# LLDB + Chisel

*debug your apps, but better this time*

**Peter Kos // 9/13/22**

Debugging + LLDB + Chisel

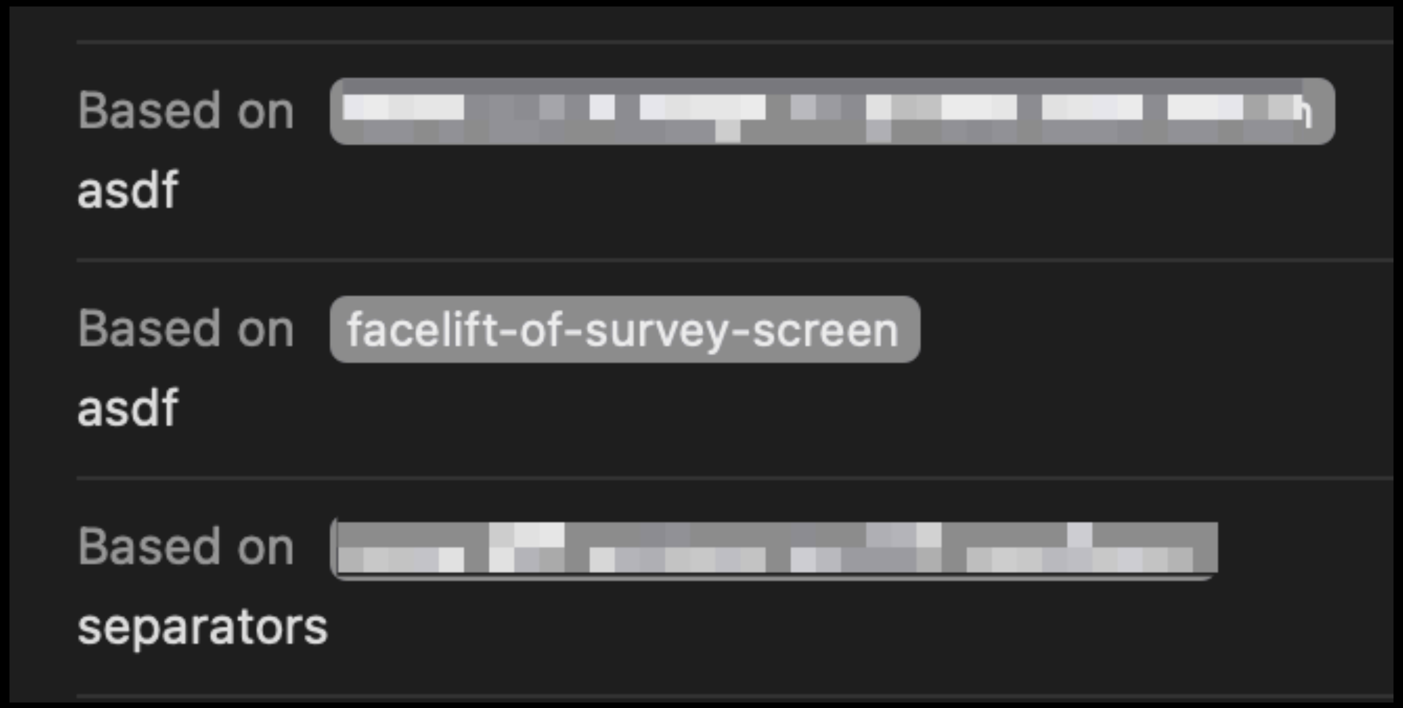
(again)

## Debugging

| | | |
|---|---|---|
| Uses Rust | Conditional Breakpoints | print(“On this method, X was hit”) |
| LAWFUL GOOD | NEUTRAL GOOD | CHAOTIC GOOD |
| print(“aaaa”) | Breakpoints | Pauses output repeatedly to catch the faulty code |
| LAWFUL NEUTRAL | TRUE NEUTRAL | CHAOTIC NEUTRAL |
| No debugging | fatalError(“a”) | #nofix all bugs in Jira |
| LAWFUL EVIL | NEUTRAL EVIL | CHAOTIC EVIL |

## *i'll have to admit…*

```
print("first")
print("here")
print("here 2")
```



Based on [redacted]
asdf

Based on facelift-of-survey-screen
asdf

Based on [redacted]
separators

my git stashes

```
po user
po user.name
po user?.name
po user!.name!

po model.user
po model!.user!
```

## *but, every tool has its use:*

*but, every tool has its use:*

## Widget lifecycle

```
    console.app
      sysdump
symb. breakpoints
```

## *but, every tool has its use:*

| Widget lifecycle | CoreData |
|---|---|
| console.app | sysdump |
| sysdump | a religion |
| symb. breakpoints | cond. breakpoints |

## *but, every tool has its use:*

### Widget lifecycle

console.app
sysdump
symb. breakpoints

### CoreData

sysdump
a religion
cond. breakpoints

### Day to day code

print debugging
commenting
breakpoints

## *but, every tool has its use:*

### Widget lifecycle

console.app
sysdump
symb. breakpoints

### CoreData

sysdump
a religion
cond. breakpoints

### Day to day code

print debugging
commenting
breakpoints

*you might not need these!*

## *but, every tool has its use:*

| Widget lifecycle | CoreData | Day to day code |
|---|---|---|
| console.app | sysdump | print debugging |
| sysdump | a religion | commenting |
| symb. breakpoints | cond. breakpoints | breakpoints |

instruments.app

memory graph

console.app

commenting

breakpoints

sysdump

visual debugger

a religion

print debugging

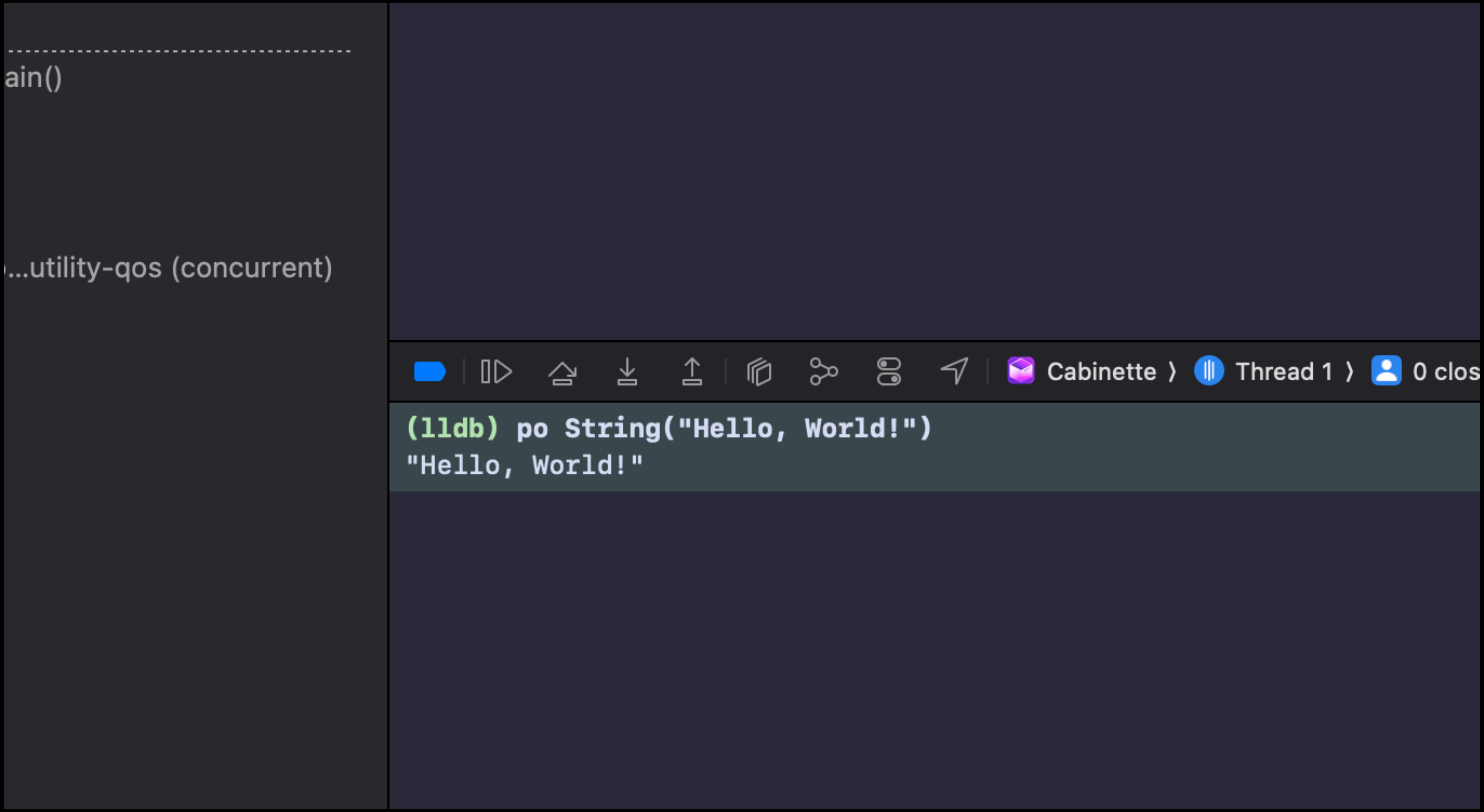
symb. breakpoints

cond. breakpoints

atos

dSYMs

## LLDB



```
(lldb) po String("Hello, World!")
"Hello, World!"
```

## *the* most *most-funnest terminal:*

```
(lldb)
```

## *the* most *most-funnest terminal:*

```
(lldb)
(lldb) po user?.name
(lldb) e -l Swift -- callFunc()
```

## *the* quirky *c-like truck:*

```
(lldb) expr -l objc++ -O -- [[UIWindow key
(lldb) e -l Swift -- unsafeBitCast(0x7fc72
```

## *the* quirky *c-like truck:*

```
(lldb) expr -l objc++ -O -- [[UIWindow keyWindow] _autolayoutTrace]
```

```
(lldb) e -l Swift -- unsafeBitCast(0x7fc72c8bc980, to: UITextView.self)
                     .backgroundColor = UIColor.blue
```

## *the* quirky *c-like truck:*

print the view hierarchy from auto layout's pov

```
(lldb) expr -l objc++ -O -- [[UIWindow keyWindow] _autolayoutTrace]
```

change a TextView's background color to blue

```
(lldb) e -l Swift -- unsafeBitCast(0x7fc72c8bc980, to: UITextView.self)
.backgroundColor = UIColor.blue
```

## *the* quirky *c-like truck:*

print the view hierarchy from auto layout's pov

```
(lldb) expr -l objc++ -O -- [[UIWindow keyWindow] _autolayoutTrace]
```

change a TextView's background color to blue

```
(lldb) e -l Swift -- unsafeBitCast(0x7fc72c8bc980, to: UITextView.self)
                     .backgroundColor = UIColor.blue
```

## *the* quirky *c-like truck:*

print the view hierarchy from auto layout's pov

```
(lldb) expr -l objc++ -O -- [[UIWindow keyWindow] _autolayoutTrace]
```

change a TextView's background color to blue

```
(lldb) e -l Swift -- unsafeBitCast(0x7fc72c8bc980, to: UITextView.self)
                     .backgroundColor = UIColor.blue
```

## *actually useful things:*

assign memory addresses to vars

```
(lldb) e -l Swift -- let $pinAddr = 0x7df67c50
```

recast these to views

```
(lldb) e -l Swift -- let $pin = unsafeBitCast($pinAddr, to: MKPinAnnotationView.self)
```

## *actually useful things:*

po

p

v

Chisel

*What if all this was ✨easier?*

## *What if all this was ✨easier?*

Python Scripting

LLDB has been structured from the beginning to be scriptable in two ways – a Un
using LLDB; and within the LLDB debugger tool, Python scripts can be used to he

lldb.llvm.org/use/python.html

*What if all this was ✨easier?*

*What if all this was ✨easier?*

*Chisel gives us a bunch of scripts!*

## *Chisel gives us a bunch of scripts!*

| | | | |
|---|---|---|---|
| Print recursive VC description | `pvc` | `fvc` | Find VC name w/ regex |
| Generate screenshot of a view | `visualize` | `dismiss` | Dismiss a VC |
| Show/hide a view | `show/hide` | `alamborder` | Border ambiguous position views |
| Border/unborder a view | `border/unborder` | `pcurl` | Print NSURLSession as curl |

## *Chisel gives us a bunch of scripts!*

| Description | Command | Command | Description |
|---|---|---|---|
| Print recursive VC description | `pvc` | `fvc` | Find VC name w/ regex |
| Generate screenshot of a view | `visualize` | `dismiss` | Dismiss a VC |
| Show/hide a view | `show/hide` | `alamborder` | Border ambiguous position views |
| Border/unborder a view | `border/unborder` | `pcurl` | Print NSURLSession as curl |

All happens **without resuming!**

*most of these have arguments, too:*

*most of these have arguments, too:*

```
alamborder
    --color/-c <color>
        A color name such as 'red', ‘blue'
    --width/-w <width>
        Desired width of border.
```

*most of these have arguments, too:*

```
alamborder
    --color/-c <color>
        A color name such as 'red', ‘blue'
    --width/-w <width>
        Desired width of border.
```

```
⇝ alamborder -c “red” -w 2.0
```

## *whatsit work like?*

<do live demo>

## ~~*whatsit work like?*~~

## *some extra help*

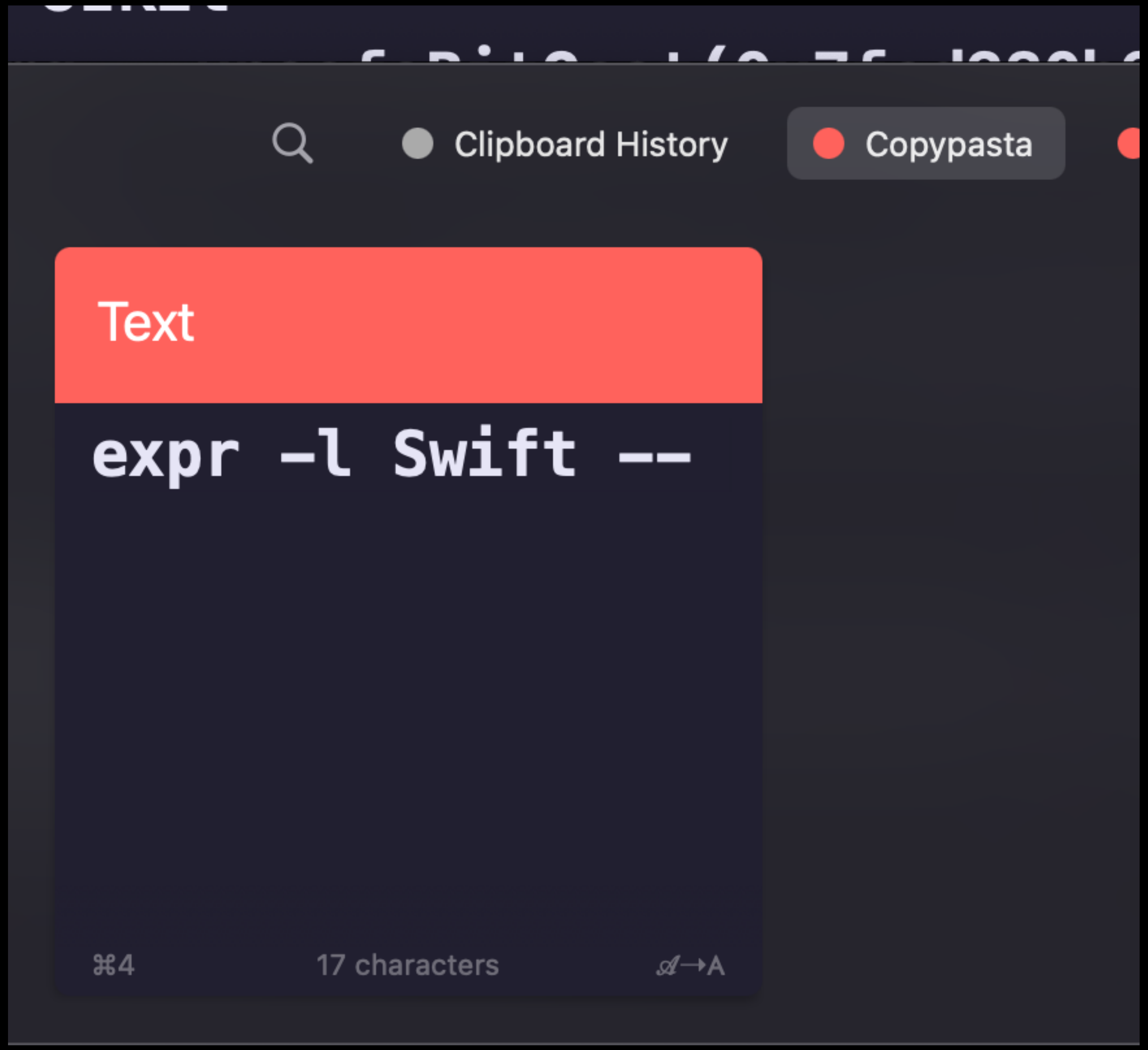
A better way to copy and paste

Paste stores everything you copy on your Mac, iPhone, and iPad, so it's always there whenever you need it.

pasteapp.io

## ~~whatsit work like?~~

## some extra help



```
expr -l Swift --
```

If you are also tired of typing

github.com/facebook/chisel

Debugging + LLDB + Chisel

(again)

# LLDB + Chisel

*debug your apps, but better this time*

**Peter Kos // 9/13/22**