CocoaHeads Aachen

From creation to deployment [sans the development part]

Introduction to **fastlane.tools**

Tobias Arends

11/24/2016

@1b0t

Rocky Road Ahead

New app project approved! - Client wants beta by tomorrow :)

- “Let’s quickly set this up,…but wait.”
  - Dev Portal
  - iTunes Connect
  - Xcode

- Especially if you don’t do this every day this can take a lot of time.
- Many shops add 1-2 days of “development time” for initial setup and beta distribution.

Rocky Road Ahead (alternative)

New app project approved! - Client wants beta by tomorrow…

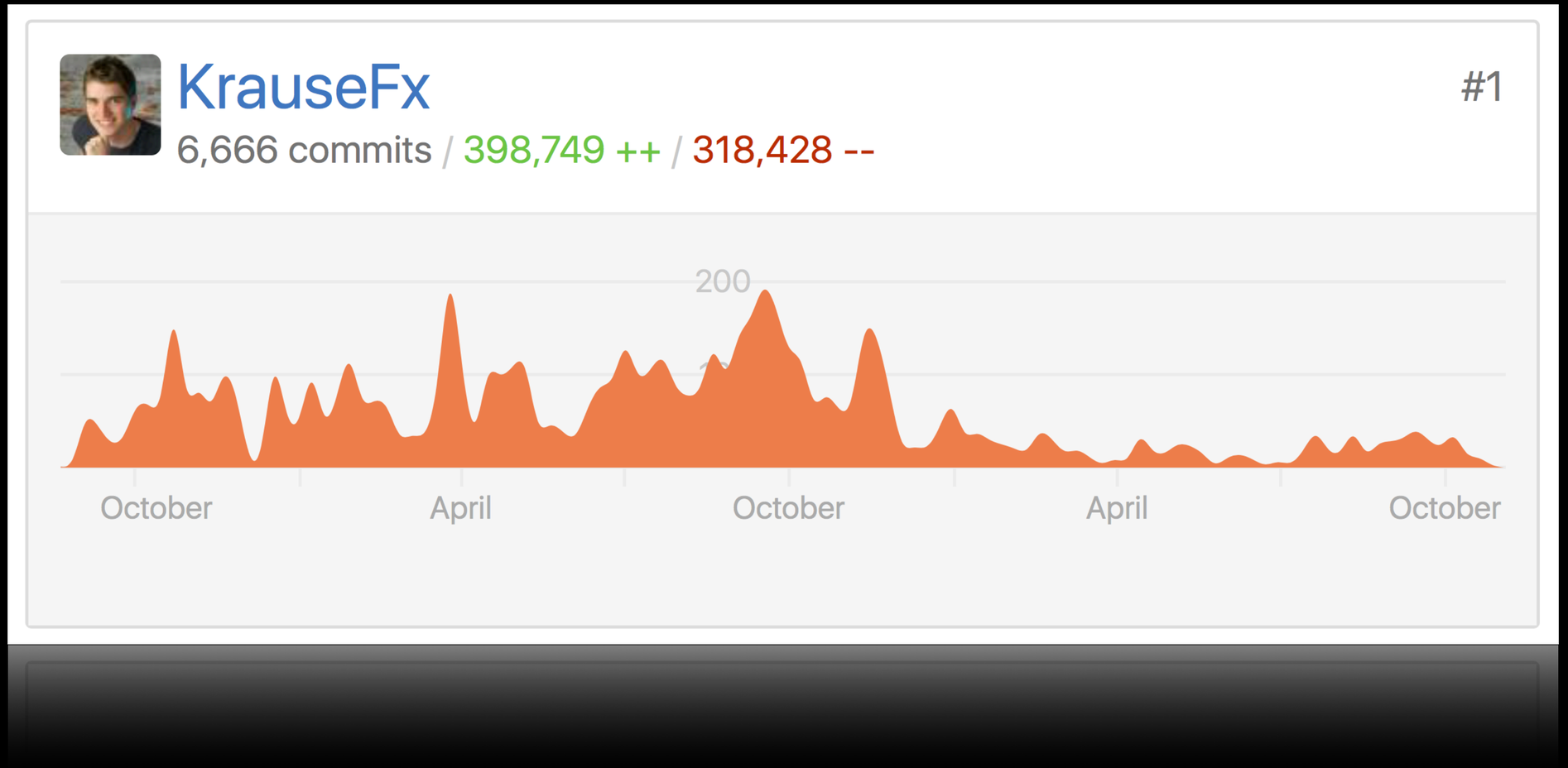
“Damn it! Mike is on vacation.”

fastlane.tools?

- Open Source [MIT]
- Initial commit: Sep. 9th 2014 by Felix Krause
- very well maintained
  - 12259 commits to date* | avg. ~14.5 commits/day
  - uses (partly) non-public APIs
    - iTC/dev portal API changes frequently adopted within hours
  - now part of fabric.io (twitter)

Felix Krause
@KrauseFx

** stand 11/21/2016*


KrauseFx
#1
6,666 commits / 398,749 ++ / 318,428 --
200
October
April
October
April
October

tools

scan

produce

frameit

deliver

sigh

gym

snapshot

pem

cert

match

Spaceship

SpeedyHeads 🚀

---

create…

produce

provision & sign…

match

test…

pilot

deliver…

snapshot

deliver

Let’s create an app

the typical way

- Log into Dev Center
  - Certificates & …
  - create App ID
    - bundle identifier

- Log into iTunes Connect
  - My Apps
  - create App
    - select App ID
    - specify App name

Let’s create an app

the fast way

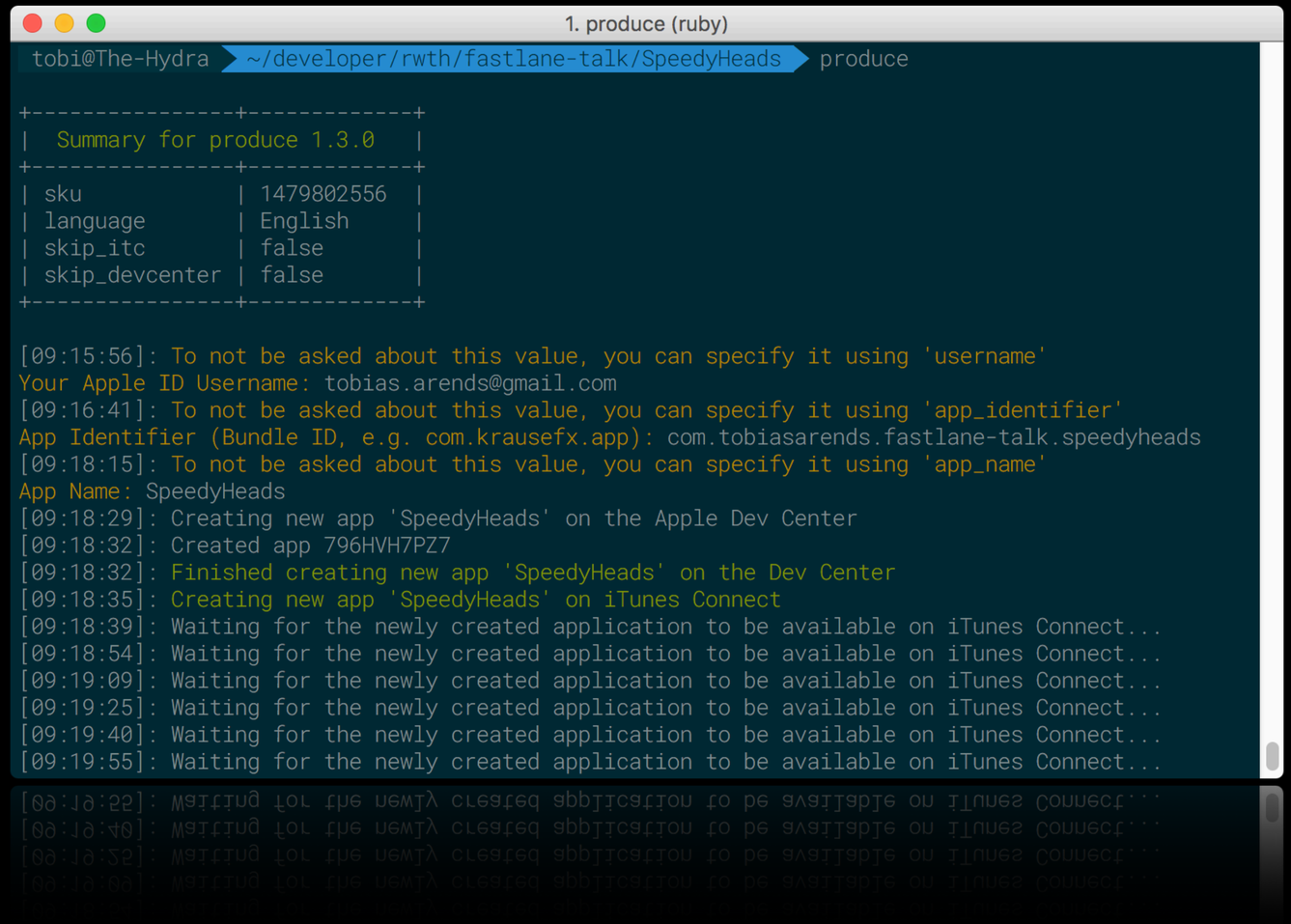
produce

```
$ produce --username        tobias.arends@gmail.com
          --app_identifier  com.tobiasarends.fastlane-talk.speedyheads
          --app_name        SpeedyHeads
```

Let’s create an app

the fast way

produce



CocoaHeads Aachen | **fastlane.tools** (Tobias Arends) | 11/24/2016

(this is the part where we typically code our app)

Code Signing Identity

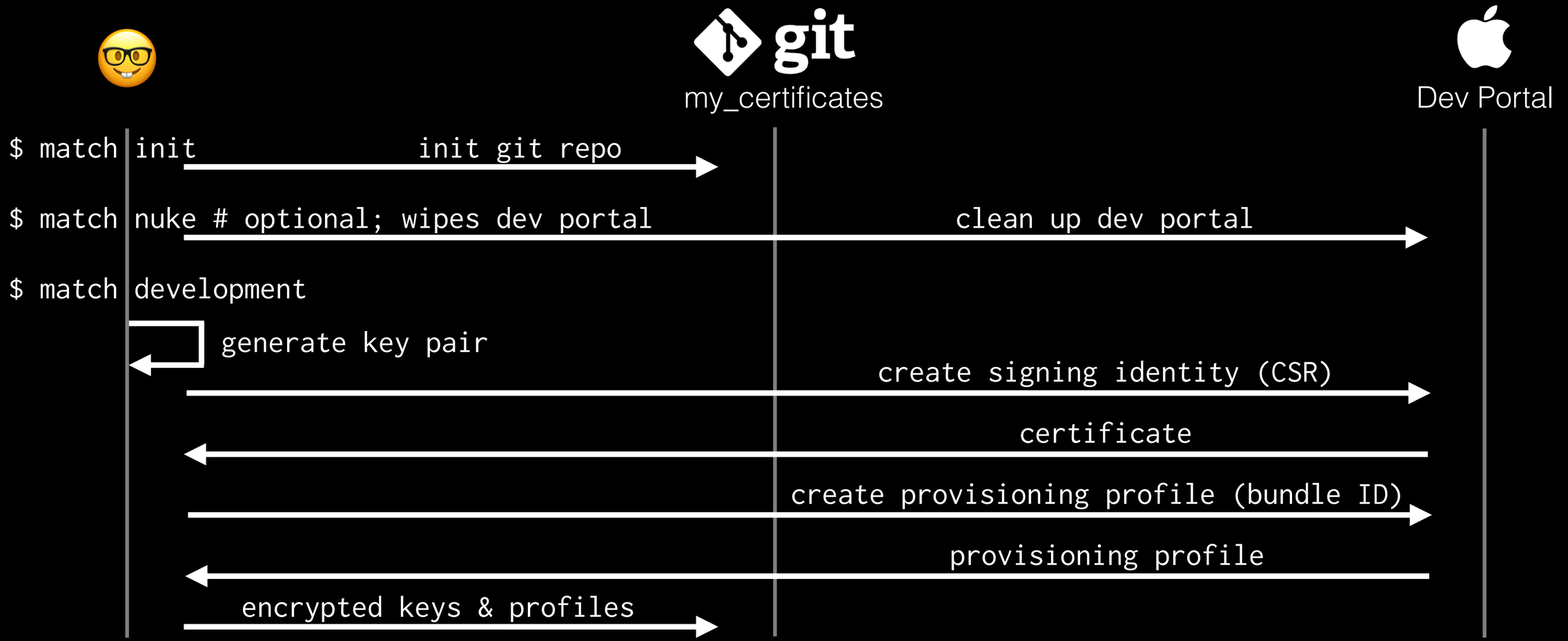
- private key (only on your Mac) + certificate

Provisioning Profiles

- glue specific Code Signing Identity to App Identifier

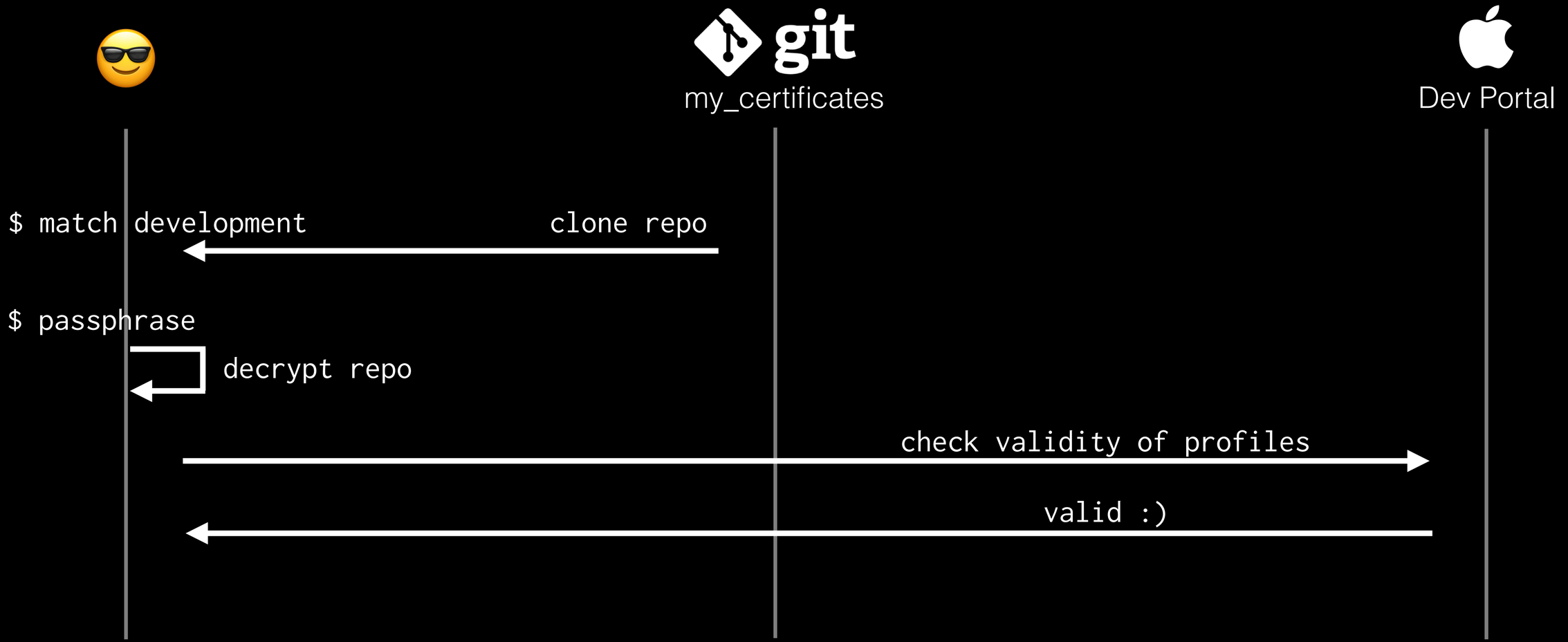
match

initial run (simplified)



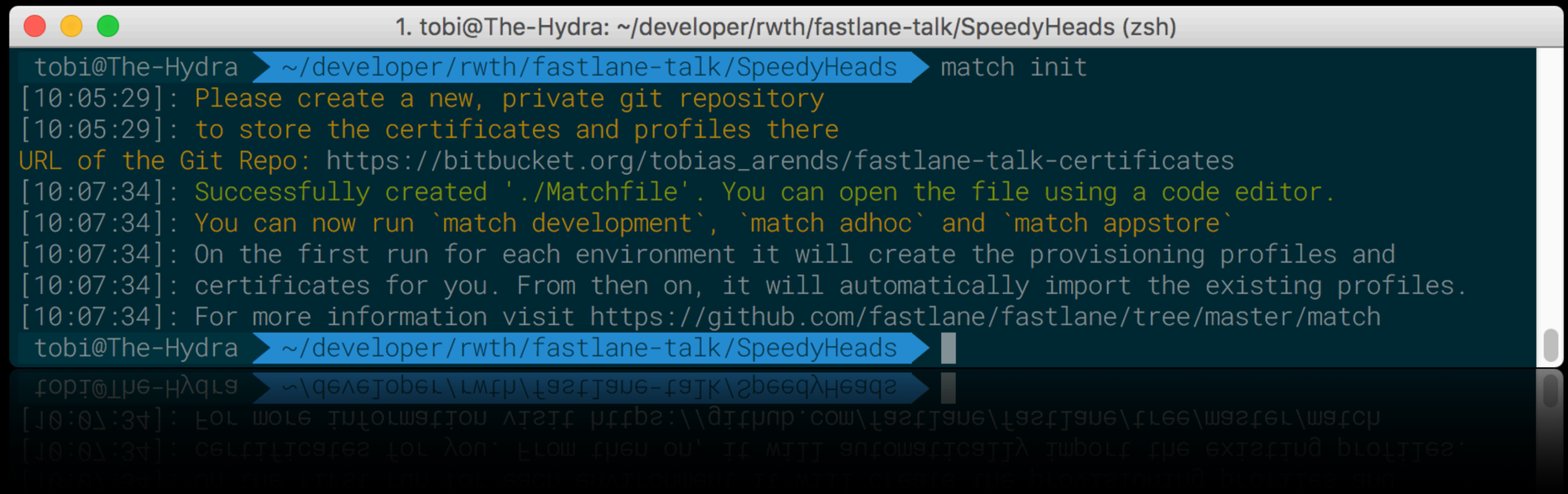
match

subsequent runs (simplified)



Code Signing & Provisioning Profiles

match



Security considerations

match

- not the recommended approach by Apple

access control

- you don’t have to grant access to dev portal to every engineer/freelancer
  - enables shared development with personal accounts
- granular access control via git

security

- certificates and profiles are encrypted using OpenSSL / passphrase
- you have to trust or review match

Security considerations

match

What can happen if repo and passphrase get compromised?

- App Store Profiles
  - nothing (unless attacker compromised iTC credentials as well)
- Development & Ad Hoc Profiles
  - attacker could potentially install apps on registered devices
- Enterprise Profiles
  - really bad things; because of this these are not supported by match

Deployment

The *Fastfile*

example: beta delivery


```
platform :ios do

  # lanes
  desc "Submit a new Beta Build to Apple TestFlight"
  lane :beta do

    # actions
    ensure_git_status_clean
    ensure_git_branch(branch: 'master')
    increment_build_number
    commit_version_bump(xcodeproj: './SpeedyHeads.xcodeproj')
    git_pull
    push_to_git_remote

    match(
      type: "appstore",
      app_identifier: “com.tobiasarends.fastlane-talk.speedyheads”,
      # readonly: true # use this if you don’t have access to dev portal
      )
    gym(scheme: "SpeedyHeads") # Build your app
    pilot(skip_submission: false) # upload your app to TestFlight
  end

end
```

Lanes

- define multiple lanes

Actions

- information is passed from one action to the next
- `$ fastlane action`
  # lists available actions

Release

Screenshots

setup; common issues


```
/* the gist; based on UITest */

import XCTest
import UIKit

class SpeedyHeadsUITests: XCTestCase {

    override func setUp() {
        super.setUp()

        let app = XCUIApplication()

        // (1)
        setupSnapshot(app)
        app.launch()
    }


    func testScreenshots() {

        let app = XCUIApplication()

        // (2) TODO: navigate to screenshot position
        // (fun times 😬🔫;  UIRefreshControl)
        // (3) start shooting screenshots
        snapshot("0_myFirstAutomatedScreenshot")
    }
}
```

- Why automate screenshot generation? It's a tedious task!
  - screenshots for every update, device type, language
  - Bonus: runs UI Tests for every combination
- Common Issues
  - simulator state (authenticated user, user settings, user data)
    - app requires setup (login, …)
      - best: reset state & re-setup on app launch
    - *snapshot* can reset all simulators for you
  - async data fetches (JSON, images…)
    - best: use locally available mock data (especially when running on CI servers)

Screenshots

common issues; and how to get around them


```
/* in your test */

import XCTest
import UIKit

class SpeedyHeadsUITests: XCTestCase {

    override func setUp() {
        super.setUp()

        let app = XCUIApplication()

        // waitForLoadingIndicator is incredibly useful :)
        setupSnapshot(app, waitForLoadingIndicator: true)

        // you can pass some constant if you need to prep your App
        // for those beautiful screenshots
        app.launchArguments.append( “UI_TEST_SCREENSHOTS”)
        app.launch()
    }


    func testScreenshots() {

        let app = XCUIApplication()

        snapshot(“0_myFirstAutomatedScreenshot")
    }
}
```

```
/* in your application */

#if DEBUG
    if ProcessInfo.processInfo.arguments.contains("UI_TEST_SCREENSHOTS"){

        setupAppForScreenshotGeneration(app)
    }
#endif

/**
 * -FASTLANE_SNAPSHOT YES is passed by default as well. However, it is
 * only available when snapshot is actually running. It’s not available
 * when debugging tests.
 */
```

Submit for Review

- **fetch** metadata from iTC
- modify metadata locally
  - txt-file based
- **upload** metadata to iTC and submit for Review

Great - Let's go!

Dependencies & Installation

```
ruby # ruby version > 2.0

$ xcode-select —install # install Xcode command line tools

# install: refer to github.com/fastlane most recent instructions
$ sudo gem install fastlane -NV # install as ruby gem or via homebrew
```

about documentation

- documentation is great! - docs.fastlane.tools
- for specifics:

  ❌ don’t immediately search the web

  ✅ ask the tools!

```
Example:
fastlane action          # list available actions
fastlane action slack    # details i.e. available environment variables
```

Demo 🚀

Where to go from here?

- play around with the tools that fit your specific needs
- maybe integrate with your favourite CI Server ? (fun times) ¯_(ツ)_/¯

References

- https://fastlane.tools
- https://codesigning.guide
- https://www.raywenderlich.com/116065/fastlane-tutorial-getting-started
- https://docs.fastlane.tools/best-practices/continuous-integration/

That’s it.

Thank you!