# LLDB

# and Python Scripting

**Jake Carter**

# LLDB?

# GDB

```
(gdb) po window
<UIWindow: 0x7fa1014ab2f0;…
```

# LLDB

```
(lldb) po window
<UIWindow: 0x7fa1014ab2f0;…
```

# LLDB

**Shorthand**

```
po window
```

**Longhand**

```
expr [-O | --object-description] -- window
```

# LLDB Command Syntax

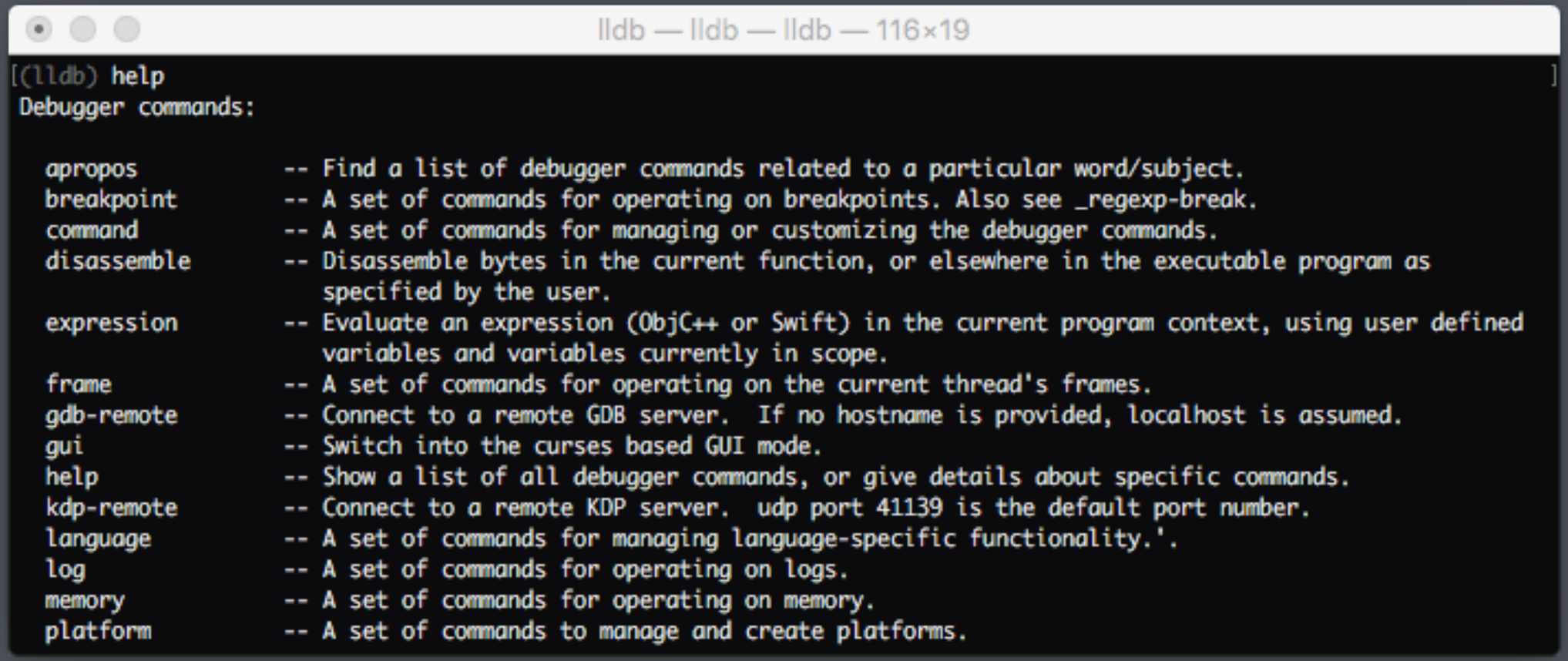
```
<command> [<subcommand>] <action> [-options [option-value]] -- [argument]
```

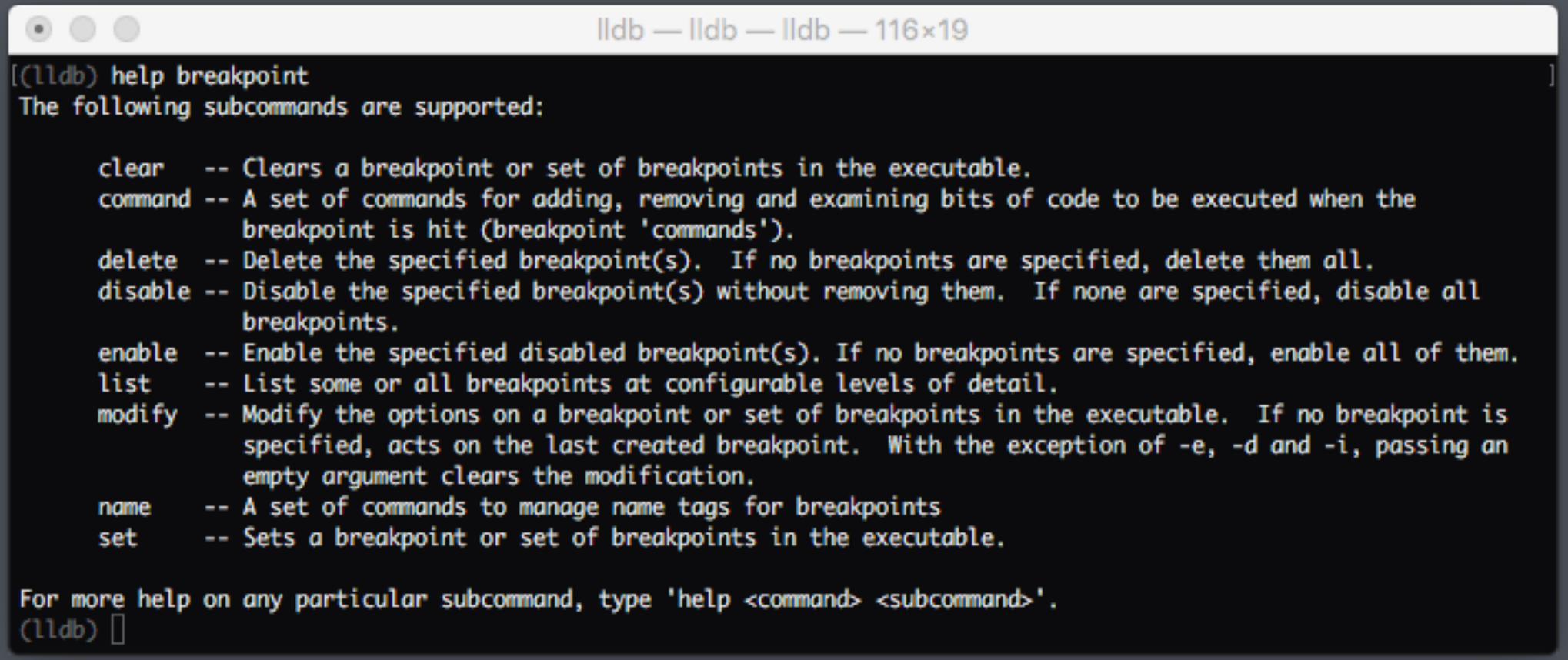
**Examples**

```
breakpoint [clear | set | list | …]

memory [find | history | read | …]

command [alias | history | script | …]
```


lldb — lldb — lldb — 116×19
(lldb) help
Debugger commands:
apropos -- Find a list of debugger commands related to a particular word/subject.
breakpoint -- A set of commands for operating on breakpoints. Also see _regexp-break.
command -- A set of commands for managing or customizing the debugger commands.
disassemble -- Disassemble bytes in the current function, or elsewhere in the executable program as specified by the user.
expression -- Evaluate an expression (ObjC++ or Swift) in the current program context, using user defined variables and variables currently in scope.
frame -- A set of commands for operating on the current thread's frames.
gdb-remote -- Connect to a remote GDB server. If no hostname is provided, localhost is assumed.
gui -- Switch into the curses based GUI mode.
help -- Show a list of all debugger commands, or give details about specific commands.
kdp-remote -- Connect to a remote KDP server. udp port 41139 is the default port number.
language -- A set of commands for managing language-specific functionality.'.
log -- A set of commands for operating on logs.
memory -- A set of commands for operating on memory.
platform -- A set of commands to manage and create platforms.


lldb — lldb — lldb — 116×19
(lldb) help breakpoint
The following subcommands are supported:
clear -- Clears a breakpoint or set of breakpoints in the executable.
command -- A set of commands for adding, removing and examining bits of code to be executed when the breakpoint is hit (breakpoint 'commands').
delete -- Delete the specified breakpoint(s). If no breakpoints are specified, delete them all.
disable -- Disable the specified breakpoint(s) without removing them. If none are specified, disable all breakpoints.
enable -- Enable the specified disabled breakpoint(s). If no breakpoints are specified, enable all of them.
list -- List some or all breakpoints at configurable levels of detail.
modify -- Modify the options on a breakpoint or set of breakpoints in the executable. If no breakpoint is specified, acts on the last created breakpoint. With the exception of -e, -d and -i, passing an empty argument clears the modification.
name -- A set of commands to manage name tags for breakpoints
set -- Sets a breakpoint or set of breakpoints in the executable.
For more help on any particular subcommand, type 'help <command> <subcommand>'.
(lldb)

# Setting a Breakpoint

```
(lldb)breakpoint set --file AppDelegate.m --line 23
```

# Breakpoint alias

**Usage**

```
(lldb)bfl AppDelegate.m 23
```

**Creation**

```
(lldb)command alias bfl breakpoint set --file %1 --line %2
```

~/.lldbinit

# ~/.lldbinit

```
command alias bfl breakpoint set --file %1 --line %2
```

# Class Hierarchy

```
(lldb)ch [NSString | <memory_address_to_an_NSString>]
NSString > NSObject
```

# LLDB Python Scripting

# LLDB Python API

- SB**Debugger**
- SB**CommandInterpreter**
- SB**CommandReturnObject**

# SB**Debugger**

- HandleCommand(*command*)
- **GetCommandInterpreter()**

# SB**CommandInterpreter**

- HandleCommand(*command*, *returnObject*)

# SB**CommandReturnObject**

- GetOutput()
- Succeeded()

# LLDB Python Script

```
#!/usr/bin/python

import lldb

def helloWorld(debugger, command, result, internal_dict):
    """Prints 'Hello, World!'"""
    print >>result, "Hello, World!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand("command script add -f hello.helloWorld hello")
    print 'The "hello" python command has been installed and is ready for use.'
```

# ~/.lldbinit

```
...

command script import /path/to/hello.py

...
```

# Class Hierarchy Python Command

```
(lldb)ch [NSString | <memory_address_to_an_NSString>]
NSString > NSObject
```

# LLDB

```
expr [-O | --object-description] -- window
```

# LLDB

```
expr [-O | --object-description] -- window

expr -l objc++ -O -- [self window]
```

# Demo

```
#!/usr/bin/python

import lldb

def ch(debugger, command, result, internal_dict):
    """Prints the class hirarchy for a given className or instance via memeory address. (Ex. 'ch NSString|<memory_address_to_an_NSString>' will output 'NSString > NSObject')"""

    ci = debugger.GetCommandInterpreter()
    ro = lldb.SBCommandReturnObject()

    command = command.strip()

    if len(command) < 1:
        print >>result, "Empty argument. Try 'ch NSString' or 'ch <memory_address_to_an_NSString>'"
        return

    ci.HandleCommand("expr -l objc++ -O -- [{} class]".format(command), ro)
    className = ro.GetOutput().strip()
    if len(className) == 0:
        print >>result, "Could not get class from {}".format(command)
        return

    acc = []

    while className != "nil":
        acc.append(className)
        ci.HandleCommand("expr -l objc++ -O -- [{} superclass]".format(className), ro)
        className = ro.GetOutput().strip()

    print >>result, " > ".join(acc)

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand("command script add -f ch.ch ch")
    print 'The "ch" python command has been installed and is ready for use.'
```

# Gotchas

- Create new SBCommandReturnObject for use with HandleCommand(), don't reuse `result` argument.
- Reuse new SBCommandReturnObject in HandleCommand() calls.
- SBCommandReturnObject.Succeeded() notifies that the command was executed without error.
- Don't forget both `command script import...` and `command script add...`

# References

- LLDB Homepage
  - LLDB Python API Reference
- LLDB Quick Start Guide
- lldb-scripts
- AverageJake.com
- @JakeCarter