Using Swift to build CLI Utilities and *Scripts* *

---

* *Narrator Voice: they weren't real scripts*

About Me [1]

- Diego Freniche
- **iOS dev**, sometimes Appc + CoffeeScript + Android + Kotlin... 😱
- working At The Best Company. Period: **Teamwork**

---

[1] best middle-aged man moment of the year

😱 I want to quit and open a bar near the beach, burn all computers or get run over by a bus

Why CLI tools?

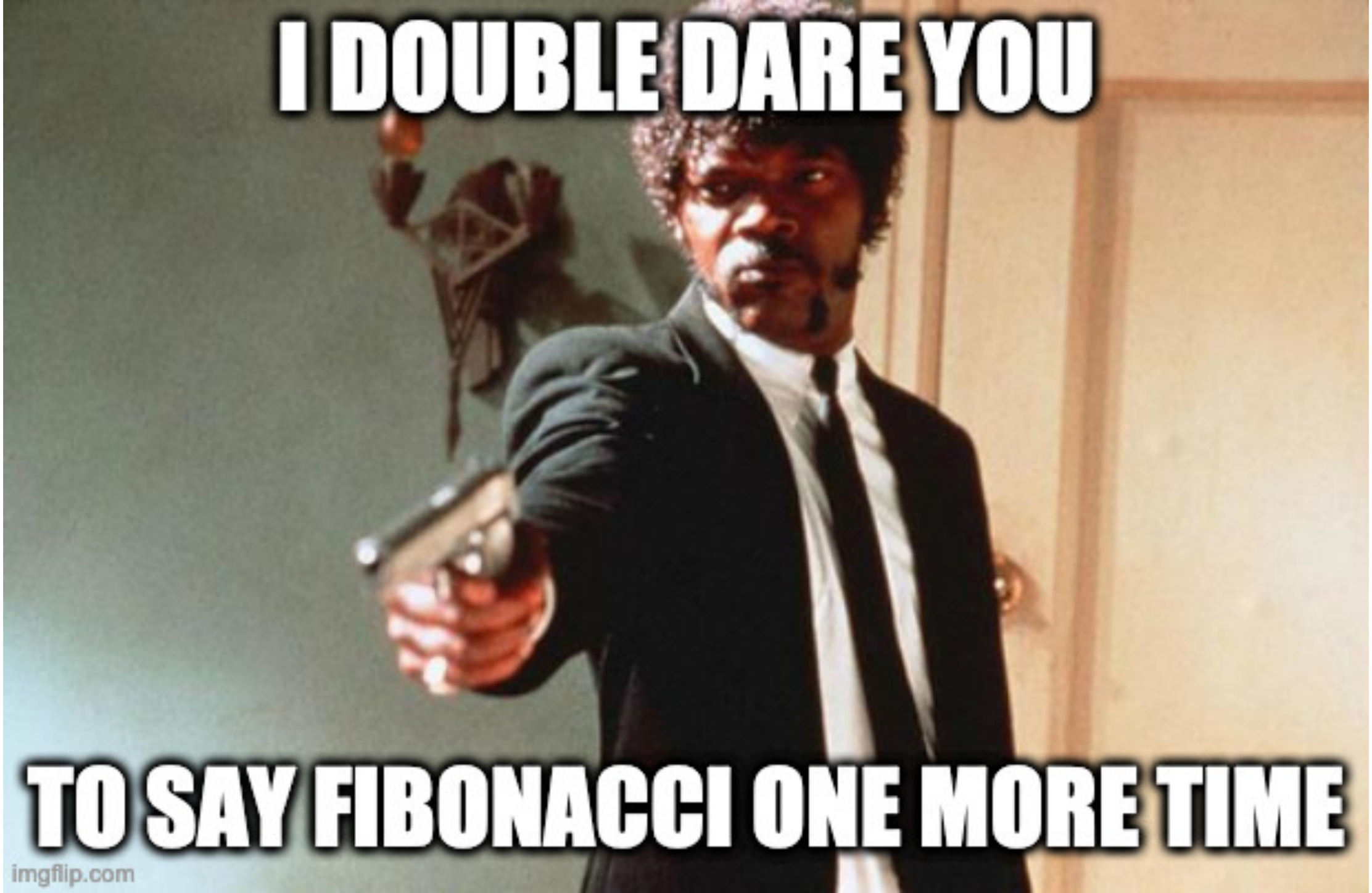
- small scripts to scratch an itch
- to automate things
- because they look cool

Swift as a "scripting" language 🤔

	Scripting	**Non Scripting**	**Swift**
Interpreted	✅		
Compiled		✅	✅
Dynamically typed	✅		
Statically Typed		✅	✅

Running Swift "Scripts" from CLI

```
// fibonacci.swift
// from: https://jblevins.org/log/swift

#!/usr/bin/swift

func fibonacci(_ n: Int) -> Int {
    if n <= 2 {
        return 1
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2)
    }
}

print(fibonacci(10))
```

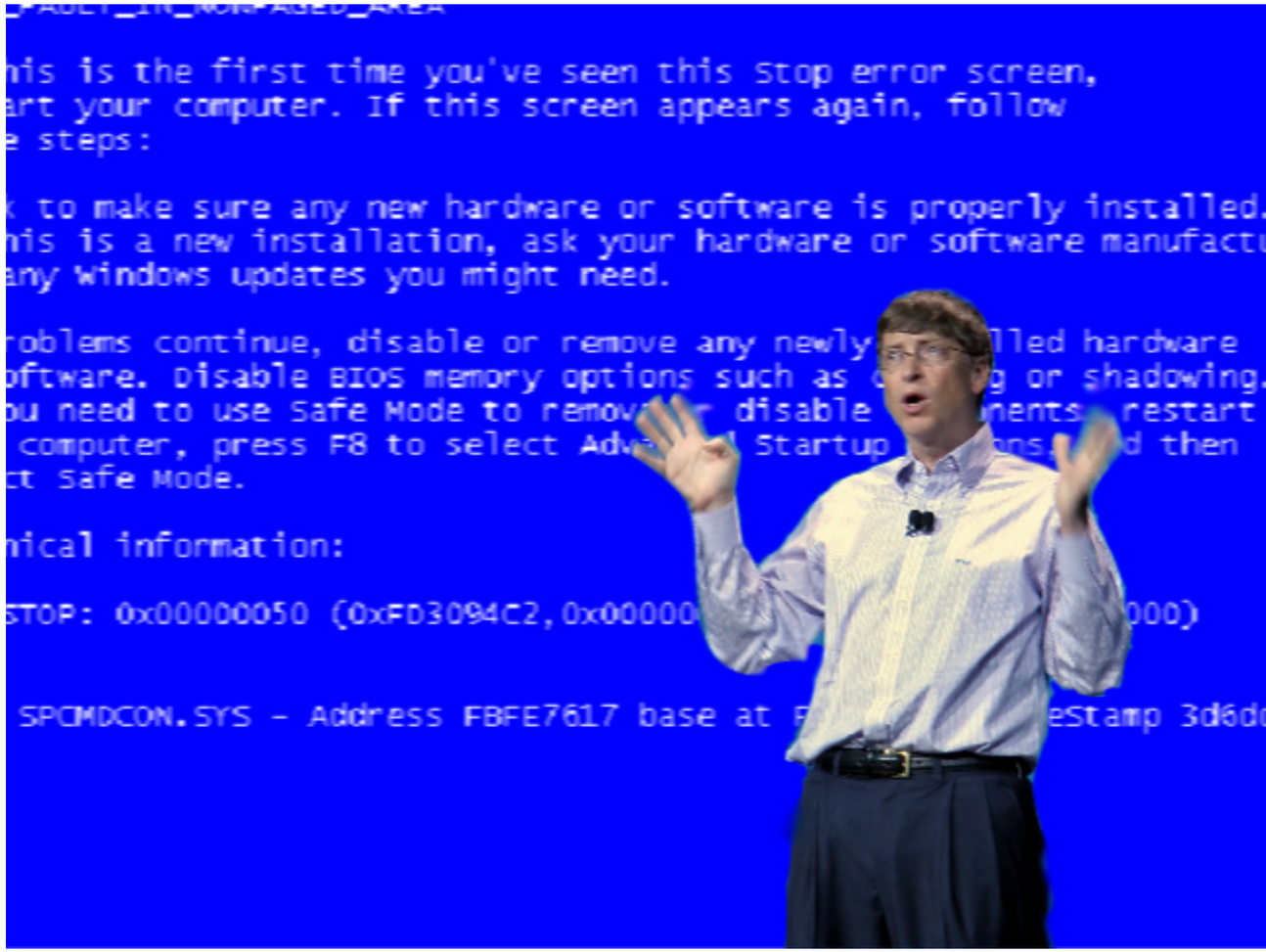

I DOUBLE DARE YOU
TO SAY FIBONACCI ONE MORE TIME
imgflip.com

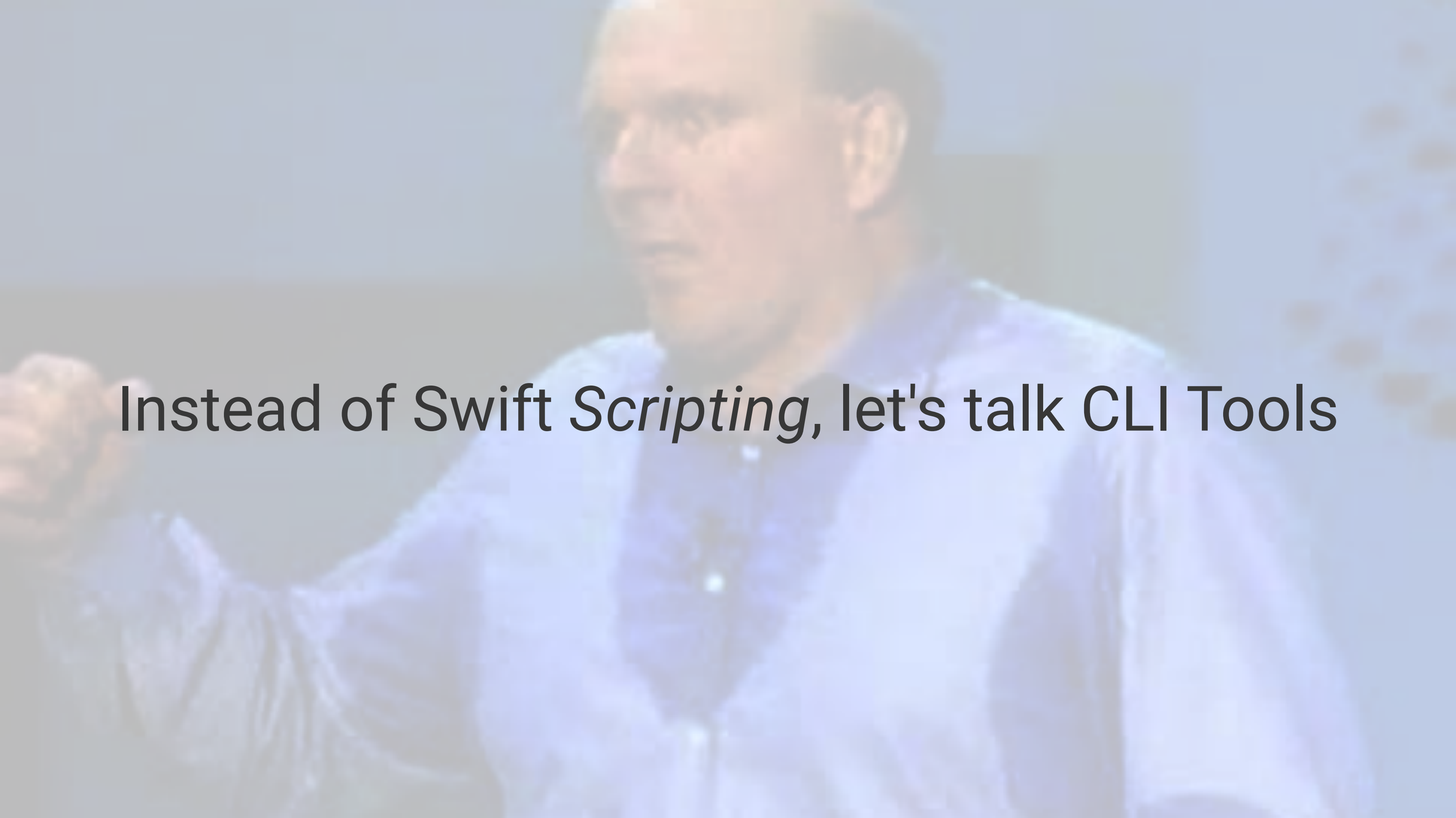
Xcode 12 Beta == :sad face:

```
$ swift -v -sdk macosx10.16  fibonacci.swift
Apple Swift version 5.3 (swiftlang-1200.0.16.9 clang-1200.0.22.5)
Target: x86_64-apple-darwin19.5.0
<unknown>:0: warning: no such SDK: 'macosx10.16'
/Applications/Xcode-beta.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/swift
-frontend -interpret fibonacci.swift -enable-objc-interop -stack-check -sdk macosx10.16 -color-
diagnostics -module-name fibonacci
<unknown>:0: warning: no such sysroot directory: 'macosx10.16'
<unknown>:0: error: unable to load standard library for target 'x86_64-apple-macosx10.15'
```

⚠️ Limitations of Swift Scripts

- only one file per script
  - no `include` other .swift file
- no easy way to add 3rd party code (well, you can create a Makefile, then pass-in options to the linker, etc.)
- compiled anyway





Instead of Swift *Scripting*, let's talk CLI Tools

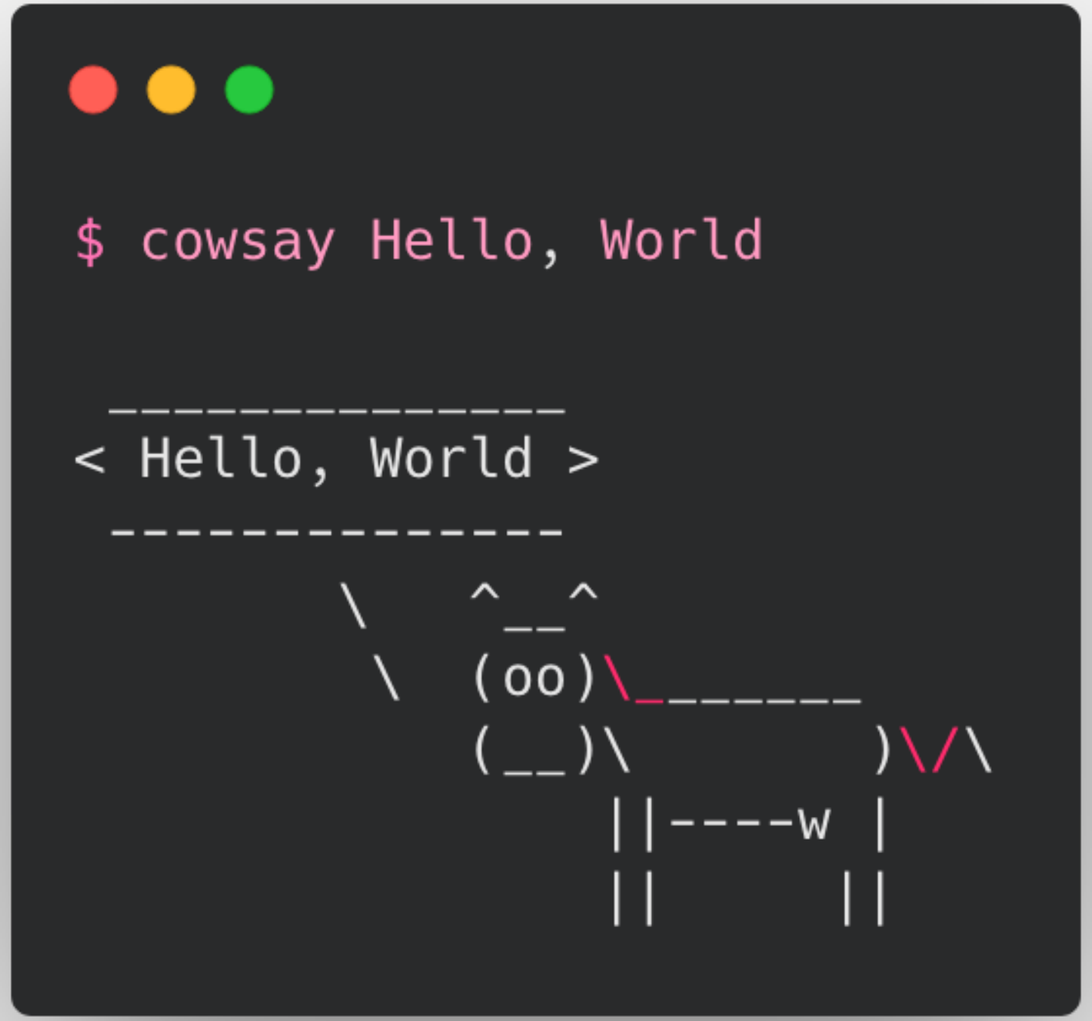
CLI TOOLS

Command line tools 🛠️ are software. Structuring their code in massive files that read like bash scripts is missing an opportunity to leverage the abstractions of the programming language and platform to have a codebase that is easier to reason about. [T1]

[T1] https://twitter.com/pedropbuendia/status/1230414793191890950?s=20

Our example

- based on Cowsay
- `brew install cowsay`
- for extra dramatism try `cowsay Hello, World && say "Hello, World"`

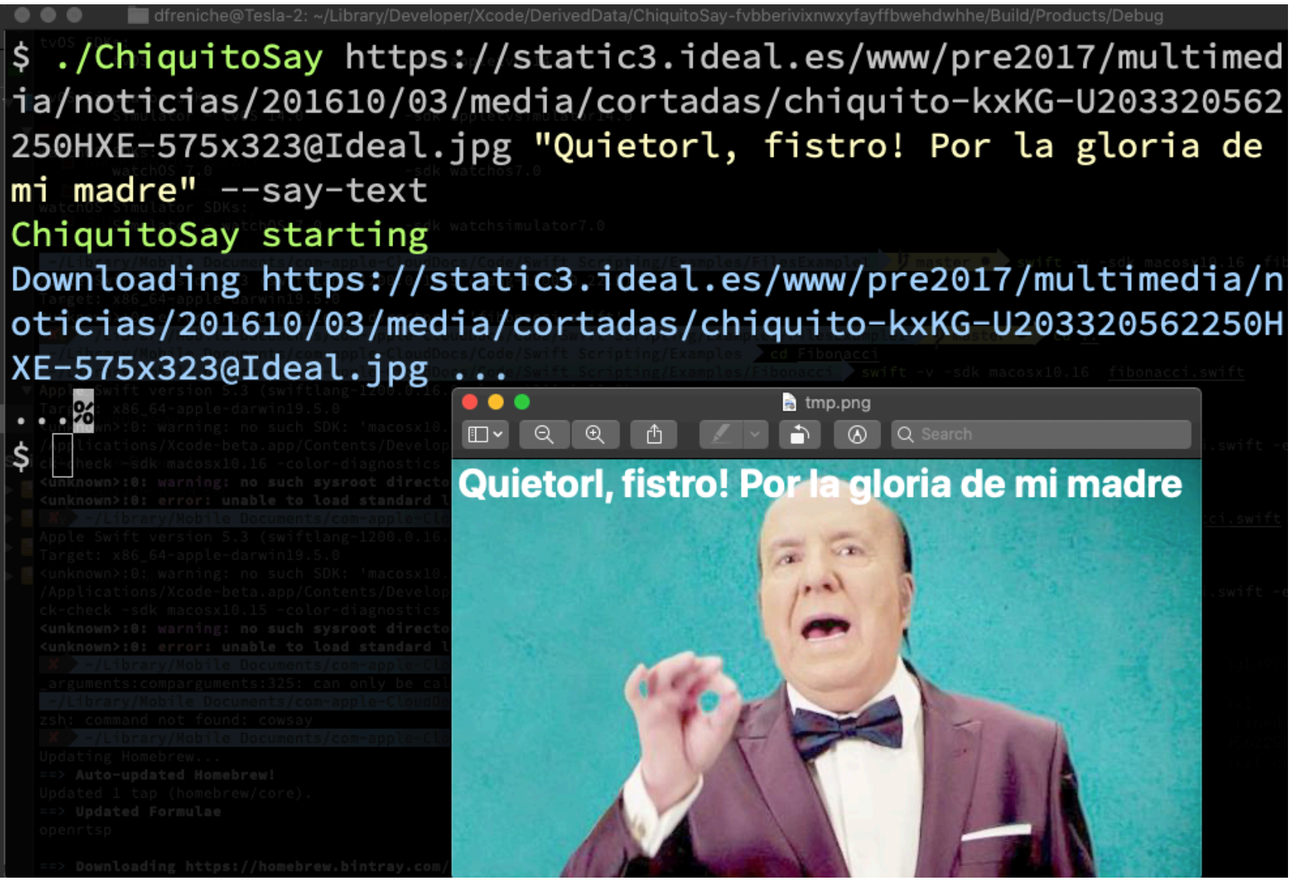


ChiquitoSay

- Based on Chiquito de la Calzada images [Don Gregorio]
- Will show an image from an URL we pass in
- Should "print" text on that image
- Should also "say" the text we're passing in

Don Gregorio https://es.wikipedia.org/wiki/Chiquito*de*la_Calzada

ChiquitoSay in action



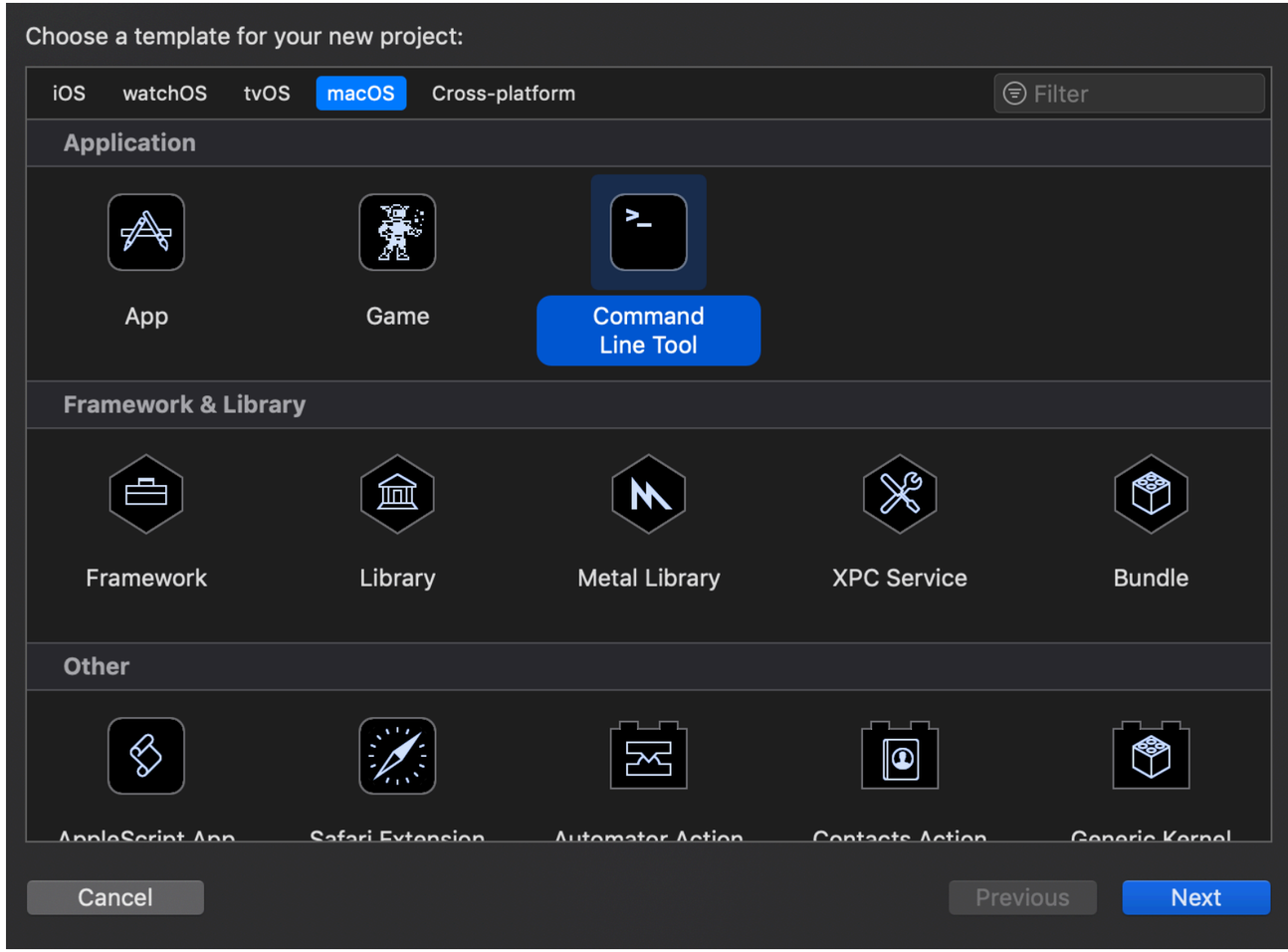
- Modern Swift 5.3, Xcode 12, CLI macOS app
- Argument parsing using SwiftArgumentParser Apple Package
- Downloads an image in a background thread, using semaphores
- Prints some text over that image using Core Image / AppKit
- Saves that image to a temp file
- Shows that temp file invoking open command

Options for creating CLI Tools

- 🛠️ start working in Xcode
  - easy and quick start
  - bit less complicated later on
- 💻 start working from Terminal
  - bit more difficult at the beginning (you have to type)
  - more powerful

Starting in Xcode

- New Project
- Mac > Command Line Tool
- 🙈 No tests!
  - if you want tests (you should) create instead a Framework and use that Framework from the CLI app.



Command-line tool project

- put code in `main.swift`
- hit Cmd + R
- watch for output in Xcode's Terminal

Program Starting point: *Script Style*

```
// main.swift: put code here
// just put code in global scope

print("Tea duck queen?")
```

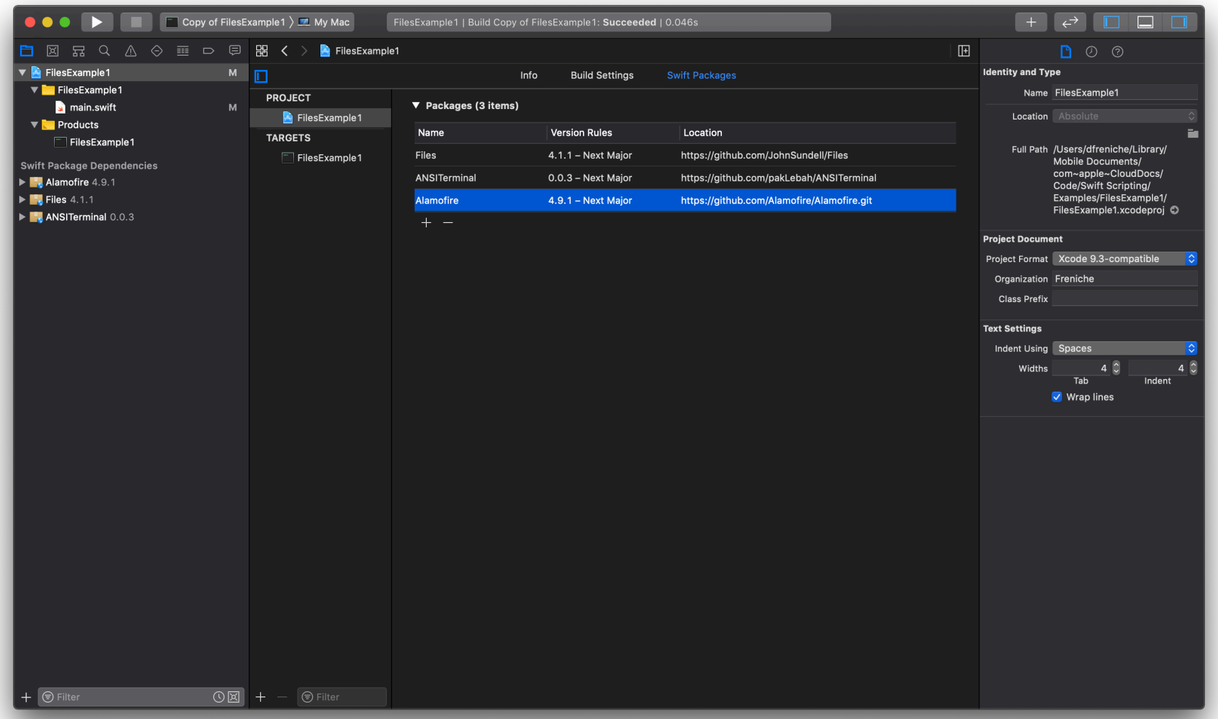
Program Starting point: Old way

```
// This file has to be called
// main.swift

struct OldApp {
    func run() {
        print("Tea duck queen?")
    }
}

let app = OldApp()
app.run()
```

Program Starting point: New Way

```swift
// We can call this whatever we want
// ChiquitoSay.swift

import Foundation

@main
struct MainChiquitoSay {

    static func main() {
        print("Tea duck queen?")
    }
}
```

3rd party dependencies: SPM



Go to `Project > Swift Packages`

1. Add new packages
2. See list of packages
3. Change versions, etc.

`File > Swift Packages`: bunch of interesting options

Where are my dependencies?

Inside a hidden `Package.resolved` file

```
$ cd ChiquitoSay.xcodeproj/project.xcworkspace/xcshareddata/swiftpm
$ ls Package.resolved
```

Package.resolved

```
{
  "object": {
    "pins": [
      {
        "package": "Alamofire",
        "repositoryURL": "https://github.com/Alamofire/Alamofire.git",
        "state": {
          "branch": null,
          "revision": "747c8db8d57b68d5e35275f10c92d55f982adbd4",
          "version": "4.9.1"
        }
      },
      ...
      // more packages
    ]
  },
  "version": 1
}
```

Our packages

- ANSITerminal: https://github.com/pakLebah/ANSITerminal
- Files: https://github.com/johnsundell/files
- swift-argument-parser: https://github.com/apple/swift-argument-parser
- SwiftFigletKit: https://github.com/dfreniche/SwiftFiglet

Main pieces of the CLI puzzle 🧩

- 🖥️ printing/reading to/from console
- 🔍 argument parsing
- 🎨 nice ANSI colors if possible
- 🗃️ reading, creating, deleting, moving files and folders
- 🚀 launching other processes
- ⚡ accessing the network

🖥️ printing to console

- just use good old `print`
- printing Optionals is ugly

```swift
let myOptional: String? = "Hello"
print("\(String(describing: myOptional))")
// Optional("Hello")
```

Printing Optionals is ugly: solution

```
prefix operator ^

extension Optional where Wrapped == String? {

    public static prefix func ^ (rhs: Wrapped?) -> String {

        if let unWrapped = rhs, let data = unWrapped {
            return data
        }
        return ""
    }
}

extension String {

    public static prefix func ^ (rhs: String) -> String {

        return rhs
    }
}
```

🖥 printing and reading from console

But that terminal...

- ~~just read from command line~~ fixed in Xcode 12 Beta!
- or try to print something with ANSI colors

🖥️ Running the app in a real Terminal 1/2

- we'll duplicate and modify a Scheme
- select binary in `Provide build settings from`
- add a post-build run script phase:

```
#!/bin/bash

open $BUILT_PRODUCTS_DIR/$PRODUCT_NAME
```

🖥️ Running the app in a real Terminal 2/2

- open binary in its folder
- run from command line

🖥️ Reading from console

- `readLine()`, `readLine(strippingNewline: true)`
- returns String?, so better do `readLine() && ""`
- use `print(message, separator: "", terminator: "")` if you don't want CR + LF before your message
- `readLine` does not work in Playgrounds 🙈

https://developer.apple.com/documentation/swift/1641199-readline

🎨 Nice ANSI colors if possible

- there are several nice libraries out there...
- SPM: https://github.com/pakLebah/ANSITerminal
- using Swift Package Manager
- `print("ChiquitoSay starting".green)`

🔍 Argument parsing

- if you really want to dive deep into Argument Parsing check out this 47 Degrees Academy talk [47]
- add package `https://github.com/apple/swift-argument-parser`
- add `ParsableCommand` to our starting point

[47] https://www.47deg.com/academy/2020-06-18-command-line-utilities-with-swift-talk/

🤔 Argument parsing

We want these args:

```
ARGUMENTS:
  <image-url>                 URL with the image to show
  <message>                   The message to print

OPTIONS:
  --print-banner              Print a text banner instead of image
  --say-text                  Say text aloud
  -h, --help                  Show help information.
```

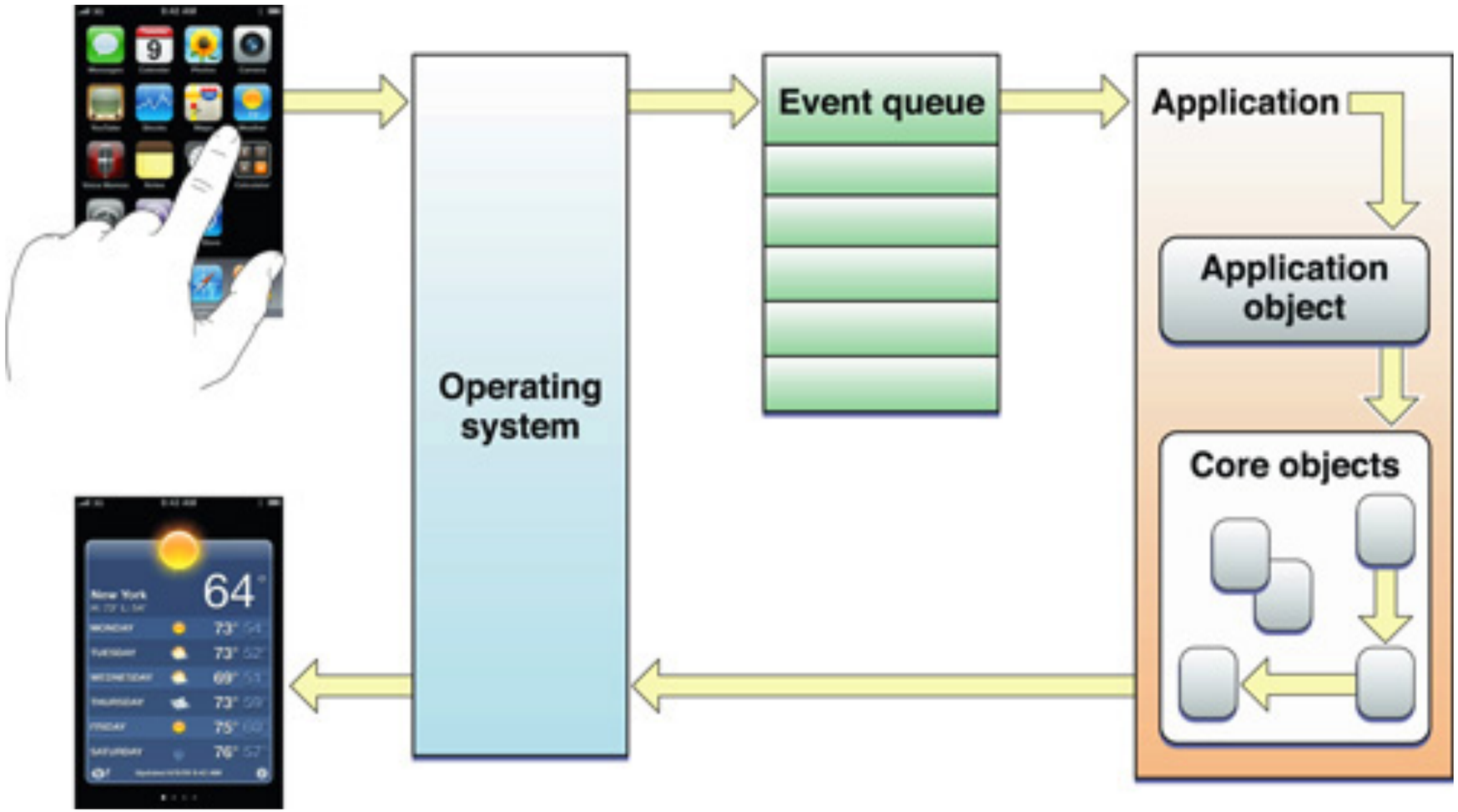
🤔 Argument parsing

```
@main
struct MainChiquitoSay: ParsableCommand {
    @Flag(help: "Print a text banner instead of image")
    var printBanner = false

    @Flag(help: "Say text aloud")
    var sayText = false

    @Argument(help: "URL with the image to show")
    var imageURL: String

    @Argument(help: "The message to print")
    var message: String
```

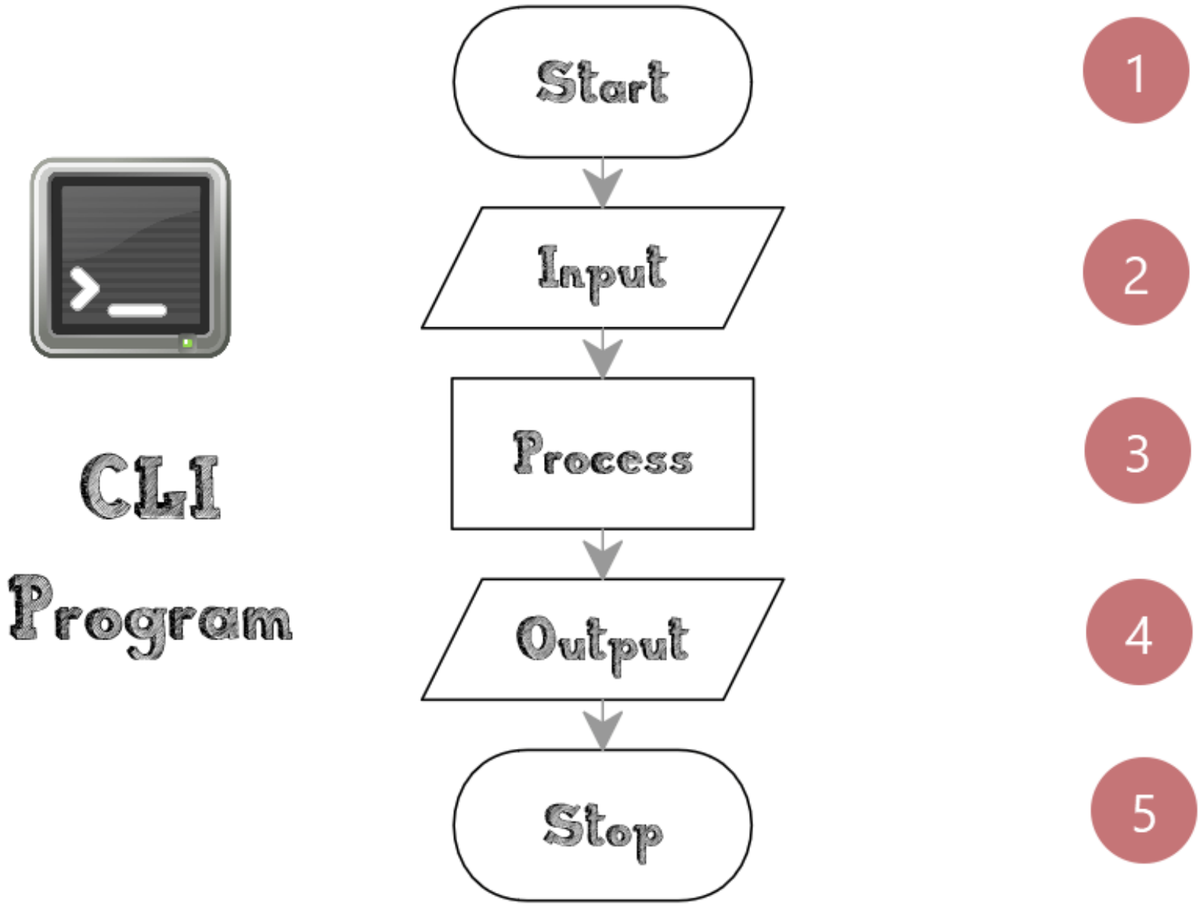
🚀 launching other processes

```
extension Process {
    ///   - command: full path to the program we want to launch
    ///   - arguments: Array of arguments passed to that program
    static func launch(command: String, arguments:[String] = []) {
        let url = URL(fileURLWithPath: command)
        do {
           try Process.run(url, arguments: arguments) { (process) in
              print("\ndidFinish: \(!process.isRunning)")
           }
        } catch {
            print("Error opening file \(error)")
        }
    }
}
```

⚡ network access



⚡ network access



⚡ network access

- we can't use GCD here *easily*
- we'll launch a background process and it will be killed: our app will continue and finish
- we need a way to wait
- we'll use semaphores because they are scary
- more info on GCD + CLI here: https://stackoverflow.com/questions/8366195/using-grand-central-dispatch-outside-of-an-application-or-runloop

⚡ network access

```
func alAtaqueerl() {
    print("Downloading \( imageURL ) ...".blue)
    var end = false

    let sema = DispatchSemaphore( value: 0)

    let task = URLSession.shared.dataTask(with: imageURL) { (data, response, error) in
        do {
            if let data = data, let originalImage = NSImage(data: data) {
                let imageWithText = originalImage.print(text: message)
                let imageWithTextFile = try Folder.temporary.createFile(at: "tmp.png", contents: imageWithText.png)

                Process.launch(command: "/usr/bin/open", arguments: ["\(imageWithTextFile.path)"])
            }
        } catch {
            print("Error creating file")
        }

        sema.signal() // signals the process to continue

        end = true
    };

    task.resume()

    while (!end) {
        sema.wait()
        print("...", separator: "", terminator: "")
        sema.signal()
    }
}
```

⚡ network access

```
func alAtaqueerl() {
    var end = false

    let sema = DispatchSemaphore( value: 0) // semaphore starts at 0

    let task = URLSession.shared.dataTask(with: imageURL) { (data, response, error) in
        do {
            // do something with that data when download finished
        } catch {
            // error!
        }

        sema.signal() // sets semaphone - 1

        end = true
    };

    task.resume() // we launch our background task AND CONTINUE executing down there 👇

    while (!end) {
        sema.wait() // this increments the semaphore + 1
        print("...", separator: "", terminator: "")
        sema.signal() // decrements - 1
    }
}
```

🗂 reading, creating, deleting, moving files and folders

SPM: https://github.com/JohnSundell/Files

```
extension Folder {

    public static func cd(path: String) -> Folder {

        do {

            let parent = try Folder.init(path: path)
            return parent
        } catch {

            return Folder.current
        }
    }
}
```

Final touches

- printing text on the image
- calling say
- printing that banner

Thanks!

Extra: SPM from console

```
swift package generate-xcodeproj --help
```

- We can get help on all `package` options:
  `swift package init --help`
- This creates the Package:
  `swift package init --type executable  --name mycoolapp`
- But if we also want a nice Project to use it with Xcode
  `swift package generate-xcodeproj`
- Run tests from command line

Images:

https://www.oldbookillustrations.com/wp-content/high-res/1830/like-rash-buss-768.jpg