

# Better fastlane with environments

## APP DEVELOPMENT

fastlane Android iOS



Jared Sorge

MARCH 28, 2024

fastlane is awesome. It can save oodles of time on those repetitive tasks that tend to fill your weekly iOS routine: making builds for TestFlight and the App Store, running unit tests, and capturing marketing screenshots. If fastlane is new to you, be sure to check out this [primer](#) on the Runway blog to get started.

If you're a small team working with one or two configurations of a single app, you'll likely have a single Fastfile that uses a handful of basic, out-of-the-box fastlane lanes. But as your team grows and your needs expand beyond the basic setup to include enterprise test apps, multiple platforms, and even more build configurations, you'll quickly find that managing lanes in a single (or multiple!) Fastfiles can quickly become unwieldy, verbose, and difficult to maintain.

That's where **fastlane environments** come in. Fastlane environments provide a way to customize our fastlane lanes to work with many different configurations and allow for easily swapping between them all. This can be a really helpful technique if you find that your Fastfiles have lanes that do basically the same things, just with some values changed for the particular app or configuration that the lane is handling. fastlane environments can help you narrow down the number of lanes you have to manage and keep those lanes flexible to work with any of your apps.

Taking advantage of fastlane environments is a great way to reduce code duplication in your setup, and it will make your fastlane scripting more understandable, extensible, and robust as your team's needs grow.

Typical fastlane configuration via \*files

result. These files are one place to define configurations for the actions that are later called by our lanes. For example, in our earlier fastlane post we defined the app's identifier in the `Appfile` and used that in a lane as follows:

```
app_identifier = CredentialsManager::AppfileConfig.try_fetch_value(:app_identifier)
```

It's great that we have the ability to use this to get at our app's identifier! But what if we didn't need to have bespoke `*file` files for each use case and could instead use different configuration files that can be swapped in as needed?

## Enter fastlane environments

It turns out that we can do just this! Using environment configuration files we can unleash extra flexibility by declaring variables that fastlane can inject into the environment environment [using `dotenv`](#), a package that lets us use key-value pairs to customize what our lanes do. Here's an example:

```
APP_IDENTIFIER = com.my-app.myapp
APP_NAME = MyApp
```

In our Fastfile we can access the environment using the `ENV` variable, which is a dictionary available to all of our lanes. So in this example, to read the app's identifier, we can call `ENV["APP_IDENTIFIER"]` in our lanes and use it how we like. This is great, but it gets even better.

We store these key-value pairs in files next to our Fastfile. The standard file is simply called `.env` and fastlane will load it for you automatically (you could also name it `.env.default` and fastlane will still pick it up). But you can also define your own custom `dotenv` files and import them as part of your lanes!

```
// .env.mac
MATCH_PLATFORM = macos
DELIVER_PLATFORM = osx

// .env.ios
MATCH_PLATFORM = ios
DELIVER_PLATFORM = ios

// Fastfile:
platform :ios do
  before_all do
    Dotenv.load ".env.ios"
  end

  lane: beta do
    match
    deliver
  end
end
```

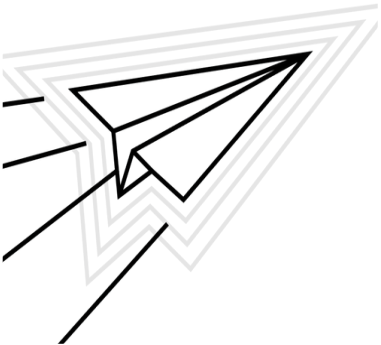
```
Dotenv.load ".env.mac"
end

lane: beta do
  match
  deliver
end
end
```

What we have here are two separate environment files ( `.env.ios` , and `.env.mac` ) which are loaded at the beginning of any lane for their respective platform. In the case of the `match` and `deliver` actions above they will always be properly configured for their correct platforms. This is super cool!

There are two main functions that we use when loading these environment files:

`Dotenv.load` and `Dotenv.overload` . The difference between the two is that `overload` will overwrite any existing values in the environment from the files being loaded. `load` is purely additive; it does not overwrite any values. So this means that you could have some base value inside of the `.env` file and then overwrite that with a value later if you want to.



### Sign up for the Flight Deck — our monthly newsletter.

We'll share our perspectives on the mobile landscape, peeks into how other mobile teams and developers get things done, technical guides to optimizing your app for performance, and more. ([See a recent issue here](#))

SIGN UP

## Next-level fastlane with environments

Let's play this out a little further. We can now see how using environments will let us swap out values at runtime and allow us to build generic lanes that can be customized by their environment. Let's say that we have an app which has different builds for beta and releases, but those processes are very similar. Here's what those lanes could look like (and notice all the overlap in the two).

```
ruby
lane :beta do

  match(app_identifier: "com.myapp.ios.beta")
```

[Product](#)[Integrations](#)[Automations](#)[Customers](#)[Pricing](#)[LOG IN](#)[CONTACT SALES](#)[G](#)

```
end

lane :release do
  match(app_identifier: "com.myapp.ios.release")
  gym(
    project: "MyProject.xcodeproj",
    scheme: "MyApp-Release"
  )
  pilot(app_identifier: "com.myapp.ios.release")
  deliver(app_identifier: "com.myapp.ios.release")
end
```

Typical fastlane configuration via  
\*files

Enter fastlane environments

Now let's extract out all the variables to environment files and see how this lets us reduce down to a single lane. To detect if we are building for release all we have to do is check if the variables for the `deliver` action are set or not (since they'll never be set in a beta release).

Next-level fastlane with  
environments

fastlane environment gotchas

Set up for automation success using  
fastlane environments

```
// .env.beta
MATCH_APP_IDENTIFIER = com.myapp.ios.beta
GYM_PROJECT = MyProject.xcodeproj
GYM_SCHEME = MyApp-Beta
PILOT_APP_IDENTIFIER = com.myapp.ios.beta
DELIVER_APP_IDENTIFIER = com.myapp.ios.beta

// .env.release
MATCH_APP_IDENTIFIER = com.myapp.ios.release
GYM_PROJECT = MyProject.xcodeproj
GYM_SCHEME = MyApp-Release
PILOT_APP_IDENTIFIER = com.myapp.ios.release
DELIVER_APP_IDENTIFIER = com.myapp.ios.release

// Fastfile
lane :build_binary do
  match
  gym
  upload_to_testflight
  if ENV["DELIVER_APP_IDENTIFIER"]
    deliver
  end
end
```

To build our lane with a given release we'll use this invocation:

```
bundle exec fastlane build_binary -env beta
```

(It's highly recommended that fastlane is installed via `bundle`, which is where `bundle exec` comes in at the start. Otherwise, if fastlane is in your `$PATH`, you can drop that.)

The end state here is that our Fastfile doesn't know anything in particular about our apps. That's all controlled by the environments that fastlane is told about. This makes our existing fastlane functionality endlessly extensible — adding new apps to our pipeline is as simple as creating a new `.env` file for each new app and filling in the necessary values. Beyond that, extracting all the app and configuration specific code from the lanes naturally makes their logic flow easier to read and reason about and, as a result, everything becomes more maintainable for engineers on the team.

## fastlane environment gotchas

Nothing is perfect, and this technique for using fastlane environments is no exception. There are a few things to keep in mind when using environments:

- It's possible that some actions may not always read the argument you're trying to pass in from the environment\*. This is the case with snapshot's `devices` and `languages` arguments [discussion about that topic](#). If you're struggling with missing environment variables then check out the docs for that particular action and don't be afraid to go spelunking for the source code if need be.  
*\*It's helpful to read up on [the docs for how fastlane processes action arguments](https://docs.fastlane.tools/advanced/fastlane/#priorities-of-parameters-and-options).*
- Environment files can't contain source code, and values can't reference other values. This will cause failures in really weird ways such as entire environment files failing to import. Programmers like their code DRY, but sometimes out in the real world things get a bit wet.

## Set up for automation success using fastlane environments

We've seen how fastlane environments can be a powerful tool to take advantage of, helping keep your Fastfiles generic and easier to maintain, and allowing you to nimbly swap out different configurations as needed. By leveraging fastlane environments to keep your setup reusable, maintainable, and expandable, you can set your team up for success as its needs for automation using fastlane continue to grow.

Cheers!

P.S. For further reading about fastlane environments, check out [their documentation](#).


Release better with Runway.

Runway integrates with all the tools you’re already using to level-up your release coordination and automation, from kickoff to release to rollout. No more cat-herding, spreadsheets, or steady drip of manual busywork.

CONTACT SALESGET STARTED


RELATED POSTS

WWDC 2021 highlights




Alessandro Martin  
JUNE 10, 2021

How — and why — to think about mobile app versioning



Bruno Rocha  
APRIL 28, 2022

Guide to the App Store Connect API: Calculate your iOS app's average user rating for each version



Jared Sorge  
FEBRUARY 28, 2024

SIGN UP FOR THE FLIGHT DECK, OUR MONTHLY NEWSLETTER

Email

SIGN UP

TRUSTED BY THE BEST MOBILE TEAMS



Product	Integrations	Resources
Mobile release management	Project management	Blog
End-to-end automation	• Jira	FAQ
Rollouts	• Linear	Documentation
Mobile insights	• Pivotal Tracker	Quickstart CI/CD
Build Distro	Version control	App review times
Automations	CI/CD	App Store Connect status page
Security	App stores	App hotfix leaderboard
Pricing	• App Store Connect	
What's new	• Google Play Console	
Explore sandbox	Slack	Company
	Monitoring	About us
	All integrations	Contact
Use cases		Terms of service
fastlane		Privacy policy
Release trains		Careers
Mobile DevOps		Status

