Building Swift CLIs

Marek Fořt

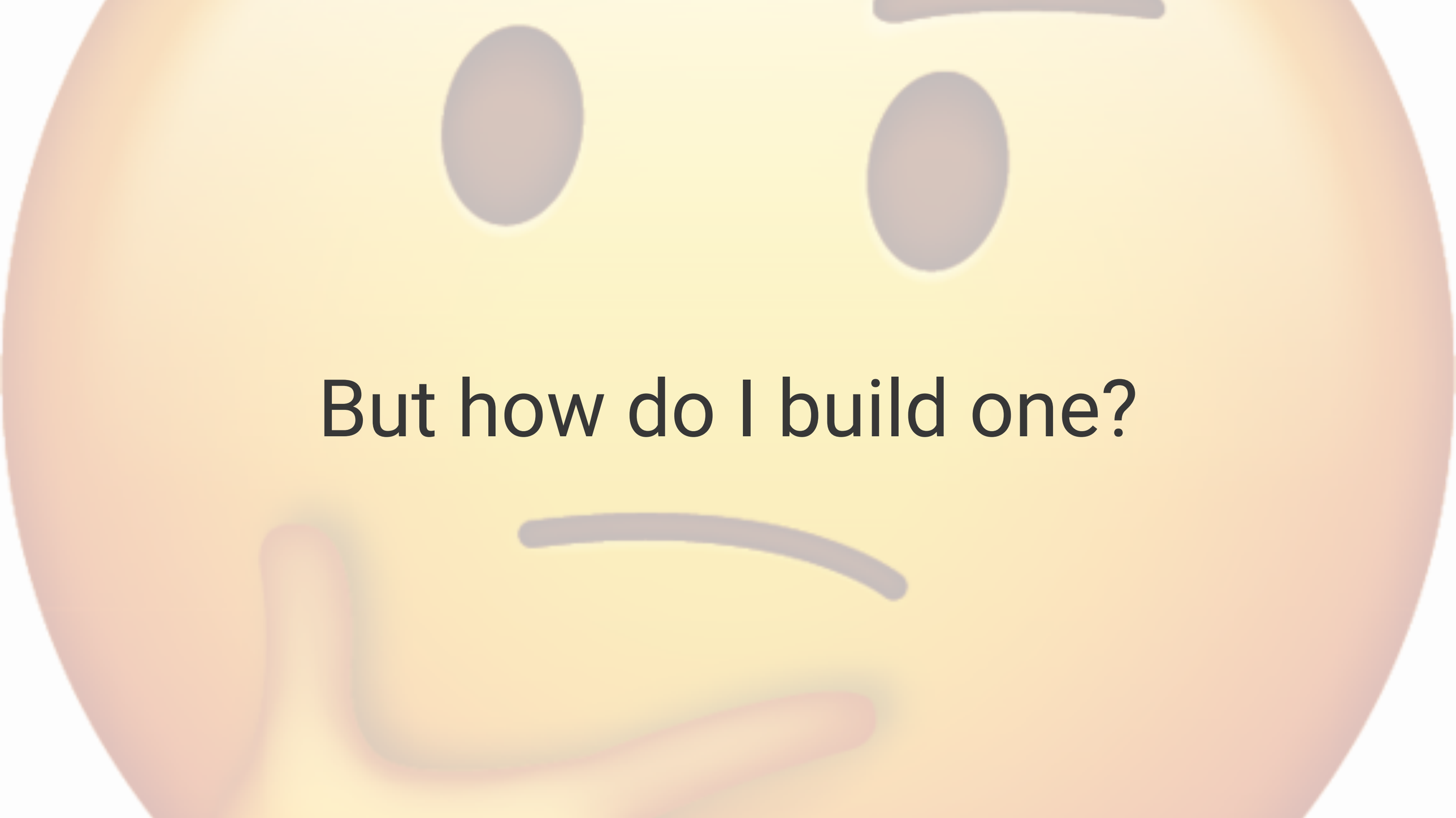

Wait a minute, who are you?

Marek Fořt

- Ackee iOS Developer
- Tuist core contributor
- CTU student

Why should I use Swift for CLI

- Tool for iOS developers
  - Easy contribution
- I love Swift
- Tools I am familiar with


But how do I build one?

Libraries
by Swift Community

Libraries
that Apple provides us

How does Apple do it?

Swift Package Manager

SPM's Swift 5.0 Package.swift

```
.library(
    name: "SwiftPM",
    targets: [
        "clibc",
        "SPMLibc",
        "POSIX",
        "Basic",
        "SPMUtility",
        "SourceControl",
        "SPMLLBuild",
        "PackageModel",
        "PackageLoading",
        "PackageGraph",
        "Build",
        "Xcodeproj",
        "Workspace"
    ]
),
```

SPM's Swift 5.0 Package.swift

```
.library(
    name: "SwiftPM",
    targets: [
        "clibc",
        "SPMLibc",
        "POSIX",
        "Basic",
        "SPMUtility",
        "SourceControl",
        "SPMLLBuild",
        "PackageModel",
        "PackageLoading",
        "PackageGraph",
        "Build",
        "Xcodeproj",
        "Workspace"
    ]
),
```

SPM's Swift 5.0 Package.swift

```
.library(
    name: "SwiftPM",
    targets: [
        "clibc",
        "SPMLibc",
        "POSIX",
        "Basic",
        "SPMUtility",
        "SourceControl",
        "SPMLLBuild",
        "PackageModel",
        "PackageLoading",
        "PackageGraph",
        "Build",
        "Xcodeproj",
        "Workspace"
    ]
),
```

Now

```
dependencies: [
    .package(url: "https://github.com/apple/swift-tools-support-core.git",
             .branch("master")),
]

.library(
    name: "SwiftToolsSupport-auto", targets: ["TSCBasic", "TSCUtility"]
)
```

Now

```
dependencies: [
    .package(url: "https://github.com/apple/swift-tools-support-core.git",
             .branch("master")),
]

.library(
    name: "SwiftToolsSupport-auto", targets: ["TSCBasic", "TSCUtility"]
)
```

TSCBasic

- File *paths*
- Process (command line commands)
- Terminal controller (colored output)
- Output streams
- Additional extensions and algorithms

Basic example

```
let myAbsolutePath = AbsolutePath("/switzerland")
let myRelativePath = RelativePath("conference")
var combinedPath = myAbsolutePath.appending(myRelativePath)
// /switzerland/conference
combinedPath = combinedPath.appending(component: "appbuilders")
// /switzerland/conference/appbuilders
combinedPath = combinedPath.appending(components: "appbuilders2020", "my_presentation")
// /switzerland/conference/appbuilders/appbuilders2020/my_presentation
```

Basic example

```
let myAbsolutePath = AbsolutePath("/switzerland")
let myRelativePath = RelativePath("conference")
var combinedPath = myAbsolutePath.appending(myRelativePath)
// /switzerland/conference
combinedPath = combinedPath.appending(component: "appbuilders")
// /switzerland/conference/appbuilders
combinedPath = combinedPath.appending(components: "appbuilders2020", "my_presentation")
// /switzerland/conference/appbuilders/appbuilders2020/my_presentation
```

Basic example

```
let myAbsolutePath = AbsolutePath("/switzerland")
let myRelativePath = RelativePath("conference")
var combinedPath = myAbsolutePath.appending(myRelativePath)
// /switzerland/conference
combinedPath = combinedPath.appending(component: "appbuilders")
// /switzerland/conference/appbuilders
combinedPath = combinedPath.appending(components: "appbuilders2020", "my_presentation")
// /switzerland/conference/appbuilders/appbuilders2020/my_presentation
```

Basic example

```
let myAbsolutePath = AbsolutePath("/switzerland")
let myRelativePath = RelativePath("conference")
var combinedPath = myAbsolutePath.appending(myRelativePath)
// /switzerland/conference
combinedPath = combinedPath.appending(component: "appbuilders")
// /switzerland/conference/appbuilders
combinedPath = combinedPath.appending(components: "appbuilders2020", "my_presentation")
// /switzerland/conference/appbuilders/appbuilders2020/my_presentation
```

Basic example

```
let myAbsolutePath = AbsolutePath("/switzerland")
let myRelativePath = RelativePath("conference")
var combinedPath = myAbsolutePath.appending(myRelativePath)
// /switzerland/conference
combinedPath = combinedPath.appending(component: "appbuilders")
// /switzerland/conference/appbuilders
combinedPath = combinedPath.appending(components: "appbuilders2020", "my_presentation")
// /switzerland/conference/appbuilders/appbuilders2020/my_presentation
```

Basic example

```
let myAbsolutePath = AbsolutePath("/switzerland")
let myRelativePath = RelativePath("conference")
var combinedPath = myAbsolutePath.appending(myRelativePath)
// /switzerland/conference
combinedPath = combinedPath.appending(component: "appbuilders")
// /switzerland/conference/appbuilders
combinedPath = combinedPath.appending(components: "appbuilders2020", "my_presentation")
// /switzerland/conference/appbuilders/appbuilders2020/my_presentation
```

TSCUtility

- Helpers for CLI development
- Argument parser
- Command definition
- Version struct
- Progress animations
- ...

Defining a Command

```
final class InitCommand: NSObject, Command {
    static var command: String = "init"

    let pathArgument: OptionArgument<String>

    init(parser: ArgumentParser) {
        let subParser = parser.add(subparser: InitCommand.command, overview: InitCommand.overview)

        pathArgument = subParser.add(option: "--path",
                                     shortName: "-p",
                                     kind: String.self,
                                     usage: "The path to the folder where the project will be generated.",
                                     completion: .filename)
    }

    func run(with arguments: ArgumentParser.Result) throws {
        let path = arguments.get(pathArgument)
    }
}
```

Defining a Command

```
final class InitCommand: NSObject, Command {
    static var command: String = "init"
    static var overview: String = "Initializes a new project"

    let pathArgument: OptionArgument<String>

    init(parser: ArgumentParser) {
        let subParser = parser.add(subparser: InitCommand.command, overview: InitCommand.overview)
        pathArgument = subParser.add(option: "--path",
                                     shortName: "-p",
                                     kind: String.self,
                                     usage: "The path to the folder where the project will be generated (Default: Current directory).",
                                     completion: .filename)
    }

    func run(with arguments: ArgumentParser.Result) throws {
        let path = arguments.get(pathArgument)
    }
}
```

Only 2019 kids will remember this...

Swift Argument Parser

```
struct InitCommand: ParsableCommand {
    @Option(name: .shortAndLong)
    var path: String?

    func run() throws {
        // Generate to path
    }
}

InitCommand.main()

init --path my_path
init -p my_path
```

```
struct InitCommand: ParsableCommand {
    @Option(name: .shortAndLong)
    var path: String?

    func run() throws {
        // Generate to path
    }
}

InitCommand.main()
```

```
init --path my_path
init -p my_path
```

```
struct InitCommand: ParsableCommand {
    @Option(name: .shortAndLong)
    var path: String?

    func run() throws {
        // Generate to path
    }
}

InitCommand.main()

init --path my_path
init -p my_path
```

```
struct InitCommand: ParsableCommand {
    @Option(name: .shortAndLong)
    var path: String?

    func run() throws {
        // Generate to path
    }
}

InitCommand.main()

init --path my_path
init -p my_path
```

```
struct InitCommand: ParsableCommand {
    @Option(name: .shortAndLong)
    var path: String?

    func run() throws {
        // Generate to path
    }
}

InitCommand.main()

init --path my_path
init -p my_path
```

Tests ✅

- Developers expect reliable tools
- Errors as first-class citizens
- Fun to write

Dependency injection

```
struct InitCommand: ParsableCommand {
    ...
    private let myService: MyServicing
    init(myService: MyServicing) {
        self.myService = myService
    }
}
```

🛑 Type 'InitCommand' does not conform to protocol 'ParsableArguments'

```
public protocol ParsableArguments: Decodable {
  /// Creates an instance of this parsable type using the definitions
  /// given by each property's wrapper.
  init()

  /// Validates the properties of the instance after parsing.
  ///
  /// Implement this method to perform validation or other processing after
  /// creating a new instance from command-line arguments.
  mutating func validate() throws
}
```

Introducing Services

```
struct InitService {
    private let myService: MyServicing
    init(myService: MyServicing = MyService()) {
        self.myService = myService
    }

    func run() throws {
        myService.someAction()
    }
}

struct InitCommand: ParsableCommand {
    ...
    func run() throws {
        try InitService().run()
    }
}
```

Introducing Services

```
struct InitService {
    private let myService: MyServicing
    init(myService: MyServicing = MyService()) {
        self.myService = myService
    }

    func run() throws {
        myService.someAction()
    }
}

struct InitCommand: ParsableCommand {
    ...
    func run() throws {
        try InitService().run()
    }
}
```

Introducing Services

```
struct InitService {
    private let myService: MyServicing
    init(myService: MyServicing = MyService()) {
        self.myService = myService
    }

    func run() throws {
        myService.someAction()
    }
}

struct InitCommand: ParsableCommand {
    ...
    func run() throws {
        try InitService().run()
    }
}
```

Introducing Services

```
struct InitService {
    private let myService: MyServicing
    init(myService: MyServicing = MyService()) {
        self.myService = myService
    }

    func run() throws {
        myService.someAction()
    }
}

struct InitCommand: ParsableCommand {
    ...
    func run() throws {
        try InitService().run()
    }
}
```

Introducing Services

```
struct InitService {
    private let myService: MyServicing
    init(myService: MyServicing = MyService()) {
        self.myService = myService
    }

    func run() throws {
        myService.someAction()
    }
}

struct InitCommand: ParsableCommand {
    ...
    func run() throws {
        try InitService().run()
    }
}
```

Services

- Solves dependency injection
- Commands become plain parsers without any logic
  - Easy testability

Demo time