











iOS 持续交付之 Fastlane

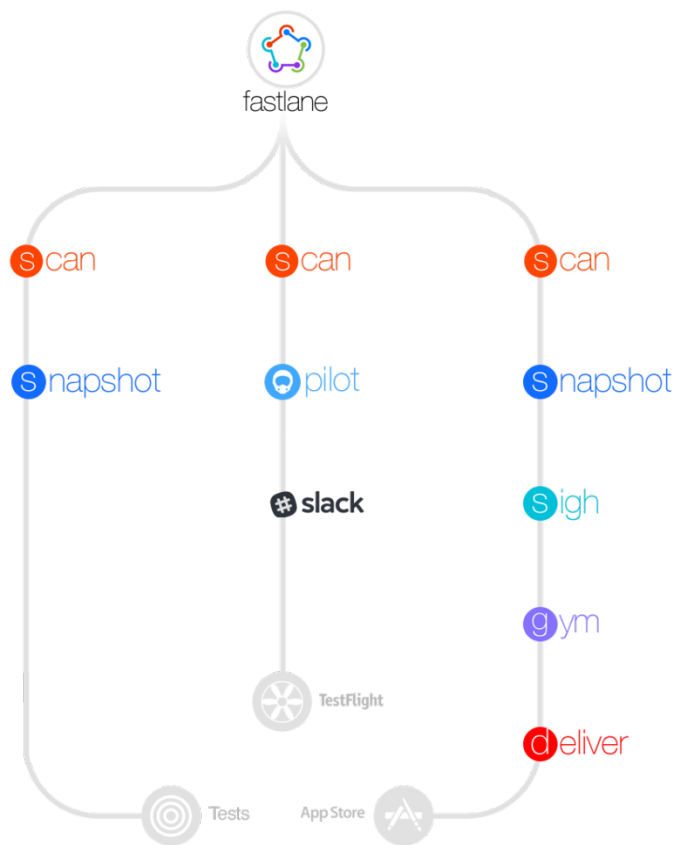


小目标：使用 `Jenkins` 一键构建，并自动上传到 `App Store`。

一、为什么选择 [Fastlane](#)?

fastlane	
	Save hours every time you push a new release to the store or beta testing service
	Integrates with all your existing tools and services (170 actions currently)
	100% open source under the MIT license
	Easy setup assistant to get started in a few minutes
	Runs on your machine, it's your app and your data
	Integrates with all major CI systems
	Supports iOS, Mac, and Android apps
	Extend and customise <i>fastlane</i> to fit your needs, you're not dependent on anyone
	Never remember any commands any more, just <i>fastlane</i>
	Deploy from any computer, including a CI server

`fastlane` 是为 `iOS` 和 `Android` 应用程序自动化测试部署和发布的最简单方法。🐼 处理所有繁琐的任务，如生成屏幕截图，处理代码签名以及发布应用程序。



使用场景

- 提交时执行测试（包括单元测试和集成测试）。
- 构建并分发内部测试，公开测试版本。
- 构建生产版本并上传至 ITC（包括更新配置文件,创建新的屏幕截图,上传应用并提交审核）。
- ...

工具集

fastlane 将如下的工具套件有机地结合起来,从管理证书到单元测试,从编译打包到上传发布,都能通过命令行轻松完成.该套件支持与 **Jenkins** 和 **CocoaPods**, **xctools** 等其他第三方工具的集成,并且能够定义多个通道 (lanes) 以支持不同的部署目标。

- 测试工具
 - scan: 自动运行测试工具，可以生成漂亮的HTML报告。
- 生成证书、配置工具
 - cert: 自动创建iOS代码签名证书(.cert文件)。
 - sigh: 创建、更新、下载和修复 provisioning profiles,支持App Store, Ad Hoc, Development和企业 profiles。
 - pem: 自动生成、更新推送配置文件。
- 截图、上传、描设备边框
 - deliver: 上传截图、元数据、App到iTunesConnect。
 - snapshot: 依靠 UI Test 完成截图。
 - frameit: 快速地把应用截图放入设备框里。
- 自动化编译工具
 - gym: 编译、打包iOS app,生成签名的ipa文件。
- App 公测工具
 - pilot: 管理TestFlight测试用户，上传二进制文件。
 - firm: 管理firm。

• ...

二、准备工作，很重要的哟

1. 配置当前设备环境，最新的 `fastlane(2.75.1)` 需要 `2.1` 以上的 `ruby` 版本，正常的版本低点的，也是要求 `2.0` 以上的。因为 `fastlane` 工具是使用 `ruby` 写的。版本过低的建议安装 `rvm` 来升级 `ruby`。

```
curl -L get.rvm.io | bash -s stable
```

```
source ~/.bashrc  
source ~/.bash_profile
```

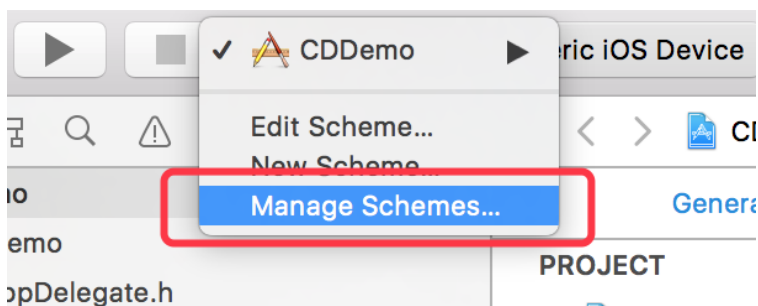
```
# 检测是否安装成功  
rvm -v
```

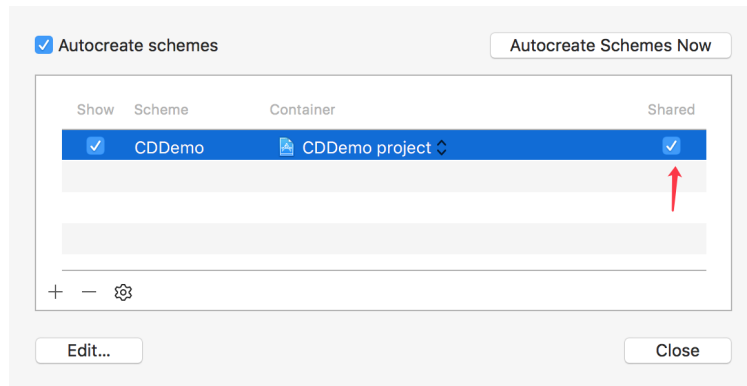
2. 设置环境变量，`fastlane` 需要设置一些环境变量才能正确运行，如果当前的语言环境没有设置为 `UTF-8`，会导致构建和上传的时候出现问题。在 `~/.bashrc`，`~/.bash_profile` 或者 `~/.zshrc` 文件下添加如下内容：

```
export LC_ALL=en_US.UTF-8  
export LANG=en_US.UTF-8
```

3. 安装 `Xcode` 命令行工具 `xcode-select --install`，如果已经安装会提示 `xcode-select: error: command line tools are already installed, use "Software Update" to install updates`。
4. 创建 `App ID`，`证书`，在 `iTunes connect` 创建一个用于测试的 app。
5. [安装fastlane](#)。
6. 创建一个测试demo。

1. 并将其 `scheme` 设置为 `shared`，不然 `fastlane init` 的时候会失败。





2. 设置好签名配置文件。
3. 为app添加 `icon`。
4. 修改 `devices` 为 `iPad` 或者 `iPhone`。

三、实践

fastlane init

1. cd 到项目目录下，对于ruby安装程序，使用命令 `sudo fastlane init`。(swift使用 `fastlane init` `swift`，Swift安装仍在测试阶段。有关更多信息，请参阅[Fastlane.swift](#)文档。)

```
6. ruby
Last login: Sat Jan 13 11:42:04 on ttys005
laiyoung_@LaiYoung-deMacBook-Pro:~$ cd /Users/laiyoung_/Documents/iOS/Test/CDDemo
laiyoung_@LaiYoung-deMacBook-Pro:~/Documents/iOS/Test/CDDemo$ fastlane init
```

2. 会问你想使用 `fastlane` 做什么？这里我们输入 `3`，自动发布到 `Apple Store`。

```
[19:28:50]: What would you like to use fastlane for?
1. 📸 Automate screenshots
2. 🚀 Automate beta distribution to TestFlight
3. 🚀 Automate App Store distribution
4. 🛠 Manual setup - manually setup your project to automate your tasks
```

执行过程中会要求你输入Apple开发证书的 `Apple ID`，如果有多个Team，会让你选择team。

3. 接着会问是否想用 `fastlane` 来管理你的 `app metadata`。

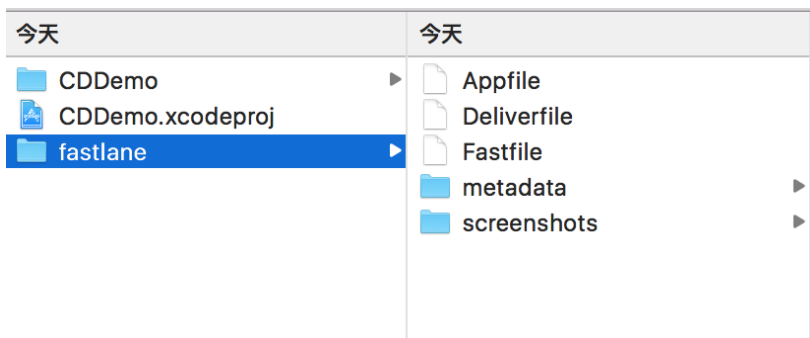
```
[20:38:19]: -----
[20:38:19]: --- Manage app metadata? ---
[20:38:19]: -----
[20:38:19]: Would you like to have fastlane manage your app's metadata?
[20:38:19]: If you enable this feature, fastlane will download your existing metadata and screenshots.
[20:38:19]: This way, you'll be able to edit your app's metadata in local '.txt' files.
[20:38:19]: After editing the local '.txt' files, just run fastlane and all changes will be pushed up.
[20:38:19]: If you don't want to use this feature, you can still use fastlane to upload and distribute new builds to the App Store
[20:38:19]: Would you like fastlane to manage your app's metadata? (y/n)
```

- 输入 `y`，`fastlane` 则会下载现有的元数据和屏幕截图。如果我们编辑了download下来的 `.txt` 文件，在使用 `fastlane` 上传app到 `iTunes connect` 的时候也会将这些内容上传到 `iTunes connect`。
- 输入 `n`，不做任何操作，仍然能使用 `fastlane` 上传app到 `App Store`。

4. 如果最后出现 `fastlane release`，就表示init成功了。

```
[20:40:18]: -----
[20:40:18]: --- Where to go from here? ---
[20:40:18]: -----
[20:40:18]: 📖 Learn more about how to automatically generate localized App Store screenshots:
[20:40:18]:      https://docs.fastlane.tools/getting-started/ios/screenshots/
[20:40:18]: 📖 Learn more about distribution to beta testing services:
[20:40:18]:      https://docs.fastlane.tools/getting-started/ios/beta-deployment/
[20:40:18]: 📖 Learn more about how to automate the App Store release process:
[20:40:18]:      https://docs.fastlane.tools/getting-started/ios/appstore-deployment/
[20:40:18]: 📖 Learn more about how to setup code signing with fastlane
[20:40:18]:      https://docs.fastlane.tools/codesigning/getting-started/
[20:40:18]:
[20:40:18]: To try your new fastlane setup, just enter and run
[20:40:18]: $ fastlane release
```

5. 此时项目目录下会多出一个 `fastlane` 的文件夹。



如果 `Deliverfile`，`screenshots` 和 `metadata` 目录没被创建，可以运行 `deliver init` 来创建。

在 `Deliverfile` 文件里，添加 `force true`，不然会在上传到 `iTunes connect` 的时候会弹出一个 `Preview.html` 网页。

使用Gemfile

1. 在项目根目录下 `touch` 一个 `Gemfile` 文件，添加以下内容

```
source "https://rubygems.org"
gem "fastlane"
```

2. 执行如下命令：

```
# 安装bundler
sudo gem install bundler
```

```
# 更新 bundle，成功之后会生成一个版本控制的Gemfile.lock文件
[sudo] bundle update
```

3. 执行命令：`bundle exec fastlane [lane_name]`，执行 `lane_name` 脚本。这里的 `lane_name` 是脚本的名称，我们可以理解为函数名，如果我们只执行 `bundle exec fastlane` 命令，则会有一个让我们选择的地方，选择需要执行的脚本。

```

laiyoung_@LaiYoung-deMBP:~/Documents/iOS/Test/CDDemo$ bundle exec fastlane
[21:36:48]: -----
[21:36:48]: --- Step: default_platform ---
[21:36:48]: -----
[21:36:48]: Welcome to fastlane! Here's what your app is setup to do:
+-----+
| Available lanes to run |
+-----+
| Number | Lane Name | Description |
+-----+
| 1       | ios release | Push a new release |
|         |             | build to the App |
|         |             | Store |
| 0       | cancel      | No selection, exit |
|         |             | fastlane! |
+-----+
[21:36:48]: Which number would you like run?
1

```

会将项目名，`ipa` 存放的路径，`app_identifier` 等一系列信息打印出来。

```

[21:01:43]: Running lane 'ios release'. Next time you can do this by directly typing 'fastlane ios release'
[21:01:43]: Driving the lane 'ios release'
[21:01:43]: -----
[21:01:43]: --- Step: build_app ---
[21:01:43]: -----
[21:01:43]: $ xcodebuild -list -project ./CDDemo.xcodeproj
[21:01:43]: $ xcodebuild -showBuildSettings -scheme CDDemo -project ./CDDemo.xcodeproj
[21:01:48]: Detected provisioning profile mapping: {"[REDACTED]Demo"=>"QTCDTestDemo"}

Summary for gym 2.75.1
+-----+
| scheme | CDDemo |
| project | ./CDDemo.xcodeproj |
| destination | generic/platform=iOS |
| export_options.provisioningProfiles | QTCDTestDemo |
| output_name | CDDemo |
| build_path | /Users/laiyoung/Library/Developer/Xcode/Archives/2018-01-12 |
| clean | false |
| output_directory | |
| silent | false |
| skip_package_ipa | false |
| buildlog_path | ~/Library/Logs/gym |
| skip_profile_detection | false |
| xcode_path | /Applications/Xcode.app |
+-----+

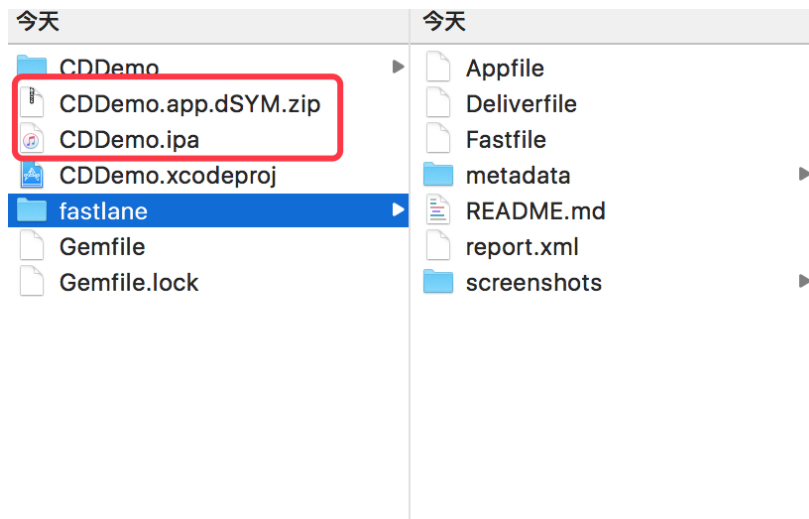
```

```

+-----+
| deliver 2.75.1 Summary |
+-----+
| ipa | /Users/laiyoung/Documents/iOS/Test/CDDemo/CDDemo.ipa |
| screenshots_path | ./fastlane/screenshots |
| metadata_path | ./fastlane/metadata |
| app_version | 1.0 |
| username | [REDACTED] |
| app_identifier | [REDACTED] |
| edit_live | false |
| platform | ios |
| skip_binary_upload | false |
| skip_screenshots | false |
| skip_metadata | false |
| skip_app_version_update | false |
| force | false |
| submit_for_review | false |
| automatic_release | false |
| phased_release | false |
| team_id | [REDACTED] |
| dev_portal_team_id | [REDACTED] |
| overwrite_screenshots | false |
| run_precheck_before_submit | true |
| precheck_default_rule_level | warn |
| ignore_language_directory_validation | false |
| precheck_include_in_app_purchases | true |
+-----+

```

`ipa` 和 `dSYM` 文件都存放在项目根目录。



紧接着会自动上传 `metadata` 和 `ipa` 到 `iTunes Connect`。

```
[✓] Uploading metadata to iTunes Connect
[21:59:15]: Successfully uploaded set of metadata to iTunes Connect
[21:59:17]: Starting with the upload of screenshots...
[21:59:17]: Successfully uploaded screenshots to iTunes Connect
[21:59:19]: Uploading binary to iTunes Connect
[21:59:19]: Going to upload updated app to iTunes Connect
[21:59:19]: This might take a few minutes. Please don't interrupt the script.
[22:01:33]: iTunes Transporter successfully finished its job
[22:01:33]: -----
[22:01:33]: Successfully uploaded package to iTunes Connect. It might take a few minutes until it's visible online.
[22:01:33]: -----
[22:01:33]: Finished the upload to iTunes Connect
```

最后会输出每个脚本执行所消耗的时间(s)。

```
+-----+-----+-----+
|               fastlane summary               |
+-----+-----+-----+
| Step | Action                | Time (in s) |
+-----+-----+-----+
| 1    | default_platform      | 0            |
| 2    | build_app              | 9            |
| 3    | upload_to_app_store    | 421          |
+-----+-----+-----+

[22:01:46]: fastlane.tools just saved you 7 minutes! 🎉
```

进阶

如果只是很简单的上传到 `iTunes connect`，上面的操作就可以满足。

如果我们是多个 `target` 或者需要配置一些 `ITC` 上面的内容，则需要进一步的深入。

metadata

`metadata` 是包含应用在 `ITC` 上面的各种信息，可以使用它配置我们的 `ITC`，建议使用 `Deliverfile`。

screenshots

屏幕截图数据。

[Appfile](#)

存储App信息，比如 `Apple ID`，`bundle ID` 等信息。

[Deliverfile](#)

交付文件。在这个文件里面可以设置 `iTunes connect` 的所有配置项，例如：

- `release_notes`，此版本新增内容。
- `copyright`，版权信息。
- `submit_for_review`，上传完成后是否直接提交新版本进行审查。
- `force`，跳过HTML报告文件验证。
- ...

请在设置 `release_nores`、`support_url`、`private_url` 等配置的时候，采用 `hash` 的方式写，[国家代码](#)，例如：

```
release_notes(  
  # 中国  
  'zh-Hans' => ENV['RELEASE_NOTES'],  
  # 澳大利亚  
  'en-au' => ENV['RELEASE_NOTES_AU'],  
  # 美国  
  'en-us' => ENV['RELEASE_NOTES_US']  
)
```

[Fastfile](#)

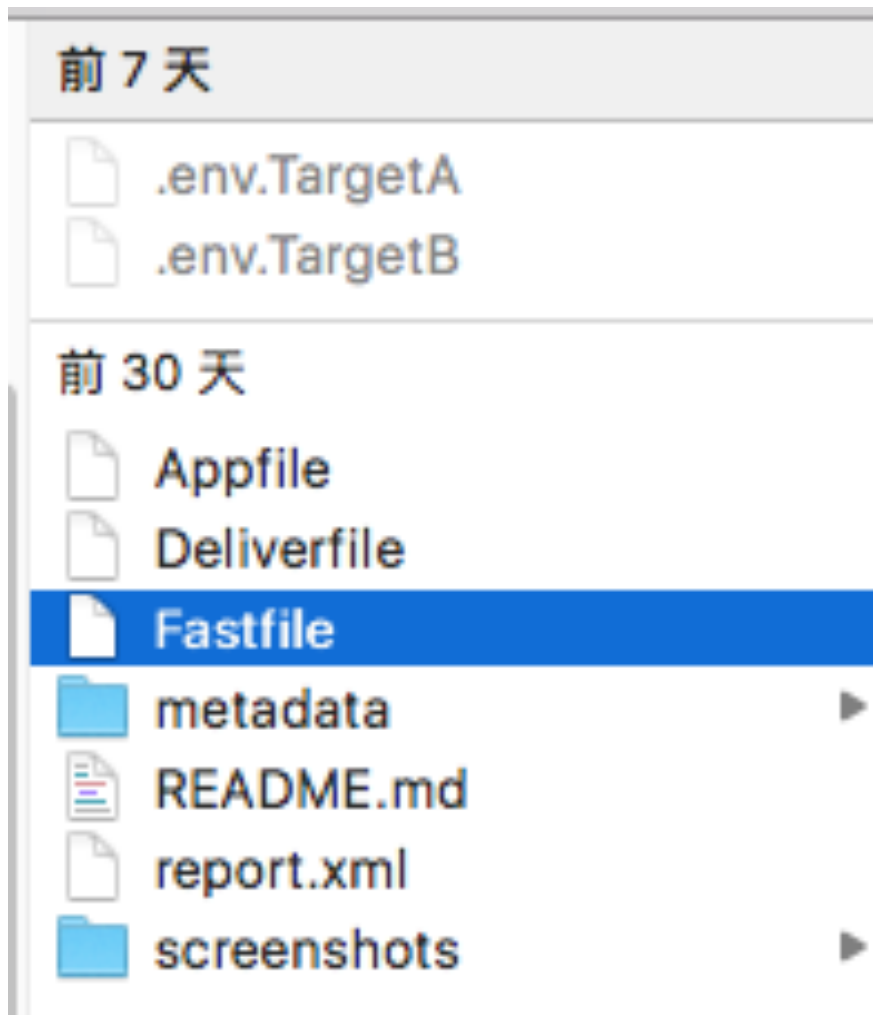
自动化脚本配置文件。是我们脚本的入口，所有的事件驱动都是在这个文件来调度的。

```
default_platform(:ios)  
  
platform :ios do  
  
  desc "demo upload_to_app_store"  
  lane :Archive_TargetA do |options|  
    scheme = options[:scheme]  
    date = Time.new.strftime("%Y%m%d-%h%M")  
  
    # export_method 支持 app-store, ad-hoc, package, enterprise,  
development  
    gym(  
      scheme: "#{scheme}",  
      output_name: "#{scheme}-#{date}.ipa",  
      clean: true,  
      export_method: 'app-store',  
    )  
  
    # upload_to_app_store  
    deliver # 当deliverfile为空的时候, 同 upload_to_app_store 作用一样  
  end  
end  
  
end
```

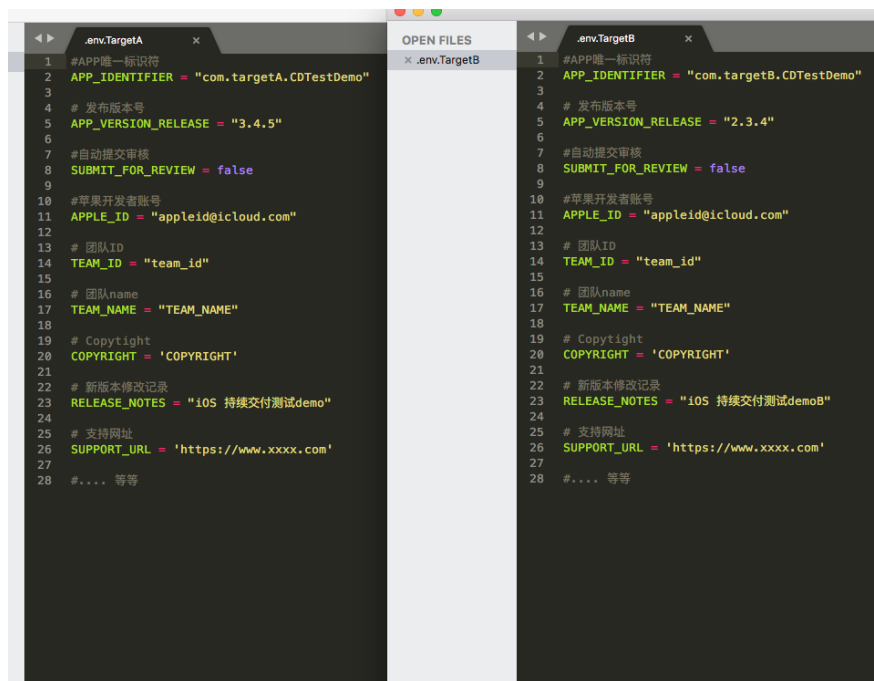
cd到项目根目录执行命令：`bundle exec fastlane Archive_TargetA scheme:"CDDemo"`，后面的 `scheme` 是带的参数。

Multi-Target

如果我们需要配置多个target进行打包的话，我们可以使用[环境变量](#)，来进行配置。假如我们现在有两个`target`，`targetA` 和 `targetB`，则我们需要创建两个`.env`文件，例如`.env.targetA`，`.env.targetB`，放在`Fastfile`文件同级目录下



在`.env`文件里面我们可以配置一些不同的内容（非公共），比如`app_identifier`，`release_notes`等等。截图如下：



在 `Appfile`, `Deliverfile`, `Fastfile` 等文件, 我们都可以直接使用 `.env` 文件里面的内容。

Appfile

```
# Appfile

#The bundle identifier of your app
app_identifier ENV['APP_IDENTIFIER']

# Your Apple email address
apple_id ENV['APPLE_ID']

# Developer Portal Team ID
team_id ENV['TEAM_ID']
```

Deliverfile, 请在设置 `release_nores`、`support_url`、`private_url` 等配置的时候, 采用 `hash` 的方式写。

```
# app_identifier
app_identifier ENV['APP_IDENTIFIER']

# 用户名,Apple ID电子邮件地址
username ENV['APPLE_ID']

# 团队ID
team_id ENV['TEAM_ID']

# 团队name
team_name ENV['TEAM_NAME']

# copyright
copyright ENV['COPYRIGHT']

# 关键字
keywords(
  'zh-Hans' => ENV['KEYWORDS'],
)

# 新版本修改记录
release_notes(
```

```

# 中国
'zh-Hans' => ENV['RELEASE_NOTES'],
# 澳大利亚
'en-au' => ENV['RELEASE_NOTES_AU'],
# 美国
'en-us' => ENV['RELEASE_NOTES_US']
)

# 支持网址
support_url(
  # 中国
  'zh-Hans' => ENV['SUPPORT_URL'],
  # 澳大利亚
  'en-au' => ENV['SUPPORT_URL_AU'],
  # 美国
  'en-us' => ENV['SUPPORT_URL_US']
)

# 隐私政策网址 国家代码 https://www.cnblogs.com/Mien/archive/2008/08/22/1273950.html
privacy_url(
  # 中国
  'zh-Hans' => ENV['PRIVACY_URL'],
  # 澳大利亚
  'en-au' => ENV['PRIVACY_URL_AU'],
  # 美国
  'en-us' => ENV['PRIVACY_URL_US']
)

# 上传完成后提交新版本进行审查
submit_for_review false

# 跳过HTML报告文件验证
force true

# 启用iTC的分阶段发布功能 灰度发布
phased_release true

# 应用审核小组的联系信息 app 审核信息
app_review_information(
  first_name: "xx",
  last_name: "xx",
  phone_number: "+86 18888888888",
  email_address: "xxxx",
  demo_user: "test1@test.com",
  demo_password: "test123"
)

...

```

Fastfile文件里面使用环境变量，跟上面略有不同。在Fastfile里面，我们需要告诉lane 要使用那个.env文件，这时候我们需要使用sh脚本命令的形式来调用一个lane 后面跟上--env 环境变量文件名，此时调用的lane 不能声明为private_lane，调用方式如下：

```

lane :releaseDemo2 do
  # 无参数
  sh "fastlane Archive_TargetA --env TargetA"

  # 有参数
  sh 'fastlane Archive_TargetA type:\'哈哈哈哈哈\' --env TargetA'
end

外部直接调用（带参数）
bundle exec fastlane Archive_TargetA type:"haha" --env TargetA

```

然后我们在 `Archive_TargetA` lane 里面使用 `ENV['xx']` 方式，读取出来的内容就是从 `.env.TargetA` 文件读取出来的。同理，`deliver` Action 对应的 `DeliverFile` 文件里面的内容也是从 `.env.TargetA` 文件读取出来的。

```
private_lane : Archive_TargetA do |options|
  scheme = ENV['SCHEME'] # 这时候读取出来的'scheme'就是'TargetA',
  从'.env.TargetA'读取出来的

  # export_method 支持 app-store, ad-hoc, package, enterprise, development
  gym(
    scheme: "#{scheme}",
    output_name: "#{scheme}.ipa",
    clean: true,
    export_method: 'app-store',
  )

  deliver # 这时候deliverfile里面读取的内容就是从'.env.TargetA'文件读取的
end
```

lane之间的调用

跟我们自己写方法调用一样，例如：

```
desc "打包统一入口"
lane :Archive do |options|
  # 如果我们传入的参数'type'是targetA, 那么我们就执行Archive_TargetA 这个lane。。。
  type = options[:type]
  if type == "TargetA"
    Archive_TargetA(options)
  elsif type == "TargetB"
    Archive_TargetB(options)
  else
    Archive_TargetA(options)
  end
end
```

系统级lane

`fastlane` 默认有 lane。

- `before_all`，就是在执行一次脚本之前首先执行的代码，我们可以在这里面执行一些公共的东西，比如 `git_pull`，`cocoapods`。

```
before all do
  # 检出到 Developer 分支
  sh 'git checkout Developer'
  git pull
  cocoapods(repo_update: true)
end
```

- `after_all`，成功结束之后，处理共有的后置逻辑。
- `before_each`，每次执行 lane 之前都会执行一次。
- `after_each`，每次执行 lane 之后都会执行一次。
- `error`，在执行上述情况任意环境报错都会中止并执行一次。

执行顺序

执行顺序	方法名	说明
1	before_all	在执行 lane 之前只执行一次。
2	before_each	每次执行 lane 之前都会执行一次。
3	lane	自定义的任务。
4	after_each	每次执行 lane 之后都会执行一次。
5	after_all	在执行 lane 成功结束之后执行一次。
6	error	在执行上述情况任意环境报错都会中止并执行一次。

Error

- 出现 `Command timed out after 10 seconds on try 1 of 4, trying again...`, 在 `fastlane` 文件开头加上:

```
ENV["FASTLANE_XCODEBUILD_SETTINGS_TIMEOUT"] = "180"  
ENV["FASTLANE_XCODE_LIST_TIMEOUT"] = "180"
```

插件

- `versioning`, 用来修改 `build` 版本号和 `version` 版本号。 `Fastlane` 内嵌的 `actionincrement_build_number` 使用的是苹果提供的 `agvtool`, `agvtool` 在更改 Build 的时候会改变所有 target 的版本号。这时如果你在一个工程里有多个产品的话, 每次编译, 所有的 Build 都要加1, 最后就不知道高到哪里去了。 `fversioning` 不仅可以指定 target 增加 Build, 而且可以按照「语义化版本」规范增加 Version, 当然也可以直接设定 Version。
- `firim`, 直接把 `AdHoc` 或者 `InHouse` 打包的 ipa 上传到 `fir.im`, 供测试下载。

自定义插件

插件安装格式

`fastlane add_plugin [name]`, 需要到项目根目录下执行。

`fastlane update_plugins` 插件更新, 同上, 需要 cd 到项目根目录下。

注意

- 保持打包机器的 Xcode 和 证书是最新的。
- 使用脚本命令形式调用的时候不能设置成 `private_lane`。
- `private_lane` 表示私有 lane, 使用 `bundle exec fastlane` 命令, 声明为 `private_lane` 的是不是显示出来的, 使用脚本命令形式调用的时候不能设置成 `private_lane`。

其它

可以直接在 `lane` 里面执行 git 命令, 例如 `sh 'git checkout Developer'`, 检出 `Developer` 分支。

由于本人的水平有限, 难免会有错误和疏漏, 也欢迎各位同学指正, 如果大家在 `Fastlane` 的使用上, 有更好的案例, 也欢迎交流和分享。

Demo

[CDDemo](#)

参考文章

- [老邢Thierry的fatlane实战系列](#)
- [iOS-持续交付](#)
- [Fastlane为iOS带来持续部署](#)
- [fastlaneTools](#)
- [fastlane文档](#)
- [Fastlane – 移动开发自动化之道](#)
- [Fastlane的黑魔法:一键打包编译上传 AppStore](#)
- [iOS中使用Fastlane实现自动化打包和发布](#)
- [小团队的自动化发布 – Fastlane带来的全自动化发布](#)
- [fastlane使用说明书](#)