LLDB LLVM debugger

START HERE

`lldb -- <prog> <args>`
Debug a program with arguments

`bt`
Show the call stack

`b <func>`
Stop at a function

`frame variable`
Show locals and arguments

`run`
Start the program

`p <expr>`
Evaluate an expression

`n / s`
Over / into the next line

`c`
Resume until the next stop

CONCEPTS

Compile with -g (and -O0 for sane stepping), otherwise LLDB has no source lines or local variables.

Debugger settings, aliases
- **Target** executable, breakpoints
  - **Process** running program
    - **Thread** bt, step
      - **Frame** variables, registers

Commands are noun verb (breakpoint set, frame select); most have short aliases such as b, n, s, c, bt, p, f.

Abbreviate any word to a unique prefix, and use help <cmd> or apropos <word> to explore.

START AND ATTACH

`lldb <prog>`
Load a program

`lldb -- <prog> <args>`
Load a program with arguments

`lldb -p <pid>`
Attach to a running process
Often needs root or ptrace permission (kernel.yama.ptrace_scope)

`lldb -c <core> <prog>`
Inspect a core dump

`lldb -b -o run -o bt -- <prog>`
Non-interactive: run, print a backtrace, exit

`run [args]`
Start the program

`process attach --name <name>`
Attach by process name

`settings set target.run-args <args>`
Set arguments for later runs

`settings set target.env-vars <VAR>=<val>`
Set an environment variable

`process detach`
Release the process and keep it running

`process kill`
Kill the program being debugged

`quit`
Leave LLDB

BREAKPOINTS

`b <func>`
Break at a function

`b <file>:<line>`
Break at a line of a file

`breakpoint set --name <func> --condition '<cond>'`
Break only when the condition holds

`breakpoint list`
List breakpoints and hit counts

`breakpoint delete <n>`
Delete a breakpoint

`breakpoint disable <n>`
Disable a breakpoint

`breakpoint enable <n>`
Re-enable a breakpoint

`breakpoint set --func-regex <regex>`
Break on every function matching a regex

`breakpoint set --method <name>`
Break on a C++ method by base name

`breakpoint modify --condition '<cond>' <n>`
Add or change a condition

`breakpoint command add <n>`
Run commands when it hits

WATCHPOINTS

`watchpoint set variable <var>`
Stop when a variable changes

`watchpoint set expression -- <addr>`
Watch the memory at an address

`watchpoint list`
List watchpoints

`watchpoint delete <n>`
Delete a watchpoint

RUN CONTROL

`c`
Continue until the next stop

`n`
Step over the next line

`s`
Step into the next line

`finish`
Run until the current function returns

`thread until <line>`
Run to a line in this frame

`si`
Step one instruction

`ni`
Step over one instruction (calls too)

RUN CONTROL (cont.)

`process status`
Show the process state and stop reason

STACK AND THREADS

`bt`
Backtrace of the current thread

`bt all`
Backtrace of every thread

`frame select <n>`
Switch to frame n

`up`
Go to the caller frame

`down`
Go back toward the callee frame

`frame info`
Show the current frame

`thread list`
List threads

`thread select <n>`
Switch to thread n

INSPECT DATA

`frame variable`
Show arguments and locals

`frame variable <var>`
Show one variable

`frame variable --format x <var>`
Show a variable in hex

`target variable <var>`
Show a global or static variable

`p <expr>`
Evaluate a C/C++ expression

`expr -- <expr>`
Evaluate, with options before the --

`parray <count> <ptr>`
Print count elements behind a pointer

`target stop-hook add -o "frame variable <var>"`
Show a variable on every stop

MEMORY AND REGISTERS

`memory read --size 4 --format x --count 4 <addr>`
Dump four 32-bit words in hex

`x/4xw <addr>`
Same, in GDB shorthand

`register read`
Show general registers

`register read <reg>`
Show one register

SOURCE AND SYMBOLS

`disassemble --frame`
Disassemble the current function

`disassemble --name <func>`
Disassemble a function by name

`disassemble --start-address <addr> --count <n>`
Disassemble n instructions from an address

`image list`
List loaded executables and shared libraries

`image lookup --address <addr>`
Which symbol is at this address?

`image lookup -r -n <regex>`
Find functions with debug info by regex

`image lookup --type <type>`
Show a type's layout

FROM GDB

GDB to LLDB

GDB	LLDB
`break foo`	b foo
`rbreak <re>`	breakpoint set -r <re>
`info locals`	frame variable
`info break`	breakpoint list
`watch <var>`	watchpoint set variable <var>
`info threads`	thread list
`disassemble`	disassemble --frame
`info registers`	register read

RECIPES

Crash post-mortem

```
ulimit -c unlimited
./<prog>            # crashes, writes core
lldb -c <core> <prog>
(lldb) bt
```

Stop in a loop on a condition

```
(lldb) b <file>:<line>
(lldb) breakpoint modify -c 'i == 99' 1
(lldb) run
(lldb) frame variable
```

Who changes this variable?

```
(lldb) b main
(lldb) run
(lldb) watchpoint set variable <var>
(lldb) c    # stops at the writing line
(lldb) bt
```

Log and go on

```
(lldb) b <func>
(lldb) breakpoint command add 1
> frame variable <var>
> continue
> DONE
```