The Command Line Toolbox 🧰

A Crash Course on building your own CLI tools

Michael Bates

Kentucky 🏇🥃
Swing Dance 💃🕺
Learn Swift Louisville Organizer

Day job: Python & JS 😕
CLI tools: Swift 🥰

Goals

Learn Tips & Tricks

Discover Open Source Packages

Build your own tools!

Command Lines Shells

Necessary 👍

Super-powerful 💪

Not great languages 👎

🏃‍♀️ Running Scripts

🔨 Building Interfaces

💬 Effective I/O

Running Scripts

🏃‍♀️ Running Scripts

1. Single executable file
2. Swift Package Manager

Single Executable File

```
//FoodPls.swift

print("🍔🍕🌭🥪🥗🍳🥧🌮🌯🍗🍜🍲🍣🍱🥡🍛🍺".randomElement()!)
```

Single Executable File

```
//FoodPls.swift

print("🍔🍕🌭🥪🥗🍳🥧🌮🌯🍗🍜🍲🍣🍱🥡🍛🍺".randomElement()!)

> swift FoodPls.swift
🥡
```

Single Executable File

```
//FoodPls.swift

print("🍔🍕🌭🥪🥗🍳🥧🌮🌯🍗🍜🍲🍣🍱🥡🍛🍺".randomElement()!)

> swift FoodPls.swift
🥡

> foodpls
🌭
```

Getting Fancy

1. `chmod +x FoodPls.swift`

Getting Fancy

1. `chmod +x FoodPls.swift`
2. `#!/usr/bin/xcrun swift`

Getting Fancy

1. `chmod +x FoodPls.swift`
2. `#!/usr/bin/xcrun swift`
3. `cp FoodPls.swift /usr/local/bin/foodpls`

Getting Fancy

1. `chmod +x FoodPls.swift`
2. `#!/usr/bin/xcrun swift`
3. `cp FoodPls.swift /usr/local/bin/foodpls`

```
> foodpls
🍕
```

Swift Package Manger

Executable target

Swift Package Manger

Executable target — from scratch

```
> xcrun swift package init --type=executable
```

Swift Package Manger

Executable target — existing package

```
targets: [
    .target(name: "foodpls", dependencies: []),
    // ...
]

// Sources/foodpls/main.swift
```

Swift Package Manager

"Installing" the binary

```
> swift build
> cp .build/debug/foodpls /usr/local/bin/
```

Marathon 

by John Sundell

github.com/JohnSundell/Marathon

Marathon 🏃

✔ Write, run, and install single-file scripts

✔ Install dependencies

Marathon 🏃

```
> marathon run FoodPls.swift
```

Marathon 🏃

```
> marathon edit FoodPls.swift
🏃   Updating packages...
✏️   Opening foodpls.xcodeproj/
```

Marathon 🏃

```
> marathon install FoodPls.swift
🏃   Compiling script...
     Installing binary...
💻   FoodPls.swift installed at /usr/local/bin/foodpls
```

Marathon 🏃

```
import Files // marathon:https://github.com/JohnSundell/Files.git
```

Marathon 🏃

```
import Files // marathon:https://github.com/JohnSundell/Files.git
```

```
> marathon add https://github.com/JohnSundell/Files.git
```

Marathon 🏃

```
> marathon install https://example.com/script.swift
```

GitHub Gists: 👍

Mint 🌱

by Yonas Kolb

github.com/yonaskolb/mint

Mint 🌱

Install executables from any Swift Package

— Carthage
— SwiftLint
— Sourcery
— Your own tools!

Mint 🌱

```
> mint install realm/SwiftLint

🌱  Finding latest version of SwiftLint
🌱  Resolved latest version of SwiftLint to 0.28.0
🌱  Cloning https://github.com/realm/SwiftLint.git 0.28.0...
🌱  Building SwiftLint Package with SPM...
🌱  Installing SwiftLint...
🌱  Installed SwiftLint 0.28.0
🌱  Linked swiftlint 0.28.0 to /usr/local/bin.

> swiftlint version
0.28.0
```

Mint 🌱

```
> mint run realm/SwiftLint@0.22.0 swiftlint
```

Mint 🌱

Mintfile

```
Carthage/Carthage
realm/SwiftLint
krzysztofzablocki/Sourcery@0.15.0
...
```

Better than a readme 👍

Gotchas — Mint 🌱 and Marathon 🏃

Building from source

Check your version

`Swiftenv` considered harmful:
`Package.swift not found`

Building Interfaces

🔨 Building Interfaces

Complex Behavior + Good UI = 🏆

🔨 Building Interfaces

Complex Behavior + Good UI = 🏆

GUI 👉 Animations
CLI 👉 Sub-commands

🔨 Building Interfaces

```
> todos add "buy milk"
```

🔨 Building Interfaces

```
> todos add "buy milk"
```

1. DIY sub-commands
2. Packages

DIY Sub-Commands

```
let args = CommandLine.arguments.dropfirst()
```

DIY Sub-Commands

```
let args = CommandLine.arguments.dropfirst()
```

`dropFirst`: removes program name

DIY Sub-Commands

```
let args = CommandLine.arguments.dropfirst()
let cmd = args[0]
```

DIY Sub-Commands

```
let args = CommandLine.arguments.dropfirst()

enum Command: String {
  case add // rm, list, etc.
}

let cmd = args.first.map(Command(rawValue:))
```

DIY Sub-Commands

```
switch cmd {
case nil:
  fatalError("Unrecognized command")
case .add:
  addTodo(title: CommandLine.arguments[1])
}
```

DIY Sub-Commands

```
switch cmd {
case nil:
  fatalError("Unrecognized command")
case .add:
  addTodo(title: CommandLine.arguments[1])
}

> todos add "buy milk"
```

DIY Sub-Commands

Good for simple scripts 👍

Doesn't scale well 👎

Manual type conversion 👎

DIY Sub-Commands

```
> todos help
Available commands:
   add    Create a new task
   do     Complete tasks by ID
   edit   Change the title of a task
   ls     List outstanding tasks
   rm     Remove tasks
   undo   Un-complete tasks by ID
```

See mklbtz/finch for a full implementation of this!

Commander

by Kyle Fuller

github.com/kylef/Commander

Commander

```
import Commander

Group {
  $0.command("add") { (title: String) in
    addTask(title: title)
  }
}.run()
```

Commander

```
import Commander

Group {
  $0.command("add") { (title: String) in
    addTask(title: title)
  }
}.run()

> todos add "buy milk"
```

Commander

Type inference 👌

Good for medium complexity 👍

Automatic Numeric & Array conversion 👍

Commandant

by Carthage
github.com/Carthage/Commandant

Commandant

```
import Commandant

let commands = CommandRegistry<String>()

commands.register(AddCommand(manager: try TaskManager()))

commands.register(HelpCommand(registry: commands))

commands.main(defaultVerb: "help") { error in
  print(error)
}
```

Commandant

```
struct AddCommand: CommandProtocol {
  let verb = "add"
  let function = "Create a new task"

  func run(_ options: Options) -> Result<Void, String> {
    // ...
  }
}
```

Commandant

```
extension AddCommand {
  struct Options: OptionsProtocol {
    let title: String

    static func evaluate(_ m: CommandMode)
      -> Result<Options, CommandantError<String>> {
      return Options.init
        <*> m <|* Argument<String>(usage: "Title for task")
    }
  }
}
```

Commandant

```
extension AddCommand {
  struct Options: OptionsProtocol {
    let title: String

    static func evaluate(_ m: CommandMode)
      -> Result<Options, CommandantError<String>> {
      return Options.init
        <*> m <|* Argument<String>(usage: "Title for task")
    }
  }
}
```

```
> todos add "buy milk"
```

Commandant

Good for complex tools with lots of options 👍

Battle-tested by Carthage 👍

Custom operators 👌

Beak 

by Yonas Kolb

github.com/yonaskolb/beak

Beak 🐦

Static-analysis with SourceKit

Generated interface & help

Dependency management like Mint 🌱

Beak 🐦

```swift
// beak.swift

/// Create a new task
public func add(title: String) {}
```

Beak 🐦

```swift
// beak.swift

/// Create a new task
public func add(title: String) {}
```

```
> beak list
Functions:
  add: Create a new task

> beak run add --title="buy milk"
```

Beak 🐦

```swift
// beak.swift

/// Create a new task
public func add(_ title: String) {}
```

```
> beak list
Functions:
  add: Create a new task

> beak run add "buy milk"
```

Beak 🐦

Great for task-runners 👍

Use with Mint for super-productivity 🤩

Effective I/O

💬 Effective I/O

1. stdin
2. stderr
3. Exit codes
4. Files

stdin

```
func readline() -> String?
```

stdin

```
print("What is your name?")
print("> ", terminator: "")

let name = readLine() ?? "stranger"

print("Hello, \(name)")
```

stdin

```
print("What is your name?")
print("> ", terminator: "")

let name = readLine() ?? "stranger"

print("Hello, \(name)")
```

```
What is your name?
> Michael
Hello, Michael
```

stdin

```
print("What is your name?")
print("> ", terminator: "")

let name = readLine() ?? "stranger"

print("Hello, \(name)")
```

```
What is your name?
> ^D
Hello, stranger
```

stdin

```
while let input = readLine() {
   // ...
}
```

stdin

```
sequence(first: "", next: { _ in
  readLine(strippingNewline: false)
}).joined()
```

stderr

stderr

```
func print<Target>(_: Any..., to: inout Target) where Target : TextOutputStream
```

stderr

```
func print<Target>(_: Any..., to: inout Target) where Target : TextOutputStream

/// Returns the file handle associated with the standard error file
class var standardError: FileHandle { get }
```

stderr

```
extension FileHandle: TextOutputStream {
  public func write(_ string: String) {
    if let data = string.data(using: .utf8) {
        self.write(data)
    }
  }
}
```

stderr

```
func errorPrint(_ item: Any) {
  var stderr = FileHandle.standardError
  print(item, to: &stderr)
}
```

Error Codes

Error Codes

```
fatalError("oops!")
```

Lots of stack dump info 👎

Error Codes

```
import Darwin
exit(-1)
```

Files

by John Sundell

github.com/JohnSundell/Files

Files

Wrapper around Foundation APIs 👍

Very convenient to use 👍

Files

```
let folder = try Folder(path: "/users/john/folder")

let file = try folder.createFile(named: "file.json")
try file.write(string: "{\"hello\": \"world\"}")

try file.delete()
try folder.delete()
```

Files

```
for file in try Folder(path: "MyFolder").files {
    try file.rename(to: file.nameWithoutExtension.capitalized)
}
```

Files

```
let origin = try Folder(path: "/users/john/folderA")
let target = try Folder(path: "/users/john/folderB")
try origin.files.move(to: target)
```

Files

✓ SPM

✓ Carthage

✓ CocoaPods

Wrap-up

Wrap-up

1. 🏃‍♀️ Running Scripts
   — single-files & packages
   — JohnSundell/Marathon 🏃
   — yonaskolb/Mint 🌱

Wrap-up

1. Building Interfaces
    - DIY
    - kylef/commander
    - Carthage/Commandant
    - yonaskolb/beak

Wrap-up

1. Effective I/O
    - stdin
    - stderr
    - error codes
    - JohnSundell/Files

The Command Line Toolbox 🧰

A Crash Course on building your own CLI tools

Michael Bates — @mklbtz — mklbtz.com