LLDB Installation and Basic Commands Usage Guide

Document Version: 2.0

Last Updated: 10th April 2026

Document Owner:
Dhruv Srivastava
dhruv.srivastava@ibm.com
IBM AIX

1. Table of Contents

2. Purpose

This document provides a practical reference for installation, ways to get started and the most commonly used LLDB commands, with reference to IBM AIX. The intention of this document is to help users debug applications efficiently by understanding the syntax and use-cases of the essential commands for program execution control, breakpoints, inspection, memory analysis, stack navigation, symbol inspection and more.

3. Installing lldb

There are two possible scenarios for installing LLDB on your AIX lpar:

1. **LLDB fileset installation:**
   If you have downloaded the lldb fileset from IBM's official AIX Web-download site, please install the fileset using the install command along with the required options. For e.g.:

   ```
   installp -acgXYd . lldb.rte
   ```

2. **Tarball package from AIX:**
   If you have received a tarball from IBM you can use the **install-lldb.sh** script received along with it to install lldb binaries in the appropriate places.
   Run the script *install-lldb.sh* to setup lldb files in */opt/lldb/bin*
   The script creates link within the */usr/bin* so you can directly run:

   ```
   lldb application
   ```

   Please make sure that both lldb and lldb-server are present in */usr/bin* together.

4. Starting lldb

You can launch a debugging session as follows:

1. For launching an application

   ```
   lldb application
   ```

3. For attaching to a running process

   ```
   lldb -p <pid>
   ```

4. For debugging an AIX coredump

```
lldb --core ./core
```

5. Command Usage:

This section gives information about functionality and usage of the most commonly used commands in a debugging session with LLDB.

a. Help and Command discovery

LLDB has an **auto-complete** feature which lets you check all the possible options related to your commands within the lldb-prompt itself, which can be utilized to check all the possible commands related to your keyword.

Command	Description
help	List all available commands
help <command>	Show help and relevant options for a specific command. This command gives details for all the command and its option variants based on your input. e.g.: *help image dump sections*
apropos <keyword>	Search command details by using any specific keyword related to it

The help command can also give you the details about a full-length command with multiple options as given in the example in the table above, and this can be utilized to know details about specialized options/keywords along with the command.

b. Program Execution Control

Command	Alias	Description
run	r	Start program execution
run arg1 arg2 …	r arg1 arg2 …	Start execution with arguments
continue	c	Resume execution after stopping
process kill	kill k	Terminate the current process
quit	q	Exit lldb

c. Breakpoints

Command	Alias	Description

breakpoint set --name <func>	b <func>	Set breakpoint in function named <func>
breakpoint set --file <file.c> --line <line >	b <file.c>:<line>	Set breakpoint in file at a particular line number
breakpoint set --address <addr>	b <addr>	Set breakpoint at an address
breakpoint list	br list	List all existing breakpoints
breakpoint delete <id>	br del <id>	Delete breakpoint with id <id>
breakpoint disable <id>	br di <id>	Disable breakpoint with id <id>
breakpoint enable <id>	br e <id>	Enable breakpoint with id <id>
breakpoint command add <id>		Add additional commands to watchpoints
breakpoint command list <id>		List all the commands attached to watchpoint <id>
breakpoint command delete <id>		Delete the set of commands from a watchpoint

Breakpoint modify:

You can add conditions to breakpoints after setting a breakpoint by using the modify command on an existing watchpoint like this:

```
breakpoint modify 1 -c 'x > 10'
```

Breakpoint command:
You can attach other commands to breakpoint hits to ask lldb to execute them on every watchpoint hit like below:

```
breakpoint command add 1
> print a
> continue
> DONE
```

d. Watchpoints

Few points to note for AIX platform:
1. AIX only supports **1 hardware watchpoint** per process per debugging session. You cannot simultaneously put more than one watchpoint in different locations. To watch a different location, you will need to remove the watchpoint from the current

location being watched.
2. AIX only supports **write watchpoints**. A read watchpoint is not supported

3. AIX watchpoints are of **8 bytes and 8 byte aligned**. Though lldb can internally handle watching smaller sizes.

Command	Alias	Description
watchpoint set variable <var>	wa s v <var>	Set a watchpoint on a variable
watchpoint set variable -s <size> <var>	wa s v -s <size> <var>	Set watchpoint on a variable with specific bytes
watchpoint list	wa l	List all watchpoints
watchpoint enable <id>	wa en <id>	Enable watchpoint with id <id>
watchpoint disable <id>	wa di <id>	Disable watchpoint with id <id>
watchpoint delete <id>	wa del <id>	Delete watchpoint with id <id>
watchpoint modify -c <condition> <id>		Add a condition to the watchpoint
watchpoint set expression -- <expr>	wa s expr -- <expr>	Set watchpoint on an address from the expression <expr>
watchpoint set expression -s <size> -- <expr>	wa s expr -s 4 -- <expr>	Set watchpoint on address from expression with given size
watchpoint ignore -i <count> <id>		Ignore watchpoint for the first -i hit counts starting from 1
watchpoint command add <id>		Add additional commands to watchpoints
watchpoint command list <id>		List all the commands attached to watchpoint <id>
watchpoint command delete <id>		Delete the set of commands from a watchpoint

Watchpoint modify:

You can add conditions to watchpoints after setting a watchpoint by using the modify command on an existing watchpoint like this:

```
watchpoint modify 1 -c 'x > 10'
```

Watchpoint command:
You can attach other commands to watchpoint hits to ask lldb to execute them on every watchpoint hit like below:

```
watchpoint command add 1
```

```
> print a
> continue
> DONE
```

e. Stepping through Code

Command	Alias	Description
step	s	Step into function
next	n	Move to the next line OR Step over function
finish		Run until the current function returns
thread step-inst	stepi	Step one instruction
thread step-inst-over	nexti	Step over for one instruction

f. Stack and Frame Inspection

Command	Alias	Description
bt	bt	Show backtrace of the current thread's call stack
frame info		Show current frame
frame variable	fr v	Show all local variables
frame variable x	fr v x	Show variable x's details

g. Variable, Memory and Register Inspection

Command	Alias	Description
print <var>	p <var>	Print variable
memory read <address>	m read <addr>	Read memory at address
register read	reg read	Display all register values
register info	reg info	Display register information
register write <reg> <value>	reg w <reg> <val>	Modify register value

h. Disassembly and Symbols

Command	Alias	Description
disassemble	di	Disassemble from current instruction
disassemble –start-address <address>	di -a <addr>	Disassemble from current address
image lookup -n <symbol>		Lookup the given symbol
image lookup -a <address>		Lookup the symbol related to the given address
image dump symtab		Dump the symbol table of the target module

i. Shared libraries and Modules

Command	Description
image list	List current executable and dependent shared library images
image dump sections	Dump the sections from one or more target modules
image dump symfile	Dump the debug symbol file for one or more target modules

j. Process and Thread Information

Command	Description
process status	Process state details
process attach -p <pid>	Attach to a process with pid <pid>
detach	Detach from the current process

k. Logging and Debugging

The lldb logging channel is a pretty friendly interface to get the lldb workflow logs during your debugging session. It is divided into four channels which is further divided into subchannels. You can enable multiple subchannels at once as per the needs to get the debugging sessions log information. Use the below command to get info on all the available logging channels and subchannels:

```
log list
```

You can use them as follows:

```
log enable lldb all
```

```
log enable lldb process platform target
```

```
log enable dwarf info
```

```
log disable
```

l. Useful Settings

You can also configure LLDB for a ton of configurable settings and make it behave accordingly. All these configurable settings can be listed using:

```
settings show
```

Add the required setting using settings set, as given below for example:

```
settings set target.process.thread.step-avoid-libraries
```

6. Quick References for Common Command Aliases

Alias	Command
b main	breakpoint set --name main
b file.c:20	breakpoint set --file file.c --line 20
r	run
c	continue
n	next
s	step
bt	thread backtrace
p x	print x
fr v	frame variable
di	disassembly
reg r	register read
x <addr>	memory read <addr>
q	quit

7. Command Mapping – DBX vs GDB vs LLDB

Command	DBX	LLDB	GDB

Breakpoint Commands	stop in FuncName/LineNum st in FuncName/LineNum	Breakpoint at Function ▪ breakpoint set -- name FuncName ▪ br s -n FuncName ▪ b FuncName Breakpoint at a line ▪ breakpoint set --file FileName --line LineNum ▪ br s -f FileName -l LineNumber b FileName: LineNumber	break FuncName break FileName:LineNumber
List all Breakpoints	status	breakpoint list br l	info break
Delete a Breakpoint	delete BreakPointNumber	breakpoint delete BreakPointNumber br del BreakPointNumber	delete BreakPointNumber
Disable a Breakpoint	disable BreakPointNumber	breakpoint disable BreakPointNumber br dis BreakPointNumber	disable BreakPointNumber
Enable a Breakpoint	enable BreakPointNumber	breakpoint enable BreakPointNumber br en BreakPointNumber	enable BreakPointNumber
Launch a process with and without arguments	run <args> r <args> run r	process launch -- <args> run <args> r <args> process launch run r	run <args> r <args> run r
Do a source level single step in the currently selected thread	step s	step s	step s
Do an instruction level single step in the currently selected thread	stepi	thread step-inst stepi si	stepi si
Do an instruction level single step over in the currently selected thread	nexti	thread step-inst-over nexti ni	nexti ni
Step out of the currently selected frame	return	thread step-out finish	finish
Examining Variables	print VariableName p VariableName	print VariableName p VariableName	print VariableName p VariableName ta v VariableName (global) fr v VariableName (local)

Show the stack backtrace for the current/all thread	where	thread backtrace bt thread backtrace all bt all	bt thread apply all bt
List of all commands in a debugger	help	help	help
Want to come out of debugging	quit q	quit q	quit q
Data Type of variable	whatis VariableName	frame variable -T VariableName fr v -T VariableName fr v VariableName type lookup TypeName (structure)	what VariableName whatis VariableName
show the general purpose registers for the current thread	registers	register read	info registers
Write a new decimal Value to the current thread register RegisterName	registers	register write RegisterName Value	p RegisterName = Value
Show all registers in all register sets for the current thread	registers	register read --all re r -a	info all-registers
Continuing Program Execution Until Next Breakpoint or Termination	cont c	continue cont c	continue cont c
List the threads in your program	thread	thread list	info threads
View Source Code Context	list	list	list
Attach to the process	attach -a PID	process attach --pid PID attach -p PID process attach --name BinaryName pro at -n BinaryName process attach --name BinaryName --waitfor pro at -n BinaryName -w	attach PID attach BinaryName attach --waitfor BinaryName
List the main executable and all dependent shared libraries	map	image list	info shared

Dump all sections from the main executable and any shared libraries	map	image dump sections	maintenance info sections
display disassembled machine code	listi	di disassemble –frame (current function) di -f disassemble --name Func (Any function named Func) di -n Func disassemble --start-address ADDR1 --end-address ADDR2 (Address Range) di -s ADDR1 -e ADDR2 di -a ADDR	disassemble disassemble Func (Any function named Func) disassemble ADDR1 ADDR2 (Address Range)
Show the current frame and source line	frame	frame select f process status	frame
Set Hardware watchpoint	stophwp &VariableName bytes	watchpoint set variable VariableName wa s v VariableName	Watch VariableName
List all Watchpoints	status	watchpoint list wa l	Info break
Delete a watchpoint	delete WatchPointNumber	watchpoint delete WatchPointNumber wa del WatchPointNumber	delete WatchPointNumber

8. Sample Debugging Workflow

Below is an example debugging session:

```
Example testcase: test1.c
-------------------------
int func(int x)
{
    x++;
    return x;
}
int main()
```

```
{
   int a = 10;
      return a + func(a);
}

Compilation:
------------
ibm-clang_r test.c -o test -g

# lldb /home/dhruv/LLDB/tests/test1
(lldb) target create "/home/dhruv/LLDB/tests/test1"
Current executable set to '/home/dhruv/LLDB/tests/test1' (powerpc64).
(lldb) b main
Breakpoint 1: where = test1`main + 28 at test1.c:8, address = 0x000000010000097c
(lldb) r
Process 21823764 launched: '/home/dhruv/LLDB/tests/test1' (powerpc64)
Process 21823764 stopped
* thread #1, name = 'test1', stop reason = breakpoint 1.1
   frame #0: 0x000000010000097c test1`main at test1.c:8
  5    }
  6    int main()
  7    {
-> 8       int a = 10;
  9          return a + func(a);
  10   }
(lldb) bt
* thread #1, name = 'test1', stop reason = breakpoint 1.1
  * frame #0: 0x000000010000097c test1`main at test1.c:8
   frame #1: 0x00000001000004ac test1`__start + 116
(lldb) n
Process 21823764 stopped
* thread #1, name = 'test1', stop reason = step over
   frame #0: 0x0000000100000984 test1`main at test1.c:9
  6    int main()
  7    {
  8       int a = 10;
-> 9          return a + func(a);
  10   }
(lldb) s
Process 21823764 stopped
* thread #1, name = 'test1', stop reason = step in
   frame #0: 0x0000000100000924 test1`func(x=10) at test1.c:3
  1    int func(int x)
  2    {
-> 3          x++;
  4          return x;
  5    }
  6    int main()
  7    {
(lldb) di
test1`func:
   0x100000920 <+0>:  stw    3, -12(1)
->  0x100000924 <+4>:  lwz    3, -12(1)
   0x100000928 <+8>:  addi   3, 3, 1
   0x10000092c <+12>: stw    3, -12(1)
   0x100000930 <+16>: lwa    3, -12(1)
   0x100000934 <+20>: blr
   0x100000938 <+24>: <unknown>
   0x10000093c <+28>: <unknown>
```

```
   0x100000940 <+32>: lwz    0, 257(0)
   0x100000944 <+36>: <unknown>
   0x100000948 <+40>: <unknown>
   0x10000094c <+44>: <unknown>
(lldb) fr v
(int) x = 10
(lldb) p x
(int) 10
(lldb) image list
[  0]                    0x0000000100000438 /home/dhruv/LLDB/tests/test1
[  1]                    0x09fffffff0001000 /usr/ccs/bin/usla64
[  2]                    0x0900000000696200 /usr/lib/libpthreads.a (_shr_xpg5_64.o)
[  3]                    0x0900000000695420 /usr/lib/libcrypt.a (shr_64.o)
[  4]                    0x09000000001fd380 /usr/lib/libc.a (_shr_64.o)
[  5]                    0x09000000000891f8 /usr/lib/libpthreads.a (shr_xpg5_64.o)
[  6]                    0x09000000000001f8 /usr/lib/libc.a (shr_64.o)
(lldb) q
Quitting LLDB will kill one or more processes. Do you really want to proceed: [Y/n] y

#
```

9. Document Details

Document Owner: Dhruv Srivastava (dhruv.srivastava@ibm.com)

Organization: IBM AIX

Version History:

Version: 1.0; **Date:** 21st Jan 2026

Details: Initial version of the AIX lldb installation and command guide

Current Version: 2.0

Dated: 10th April 2026

Details: Updated Hardware Watchpoint commands and added some more Breakpoint commands to the existing list.

End of Document