| | |
|---|---|
| **(gdb)** `b foo.c:12` | **(lldb)** `settings set target.inline-breakpoint-strategy always`<br>**(lldb)** `br s -f foo.c -l 12` |

Set a breakpoint by regular expression on source file contents.

| | |
|---|---|
| **(gdb)** `shell grep -e -n pattern source-file`<br>**(gdb)** `break source-file:CopyLineNumbers` | **(lldb)** `breakpoint set --source-pattern regular-expression --file SourceFile`<br>**(lldb)** `br s -p regular-expression -f file` |

Set a conditional breakpoint

| | |
|---|---|
| **(gdb)** `break foo if strcmp(y,"hello") == 0` | **(lldb)** `breakpoint set --name foo --condition '(int)strcmp(y,"hello") == 0'`<br>**(lldb)** `br s -n foo -c '(int)strcmp(y,"hello") == 0'` |

List all breakpoints.

| | |
|---|---|
| **(gdb)** `info break` | **(lldb)** `breakpoint list`<br>**(lldb)** `br l` |

Delete a breakpoint.

| | |
|---|---|
| **(gdb)** `delete 1` | **(lldb)** `breakpoint delete 1`<br>**(lldb)** `br del 1` |

## Watchpoint Commands

| GDB | LLDB |
|---|---|

Set a watchpoint on a variable when it is written to.

| | |
|---|---|
| **(gdb)** `watch global_var` | **(lldb)** `watchpoint set variable global_var`<br>**(lldb)** `wa s v global_var` |

Set a watchpoint on a memory location when it is written into. The size of the region to watch for defaults to the pointer size if no '-x byte_size' is specified. This command takes raw input, evaluated as an expression returning an unsigned integer pointing to the start of the region, after the '--' option terminator.

| GDB | LLDB |
|---|---|
| `(gdb) watch -location g_char_ptr` | `(lldb) watchpoint set expression -- my_ptr`<br>`(lldb) wa s e -- my_ptr` |

Set a condition on a watchpoint.

```
(lldb) watch set var global
(lldb) watchpoint modify -c
'(global==5)'
(lldb) c
...
(lldb) bt
* thread #1: tid = 0x1c03,
0x0000000100000ef5
a.out`modify + 21 at
main.cpp:16, stop reason =
watchpoint 1
frame #0: 0x0000000100000ef5
a.out`modify + 21 at
main.cpp:16
frame #1: 0x0000000100000eac
a.out`main + 108 at
main.cpp:25
frame #2: 0x00007fff8ac9c7e1
libdyld.dylib`start + 1
(lldb) frame var global
(int32_t) global = 5
```

List all watchpoints.

| GDB | LLDB |
|---|---|
| `(gdb) info break` | `(lldb) watchpoint list`<br>`(lldb) watch l` |

Delete a watchpoint.

| GDB | LLDB |
|---|---|
| `(gdb) delete 1` | `(lldb) watchpoint delete 1`<br>`(lldb) watch del 1` |

## Examining Variables

Show the arguments and local variables for the current frame.

| GDB | LLDB |
|---|---|
| `(gdb) info args`<br>and<br>`(gdb) info locals` | `(lldb) frame variable`<br>`(lldb) fr v` |

Show the local variables for the current frame.

| GDB | LLDB |
|---|---|
| `(gdb) info locals` | `(lldb) frame variable --no-args`<br>`(lldb) fr v -a` |

Show the contents of local variable "bar".

| | |
|---|---|
| **(gdb)** p bar | **(lldb)** frame variable bar<br>**(lldb)** fr v bar<br>**(lldb)** p bar |

Show the contents of local variable "bar" formatted as hex.

| | |
|---|---|
| **(gdb)** p/x bar | **(lldb)** frame variable --format x bar<br>**(lldb)** fr v -f x bar |

Show the contents of global variable "baz".

| | |
|---|---|
| **(gdb)** p baz | **(lldb)** target variable baz<br>**(lldb)** ta v baz |

Show the global/static variables defined in the current source file.

| | |
|---|---|
| n/a | **(lldb)** target variable<br>**(lldb)** ta v |

Display the variables "argc" and "argv" every time you stop.

| | |
|---|---|
| **(gdb)** display argc<br>**(gdb)** display argv | **(lldb)** target stop-hook add --one-liner "frame variable argc argv"<br>**(lldb)** ta st a -o "fr v argc argv"<br>**(lldb)** display argc<br>**(lldb)** display argv |

Display the variables "argc" and "argv" only when you stop in the function named **main**.

| | |
|---|---|
| | **(lldb)** target stop-hook add --name main --one-liner "frame variable argc argv"<br>**(lldb)** ta st a -n main -o "fr v argc argv" |

Display the variable "*this" only when you stop in c class named **MyClass**.

| | |
|---|---|
| | **(lldb)** target stop-hook add --classname MyClass --one-liner "frame variable *this"<br>**(lldb)** ta st a -c MyClass -o "fr v *this" |

## Evaluating Expressions

| GDB | LLDB |
|---|---|

Evaluating a generalized expression in the current frame.

| | |
|---|---|
| **(gdb)** print (int) printf ("Print nine: %d.", 4 + 5) or if you don't want to see void returns: **(gdb)** call (int) printf ("Print nine: %d.", 4 + 5) | **(lldb)** expr (int) printf ("Print nine: %d.", 4 + 5) or using the print alias: **(lldb)** print (int) printf ("Print nine: %d.", 4 + 5) |

Creating and assigning a value to a convenience variable.

| | |
|---|---|
| **(gdb)** set $foo = 5 **(gdb)** set variable $foo = 5 or using the print command **(gdb)** print $foo = 5 or using the call command **(gdb)** call $foo = 5 and if you want to specify the type of the variable: **(gdb)** set $foo = (unsigned int) 5 | In lldb you evaluate a variable declaration expression as you would write it in C: **(lldb)** expr unsigned int $foo = 5 |

Printing the ObjC "description" of an object.

| | |
|---|---|
| **(gdb)** po [SomeClass returnAnObject] | **(lldb)** expr -o -- [SomeClass returnAnObject] or using the po alias: **(lldb)** po [SomeClass returnAnObject] |

Print the dynamic type of the result of an expression.

| | |
|---|---|
| **(gdb)** set print object 1 **(gdb)** p someCPPObjectPtrOrReference only works for C++ objects. | **(lldb)** expr -d 1 -- [SomeClass returnAnObject] **(lldb)** expr -d 1 -- someCPPObjectPtrOrReference or set dynamic type printing to be the default: **(lldb)** settings set target.prefer-dynamic run-target |

Calling a function so you can stop at a breakpoint in the function.

| | |
|---|---|
| **(gdb)** set unwindonsignal 0 **(gdb)** p function_with_a_breakpoint() | **(lldb)** expr -i 0 -- function_with_a_breakpoint() |

Calling a function that crashes, and stopping when the function crashes.

| | |
|---|---|
| **(gdb)** set unwindonsignal 0 **(gdb)** p function_which_crashes() | **(lldb)** expr -u 0 -- function_which_crashes() |

## Examining Thread State

| GDB | LLDB |
|---|---|
| List the threads in your program. | |
| `(gdb) info threads` | `(lldb) thread list` |
| Select thread 1 as the default thread for subsequent commands. | |
| `(gdb) thread 1` | `(lldb) thread select 1`<br>`(lldb) t 1` |
| Show the stack backtrace for the current thread. | |
| `(gdb) bt` | `(lldb) thread backtrace`<br>`(lldb) bt` |
| Show the stack backtraces for all threads. | |
| `(gdb) thread apply all bt` | `(lldb) thread backtrace all`<br>`(lldb) bt all` |
| Backtrace the first five frames of the current thread. | |
| `(gdb) bt 5` | `(lldb) thread backtrace -c 5`<br>`(lldb) bt 5` (*lldb-169 and later*)<br>`(lldb) bt -c 5` (*lldb-168 and earlier*) |
| Select a different stack frame by index for the current thread. | |
| `(gdb) frame 12` | `(lldb) frame select 12`<br>`(lldb) fr s 12`<br>`(lldb) f 12` |
| List information about the currently selected frame in the current thread. | |
| | `(lldb) frame info` |
| Select the stack frame that called the current stack frame. | |
| `(gdb) up` | `(lldb) up`<br>`(lldb) frame select --relative=1` |
| Select the stack frame that is called by the current stack frame. | |
| `(gdb) down` | `(lldb) down`<br>`(lldb) frame select --relative=-1`<br>`(lldb) fr s -r-1` |
| Select a different stack frame using a relative offset. | |
| | `(lldb) frame select --relative 2` |

| | |
|---|---|
| `(gdb) up 2`<br>`(gdb) down 3` | `(lldb) fr s -r2`<br><br>`(lldb) frame select --relative -3`<br>`(lldb) fr s -r-3` |

Show the general purpose registers for the current thread.

| | |
|---|---|
| `(gdb) info registers` | `(lldb) register read` |

Write a new decimal value '123' to the current thread register 'rax'.

| | |
|---|---|
| `(gdb) p $rax = 123` | `(lldb) register write rax 123` |

Skip 8 bytes ahead of the current program counter (instruction pointer). Note that we use backticks to evaluate an expression and insert the scalar result in LLDB.

| | |
|---|---|
| `(gdb) jump *$pc+8` | ``(lldb) register write pc `$pc+8` `` |

Show the general purpose registers for the current thread formatted as **signed decimal**. LLDB tries to use the same format characters as **printf(3)** when possible. Type "help format" to see the full list of format specifiers.

| | |
|---|---|
| | `(lldb) register read --format i`<br>`(lldb) re r -f i`<br><br>*LLDB now supports the GDB shorthand format syntax but there can't be space after the command:*<br>`(lldb) register read/d` |

Show all registers in all register sets for the current thread.

| | |
|---|---|
| `(gdb) info all-registers` | `(lldb) register read --all`<br>`(lldb) re r -a` |

Show the values for the registers named "rax", "rsp" and "rbp" in the current thread.

| | |
|---|---|
| `(gdb) info all-registers rax rsp rbp` | `(lldb) register read rax rsp rbp` |

Show the values for the register named "rax" in the current thread formatted as **binary**.

| | |
|---|---|
| `(gdb) p/t $rax` | `(lldb) register read --format binary rax`<br>`(lldb) re r -f b rax`<br><br>*LLDB now supports the GDB shorthand format syntax but there can't be space after the command:* |

```
(lldb) register read/t rax
(lldb) p/t $rax
```

Read memory from address 0xbffff3c0 and show 4 hex uint32_t values.

```
(gdb) x/4xw 0xbffff3c0
```

```
(lldb) memory read --size 4 --format x --count 4 0xbffff3c0
(lldb) me r -s4 -fx -c4 0xbffff3c0
(lldb) x -s4 -fx -c4 0xbffff3c0
```

*LLDB now supports the GDB shorthand format syntax but there can't be space after the command:*

```
(lldb) memory read/4xw 0xbffff3c0
(lldb) x/4xw 0xbffff3c0
(lldb) memory read --gdb-format 4xw 0xbffff3c0
```

Read memory starting at the expression "argv[0]".

```
(gdb) x argv[0]
```

```
(lldb) memory read `argv[0]`
```

***NOTE:*** *any command can inline a scalar expression result (as long as the target is stopped) using backticks around any expression:*

```
(lldb) memory read --size `sizeof(int)` `argv[0]`
```

Read 512 bytes of memory from address 0xbffff3c0 and save results to a local file as **text**.

```
(gdb) set logging on
(gdb) set logging file /tmp/mem.txt
(gdb) x/512bx 0xbffff3c0
(gdb) set logging off
```

```
(lldb) memory read --outfile /tmp/mem.txt --count 512 0xbffff3c0
(lldb) me r -o/tmp/mem.txt -c512 0xbffff3c0
(lldb) x/512bx -o/tmp/mem.txt 0xbffff3c0
```

Save binary memory data starting at 0x1000 and ending at 0x2000 to a file.

```
(gdb) dump memory /tmp/mem.bin 0x1000 0x2000
```

```
(lldb) memory read --outfile /tmp/mem.bin --binary 0x1000 0x2000
(lldb) me r -o /tmp/mem.bin -b 0x1000 0x2000
```

Get information about a specific heap allocation (available on macOS only).

```
(lldb) command script import
```

| | |
|---|---|
| **(gdb)** `info malloc 0x10010d680` | `lldb.macosx.heap`<br>**(lldb)** `process launch --environment MallocStackLogging=1 -- [ARGS]`<br>**(lldb)** `malloc_info --stack-history 0x10010d680` |

Get information about a specific heap allocation and cast the result to any dynamic type that can be deduced (available on macOS only)

| | |
|---|---|
| | **(lldb)** `command script import lldb.macosx.heap`<br>**(lldb)** `malloc_info --type 0x10010d680` |

Find all heap blocks that contain a pointer specified by an expression EXPR (available on macOS only).

| | |
|---|---|
| | **(lldb)** `command script import lldb.macosx.heap`<br>**(lldb)** `ptr_refs EXPR` |

Find all heap blocks that contain a C string anywhere in the block (available on macOS only).

| | |
|---|---|
| | **(lldb)** `command script import lldb.macosx.heap`<br>**(lldb)** `cstr_refs CSTRING` |

Disassemble the current function for the current frame.

| | |
|---|---|
| **(gdb)** `disassemble` | **(lldb)** `disassemble --frame`<br>**(lldb)** `di -f` |

Disassemble any functions named **main**.

| | |
|---|---|
| **(gdb)** `disassemble main` | **(lldb)** `disassemble --name main`<br>**(lldb)** `di -n main` |

Disassemble an address range.

| | |
|---|---|
| **(gdb)** `disassemble 0x1eb8 0x1ec3` | **(lldb)** `disassemble --start-address 0x1eb8 --end-address 0x1ec3`<br>**(lldb)** `di -s 0x1eb8 -e 0x1ec3` |

Disassemble 20 instructions from a given address.

| | |
|---|---|
| **(gdb)** `x/20i 0x1eb8` | **(lldb)** `disassemble --start-address 0x1eb8 --count 20`<br>**(lldb)** `di -s 0x1eb8 -c 20` |

Show mixed source and disassembly for the current function for the current frame.

| | |
|---|---|
| | **(lldb)** `disassemble --frame --` |

| | |
|---|---|
| n/a | mixed<br>**(lldb)** di -f -m |

Disassemble the current function for the current frame and show the opcode bytes.

| | |
|---|---|
| n/a | **(lldb)** disassemble --frame --bytes<br>**(lldb)** di -f -b |

Disassemble the current source line for the current frame.

| | |
|---|---|
| n/a | **(lldb)** disassemble --line<br>**(lldb)** di -l |

## Executable and Shared Library Query Commands

| GDB | LLDB |
|---|---|

List the main executable and all dependent shared libraries.

| | |
|---|---|
| **(gdb)** info shared | **(lldb)** image list |

Look up information for a raw address in the executable or any shared libraries.

| | |
|---|---|
| **(gdb)** info symbol 0x1ec4 | **(lldb)** image lookup --address 0x1ec4<br>**(lldb)** im loo -a 0x1ec4 |

Look up functions matching a regular expression in a binary.

| | |
|---|---|
| **(gdb)** info function `<FUNC_REGEX>` | This one finds debug symbols:<br>**(lldb)** image lookup -r -n `<FUNC_REGEX>`<br><br>This one finds non-debug symbols:<br>**(lldb)** image lookup -r -s `<FUNC_REGEX>`<br><br>Provide a list of binaries as arguments to limit the search. |

Find full source line information.

| | |
|---|---|
| **(gdb)** info line 0x1ec4 | This one is a bit messy at present. Do:<br><br>**(lldb)** image lookup -v --address 0x1ec4<br><br>and look for the LineEntry |

| | |
|---|---|
| | line, which will have the full source path and line range information. |

Look up information for an address in **a.out** only.

| | |
|---|---|
| | **(lldb)** image lookup --address 0x1ec4 a.out<br>**(lldb)** im loo -a 0x1ec4 a.out |

Look up information for for a type `Point` by name.

| | |
|---|---|
| **(gdb)** ptype Point | **(lldb)** image lookup --type Point<br>**(lldb)** im loo -t Point |

Dump all sections from the main executable and any shared libraries.

| | |
|---|---|
| **(gdb)** maintenance info sections | **(lldb)** image dump sections |

Dump all sections in the **a.out** module.

| | |
|---|---|
| | **(lldb)** image dump sections a.out |

Dump all symbols from the main executable and any shared libraries.

| | |
|---|---|
| | **(lldb)** image dump symtab |

Dump all symbols in **a.out** and **liba.so**.

| | |
|---|---|
| | **(lldb)** image dump symtab a.out liba.so |

## Miscellaneous

| GDB | LLDB |
|---|---|

Search command help for a keyword.

| | |
|---|---|
| **(gdb)** apropos keyword | **(lldb)** apropos keyword |

Echo text to the screen.

| | |
|---|---|
| **(gdb)** echo Here is some text\n | **(lldb)** script print "Here is some text" |

Remap source file pathnames for the debug session. If your source files are no longer located in the same location as when the program was built --- maybe the program was built on a different computer --- you need to tell the debugger how to find the sources at their local file path instead of the build system's file path.

| | |
|---|---|
| **(gdb)** set pathname- | **(lldb)** settings set |

| | |
|---|---|
| `substitutions /buildbot/path /my/path` | `target.source-map /buildbot/path /my/path` |

Supply a catchall directory to search for source files in.

| | |
|---|---|
| **(gdb)** `directory /my/path` | (*No equivalent command - use the source-map instead.*) |