**Command abbreviations**

**add-dsym** -- Add a debug symbol file to one of the targets current modules by specifying a path to a debug symbols file or by using the options to specify a module.

Syntax: add-dsym <cmd-options>

Command Options Usage:

add-dsym [-u <module-uuid>] [-s <shlib-name>]

add-dsym [-FS]

-F ( --frame )

Locate the debug symbols for the currently selected frame.

-S ( --stack )

Locate the debug symbols for every frame in the current call stack.

-s <shlib-name> ( --shlib <shlib-name> )

Locate the debug symbols for the shared library specified by name.

-u <module-uuid> ( --uuid <module-uuid> )

A module UUID value.

*'add-dsym' is an abbreviation for 'target symbols add'*

**attach** -- Attach to process by ID or name.

Syntax: \_regexp-attach <pid> | <process-name>

*'attach' is an abbreviation for '\_regexp-attach'*

**b** -- Set a breakpoint using one of several shorthand formats.

Syntax:

\_regexp-break <filename>:<linenum>:<colnum>

main.c:12:21 // Break at line 12 and column 21 of main.c

\_regexp-break <filename>:<linenum>

main.c:12 // Break at line 12 of main.c

\_regexp-break <linenum>

12 // Break at line 12 of current file

\_regexp-break 0x<address>

0x1234000 // Break at address 0x1234000

\_regexp-break <name>

main // Break in 'main' after the prologue

\_regexp-break &<name>

&main // Break at first instruction in 'main'

\_regexp-break <module>`<name>

libc.so`malloc // Break in 'malloc' from 'libc.so'

\_regexp-break /<source-regex>/

/break here/ // Break on source lines in current file

// containing text 'break here'.

*'b' is an abbreviation for '\_regexp-break'*

**bt** -- Show the current threads call stack. Any numeric argument displays at most that many frames. The argument 'all' displays all threads. Use 'settings set frame-format' to customize the

printing of individual frames and 'settings set thread-format' to customize the thread header. Expects 'raw' input (see 'help raw-input'.)

Syntax: bt [<digit> | all]

*'bt' is an abbreviation for '\_regexp-bt'*

**c** -- Continue execution of all threads in the current process.

Syntax: c <cmd-options>

Command Options Usage:

c [-i <unsigned-integer>]

-i <unsigned-integer> ( --ignore-count <unsigned-integer> )

Ignore <N> crossings of the breakpoint (if it exists) for the currently selected thread.

*'c' is an abbreviation for 'process continue'*

**call** -- Evaluate an expression on the current thread. Displays any returned value with LLDBs default formatting.

Syntax: call <expr>

*'call' is an abbreviation for 'expression --'*

**continue** -- Continue execution of all threads in the current process.

Syntax: continue <cmd-options>

Command Options Usage:

continue [-i <unsigned-integer>]

-i <unsigned-integer> ( --ignore-count <unsigned-integer> )

Ignore <N> crossings of the breakpoint (if it exists) for the currently selected thread.

*'continue' is an abbreviation for 'process continue'*

**detach** -- Detach from the current target process.

Syntax: detach <cmd-options>

Command Options Usage:

detach [-s <boolean>]

-s <boolean> ( --keep-stopped <boolean> )

Whether or not the process should be kept stopped on detach

(if possible).

*'detach' is an abbreviation for 'process detach'*

**di/dis** -- Disassemble specified instructions in the current target. Defaults to the current function for the current thread and stack frame.

Syntax: dis <cmd-options>

Command Options Usage:

dis [-bmr] -s <address-expression> [-A <arch>] [-C <num-lines>] [-e <address-expression>] [-F <disassembly-flavor>] [-P <plugin>]

dis [-bmr] -s <address-expression> [-A <arch>] [-C <num-lines>] [-c <num-lines>] [-F <disassembly-flavor>] [-P <plugin>]

dis [-bmr] [-A <arch>] [-C <num-lines>] [-c <num-lines>] [-F <disassembly-flavor>] [-n <function-name>] [-P <plugin>]

dis [-bfmr] [-A <arch>] [-C <num-lines>] [-c <num-lines>] [-F <disassembly-flavor>] [-P <plugin>]

dis [-bmpr] [-A <arch>] [-C <num-lines>] [-c <num-lines>] [-F <disassembly-flavor>] [-P <plugin>]

dis [-blmr] [-A <arch>] [-C <num-lines>] [-F <disassembly-flavor>] [-P <plugin>]

dis [-bmr] [-a <address-expression>] [-A <arch>] [-C <num-lines>] [-c <num-lines>] [-F <disassembly-flavor>] [-P <plugin>]

--force

Force dissasembly of large functions.

-A <arch> ( --arch <arch> )

Specify the architecture to use from cross disassembly.

-C <num-lines> ( --context <num-lines> )

Number of context lines of source to show.

-F <disassembly-flavor> ( --flavor <disassembly-flavor> )

Name of the disassembly flavor you want to use. Currently the

only valid options are default, and for Intel architectures,

att and intel.

-P <plugin> ( --plugin <plugin> )

Name of the disassembler plugin you want to use.

-a <address-expression> ( --address <address-expression> )

Disassemble function containing this address.

-b ( --bytes )

Show opcode bytes when disassembling.

-c <num-lines> ( --count <num-lines> )

Number of instructions to display.

-e <address-expression> ( --end-address <address-expression> )

Address at which to end disassembling.

-f ( --frame )

Disassemble from the start of the current frames function.

-l ( --line )

Disassemble the current frames current source line

instructions if there is debug line table information, else

disassemble around the pc.

-m ( --mixed )

Enable mixed source and assembly display.

-n <function-name> ( --name <function-name> )

Disassemble entire contents of the given function name.

-p ( --pc )

Disassemble around the current pc.

-r ( --raw )

Print raw disassembly with no symbol information.

-s <address-expression> ( --start-address <address-expression> )

Address at which to start disassembling.

*'dis' is an abbreviation for 'disassemble'*

**display --** Evaluate an expression at every stop (see 'help target stop-hook'.)

Syntax: \_regexp-display expression

*'display' is an abbreviation for '\_regexp-display'*

**down --** Select a newer stack frame. Defaults to moving one frame, a numeric argument can specify an arbitrary number.

Syntax: \_regexp-down [<count>]

*'down' is an abbreviation for '\_regexp-down'*

**env --** Shorthand for viewing and setting environment variables.

Syntax:

\_regexp-env // Show environment

\_regexp-env <name>=<value> // Set an environment variable

*'env' is an abbreviation for '\_regexp-env'*

**exit --** Quit the LLDB debugger.

Syntax: exit

*'exit' is an abbreviation for 'quit'*

**f --** Select the current stack frame by index from within the current thread (see ‘thread backtrace’.)

Syntax: f <cmd-options> [<frame-index>]

Command Options Usage:

f [-r <offset>] [<frame-index>]

-r <offset> ( --relative <offset> )

A relative frame index offset from the current frame index.

This command takes options and free-form arguments. If your arguments resemble option specifiers (i.e., they start with a – or --), you must use ‘ – ‘ between the end of the command options and the beginning of the arguments.

*‘f’ is an abbreviation for ‘frame select’*

**file** -- Create a target using the argument as the main executable.

Syntax: file <cmd-options> <filename>

Command Options Usage:

file [-a <arch>] [-p <platform-name>] [-v <none>] [-b <none>] [-S <filename>] [-c <filename>] [-s <filename>] [-r <filename>] [-d[<value>]] <filename>

-S <filename> ( --sysroot <filename> )

Specify the SDK root directory that contains a root of all remote system files.

-a <arch> ( --arch <arch> )

Specify the architecture for the target.

-b <none> ( --build <none> )

Specify the initial SDK build number.

-c <filename> ( --core <filename> )

Fullpath to a core file to use for this target.

-d[<value>] ( --no-dependents=[<value>] )

Whether or not to load dependents when creating a target. If

the option is not specified, the value is implicitly

‘default’. If the option is specified but without a value, the

value is implicitly ‘true’.

Values: default | true | false

-p <platform-name> ( --platform <platform-name> )

Specify name of the platform to use for this target, creating

the platform if necessary.

-r <filename> ( --remote-file <filename> )

Fullpath to the file on the remote host if debugging remotely.

-s <filename> ( --symfile <filename> )

Fullpath to a stand alone debug symbols file for when debug

symbols are not in the executable.

-v <none> ( --version <none> )

Specify the initial SDK version to use prior to connecting.

This command takes options and free-form arguments. If your arguments resemble option specifiers (i.e., they start with a – or --), you must use ‘ – ‘ between the end of the command options and the beginning of the arguments.

*‘file’ is an abbreviation for ‘target create’*

**finish --** Finish executing the current stack frame and stop after returning. Defaults to current thread unless specified.

Syntax: finish <cmd-options> [<thread-id>]

Command Options Usage:

finish [-c <count>] [-e <linenum>] [-m <run-mode>] [-a <oolean>] [-t <function-name>] [-A <oolean>] [-r <regular-expression>] [<thread-id>]

*‘finish’ is an abbreviation for ‘thread step-out’*

**history** -- Dump the history of commands in this session. Commands in the history list can be run again using “!<INDEX>”. “!-<OFFSET>” will re-run the command that is <OFFSET> commands from the end of the list (counting the current command).

Syntax: history <cmd-options>

Command Options Usage:

history [-c <unsigned-integer>] [-e <unsigned-integer>] [-s <unsigned-integer>]

history [-C]

-C ( --clear )

Clears the current command history.

-c <unsigned-integer> ( --count <unsigned-integer> )

How many history commands to print.

-e <unsigned-integer> ( --end-index <unsigned-integer> )

Index at which to stop printing history commands.

-s <unsigned-integer> ( --start-index <unsigned-integer> )

Index at which to start printing history commands (or end to

mean tail mode).

*‘history’ is an abbreviation for ‘session history’*

**image** -- Commands for accessing information for one or more target modules.

Syntax: image

*‘image’ is an abbreviation for ‘target modules’*

**j/jump** -- Set the program counter to a new address.

Syntax:

\_regexp-jump <line>

\_regexp-jump +<line-offset> | -<line-offset>

\_regexp-jump <file>:<line>

\_regexp-jump \*<addr>

*‘jump’ is an abbreviation for ‘\_regexp-jump’*

**kill** -- Terminate the current target process.

Syntax: kill

*‘kill’ is an abbreviation for ‘process kill’*

**l/list** -- List relevant source code using one of several shorthand formats.

Syntax:

\_regexp-list <file>:<line> // List around specific file/line

\_regexp-list <line> // List current file around specified line

\_regexp-list <function-name> // List specified function

\_regexp-list 0x<address> // List around specified address

\_regexp-list –[<count>] // List previous <count> lines

\_regexp-list // List subsequent lines

*‘list’ is an abbreviation for ‘\_regexp-list’*

**n/next** -- Source level single step, stepping over calls. Defaults to current thread unless specified.

Syntax: next <cmd-options> [<thread-id>]

Command Options Usage:

next [-c <count>] [-e <linenum>] [-m <run-mode>] [-a <oolean>] [-t <function-name>] [-A <oolean>] [-r <regular-expression>] [<thread-id>]

-A <oolean> ( --step-out-avoids-no-debug <oolean> )

A oolean value, if true stepping out of functions will

continue to step out till it hits a function with debug

information.

-a <oolean> ( --step-in-avoids-no-debug <oolean> )

A oolean value that sets whether stepping into functions will

step over functions with no debug information.

-c <count> ( --count <count> )

How many times to perform the stepping operation – currently

only supported for step-inst and next-inst.

-e <linenum> ( --end-linenumber <linenum> )

The line at which to stop stepping – defaults to the next line

and only supported for step-in and step-over. You can also

pass the string ‘block’ to step to the end of the current

block. This is particularly use in conjunction with

--step-target to step through a complex calling sequence.

-m <run-mode> ( --run-mode <run-mode> )

Determine how to run other threads while stepping the current

thread.

Values: this-thread | all-threads | while-stepping

-r <regular-expression> ( --step-over-regexp <regular-expression> )

A regular expression that defines function names to not to

stop at when stepping in.

-t <function-name> ( --step-in-target <function-name> )

The name of the directly called function step in should stop

at when stepping into.

This command takes options and free-form arguments. If your

arguments resemble option specifiers (i.e., they start with a – or

--), you must use ‘ – ‘ between the end of the command options and

the beginning of the arguments.

*‘next’ is an abbreviation for ‘thread step-over’*

**ni/nexti** -- Instruction level single step, stepping over calls. Defaults to current thread unless specified.

Syntax: ni <cmd-options> [<thread-id>]

Command Options Usage:

ni [-c <count>] [-e <linenum>] [-m <run-mode>] [-a <oolean>] [-t <function-name>] [-A <oolean>] [-r <regular-expression>] [<thread-id>]

-A <oolean> ( --step-out-avoids-no-debug <oolean> )

A oolean value, if true stepping out of functions will

continue to step out till it hits a function with debug

information.

-a <oolean> ( --step-in-avoids-no-debug <oolean> )

A oolean value that sets whether stepping into functions will

step over functions with no debug information.

-c <count> ( --count <count> )

How many times to perform the stepping operation – currently

only supported for step-inst and next-inst.

-e <linenum> ( --end-linenumber <linenum> )

The line at which to stop stepping – defaults to the next line

and only supported for step-in and step-over. You can also

pass the string ‘block’ to step to the end of the current

block. This is particularly use in conjunction with

--step-target to step through a complex calling sequence.

-m <run-mode> ( --run-mode <run-mode> )

Determine how to run other threads while stepping the current

thread.

Values: this-thread | all-threads | while-stepping

-r <regular-expression> ( --step-over-regexp <regular-expression> )

A regular expression that defines function names to not to

stop at when stepping in.

-t <function-name> ( --step-in-target <function-name> )

The name of the directly called function step in should stop

at when stepping into.

This command takes options and free-form arguments. If your

arguments resemble option specifiers (i.e., they start with a – or

--), you must use ‘ – ‘ between the end of the command options and

the beginning of the arguments.

*‘ni’ is an abbreviation for ‘thread step-inst-over’*

**p/print** -- Print a variable or expression.

Syntax: p <expr>

*‘p’ is an abbreviation for ‘expression –'*

**parray** -- parray <COUNT> <EXPRESSION> -- lldb will evaluate EXPRESSION to get a typed-pointer-to-an-array in memory, and will display COUNT elements of that type from the array.

Syntax: parray <expr>

‘parray’ is an abbreviation for ‘expression -Z %1 --'

**po** -- Evaluate an expression on the current thread. Displays any returned value with formatting controlled by the types author.

Syntax: po <expr>

*‘po’ is an abbreviation for ‘expression -O --'*

**poarray** -- poarray <COUNT> <EXPRESSION> -- lldb will evaluate EXPRESSION to get the address of an array of COUNT objects in memory, and will call po on them.

Syntax: poarray <expr>

*‘poarray’ is an abbreviation for ‘expression -O -Z %1 --'*

**q** -- Quit the LLDB debugger.

*‘q’ is an abbreviation for ‘quit’*

**r/run --** Launch the executable in the debugger.

Syntax: r [<run-args>]

*‘r’ is an abbreviation for ‘process launch -c /bin/bash –'*

**rbreak** -- Sets a breakpoint or set of breakpoints in the executable.

Syntax: rbreak <cmd-options>

-A ( --all-files )

All files are searched for source pattern matches.

-C <command> ( --command <command> )

A command to run when the breakpoint is hit, can be provided

more than once, the commands will get run in order left to

right.

-D ( --dummy-breakpoints )

Act on Dummy breakpoints – i.e. breakpoints set before a file

is provided, which prime new targets.

-E <source-language> ( --language-exception <source-language> )

Set the breakpoint on exceptions thrown by the specified

language (without options, on throw but not catch.)

-F <fullname> ( --fullname <fullname> )

Set the breakpoint by fully qualified function names. For C++

this means namespaces and all arguments, and for Objective-C

this means a full functionprototype with class and selector.

Can be repeated multiple times to make one breakpoint for

multiple names.

-G <oolean> ( --auto-continue <oolean> )

The breakpoint will auto-continue after running its commands.

-H ( --hardware )

Require the breakpoint to use hardware breakpoints.

-K <oolean> ( --skip-prologue <oolean> )

sKip the prologue if the breakpoint is at the beginning of a

function. If not set the target.skip-prologue setting is

used.

-L <source-language> ( --language <source-language> )

Specifies the Language to use when interpreting the

breakpoints expression (note: currently only implemented for

setting breakpoints on identifiers). If not set the

target.language setting is used.

-M <method> ( --method <method> )

Set the breakpoint by C++ method names. Can be repeated

multiple times to make one breakpoint for multiple methods.

-N <breakpoint-name> ( --breakpoint-name <breakpoint-name> )

Adds this to the list of names for this breakpoint.

-P <python-class> ( --script-class <python-class> )

The name of the class that will manage a scripted breakpoint.

-R <address> ( --address-slide <address> )

Add the specified offset to whatever address(es) the

breakpoint resolves to. At present this applies the offset

directly as given, and I try to align it to instruction

boundaries.

-S <selector> ( --selector <selector> )

Set the breakpoint by ObjC selector name. Can be repeated

multiple times to make one breakpoint for multiple Selectors.

-T <thread-name> ( --thread-name <thread-name> )

The breakpoint stops only for the thread whose thread name

matches this argument.

-X <function-name> ( --source-regexp-function <function-name> )

When used with ‘-p’ limits the source regex to source

contained in the namedfunctions. Can be repeated multiple

times.

-a <address-expression> ( --address <address-expression> )

Set the breakpoint at the specified address. If the address

maps uniquely to a particular binary, then the address will be

converted to a fileaddress, so that the breakpoint will

track that binary+offset no matter where the binary eventually

loads. Alternately, if you also specify the module – with the

-s option – then the address will be treated as a file address

in that module, and resolved accordingly. Again, this will

allow lldb to track that offset on subsequent reloads. The

module need not have been loaded at the time you specify this

breakpoint, and will get resolved when the module is loaded.

-b <function-name> ( --basename <function-name> )

Set the breakpoint by function basename (C++ namespaces and

arguments will beignored). Can be repeated multiple times to

make one breakpoint for multiplesymbols.

-c <expr> ( --condition <expr> )

The breakpoint stops only if this condition expression

evaluates to true.

-d ( --disable )

Disable the breakpoint.

-f <filename> ( --file <filename> )

Specifies the source file in which to set this breakpoint.

Note, by default lldb only looks for files that are #included

if they use the standard include file extensions. To set

breakpoints on .c/.cpp/.m/.mm files that are #included, set

target.inline-breakpoint-strategy to always.

-h <oolean> ( --on-catch <oolean> )

Set the breakpoint on exception catcH.

-I <count> ( --ignore-count <count> )

Set the number of times this breakpoint is skipped before

stopping.

-k <none> ( --structured-data-key <none> )

The key for a key/value pair passed to the implementation of a

scripted breakpoint. Pairs can be specified more than once.

-l <linenum> ( --line <linenum> )

Specifies the line number on which to set this breakpoint.

-m <oolean> ( --move-to-nearest-code <oolean> )

Move breakpoints to nearest code. If not set the

target.move-to-nearest-code setting is used.

-n <function-name> ( --name <function-name> )

Set the breakpoint by function name. Can be repeated multiple

times to makeone breakpoint for multiple names

-o <oolean> ( --one-shot <oolean> )

The breakpoint is deleted the first time it stop causes a

stop.

-p <regular-expression> ( --source-pattern-regexp <regular-expression> )

Set the breakpoint by specifying a regular expression which is

matched against the source text in a source file or files

specified with the -f can be specified more than once. If no

source files are specified, uses the current default source

file. If you want to match against all source files, pass

the –all-files option.

-q <queue-name> ( --queue-name <queue-name> )

The breakpoint stops only for threads in the queue whose name

is given by this argument.

-r <regular-expression> ( --func-regex <regular-expression> )

Set the breakpoint by function name, evaluating a

regular-expression to find the function name(s).

-s <shlib-name> ( --shlib <shlib-name> )

Set the breakpoint only in this shared library. Can repeat

this option multiple times to specify multiple shared

libraries.

-t <thread-id> ( --thread-id <thread-id> )

The breakpoint stops only for the thread whose TID matches

this argument. The token ‘current’ resolves to the current

threads ID.

-u <column> ( --column <column> )

Specifies the column number on which to set this breakpoint.

-v <none> ( --structured-data-value <none> )

The value for the previous key in the pair passed to the

implementation of a scripted breakpoint. Pairs can be

specified more than once.

-w <oolean> ( --on-throw <oolean> )

Set the breakpoint on exception throW.

-x <thread-index> ( --thread-index <thread-index> )

The breakpoint stops only for the thread whose index matches

this argument.

-y <linespec> ( --joint-specifier <linespec> )

A specifier in the form filename:line[:column] for setting

file & line breakpoints.

*‘rbreak’ is an abbreviation for ‘breakpoint set -r %1’*

**re --** Commands to access registers for the current thread and stack frame.

Syntax: re

*'re' is an abbreviation for 'register'*

**s/step. --** Source level single step, stepping into calls. Defaults to current thread unless specified.

Syntax: step <cmd-options> [<thread-id>]

Command Options Usage:

step [-c <count>] [-e <linenum>] [-m <run-mode>] [-a <boolean>] [-t <function-name>] [-A <boolean>] [-r <regular-expression>] [<thread-id>]

-A <boolean> ( --step-out-avoids-no-debug <boolean> )

A boolean value, if true stepping out of functions will

continue to step out till it hits a function with debug

information.

-a <boolean> ( --step-in-avoids-no-debug <boolean> )

A boolean value that sets whether stepping into functions will

step over functions with no debug information.

-c <count> ( --count <count> )

How many times to perform the stepping operation - currently

only supported for step-inst and next-inst.

-e <linenum> ( --end-linenumber <linenum> )

The line at which to stop stepping - defaults to the next line

and only supported for step-in and step-over. You can also

pass the string 'block' to step to the end of the current

block. This is particularly use in conjunction with

--step-target to step through a complex calling sequence.

-m <run-mode> ( --run-mode <run-mode> )

Determine how to run other threads while stepping the current

thread.

Values: this-thread | all-threads | while-stepping

-r <regular-expression> ( --step-over-regexp <regular-expression> )

A regular expression that defines function names to not to

stop at when stepping in.

-t <function-name> ( --step-in-target <function-name> )

The name of the directly called function step in should stop

at when stepping into.

This command takes options and free-form arguments. If your

arguments resemble option specifiers (i.e., they start with a - or

--), you must use ' -- ' between the end of the command options and

the beginning of the arguments.

*'step' is an abbreviation for 'thread step-in'*

**shell --** Run a shell command on the host.

Syntax: shell <shell-command>

*'shell' is an abbreviation for 'platform shell -h --'*

**si/stepi --** Instruction level single step, stepping into calls. Defaults to current thread unless specified.

Syntax: si <cmd-options> [<thread-id>]

Command Options Usage:

si [-c <count>] [-e <linenum>] [-m <run-mode>] [-a <boolean>] [-t <function-name>] [-A <boolean>] [-r <regular-expression>] [<thread-id>]

-A <boolean> ( --step-out-avoids-no-debug <boolean> )

A boolean value, if true stepping out of functions will

continue to step out till it hits a function with debug

information.

-a <boolean> ( --step-in-avoids-no-debug <boolean> )

A boolean value that sets whether stepping into functions will

step over functions with no debug information.

-c <count> ( --count <count> )

How many times to perform the stepping operation - currently

only supported for step-inst and next-inst.

-e <linenum> ( --end-linenumber <linenum> )

The line at which to stop stepping - defaults to the next line

and only supported for step-in and step-over. You can also

pass the string 'block' to step to the end of the current

block. This is particularly use in conjunction with

--step-target to step through a complex calling sequence.

-m <run-mode> ( --run-mode <run-mode> )

Determine how to run other threads while stepping the current

thread.

Values: this-thread | all-threads | while-stepping

-r <regular-expression> ( --step-over-regexp <regular-expression> )

A regular expression that defines function names to not to

stop at when stepping in.

-t <function-name> ( --step-in-target <function-name> )

The name of the directly called function step in should stop

at when stepping into.

This command takes options and free-form arguments. If your

arguments resemble option specifiers (i.e., they start with a - or

--), you must use ' -- ' between the end of the command options and

the beginning of the arguments.

*'si' is an abbreviation for 'thread step-inst'*

**sif --** Step through the current block, stopping if you step directly into a function whose name matches the TargetFunctionName.

Syntax: sif <TargetFunctionName>

Command Options Usage:

sif [-c <count>] [-e <linenum>] [-m <run-mode>] [-a <boolean>] [-t <function-name>] [-A <boolean>] [-r <regular-expression>] [<thread-id>]

-A <boolean> ( --step-out-avoids-no-debug <boolean> )

A boolean value, if true stepping out of functions will

continue to step out till it hits a function with debug

information.

-a <boolean> ( --step-in-avoids-no-debug <boolean> )

A boolean value that sets whether stepping into functions will

step over functions with no debug information.

-c <count> ( --count <count> )

How many times to perform the stepping operation - currently

only supported for step-inst and next-inst.

-e <linenum> ( --end-linenumber <linenum> )

The line at which to stop stepping - defaults to the next line

and only supported for step-in and step-over. You can also

pass the string 'block' to step to the end of the current

block. This is particularly use in conjunction with

--step-target to step through a complex calling sequence.

-m <run-mode> ( --run-mode <run-mode> )

Determine how to run other threads while stepping the current

thread.

Values: this-thread | all-threads | while-stepping

-r <regular-expression> ( --step-over-regexp <regular-expression> )

A regular expression that defines function names to not to

stop at when stepping in.

-t <function-name> ( --step-in-target <function-name> )

The name of the directly called function step in should stop

at when stepping into.

This command takes options and free-form arguments. If your

arguments resemble option specifiers (i.e., they start with a - or

--), you must use ' -- ' between the end of the command options and

the beginning of the arguments.

*'sif' is an abbreviation for 'thread step-in -e block -t %1'*

**t** -- Change the currently selected thread.

Syntax: t <thread-index>

*'t' is an abbreviation for 'thread select'*

**tbreak** -- Set a one-shot breakpoint using one of several shorthand formats.

Syntax:

\_regexp-break <filename>:<linenum>:<colnum>

main.c:12:21 // Break at line 12 and column 21 of main.c

\_regexp-break <filename>:<linenum>

main.c:12 // Break at line 12 of main.c

\_regexp-break <linenum>

12 // Break at line 12 of current file

\_regexp-break 0x<address>

0x1234000 // Break at address 0x1234000

\_regexp-break <name>

main // Break in 'main' after the prologue

\_regexp-break &<name>

&main // Break at first instruction in 'main'

\_regexp-break <module>`<name>

libc.so`malloc // Break in 'malloc' from 'libc.so'

\_regexp-break /<source-regex>/

/break here/ // Break on source lines in current file

// containing text 'break here'.

*'tbreak' is an abbreviation for '\_regexp-tbreak'*

**undisplay --** Stop displaying expression at every stop (specified by stop-hook index.)

Syntax: \_regexp-undisplay stop-hook-number

*'undisplay' is an abbreviation for '\_regexp-undisplay'*

**up** -- Select an older stack frame. Defaults to moving one frame, a numeric argument can specify an arbitrary number.

Syntax: \_regexp-up [<count>]

*'up' is an abbreviation for '\_regexp-up'*

**v/var/vo** -- Show variables for the current stack frame. Defaults to all arguments and local variables in scope. Names of argument, local, file static and file global variables can be specified.

Syntax: var <cmd-options> [<variable-name> [<variable-name> [...]]]

Command Options Usage:

var [-AFLORTacglrst] [-y <name>] [-z <name>] [-f <format>] [-G <gdb-format>] [-d <none>] [-S <boolean>] [-D <count>] [-P <count>] [-Y[<count>]] [-V <boolean>] [-Z <count>] [<variable-name> [<variable-name> [...]]]

-A ( --show-all-children )

Ignore the upper bound on the number of children to show.

-D <count> ( --depth <count> )

Set the max recurse depth when dumping aggregate types

(default is infinity).

-F ( --flat )

Display results in a flat format that uses expression paths

for each variable or member.

-G <gdb-format> ( --gdb-format <gdb-format> )

Specify a format using a GDB format specifier string.

-L ( --location )

Show variable location information.

-O ( --object-description )

Display using a language-specific description API, if

possible.

-P <count> ( --ptr-depth <count> )

The number of pointers to be traversed when dumping values

(default is zero).

-R ( --raw-output )

Dont use formatting options.

-S <boolean> ( --synthetic-type <boolean> )

Show the object obeying its synthetic provider, if available.

-T ( --show-types )

Show variable types when dumping values.

-V <boolean> ( --validate <boolean> )

Show results of type validators.

-Y[<count>] ( --no-summary-depth=[<count>] )

Set the depth at which omitting summary information stops

(default is 1).

-Z <count> ( --element-count <count> )

Treat the result of the expression as if its type is an array

of this many values.

-a ( --no-args )

Omit function arguments.

-c ( --show-declaration )

Show variable declaration information (source file and line

where the variable was declared).

-d <none> ( --dynamic-type <none> )

Show the object as its full dynamic type, not its static type,

if available.

Values: no-dynamic-values | run-target | no-run-target

-f <format> ( --format <format> )

Specify a format to be used for display.

-g ( --show-globals )

Show the current frame source file global and static

variables.

-l ( --no-locals )

Omit local variables.

-r ( --regex )

The <variable-name> argument for name lookups are regular

expressions.

-s ( --scope )

Show variable scope (argument, local, global, static).

-t ( --no-recognized-args )

Omit recognized function arguments.

-y <name> ( --summary <name> )

Specify the summary that the variable output should use.

-z <name> ( --summary-string <name> )

Specify a summary string to use to format the variable output.

Children of aggregate variables can be specified such as 'var->child.x'. In 'frame variable', the operators -> and [] do not invoke operator overloads if they exist, but directly access the specified element. If you want to trigger operator overloads use the expression command to print the variable instead.

It is worth noting that except for overloaded operators, when printing local variables 'expr local\_var' and 'frame var local\_var' produce the same results. However, 'frame variable' is more efficient, since it uses debug information and memory reads directly, rather than parsing and evaluating an expression, which may even involve JITing and running code in the target program.

This command takes options and free-form arguments. If your

arguments resemble option specifiers (i.e., they start with a - or

--), you must use ' -- ' between the end of the command options and

the beginning of the arguments.

*'var' is an abbreviation for 'frame variable'*

**x** -- Read from the memory of the current target process.

Syntax: x <cmd-options> <address-expression> [<address-expression>]

Command Options Usage:

x [-r] [-f <format>] [-c <count>] [-G <gdb-format>] [-s <byte-size>] [-l <number-per-line>] [-o <filename>] <address-expression> [<address-expression>]

x [-br] [-f <format>] [-c <count>] [-s <byte-size>] [-o <filename>] <address-expression> [<address-expression>]

x [-AFLORTr] -t <name> [-f <format>] [-c <count>] [-G <gdb-format>] [-E <count>] [-o <filename>] [-d <none>] [-S <boolean>] [-D <count>] [-P <count>] [-Y[<count>]] [-V <boolean>] [-Z <count>] <address-expression> [<address-expression>]

x -t <name> [-x <source-language>] <address-expression> [<address-expression>]

--show-tags

Include memory tags in output (does not apply to binary

output).

-A ( --show-all-children )

Ignore the upper bound on the number of children to show.

-D <count> ( --depth <count> )

Set the max recurse depth when dumping aggregate types

(default is infinity).

-E <count> ( --offset <count> )

How many elements of the specified type to skip before

starting to display data.

-F ( --flat )

Display results in a flat format that uses expression paths

for each variable or member.

-G <gdb-format> ( --gdb-format <gdb-format> )

Specify a format using a GDB format specifier string.

-L ( --location )

Show variable location information.

-O ( --object-description )

Display using a language-specific description API, if

possible.

-P <count> ( --ptr-depth <count> )

The number of pointers to be traversed when dumping values

(default is zero).

-R ( --raw-output )

Dont use formatting options.

-S <boolean> ( --synthetic-type <boolean> )

Show the object obeying its synthetic provider, if available.

-T ( --show-types )

Show variable types when dumping values.

-V <boolean> ( --validate <boolean> )

Show results of type validators.

-Y[<count>] ( --no-summary-depth=[<count>] )

Set the depth at which omitting summary information stops

(default is 1).

-Z <count> ( --element-count <count> )

Treat the result of the expression as if its type is an array

of this many values.

-b ( --binary )

If true, memory will be saved as binary. If false, the memory

is saved save as an ASCII dump that uses the format, size,

count and number per line settings.

-c <count> ( --count <count> )

The number of total items to display.

-d <none> ( --dynamic-type <none> )

Show the object as its full dynamic type, not its static type,

if available.

Values: no-dynamic-values | run-target | no-run-target

-f <format> ( --format <format> )

Specify a format to be used for display.

-l <number-per-line> ( --num-per-line <number-per-line> )

The number of items per line to display.

-o <filename> ( --outfile <filename> )

Specify a path for capturing command output.

-r ( --force )

Necessary if reading over target.max-memory-read-size bytes.

-s <byte-size> ( --size <byte-size> )

The size in bytes to use when displaying with the selected

format.

-t <name> ( --type <name> )

The name of a type to view memory as.

-x <source-language> ( --language <source-language> )

The language of the type to view memory as.

--append-outfile

Append to the file specified with '--outfile <path>'.

This command takes options and free-form arguments. If your

arguments resemble option specifiers (i.e., they start with a - or

--), you must use ' -- ' between the end of the command options and

the beginning of the arguments.

*'x' is an abbreviation for 'memory read'*