Building Swift CLIs

Marek Fořt

WHO WE ARE

2012
Ackee is founded

60
Team members

120
+
Successful projects

Most of us are Czech Technical University graduates

Tapestry

What it does

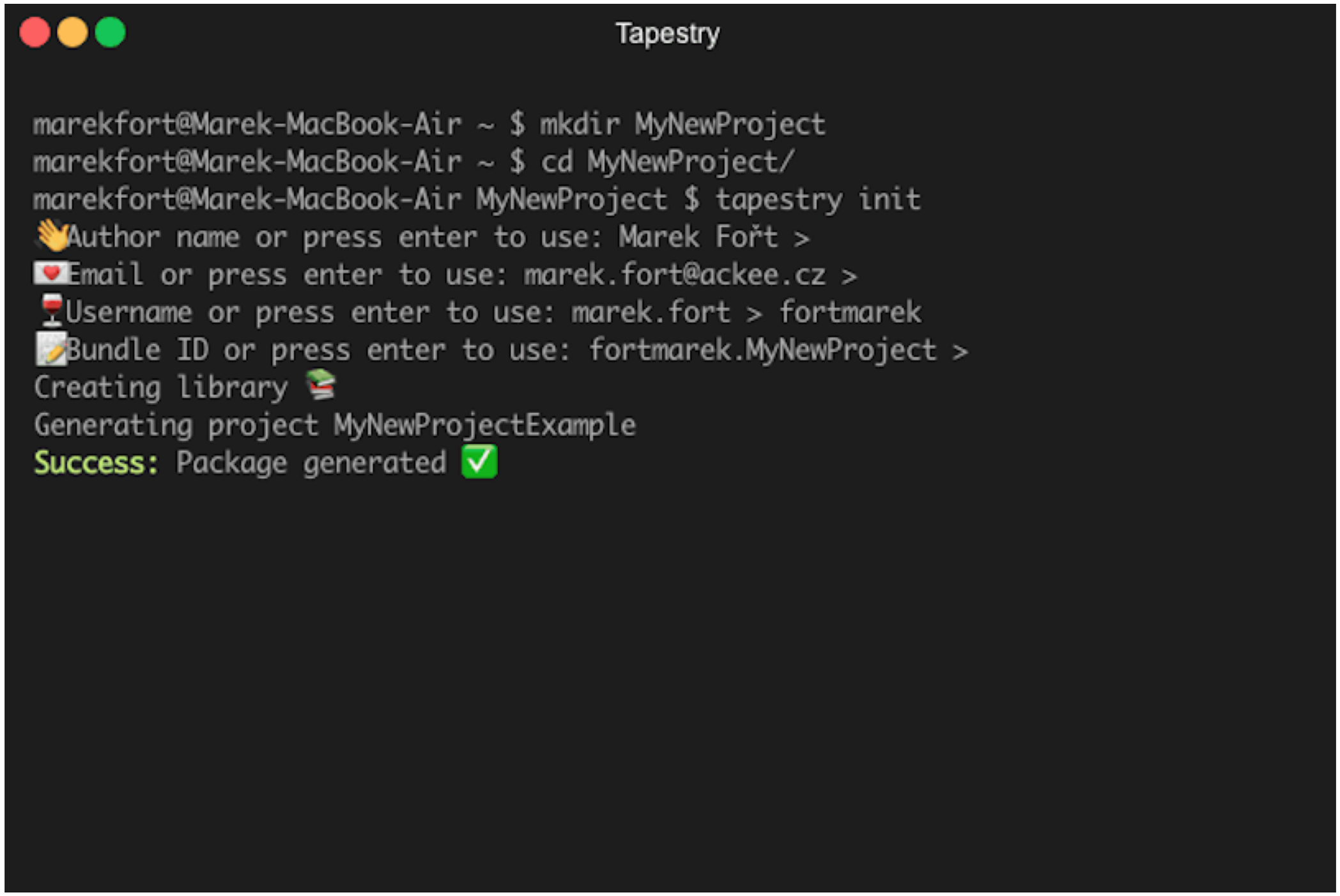
Framework template built around SPM

Eases release process with automation

Provides type-safe editable configuration file

A lot of concepts inspired by tuist (Xcode project generation tool)

Template


Tapestry
marekfort@Marek-MacBook-Air ~ $ mkdir MyNewProject
marekfort@Marek-MacBook-Air ~ $ cd MyNewProject/
marekfort@Marek-MacBook-Air MyNewProject $ tapestry init
👋Author name or press enter to use: Marek Fořt >
💌Email or press enter to use: marek.fort@ackee.cz >
🍷Username or press enter to use: marek.fort > fortmarek
📝Bundle ID or press enter to use: fortmarek.MyNewProject >
Creating library 📚
Generating project MyNewProjectExample
Success: Package generated ✅

Release


Terminalizer
(tapestry) $ tapestry release 6.2.4
Updating docs
Checking Cocoapods compatibility...
Checking Carthage compatibility...
Checking SPM compatibility...
Updating version
Success: Version updated to 6.2.4

Tapestry config

```
import PackageDescription

let config = TapestryConfig(
    release: Release(actions: [.pre(.docsUpdate),
                               .pre(.dependenciesCompatibility([.spm(.all)]))],
                     add: ["README.md",
                           "CHANGELOG.md"],
                     commitMessage: "Version \(Argument.version)",
                     push: true))
```

Before

Releasing

In this document you'll find all the necessary steps to release a new version of `xcodeproj`:

1. Re-generate the Carthage project with `tuist generate` *(Install Tuist if you don't have it installed already)*.
2. Update Carthage dependencies if they are outdated with `bundle exec rake carthage_update_dependencies`.
3. Validate the state of the project by running `bundle exec rake release_check`.
4. Update the `CHANGELOG.md` adding a new entry at the top with the next version. Make sure that all the changes in the version that is about to be released are properly formatted.
5. Update the version in the `xcodeproj.podspec` and `README.md` files.
6. Commit, tag and push the changes to GitHub.
7. Create a new release on GitHub including the information from the last entry in the `CHANGELOG.md`.
8. Push the changes to CocoaPods: `bundle exec pod trunk push --allow-warnings --verbose`.
9. Archive the Carthage framework by running `bundle exec rake arhive_carthage` and attach the `XcodeProj.framework.zip` to the GitHub release.

After

Releasing

> In this document you'll find all the necessary steps to release a new version of `xcodeproj`:

1. Run `tapestry release version-number` (eg `tapestry release 7.1.0`) *(Install tapestry and tuist if you don't have them installed already)*.
2. Create a new release on GitHub including the information from the last entry in the `CHANGELOG.md`.
3. Attach `XcodeProj.framework.zip` to the GitHub release.

Swift Package Manager 

Why to Use SPM Libraries in Your Project

+ Eliminate dependencies

+ Closely aligned with Apple’s standards

+ Wide range of tools

- Lack of documentation

CommandRegistry

```swift
public final class CommandRegistry {
    let parser: ArgumentParser
    var commands: [Command] = []
    init() {
        // SPMUtility's ArgumentParser
        parser = ArgumentParser(commandName: "tapestry", usage: "<command> <options>")
        // Registering our commands
        register(command: InitCommand.self)
        register(command: ReleaseCommand.self)
    }

    func register(command: Command.Type) {
        commands.append(command.init(parser: parser))
    }

    public func run() {
        let processedArguments = Array(ProcessInfo.processInfo.arguments)
        let parsedArguments = try parser.parse(arguments)
        // Find command and run it
        if let command = commands.first(where: { type(of: $0).command == parsedArguments.subparser(parser) }) {
            try command.run(with: arguments)
        }
    }
}
```

ackee

Command

```swift
final class ReleaseCommand: NSObject, Command {
    static var command: String = "release"
    static var overview: String = "Runs release steps defined in `TapestryConfig.swift` file"

    let versionArgument: PositionalArgument<Version>

    required init(parser: ArgumentParser) {
        let subParser = parser.add(subparser: ReleaseCommand.command, overview: ReleaseCommand.overview)
        versionArgument = subParser.add(positional: "Version", kind: Version.self)
    }

    func run(with arguments: ArgumentParser.Result) throws {
        guard let version = arguments.get(versionArgument) else { throw ReleaseError.noVersion }
    }
}
```

Run

```
import Foundation
import TapestryKit

var registry = CommandRegistry()
registry.run()
```

🔥 Errors 🔥

Errors

Make errors actionable

Provide enough information for the user to be able to debug if necessary

Make them part of your testing process

Let errors bubble up and handle them at one place

Revisiting CommandRegistry

```
public final class CommandRegistry {
    let parser: ArgumentParser
    var commands: [Command] = []
    init() {
        parser = ArgumentParser(commandName: "tapestry", usage: "<command> <options>")
        register(command: InitCommand.self)
        register(command: ReleaseCommand.self)
    }

    func register(command: Command.Type) {
        commands.append(command.init(parser: parser))
    }

    public func run() {
        do {
            let processedArguments = Array(ProcessInfo.processInfo.arguments)
            let parsedArguments = try parser.parse(processedArguments)
            if let command = commands.first(where: { type(of: $0).command == parsedArguments.subparser(parser) }) {
                try command.run(with: arguments)
            }
        // All errors fall through these two cases
        } catch let error {
            print(error.localizedDescription)
        }
    }
}
```

Error example

```
TapestryDemo (master) $ tapestry release 1.1.1
Error: Version tag 1.1.1 already exists. Either delete the tag or use a different one
```

Testability

Dependency injection

Shared dependencies for the most used dependencies

Test for specific errors

WWYC - wrap what you can

Writing tests should be as fun as writing new functionalities

ackee

System

```
public final class System: Systeming {
    // Our shared System instance
    public static var shared: Systeming = System()

    public func run(_ arguments: [String]) throws {
        // Leveraging SPMUtility's Process to run command in shell
        let process = Process(arguments: arguments)

        try process.launch()
        let result = try process.waitUntilExit()

        try result.throwIfErrored()
    }
}
```

ackee

Shared test cases

```
// Shared XCTestCase to subclass
public class TapestryUnitTestCase: XCTestCase {
    public var system: MockSystem!

    public override func setUp() {
        super.setUp()
        // Reinit System for each test case
        system = MockSystem()
        System.shared = system
    }
}
```

Testing specific error

```
func test_check_fails_when_carthage() {
    // Given
    system.errorCommand(["carthage", "build", "--no-skip-current"])

    // Then
    XCTAssertThrowsSpecific(try subject.checkCompatibility(with: [.carthage],
                                                           path: fileHandler.currentPath),
                            DependenciesCompatibilityError.carthage)
}
```

Takeaways

Use Tapestry to automate release process

SPM libraries as a go-to sophisticated tool for CLI development

Treat errors as a key part of your tool

Ensure good testability to achieve high test coverage

ackee

Krásní lidé, krásné appky

www.ackee.cz