macOS install

Contents

System requirements

To install and run Flutter, your development environment must meet these minimum requirements:

- **Operating Systems**: macOS
- **Disk Space**: 2.8 GB (does not include disk space for IDE/tools).
- **Tools**: Flutter uses `git` for installation and upgrade. We recommend installing Xcode, which includes `git`, but you can also install `git` separately.

Important: If you're installing on a Mac with the latest Apple M1 processor, you may find these supplementary notes useful reading as we complete support for the new Apple Silicon architecture.

Get the Flutter SDK

1. Download the following installation bundle to get the latest stable release of the Flutter SDK:

   flutter_macos_2.2.3-stable.zip

   For other release channels, and older builds, see the SDK releases page.

2. Extract the file in the desired location, for example:

```
$ cd ~/development
$ unzip ~/Downloads/flutter_macos_2.2.3-stable.zip
```

3. Add the `flutter` tool to your path:

```
$ export PATH="$PATH:`pwd`/flutter/bin"
```

   This command sets your `PATH` variable for the *current* terminal window only. To permanently add Flutter to your path, see Update your path.

You are now ready to run Flutter commands!

Note: To update an existing version of Flutter, see Upgrading Flutter.

Run flutter doctor

Run the following command to see if there are any dependencies you need to install to complete the setup (for verbose output, ac the `-v` flag):

```
$ flutter doctor
```

This command checks your environment and displays a report to the terminal window. The Dart SDK is bundled with Flutter; it is r necessary to install Dart separately. Check the output carefully for other software you might need to install or further tasks to perform (shown in **bold** text).

For example:

```
[-] Android toolchain - develop for Android devices
    • Android SDK at /Users/obiwan/Library/Android/sdk
    ✗ Android SDK is missing command line tools; download from https://goo.gl/XxQghQ
    • Try re-installing or updating your Android SDK,
      visit https://flutter.dev/setup/#android-setup for detailed instructions.
```

The following sections describe how to perform these tasks and finish the setup process.

Once you have installed any missing dependencies, run the `flutter doctor` command again to verify that you've set everything u correctly.

Downloading and installing with Homebrew

If you have Homebrew installed on your machine, you can install Flutter using the following command:

```
$ brew install --cask flutter
```

Then, run `flutter doctor`. That lets you know if there are other dependencies you need to install to use Flutter, such as the Andr SDK.

Downloading straight from GitHub instead of using an archive

This is only suggested for advanced use cases.

You can also use git directly instead of downloading the prepared archive. For example, to download the stable branch:

```
$ git clone https://github.com/flutter/flutter.git -b stable
```

Update your path, and run `flutter doctor`. That will let you know if there are other dependencies you need to install to use Flutte (e.g. the Android SDK).

If you did not use the archive, Flutter will download necessary development binaries as they are needed (if you used the archive, t are included in the download). You may wish to pre-download these development binaries (for example, you may wish to do this when setting up hermetic build environments, or if you only have intermittent network availability). To do so, run the following command:

```
$ flutter precache
```

For additional download options, see `flutter help precache`.

Warning: The `flutter` tool uses Google Analytics to anonymously report feature usage statistics and basic crash reports. This data is used to help improve Flutter tools over time.

Flutter tool analytics are not sent on the very first run. To disable reporting, type `flutter config --no-analytics`. To displa the current setting, type `flutter config`. If you opt out of analytics, an opt-out event is sent, and then no further information is sent by the Flutter tool.

By downloading the Flutter SDK, you agree to the Google Terms of Service. Note: The Google Privacy Policy describes how data is handled in this service.

Moreover, Flutter includes the Dart SDK, which may send usage metrics and crash reports to Google.

Update your path

You can update your PATH variable for the current session at the command line, as shown in Get the Flutter SDK. You'll probably want to update this variable permanently, so you can run `flutter` commands in any terminal session.

The steps for modifying this variable permanently for all terminal sessions are machine-specific. Typically you add a line to a file t is executed whenever you open a new window. For example:

1. Determine the path of your clone of the Flutter SDK. You need this in Step 3.
2. Open (or create) the `rc` file for your shell. Typing `echo $SHELL` in your Terminal tells you which shell you're using. If you're us Bash, edit `$HOME/.bash_profile` or `$HOME/.bashrc`. If you're using Z shell, edit `$HOME/.zshrc`. If you're using a different she the file path and filename will be different on your machine.
3. Add the following line and change `[PATH_OF_FLUTTER_GIT_DIRECTORY]` to be the path of your clone of the Flutter git repo:

```
$ export PATH="$PATH:[PATH_OF_FLUTTER_GIT_DIRECTORY]/bin"
```

4. Run `source $HOME/.<rc file>` to refresh the current window, or open a new terminal window to automatically source the f
5. Verify that the `flutter/bin` directory is now in your PATH by running:

```
$ echo $PATH
```

Verify that the `flutter` command is available by running:

```
$ which flutter
```

Note: As of Flutter's 1.19.0 dev release, the Flutter SDK contains the `dart` command alongside the `flutter` command so that you can more easily run Dart command-line programs. Downloading the Flutter SDK also downloads the compatible version of Dart, but if you've downloaded the Dart SDK separately, make sure that the Flutter version of `dart` is first in your path, as the two versions might not be compatible. The following command tells you whether the `flutter` and `dart` commands originate from the same `bin` directory and are therefore compatible.

```
$ which flutter dart
/path-to-flutter-sdk/bin/flutter
/usr/local/bin/dart
```

As shown above, the two commands don't come from the same `bin` directory. Update your path to use commands from `/path-to-flutter-sdk/bin` before commands from `/usr/local/bin` (in this case). After updating your shell for the change to take effect, running the `which` command again should show that the `flutter` and `dart` commands now come from the same directory.

```
$ which flutter dart
/path-to-flutter-sdk/bin/flutter
/path-to-flutter-sdk/bin/dart
```

To learn more about the `dart` command, run `dart -h` from the command line, or see the dart tool page.

Platform setup

macOS supports developing Flutter apps in iOS, Android, and the web (technical preview release). Complete at least one of the platform setup steps now, to be able to build and run your first Flutter app.

iOS setup

Install Xcode

To develop Flutter apps for iOS, you need a Mac with Xcode installed.

1. Install the latest stable version of Xcode (using web download or the Mac App Store).
2. Configure the Xcode command-line tools to use the newly-installed version of Xcode by running the following from the command line:

```
$ sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer
$ sudo xcodebuild -runFirstLaunch
```

This is the correct path for most cases, when you want to use the latest version of Xcode. If you need to use a different vers specify that path instead.

3. Make sure the Xcode license agreement is signed by either opening Xcode once and confirming or running `sudo xcodebuil license` from the command line.

Versions older than the latest stable version may still work, but are not recommended for Flutter development. Using old versions Xcode to target bitcode is not supported, and is likely not to work.

With Xcode, you'll be able to run Flutter apps on an iOS device or on the simulator.

Set up the iOS simulator

To prepare to run and test your Flutter app on the iOS simulator, follow these steps:

1. On your Mac, find the Simulator via Spotlight or by using the following command:

```
$ open -a Simulator
```

2. Make sure your simulator is using a 64-bit device (iPhone 5s or later) by checking the settings in the simulator's **Hardware > Device** menu.
3. Depending on your development machine's screen size, simulated high-screen-density iOS devices might overflow your scre Grab the corner of the simulator and drag it to change the scale. You can also use the **Window > Physical Size** or **Window > Pixel Accurate** options if your computer's resolution is high enough.
    - If you are using a version of Xcode older than 9.1, you should instead set the device scale in the **Window > Scale** menu

Create and run a simple Flutter app

To create your first Flutter app and test your setup, follow these steps:

1. Create a new Flutter app by running the following from the command line:

```
$ flutter create my_app
```

2. A `my_app` directory is created, containing Flutter's starter app. Enter this directory:

```
$ cd my_app
```

3. To launch the app in the Simulator, ensure that the Simulator is running and enter:

```
$ flutter run
```

Deploy to iOS devices

To deploy your Flutter app to a physical iOS device you'll need to set up physical device deployment in Xcode and an Apple Develc account. If your app is using Flutter plugins, you will also need the third-party CocoaPods dependency manager.

1. You can skip this step if your apps do not depend on Flutter plugins with native iOS code. Install and set up CocoaPods by running the following commands:

```
$ sudo gem install cocoapods
```

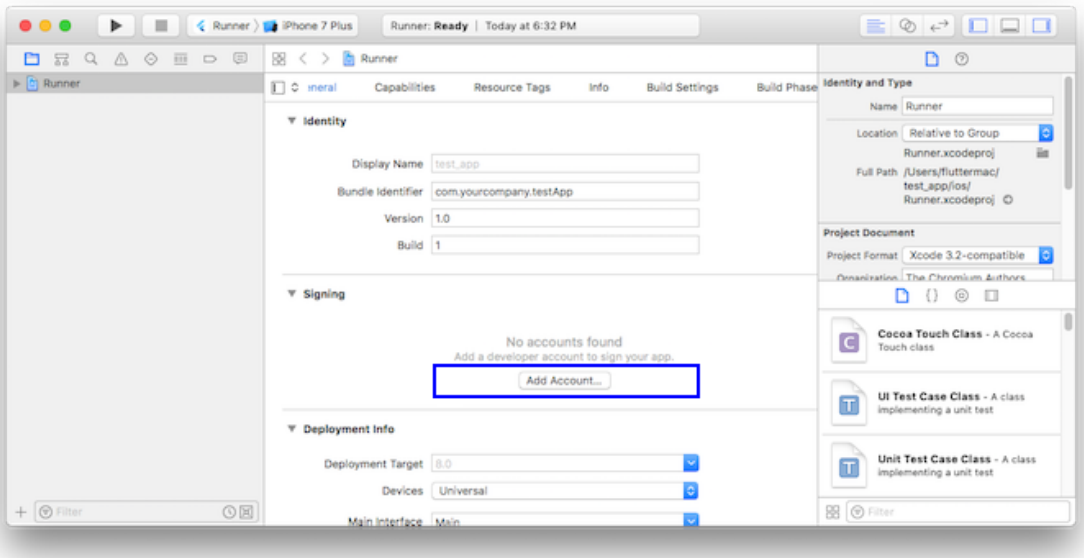
> **Note:** The default version of Ruby requires `sudo` to install the CocoaPods gem. If you are using a Ruby Version manager, you may need to run without `sudo`.

2. Follow the Xcode signing flow to provision your project:

    a. Open the default Xcode workspace in your project by running `open ios/Runner.xcworkspace` in a terminal window fr your Flutter project directory.
    b. Select the device you intend to deploy to in the device drop-down menu next to the run button.
    c. Select the `Runner` project in the left navigation panel.
    d. In the `Runner` target settings page, make sure your Development Team is selected under **Signing & Capabilities > Tear**

    When you select a team, Xcode creates and downloads a Development Certificate, registers your device with your account, and creates and downloads a provisioning profile (if needed).

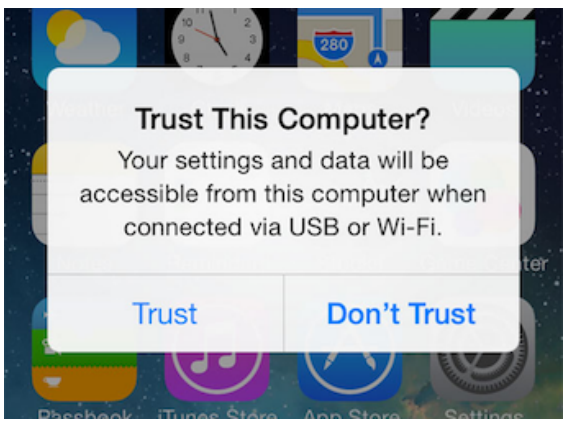
    - To start your first iOS development project, you might need to sign into Xcode with your Apple ID.



Developn

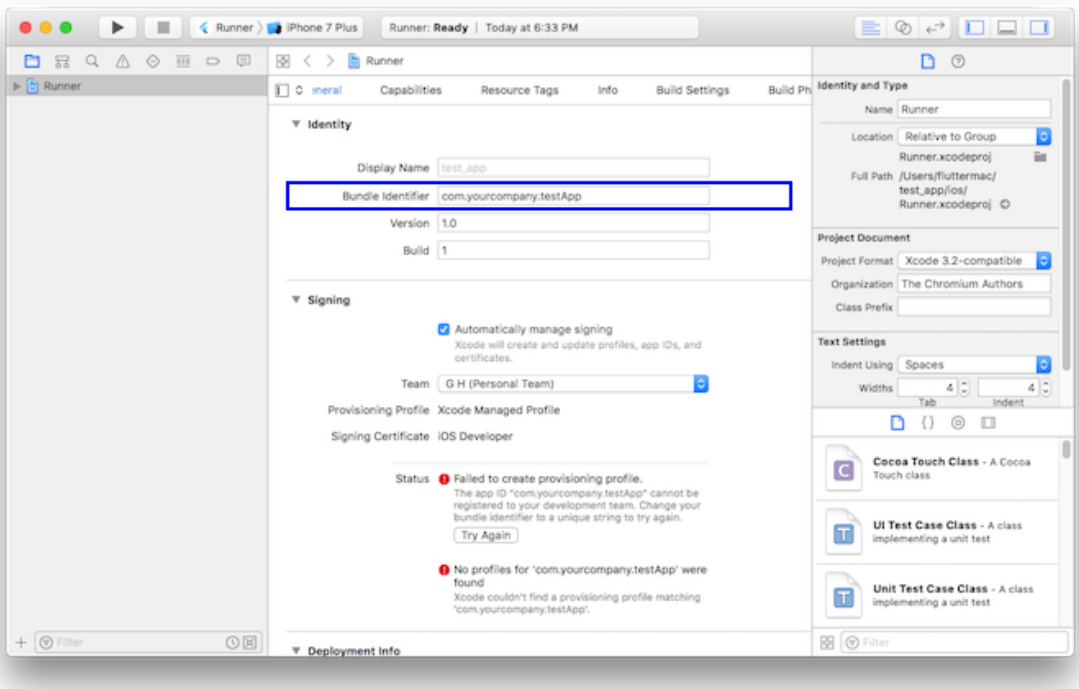
and testing is supported for any Apple ID. Enrolling in the Apple Developer Program is required to distribute your to the App Store. For details about membership types, see Choosing a Membership.

- The first time you use an attached physical device for iOS development, you need to trust both your Mac and the Development Certificate on that device. Select `Trust` in the dialog prompt when first connecting the iOS device t your Mac.



Then, go to the Settings app on the iOS device, select **General > Device Management** and trust your Certificate. F first time users, you may need to select **General > Profiles > Device Management** instead.

- If automatic signing fails in Xcode, verify that the project's **General > Identity > Bundle Identifier** value is unique.



3. Start your app by running `flutter run` or clicking the Run button in Xcode.

Android setup

> **Note:** Flutter relies on a full installation of Android Studio to supply its Android platform dependencies. However, you can write your Flutter apps in a number of editors; a later step discusses that.

Install Android Studio

1. Download and install Android Studio.
2. Start Android Studio, and go through the 'Android Studio Setup Wizard'. This installs the latest Android SDK, Android SDK Command-line Tools, and Android SDK Build-Tools, which are required by Flutter when developing for Android.
3. Run `flutter doctor` to confirm that Flutter has located your installation of Android Studio. If Flutter cannot locate it, run `flutter config --android-studio-dir <directory>` to set the directory that Android Studio is installed to.

Set up your Android device

To prepare to run and test your Flutter app on an Android device, you need an Android device running Android 4.1 (API level 16) o higher.

1. Enable **Developer options** and **USB debugging** on your device. Detailed instructions are available in the Android documentat
2. Windows-only: Install the Google USB Driver.
3. Using a USB cable, plug your phone into your computer. If prompted on your device, authorize your computer to access your device.
4. In the terminal, run the `flutter devices` command to verify that Flutter recognizes your connected Android device. By defa Flutter uses the version of the Android SDK where your `adb` tool is based. If you want Flutter to use a different installation of Android SDK, you must set the `ANDROID_SDK_ROOT` environment variable to that installation directory.

Set up the Android emulator

To prepare to run and test your Flutter app on the Android emulator, follow these steps:

1. Enable VM acceleration on your machine.
2. Launch **Android Studio**, click the **AVD Manager** icon, and select **Create Virtual Device...**
   - In older versions of Android Studio, you should instead launch **Android Studio > Tools > Android > AVD Manager** and select **Create Virtual Device...**. (The **Android** submenu is only present when inside an Android project.)
   - If you do not have a project open, you can choose **Configure > AVD Manager** and select **Create Virtual Device...**
3. Choose a device definition and select **Next**.
4. Select one or more system images for the Android versions you want to emulate, and select **Next**. An *x86* or *x86_64* image i recommended.
5. Under Emulated Performance, select **Hardware - GLES 2.0** to enable hardware acceleration.
6. Verify the AVD configuration is correct, and select **Finish**.

   For details on the above steps, see Managing AVDs.

7. In Android Virtual Device Manager, click **Run** in the toolbar. The emulator starts up and displays the default canvas for your selected OS version and device.

Agree to Android Licenses

Before you can use Flutter, you must agree to the licenses of the Android SDK platform. This step should be done after you have installed the tools listed above.

1. Make sure that you have a version of Java 8 installed and that your `JAVA_HOME` environment variable is set to the JDK's folde

   Android Studio versions 2.2 and higher come with a JDK, so this should already be done.

2. Open an elevated console window and run the following command to begin signing licenses.

```
$ flutter doctor --android-licenses
```

3. Review the terms of each license carefully before agreeing to them.
4. Once you are done agreeing with licenses, run `flutter doctor` again to confirm that you are ready to use Flutter.

macOS setup

> ⚠ **Warning: Beta!** This area covers desktop support, which is available as a beta release. Beta support still has notable feature gaps, including accessibility support. You can try a beta snapshot of desktop support on the stable channel, or you can keep up with the latest changes to desktop on the beta channel. For more information, see the **Desktop** section in What's new in Flutter 2, a free article on Medium.

Additional macOS requirements

For macOS desktop development, you need the following in addition to the Flutter SDK:

- Xcode
- CocoaPods if you use plugins

Enable desktop support

At the command line, perform the following command to enable desktop support

```
$ flutter config --enable-macos-desktop
```

For more information, see Desktop support for Flutter

Web setup

Flutter has support for building web applications in the `stable` channel. Any app created in Flutter 2 automatically builds for the v
To add web support to an existing app, follow the instructions on Building a web application with Flutter when you've completed t
setup above.

Next step

Set up your preferred editor.

Set up an edito