Command-line Automations

Unchain your Xcode Projects!

synesthesia
the digital experience company

Stefano Mondino

Mobile tech leader @Synesthesia

Twitter @puntoste
Github @stefanomondino

synesthesia
the digital experience company

Stuff we'd like to automate

- Project environment creation
- xcodeproj proper setup
- Beautification of our code (lint & format)
- Static references to resources (translations, images, etc.)

What should I do when I checkout a new project?

What “environmental tools” does my project require?

synesthesia
the digital experience company

Is Cocoapods available?

What version should I use?

synesthesia
the digital experience company

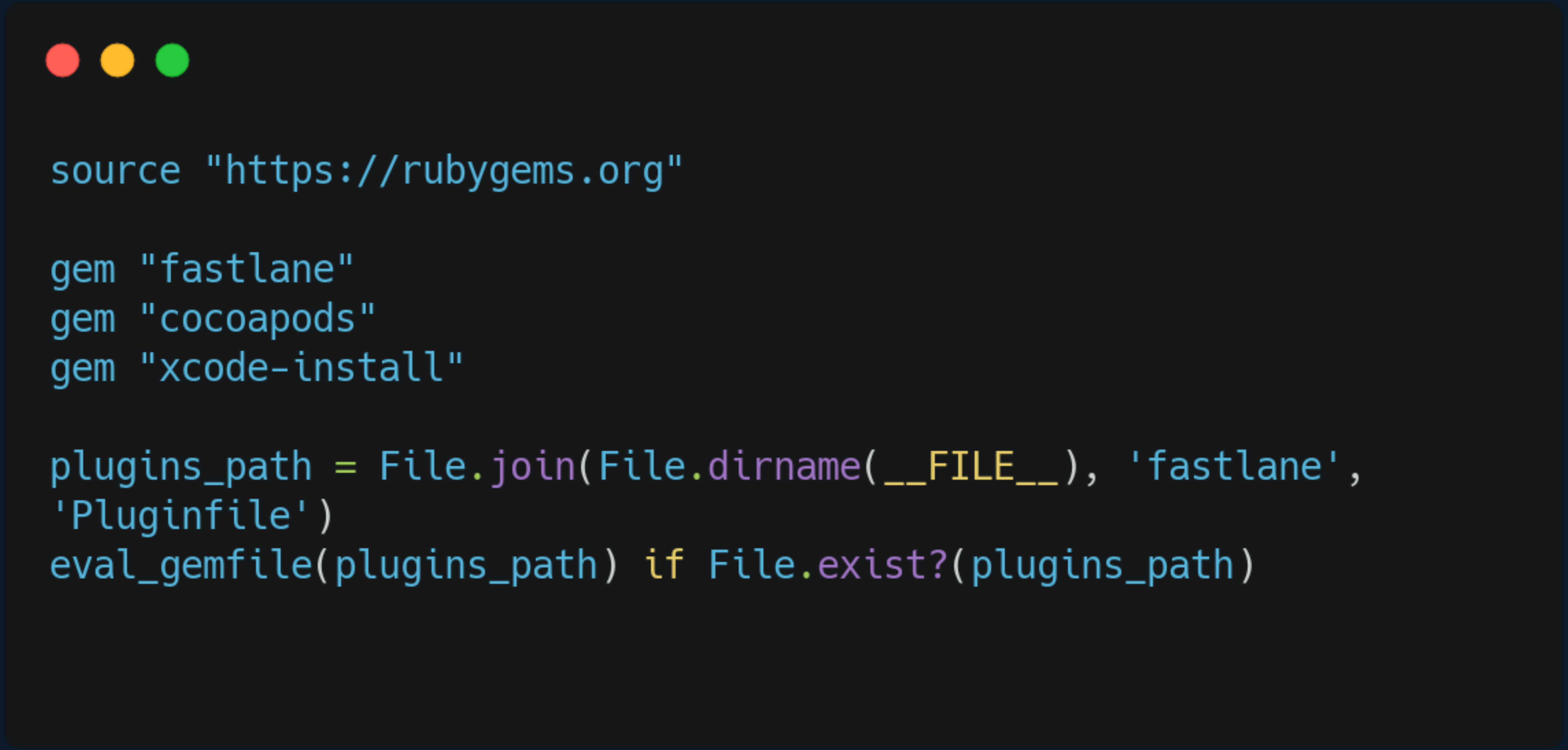
Trying to avoid this

The sandbox is not in sync with the Podfile.lock. Run 'pod install' or update your CocoaPods installation.

Gemfile

- Container of all ruby dependencies
- Ruby code: can be scripted
- Lock specific gem versions for your project
- `bundle exec` before each command (`bundle exec pod install`)
- Requires: bundler -> `gem install bundler`

Gemfile



```ruby
source "https://rubygems.org"

gem "fastlane"
gem "cocoapods"
gem "xcode-install"

plugins_path = File.join(File.dirname(__FILE__), 'fastlane',
'Pluginfile')
eval_gemfile(plugins_path) if File.exist?(plugins_path)
```

Is Carthage available?

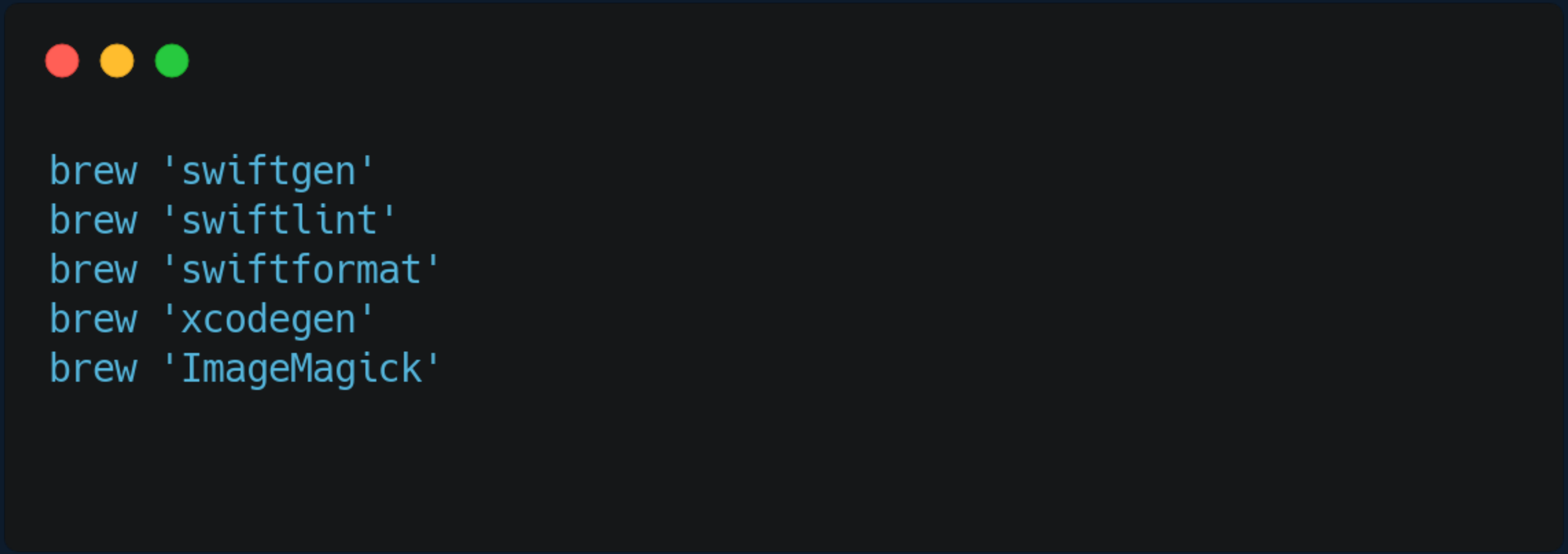
And what about SwiftLint / SwiftFormat / other non-ruby dependencies?

Brewfile

- brew install …
- carthage, swiftlint, xcodegen
- List dependencies in Brewfile
- Requires: homebrew -> see https://brew.sh

synesthesia
the digital experience company

Brewfile



```
brew 'swiftgen'
brew 'swiftlint'
brew 'swiftformat'
brew 'xcodegen'
brew 'ImageMagick'
```

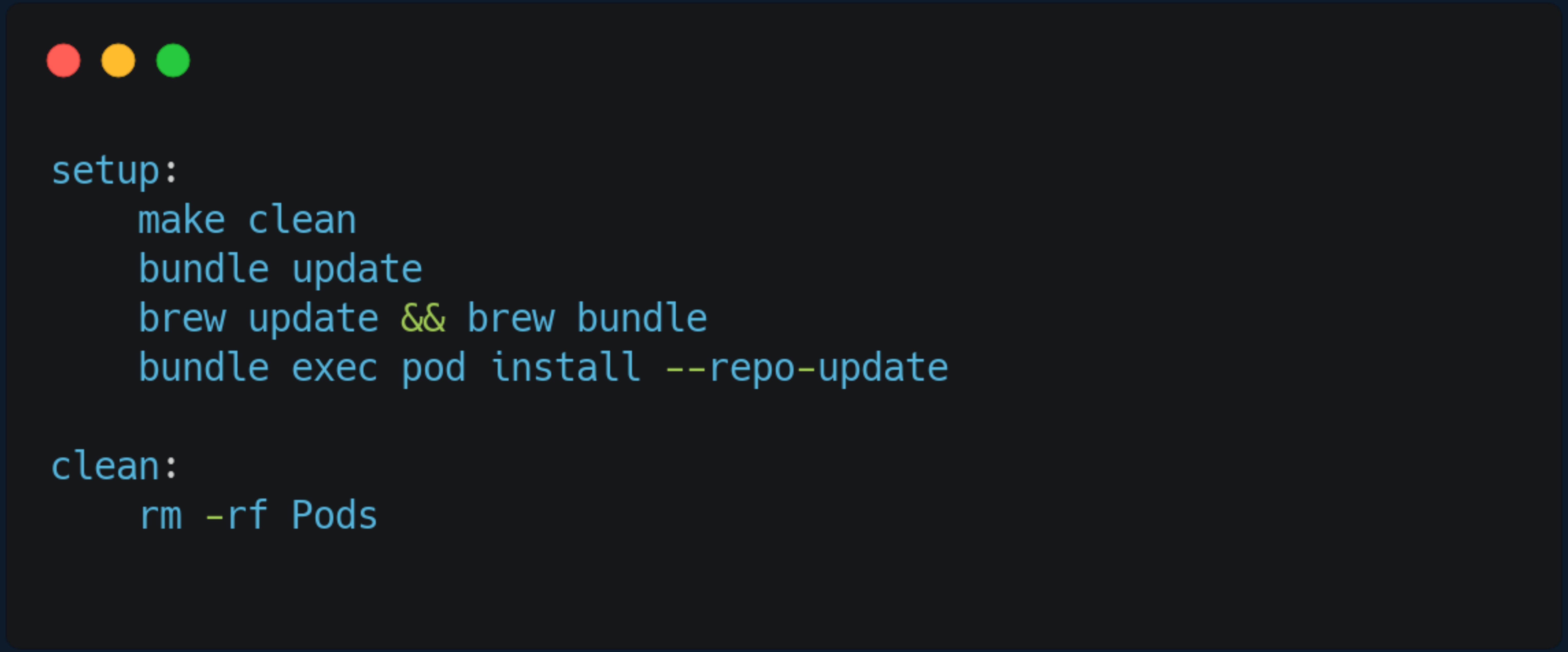
How do i chain those commands?

Should I really remember everything?

Makefile

- Contains targets: list of shell commands
- Can define environment variables
- Invoke a target with `make target`
- Requires: Command line tools

Makefile



```
setup:
    make clean
    bundle update
    brew update && brew bundle
    bundle exec pod install --repo-update

clean:
    rm -rf Pods
```

Why should I deal with xcodeproj stuff?

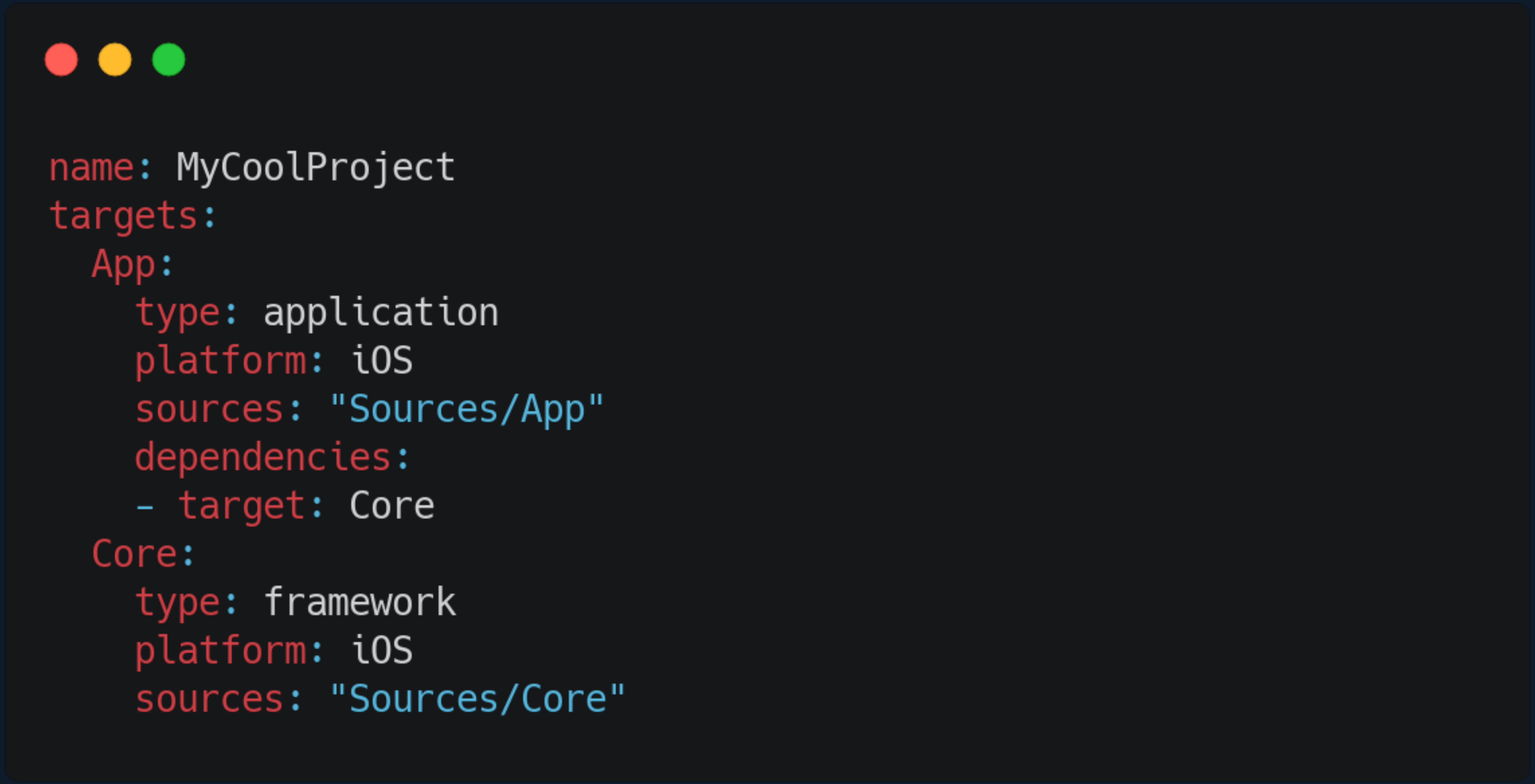
Merge conflicts are hard to solve

Errors are hard to spot

XcodeGen

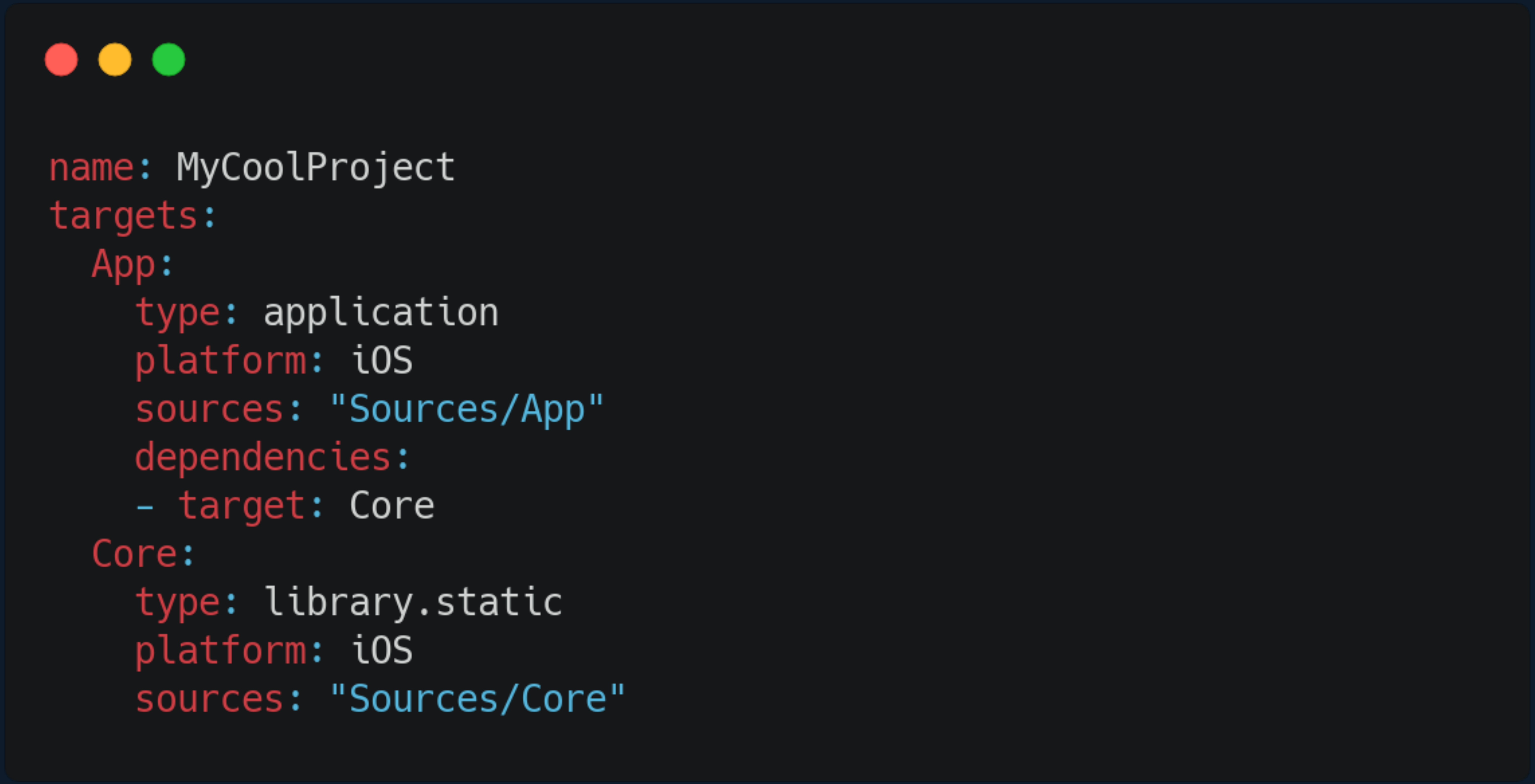
- A xcodeproj automatic generator (written in Swift ❤️)
- Driven by a YAML configuration file (project.yml)
- Supports environment variables from Makefile
- No need to check xcodeproj file into version control

project.yml



```yaml
name: MyCoolProject
targets:
  App:
    type: application
    platform: iOS
    sources: "Sources/App"
    dependencies:
    - target: Core
  Core:
    type: framework
    platform: iOS
    sources: "Sources/Core"
```

project.yml (static lib)



```yaml
name: MyCoolProject
targets:
  App:
    type: application
    platform: iOS
    sources: "Sources/App"
    dependencies:
    - target: Core
  Core:
    type: library.static
    platform: iOS
    sources: "Sources/Core"
```

Pros

- No accidental/unwanted changes to main xcodeproj
- Merge conflicts on YAML files (easier to solve)
- Names and types can be changed from environment variables
- Easy to add custom configurations or custom targets

synesthesia
the digital experience company

Cons

- It’s a third party dependency
- No-conventional way to change settings
- Not a standard: developers may need instructions

synesthesia
the digital experience company

Alternatives

- Tuist (Swift)
- Xcake (Ruby)
- struct(YAML)

How can enforce my team’s codestyle?

Everyone codes differently.

synesthesia
the digital experience company

SwiftFormat

Automatic code reformatter, based on general swift conventions

SwiftLint

Code linter -> Warnings and errors if code style is “wrong”
Autocorrection is available as separate command

Check this out!

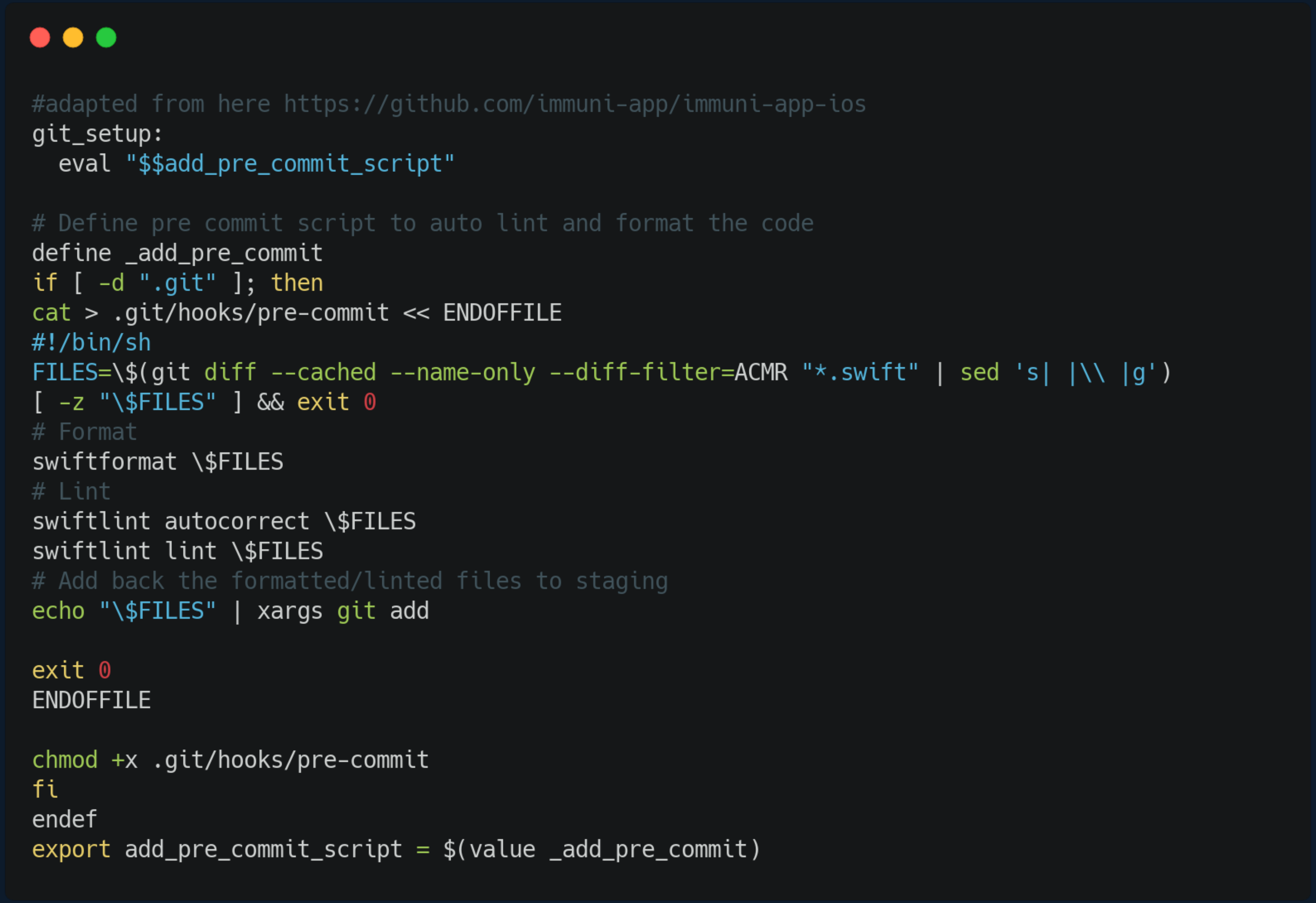
https://nshipster.com/swift-format/

{}

Format/lint automation

- pre-commit git hooks: executed right before commit
- run swift format + swiftlint only on modified files
- install git-hook with Makefile
- Credits: Immuni (italian Covid19-Tracing by Bending Spoons)

Makefile script



```
#adapted from here https://github.com/immuni-app/immuni-app-ios
git_setup:
  eval "$$add_pre_commit_script"

# Define pre commit script to auto lint and format the code
define _add_pre_commit
if [ -d ".git" ]; then
cat > .git/hooks/pre-commit << ENDOFFILE
#!/bin/sh
FILES=\$(git diff --cached --name-only --diff-filter=ACMR "*.swift" | sed 's| |\\ |g')
[ -z "\$FILES" ] && exit 0
# Format
swiftformat \$FILES
# Lint
swiftlint autocorrect \$FILES
swiftlint lint \$FILES
# Add back the formatted/linted files to staging
echo "\$FILES" | xargs git add

exit 0
ENDOFFILE

chmod +x .git/hooks/pre-commit
fi
endef
export add_pre_commit_script = $(value _add_pre_commit)
```

How can statically reference my resources?

Should I remember all those strings? What if I type them wrong?

SwiftGen

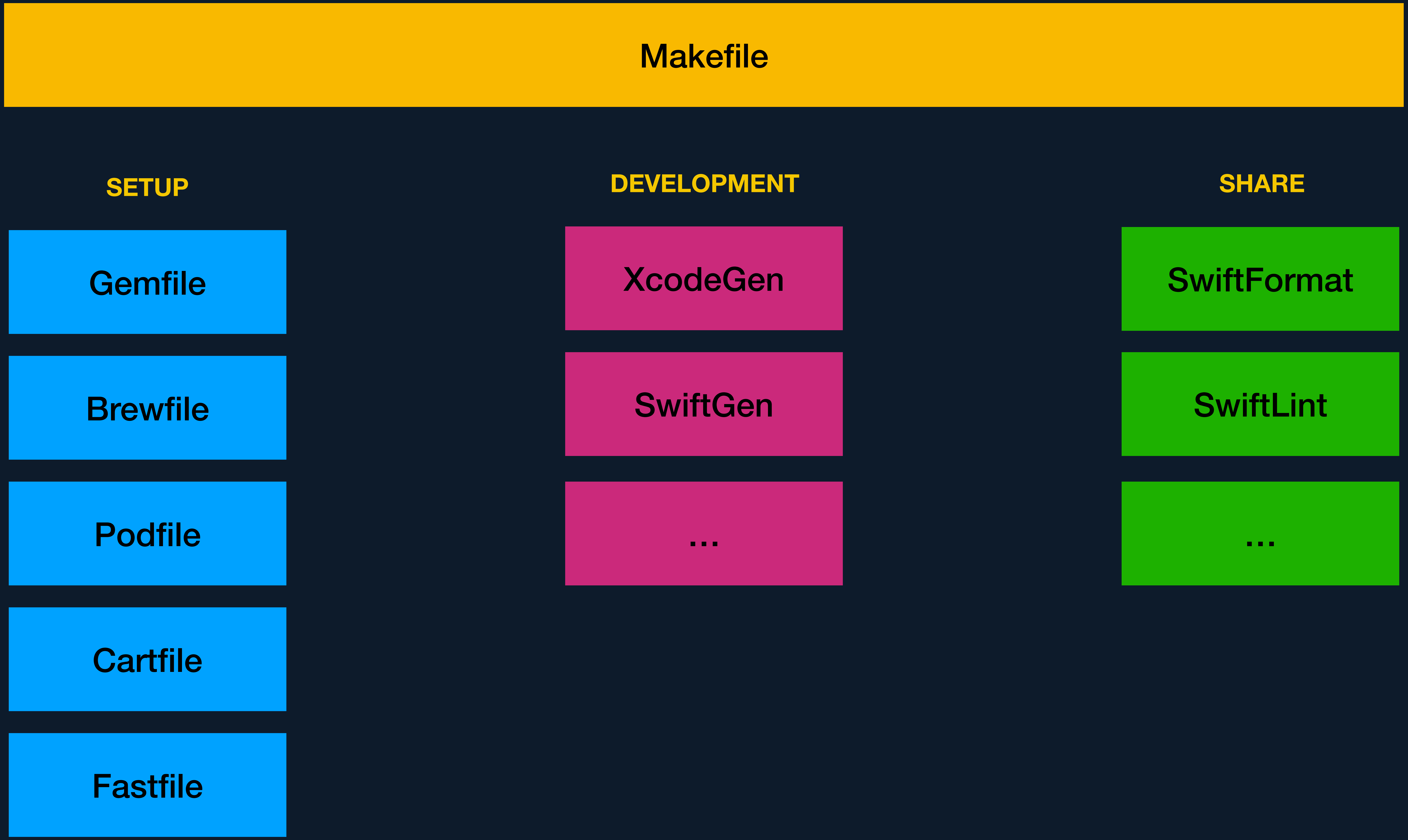
- Generates source code files through templates
- Input can be xcassets, custom JSON/YAML/pList
- Processing with Stencil templates
- Generated output is Swift code

SwiftGen - Use cases

- Swift enum with all images in a xcassets folder
- Swift enum with all Storyboard/xibs contained in a folder
- Swift custom colors from json file
- Swift enum with translations
- Your custom template for your specific use case

Wrapping up!

synesthesia
the digital experience company


Makefile
SETUP
Gemfile
Brewfile
Podfile
Cartfile
Fastfile
DEVELOPMENT
XcodeGen
SwiftGen
...
SHARE
SwiftFormat
SwiftLint
...

Showtime!

https://github.com/stefanomondino/CommandLineAutomations

Thanks!

synesthesia
the digital experience company