

# macOS install

[Docs](#) [Get started](#) [Install](#) macOS

## Contents

- [System requirements](#)
- [Get the Flutter SDK](#)
  - [Run flutter doctor](#)
  - [Update your path](#)
- [Platform setup](#)
- [iOS setup](#)
  - [Install Xcode](#)
  - [Set up the iOS simulator](#)
  - [Create and run a simple Flutter app](#)
  - [Deploy to iOS devices](#)
- [Android setup](#)
  - [Install Android Studio](#)
  - [Set up your Android device](#)
  - [Set up the Android emulator](#)
- [Web setup](#)
- [Next step](#)

# System requirements

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

- **Operating Systems:** macOS (64-bit)
- **Disk Space:** 2.8 GB (does not include disk space for IDE/tools).
- **Tools:** Flutter depends on these command-line tools being available in your environment.
  - `bash`
  - `curl`
  - `git 2.x`
  - `mkdir`
  - `rm`
  - `unzip`
  - `which`
  - `zip`

## Get the Flutter SDK

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

```
flutter_macos_v1.12.13+hotfix.9-stable.zip
```

For other release channels, and older builds, see the [SDK archive](#) page.

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

```
$ cd ~/development
$ unzip
~/Downloads/flutter_macos_v1.12.13+hotfix.9-stable.zip
```

If you don't want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code from the [Flutter repo](#) on GitHub, and change branches or tags as needed. For example:

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

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

```
$ export content_copy  
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](#).

#### 4. Optionally, pre-download development binaries:

The `flutter` tool downloads platform-specific development binaries as needed. For scenarios where pre-downloading these artifacts is preferable (for example, in hermetic build environments, or with intermittent network availability), iOS and Android binaries can be downloaded ahead of time by running:

```
$ flutter precache content_copy
```

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

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, add the `-v` flag):

```
$ flutter doctor
```

content\_copy

This command checks your environment and displays a report to the terminal window. The Dart SDK is bundled with Flutter; it is not 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

**x 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 up correctly.

**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 display the current setting, type `flutter config`. If you

opt out of analytics, an opt-out event will be sent, and then no further information will be 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 that is executed whenever you open a new window. For example:

1. Determine the directory where you placed the Flutter SDK. You need this in Step 3.

2. Open (or create) the `rc` file for your shell. For example, macOS Mojave (and earlier) uses the Bash shell by default, so edit `$HOME/.bash_profile` or `$HOME/.bashrc`. macOS Catalina uses the Z shell by default, so edit `$HOME/.zshrc`. If you are using a different shell, the file path and filename will be different on your machine.
3. Add the following line and change `[PATH_TO_FLUTTER_GIT_DIRECTORY]` to be the path where you cloned Flutter's git repo:

```
$ export PATH="$PATH: [PATH_TO_FLUTTER_GIT_DIRECTORY]/flutter/bin"
```

4. Run `source $HOME/.<rc file>` to refresh the current window, or open a new terminal window to automatically source the file.
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
```



# 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 version, specify that path instead.

3. Make sure the Xcode license agreement is signed by either opening Xcode once and confirming or running `sudo xcodebuild -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 of 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 content_copy
```

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 screen. Set the device scale under the **Window > Scale** menu in the simulator.

## 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 content_copy
```

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

```
$ cd my_app
```

content\_copy

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

```
$ flutter run
```

content\_copy

## Deploy to iOS devices

To deploy your Flutter app to a physical iOS device you need the third-party CocoaPods dependency manager and an Apple Developer account. You'll also need to set up physical device deployment in Xcode.

1. Install and set up CocoaPods by running the following commands:

```
$ sudo gem install cocoapods  
$ pod setup
```

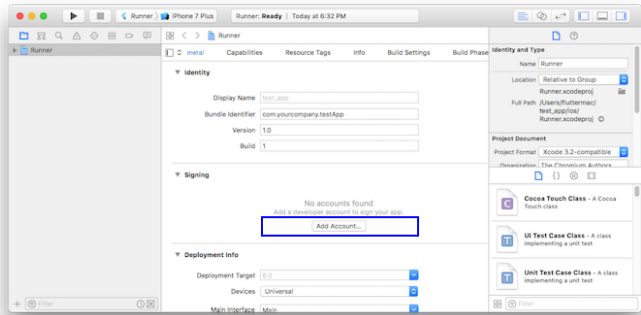
content\_copy

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 from 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. The UI varies depending on your version of Xcode.
  - For Xcode 10, look under **General > Signing > Team**.
  - For Xcode 11 and newer, look under **Signing & Capabilities > Team**.

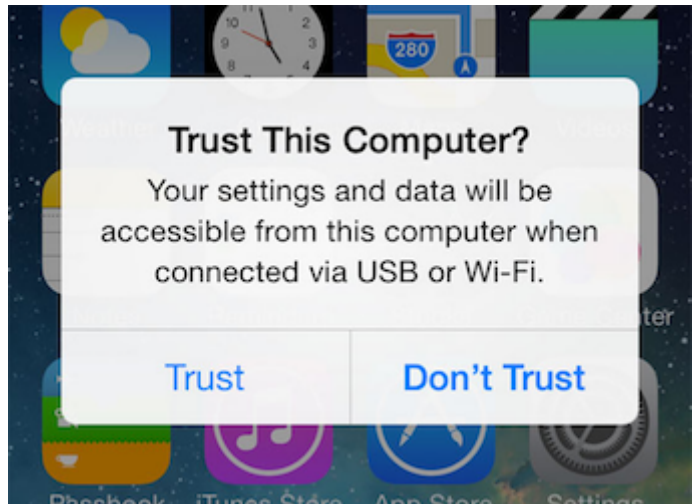
When you select a team, Xcode creates and downloads a Development Certificate, registers your device with your account, and creates and downloads a provisioning profile (if needed).

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



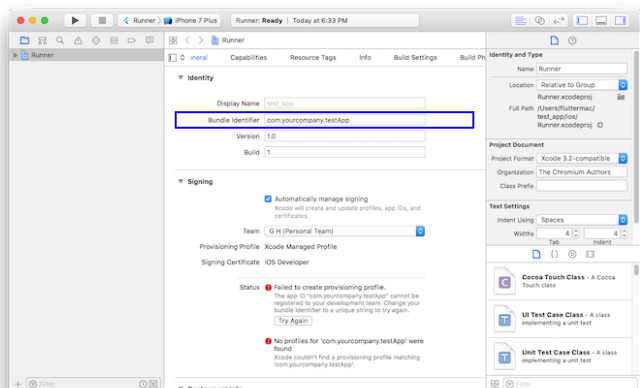
Development and testing is supported for any Apple ID. Enrolling in the Apple Developer Program is required to distribute your app to the App Store. For details about membership types, see [Choosing a Membership](#).

- 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 to your Mac.



Then, go to the Settings app on the iOS device, select **General > Device Management** and trust your Certificate.

- 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`.

# 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 will discuss 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.



**Warning:** In Android Studio 3.6 or later, you need to manually add the old version of the Android SDK Tools for Flutter to work. To do this:

1. Open the **Android Studio SDK Manager**
2. In the Android SDK tab, uncheck **Hide Obsolete Packages**
3. Check **Android SDK Tools (Obsolete)**

The dialog below shows the appropriate settings: 

This is a [known issue](#) that will be addressed in an upcoming version of Flutter.

# Set up your Android device

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

1. Enable **Developer options** and **USB debugging** on your device. Detailed

instructions are available in the [Android documentation](#).

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 default, Flutter uses the version of the Android SDK where your `adb` tool is based. If you want Flutter to use a different installation of the Android SDK, you must set the `ANDROID_HOME` 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 > Tools > Android > AVD Manager** and select **Create Virtual Device**. (The **Android** submenu is only present when inside an Android project.)
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 is 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.

# Web setup

**Note:** As of 1.12, Flutter has early support for running web applications, but you need to be running the beta channel of Flutter. If you experience a problem that hasn't yet been reported, please [file an issue](#) and make sure that “web” appears in the title.

---

To prepare to run, test, and debug your Flutter app on the web, you must [install Chrome](#), if you haven't already.