How to build the perfect fastlane pipeline for iOS

APP DEVELOPMENT

fastlane iOS

Ishara Niroshana

MARCH 28, 2024

You've spent months building an app, and when it finally comes time to distribute it to the App Store, you realize it's not going to be an easy task. Even when you're just releasing some updates, you have to perform each step in the right order and to Apple's exacting specifications: code signing, creating the app and version in App Store Connect, running tests, archiving and uploading the build, and generating and setting various metadata and screenshots. This is where the app automation platform fastlane comes in.

fastlane is an open-source suite of tools that allows you to automate your iOS or Android mobile app releases, potentially saving you hours of development time. It is powered by a Ruby configuration file called a *Fastfile*, in which you can add *lanes* to serve different purposes.

In this tutorial, you'll see how to build a local fastlane pipeline that will automate the final steps of the iOS development and deployment process. You'll add lanes for signing, testing, building, and deploying a simple "to-do" list application. At the end of this tutorial, you should have everything you need to build the perfect fastlane pipeline for your next iOS app.

Setting up fastlane

There are many ways to install fastlane, but we'll use Ruby for the task. Excitingly, there's also a Swift version of fastlane that's currently in beta.

Confirm that you have Ruby installed on your machine by opening Terminal and

```
ruby -v
```

If Ruby is not installed, follow their instructions here to install it.

Next, set up the Xcode command-line tool (CLT). It can be enabled with the following command:

```
xcode-select --install
```

Now, you're ready to install fastlane. Run the following command to add the Ruby gem for the project:

```
sudo gem install -n /usr/local/bin fastlane --verbose
```

When the installation is complete, confirm it by running the following command:

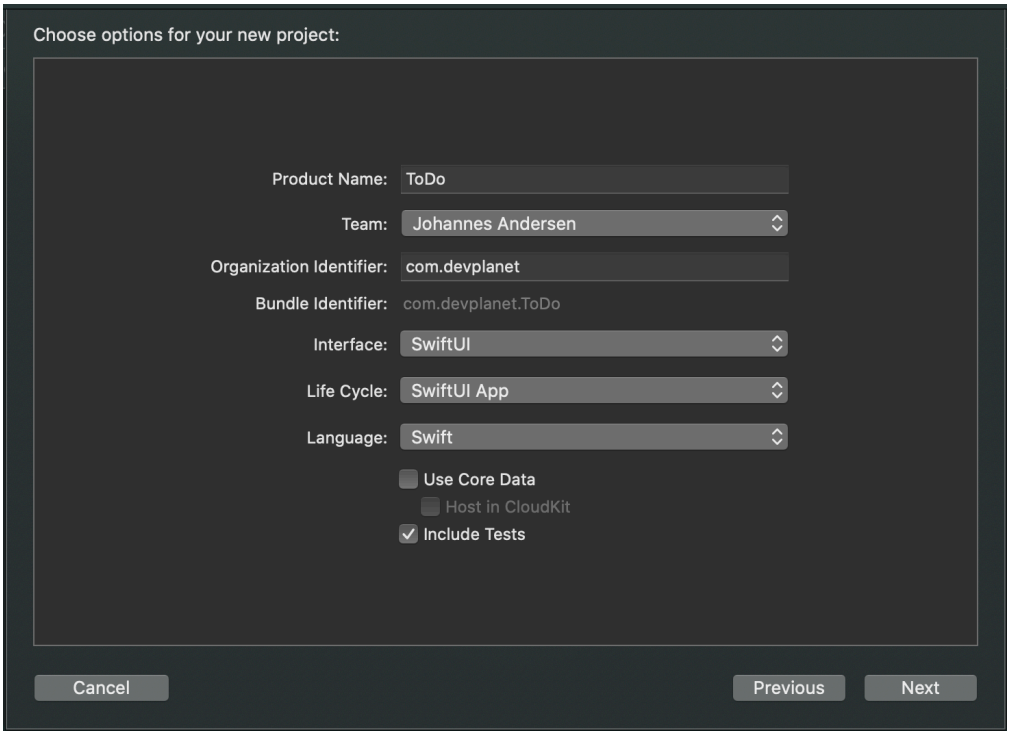
```
fastlane --version

# Output:
fastlane installation at path:
/usr/local/Cellar/fastlane/2.185.0/libexec/gems/fastlane-2.185.0/bin/fastlane
------------------------------
[✔] 🚀
fastlane 2.185.0
```

```
Congratulations! You're ready to use fastlane in a new project.
```


Configuring fastlane for a project

For this tutorial, we'll use a demo app called *ToDo*. You can find the finished source

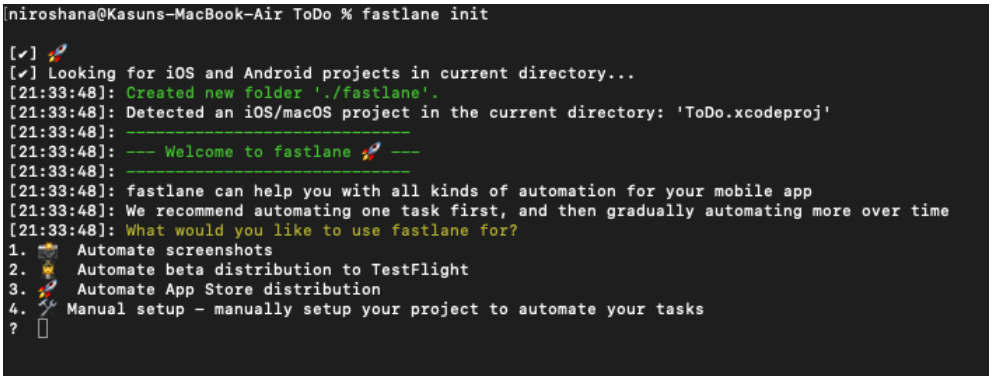
To get your new app started, open Xcode and create a new app.



To initialize fastlane inside the project, go to the project's root directory in your terminal and run the following:

```
fastlane init
```

fastlane will ask you to choose a single automated action to implement. Automated actions are pre-built actions that let you automate various aspects of your development and release workflow. You'll implement multiple automated actions in this tutorial, so just select manual setup by entering **4**.



Go to the root directory of the project. You will see a new Gemfile, which includes project dependencies, and a `./fastlane` directory containing an Appfile and a Fastfile.

- **Appfile** - contains a bundle identifier and your Apple ID.
- **Fastfile** - contains the `fastlane.tools` configuration and actions.

For this tutorial, we'll be using an App Store Connect API Key to authenticate, so you

Uncomment the app_identifier line, and enter your app's bundle identifier like so:

```
app_identifier("com.my-app.myapp")
```

You can later easily access your app identifier in any lanes:

```
lane :my_lane do
      app_identifier =
CredentialsManager::AppfileConfig.try_fetch_value(:app_identifier)
end
```

Authentication and the App Store Connect API key

fastlane offers a few different ways to authenticate against your App Store Connect account. Now that Apple provides an official, public API to interact with App Store Connect, the preferred approach involves authenticating against this API using an App Store Connect API key. Note that some fastlane actions are not yet supported by the official API, but most common actions are. (If you do find yourself needing to authenticate using one of fastlane's other methods, note that Apple now enforces two factor authentication [2FA] for all accounts, and this introduces complications and flakiness, especially if you're running your fastlane script from a bot or on a remote machine.)

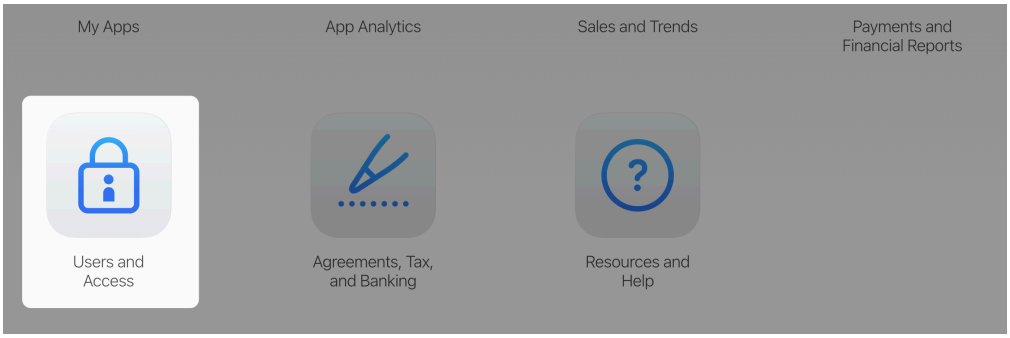
In order to use the *App Store Connect API*, fastlane requires the following:

- App Store Connect API key filepath or content
- Issuer ID
- Key ID

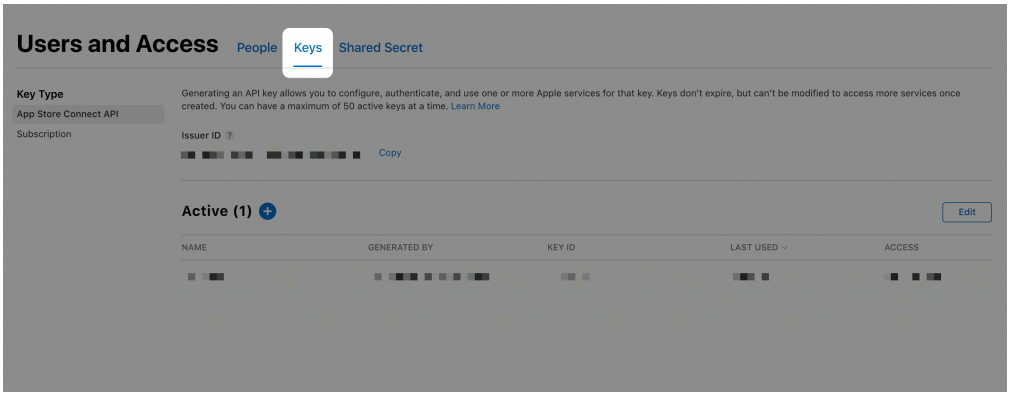
Creating an App Store Connect API key

To create a key, you need to have Admin permissions in your App Store Connect account. Here are the steps you need to follow:

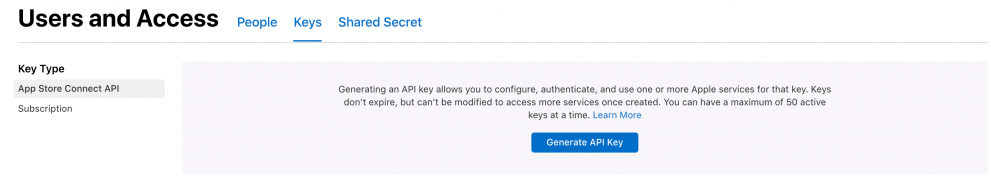
1. Log in to App Store Connect
2. Navigate to **Users and Access**



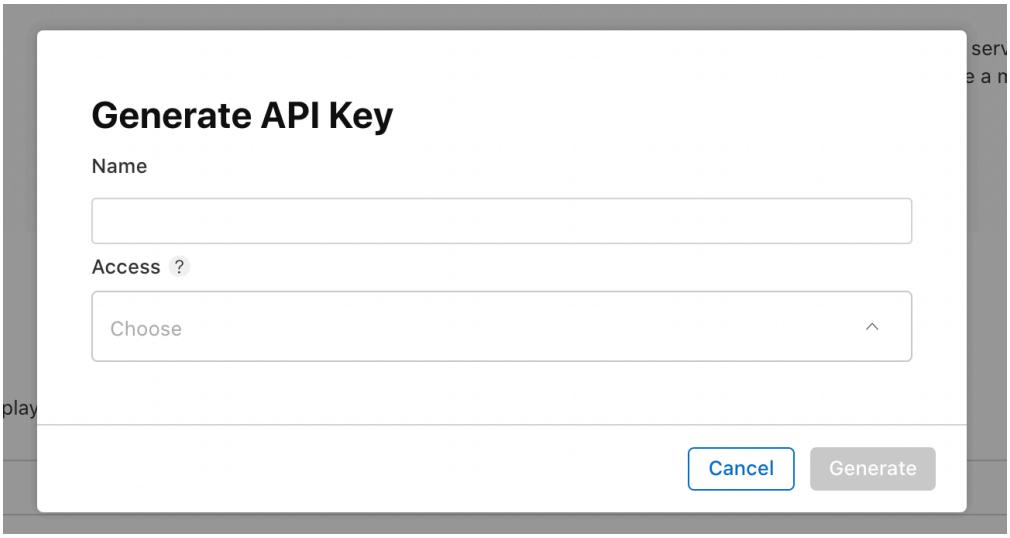
3. Select the **Keys** tab



4. Click **Generate API Key** button

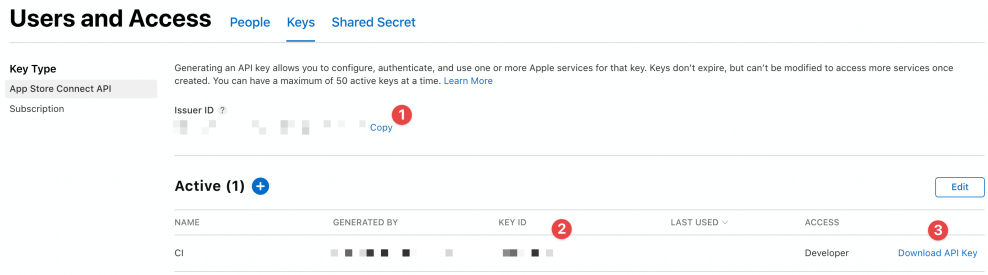


5. Enter a name for the key.



6. Select at least one role. Remember to provide the minimum level of access needed.

7. Click **Generate**



The *API Key* will be in `.p8` format. You might consider using a **base64-encoded** version to avoid running into this known issue.

To encode your key, run the command below:

```
cat [YOUR_KEY_NAME].p8 | base64
```

Now you have everything you need to authenticate against the App Store Connect API. Next, we'll use theAPI key you generated to run a fastlane lane.

Using App Store Connect API key

fastlane uses the `app_store_connect_api_key` action to authenticate. You'll pass in your *key ID*, *issuer ID* and *key filepath* as environment variables or direct values.

You can add the following call inside a lane to authenticate before performing some actions:

```
app_store_connect_api_key(
   key_id: "D83848D23",
   issuer_id: "227b0bbf-ada8-458c-9d62-3d8022b7d07f",
   key_content: "ABC123" // base64 encoded key,
   is_key_content_base64: true,
   in_house: false #boolean value if team is Enterprise or not
)
```

You can also use default *environment* variables that automatically load in if set, avoiding the need to provide the values as arguments. Here are some of the keys you might use:

- key_id: APP_STORE_CONNECT_API_KEY_KEY_ID
- issuer_id: APP_STORE_CONNECT_API_KEY_ISSUER_ID
- key_content: APP_STORE_CONNECT_API_KEY_KEY
- in_house: APP_STORE_CONNECT_API_KEY_IN_HOUSE

If you add everything as environment variables in your CLI, your call to `app_store_connect_api_key` can be as simple as:

`Actions.lane_context[SharedValues::APP_STORE_CONNECT_API_KEY]` , which other actions can easily use. That way, you don't have to provide your API key to each individual action.

For example:

```
lane :build_app do |options|
  api_key = lane_context[SharedValues::APP_STORE_CONNECT_API_KEY]

  gym(
    api_key: api_key,
  )
end
```

Adding a fastlane lane

A *lane* is a workflow of sequential tasks. Each lane has a description and name you can use to execute it.

In this tutorial, we will create lanes for:

1. Profiles & Certificate handling
2. Building the app
3. Uploading the app to TestFlight
4. Automating screenshots
5. Releasing the app

While these five lanes don't cover everything you might want to manage and automate in fastlane, they form a solid foundational fastlane setup for you to start with.

Code signing with match

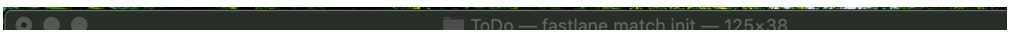
Code signing is mandatory on iOS when distributing your app. It assures that your app's code can be trusted and hasn't been modified by a third party since it was last signed.

fastlane match gives you a secure way to easily share certificates across your development team. It will keep all required certificates & provisioning profiles in encrypted storage (e.g., private git repository, Google Cloud, or Amazon S3). For this tutorial, we'll show you how to use a private git repository on GitHub.

Open your terminal and run:

```
fastlane match init
```

Select your desired storage type and enter the URL:

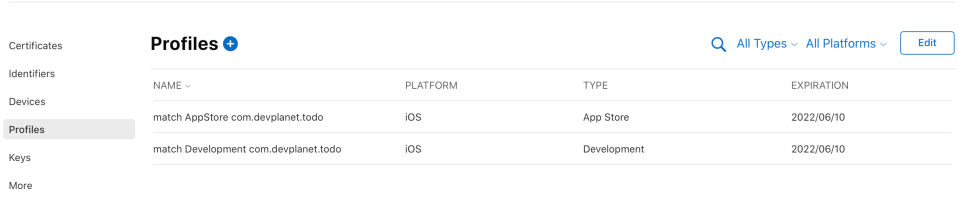


```
? 1
[23:25:45]: Please create a new, private git repository to store the certificates and profiles there
[23:25:45]: URL of the Git Repo:
```

Once you proceed with a password, certificates will be handled by fastlane match. To create profiles for development and App Store, you can execute the following commands:

```
fastlane match development
fastlane match appstore
```

Now, check out the Apple Developer Portal. You will see the profiles have been created there:



Note that, with code signing managed by fastlane match, you need to disable **automatic code signing** in your Xcode project.

You can also add a lane to sync certificates on your machine. Open your Fastfile and add the following:

```
desc "Sync certificates"
 lane :sync_certificates do
   #read-only disables match from overriding the existing certificates.
   match({readonly: true, type: "appstore"})
 end
```

There are four different types of profiles you can use for the `type` attribute:

1. **App Store profile** - Used for distributing a production app to the App Store.
2. **Development profile** - Used to install an app in *debug* mode.
3. **Enterprise/in-house distribution profile** - Used for distributing apps to non-registered devices outside of the App Store; only available with enterprise developer accounts.
4. **Ad-hoc profile**: Used to distribute an app to devices registered in the developer account.

The values to pass for each of these profiles are `appstore`, `development`, `enterprise`, and `adhoc`, respectively. The specific one you should use depends on your needs and audience, so check out fastlane's match documentation for more detail.

Building an .ipa file with gym

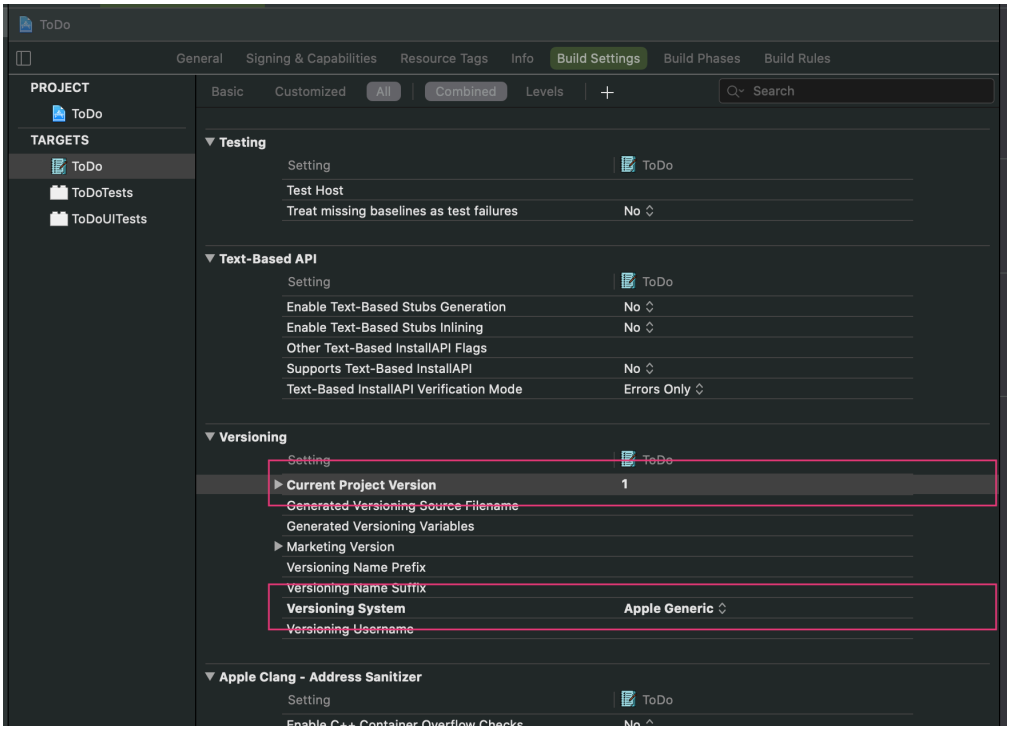
Archiving and building your app is time consuming, but with fastlane's gym action, you

```
fastlane gym init
```

fastlane will create a *Gymfile* for you. Open the Gymfile and add the following:

```
# App scheme name
scheme("ToDo")
#provide provisioning profiles to use
export_options({
   method: "app-store",
   provisioningProfiles: {
     "com.devplanet.todo" => "match AppStore com.devplanet.todo",
        }
})
# Specify the path to store .ipa file
output_directory("./fastlane/builds")
# Excludes bitcode from the build
include_bitcode(false)
# Excludes symbols from the build.
include_symbols(false)
```

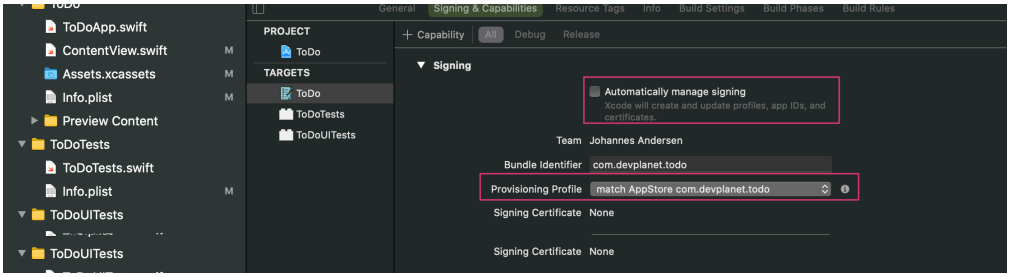
When building your app for TestFlight or the App Store, you must increment the build number each time. To automate this, enable Apple Generic Versioning by changing the app versioning settings:



Open the Fastfile and add the following lane to create an `.ipa` :

```
desc "Create ipa"
 lane :build do
   #update profiles

   sync_certificates
```

Before building the app, disable automatic code signing and select the correct **Provisioning profile** in Xcode:



Note that you can also do this using the `update_code_signing_settings` fastlane action.

Now you're ready to run `fastlane build` in your terminal.

Once you get the message that the build is completed, you can find the `.ipa` file in the `fastlane/builds` directory.

Uploading to TestFlight with pilot

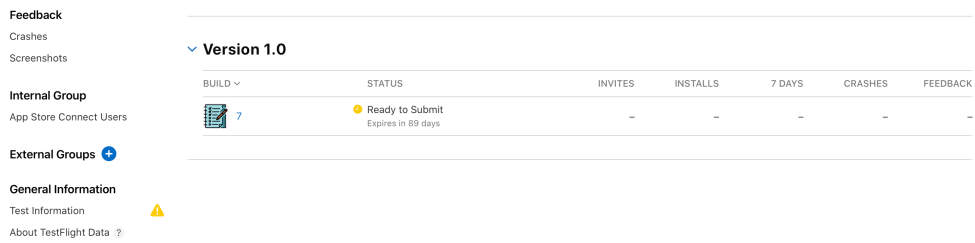
fastlane interacts with TestFlight through its pilot action (a.k.a. upload*to*testflight).

As you already have a lane to build an `.ipa` file, you just need to add the `pilot` command to your Fastfile to upload your build:

```
desc "Upload to TestFlight"
 lane :beta do
   build
   pilot
 end
end
```

If you need to upload a specific `.ipa` file to Testflight, remove `build` from the lane and add `ipa("./fastlane/builds/ToDo.ipa")`. This will upload the `.ipa` at the specified filepath.

Once fastlane finishes running this lane, you will see the build available in TestFlight:



Add a new tester to the app using the following command:

```
fastlane pilot add email@invite.com -g group-1,group-2
```

There are many other commands for pilot which you can explore in fastlane docs.

Adding screenshots

Screenshots are a big part of your app's App Store presence, and creating and updating these assets is an important part of the release process. Capturing real screenshots on a live, running app takes a lot of effort, especially as you need to generate different versions for different screen sizes and, often, languages. Let's look at how you can automate this with fastlane.

In order to automate screenshot capture, you'll need to set up some UI tests first. To easily create UI tests, you can record steps using Xcode and the simulator, automatically generating the required test method code in the process. The details are beyond the scope of this tutorial, so check out this article for more information.

When your UI tests are ready, run the following command:

```
fastlane snapshot init
```

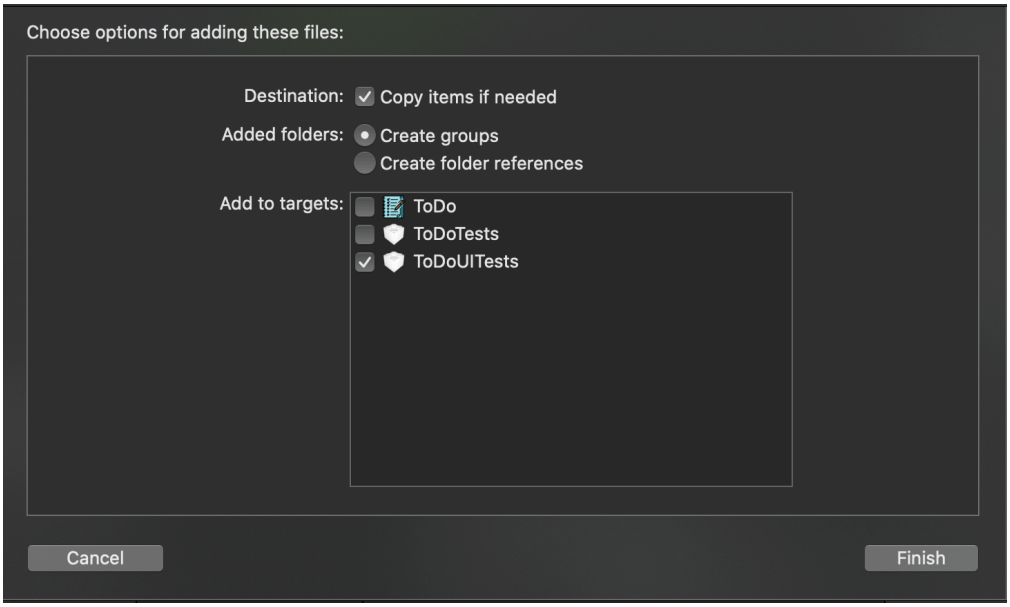
Once the command is executed, fastlane will show you the steps for configuring `snapshot`. Go to the newly-created `Snapfile` inside the `./fastlane` directory and configure it as shown below, uncommenting any relevant options as needed:

```
# A list of devices you want to take the screenshots from
devices([
 "iPad (8th generation)",
 "iPad Air (4th generation)",
 "iPad Pro (11-inch)",
 "iPad Pro (12.9-inch)",
 "iPad Pro (9.7-inch)",
 "iPhone 12",
 "iPhone 12 Mini",
 "iPhone 12 Pro",
 "iPhone 12 Pro Max",
])

languages([
```

```
# The name of the scheme which contains the UI tests
scheme("ToDoUITests")

# Where should the resulting screenshots be stored?
output_directory("./screenshots")

# Uncomment this to clear all previously generated screenshots before creating new ones
# clear_previous_screenshots(true)
```

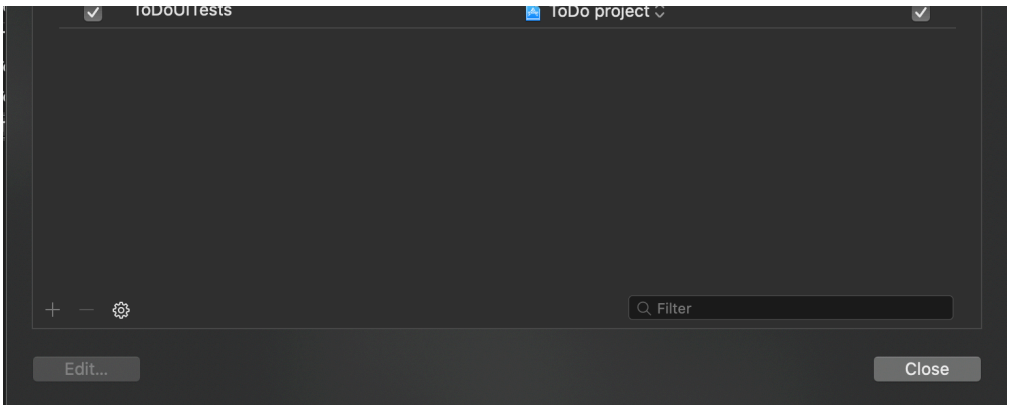
Now open Xcode and drag and drop the `SnapshotHelper.swift` file into your UI test directory. Choose the options as seen below:

- Copy items if needed
- Create groups
- ToDoUITests

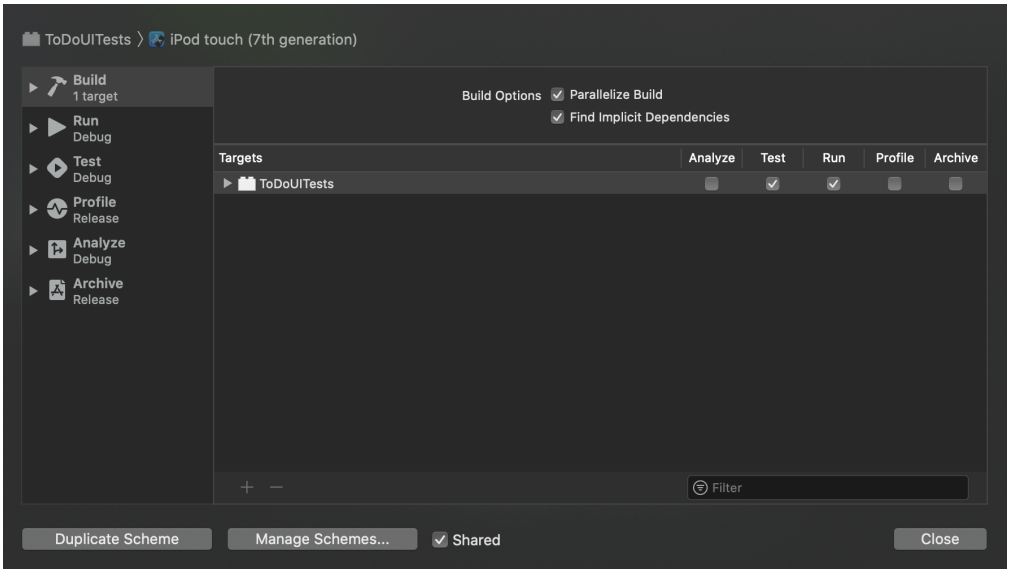


Open your UI test file (called `ToDoUITests.swift` in the example application), and add the call `setupSnapshot(app)` before `app.launch()`. Whenever you want to capture a screenshot, add a call to `snapshot("[SCREENSHOT_NAME]")`.

Add a UITest target as an Xcode scheme by opening **Product>Scheme>Manage Schemes**. Inside **Manage Schemes** add your UITest target:



Make sure to enable the *shared* property. Select the UI Test scheme and click **Build** on the scheme editor. Then select the **Test** and **Run** options and close the window.

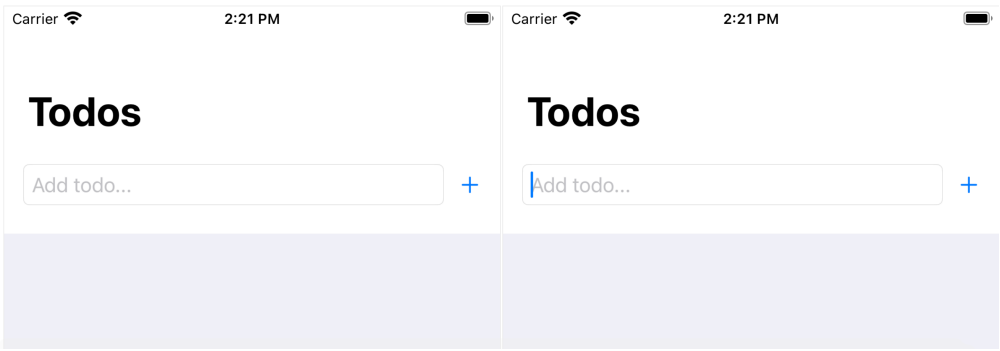


Now your Xcode configuration is complete, so you can move on to adding another lane to your Fastfile.

This lane uses **snapshot** to take screenshots as per your Snapfile's settings:

```
desc "Take screenshots"
 lane :screenshot do
   snapshot
 end
```

Run the `fastlane screenshot` command in your terminal and fastlane will automatically open a preview of your screenshots.



Deployment with fastlane deliver

At this point, you've used fastlane to create an app, build your app and generate an .ipa file, and capture screenshots, so now it's time to get all this information uploaded and submitted to the App Store. fastlane's deliver action will automate this process for you.

Go to the root directory of your project in your terminal and enter:

```
fastlane deliver
```

Confirm that you'd like to set up and pick Swift or Ruby for your fastlane configuration. We prefer Ruby because **fastlane.swift** is still in beta.

Once the *deliverfile* generation is completed, you will see one new file and two new directories inside the `./fastlane` directory:

- **metadata** - directory containing App Store metadata.
- **screenshots** - directory containing the app screenshots.
- **deliverfile** - file configuring your release and allowing you to set additional values required for App Store release, like pricing tier and export compliance responses.

You will find some text files inside the fastlane directory which are named after their corresponding fields in the App Store (e.g. *description*, *keywords*, *categories*, etc). fastlane will use these files to populate your app's metadata in App Store Connect.

Additionally, you can provide an application rating file that gives Apple the information it needs to calculate your app's Age Rating. You'll need to create a JSON file inside the metadata directory named `app_store_rating_config.json` and include the following:

```
{
 "CARTOON_FANTASY_VIOLENCE": 0,
 "REALISTIC_VIOLENCE": 0,
 "PROLONGED_GRAPHIC_SADISTIC_REALISTIC_VIOLENCE": 0,
 "PROFANITY_CRUDE_HUMOR": 0,
 "MATURE_SUGGESTIVE": 0,

 "HORROR": 0,
```

```
 "UNRESTRICTED_WEB_ACCESS": 0,
 "GAMBLING_CONTESTS": 0
}
```

You're now ready to upload to the App Store (and, optionally, submit your app for review!). Modify the deliverfile as seen below:

```
# Indicates that it's a free app.
price_tier(0)
# Answer the questions Apple would present to you upon manually submitting for review
submission_information({
   export_compliance_encryption_updated: false,
   export_compliance_uses_encryption: false,
   content_rights_contains_third_party_content: false,
   add_id_info_uses_idfa: false
})
# 3
app_rating_config_path("./fastlane/metadata/app_store_rating_config.json")
# file location
ipa("./fastlane/builds/ToDo.ipa")
# option to automatically submit the app for review (turned off here)
submit_for_review(false)
# 6
automatic_release(false)
```

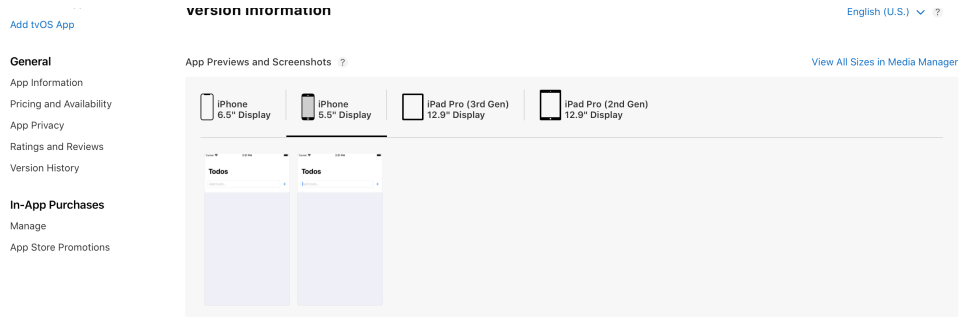
And add the following new lane to your Fastfile:

```
desc "Upload to App Store"
 lane :upload do
   deliver
 end
```

Finally, execute the fastlane upload command:

```
fastlane upload
```

Fastlane will verify the upload information by loading an HTML preview in your browser. Once the lane runs, visit App Store Connect and you will see the build is uploaded and ready to submit for review.



You can also combine all the lanes we created. This single command in your Fastfile will create the app, take screenshots, build, and upload the app to App Store Connect:

```
desc "Create app, screenshot, build and upload"
 lane :release_app do
   create_app
   build
   screenshot
   deliver
 end
```

Now run `fastlane release_app`, and fastlane will take care of everything.

You can check your overall fastlane setup by running `fastlane` in your terminal. You'll see a list of available lanes so you can run them individually as needed.

```
[21:16:49]: Welcome to fastlane! Here's what your app is set up to do:
+---------+-------------------+-------------------------+
|              Available lanes to run                   |
+---------+-------------------+-------------------------+
| Number  | Lane Name         | Description             |
+---------+-------------------+-------------------------+
| 1       | ios create_app    | Create app on Apple     |
|         |                   | Developer and App       |
|         |                   | Store Connect           |
| 2       | ios sync_profiles | Sync certificates       |
| 3       | ios build         | Create ipa              |
| 4       | ios beta          | Upload to TestFlight    |
| 5       | ios screenshot    | Take screenshots        |
| 6       | ios upload        | Upload to App Store     |
| 7       | ios release_app   | Create app,             |
|         |                   | screenshot ,build and   |
|         |                   | upload                  |
| 0       | cancel            | No selection, exit      |
|         |                   | fastlane!               |
+---------+-------------------+-------------------------+
```

Conclusion

Automating your iOS deployment and release process can save you a lot of time and headaches, and it becomes essential when you're developing and releasing iOS apps at scale. In this tutorial, you took a big step in that direction by building out an automated iOS deployment pipeline using **fastlane**. It may take some work to set up, but will save you hours of time building, testing, and releasing your app on a regular basis.

Looking for a better way to distribute all your different flavors of builds, from one-offs to nightlies to RCs?

Give Build Distro a try! Sign up for Runway and see it in action for yourself.

GET STARTED

RELATED POSTS

How to set up and run a beta testing program for your mobile app

Richard Huffaker

SEPTEMBER 18, 2023

Automate away iOS code signing renewal pain with fastlane match

Jared Sorge

JUNE 21, 2022

Runway's next big step, and the funding to take us there

NOVEMBER 16, 2021

SIGN UP FOR THE FLIGHT DECK, OUR MONTHLY NEWSLETTER

Email SIGN UP

TRUSTED BY THE BEST MOBILE TEAMS