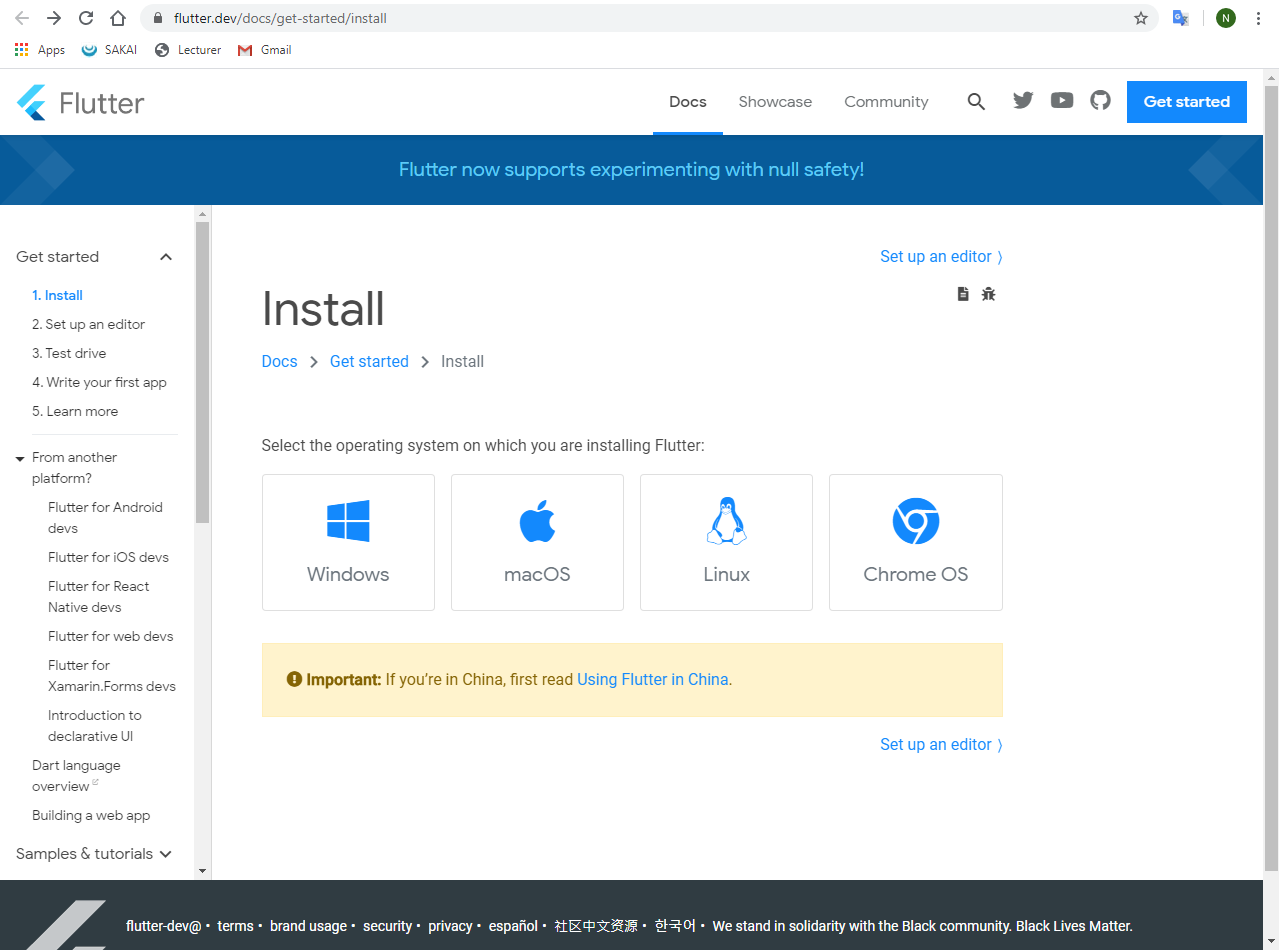
# Install flutter

# Step 1: Chose the operating system

Enter the link <https://flutter.dev/docs/get-started/install> and choose that operating system you are using.



Step2:

# Step 2:

* Windows install:

## **Case1: Windows operating system**

### **System requirements**

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

* **Operating Systems**: Windows 7 SP1 or later (64-bit), x86-64 based
* **Disk Space**: 1.32 GB (does not include disk space for IDE/tools).
* **Tools**: Flutter depends on these tools being available in your environment.
  + [Windows PowerShell 5.0](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-windows-powershell) or newer (this is pre-installed with Windows 10)
  + [Git for Windows](https://git-scm.com/download/win) 2.x, with the **Use Git from the Windows Command Prompt** option.

If Git for Windows is already installed, make sure you can run git commands from the command prompt or PowerShell.

### **Get the Flutter SDK**

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

<https://flutter.dev/docs/development/tools/sdk/releases>

For other release channels, and older builds, see the [SDK releases](https://flutter.dev/docs/development/tools/sdk/releases) page.

1. Extract the zip file and place the contained flutter in the desired installation location for the Flutter SDK (for example, C:\src\flutter).

**Warning:** Do not install Flutter in a directory like C:\Program Files\ that requires elevated privileges.

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](https://github.com/flutter/flutter) on GitHub, and change branches or tags as needed. For example:

C:\src>git clone https://github.com/flutter/flutter.git -b stable

You are now ready to run Flutter commands in the Flutter Console.

#### **Update your path**

If you wish to run Flutter commands in the regular Windows console, take these steps to add Flutter to the PATH environment variable:

* From the Start search bar, enter ‘env’ and select **Edit environment variables for your account**.
* Under **User variables** check if there is an entry called **Path**:
  + If the entry exists, append the full path to flutter\bin using ; as a separator from existing values.
  + If the entry doesn’t exist, create a new user variable named Path with the full path to flutter\bin as its value.

You have to close and reopen any existing console windows for these changes to take effect.

**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 (on macOS, linux, and chrome OS), tells you whether the flutter and dart commands originate from the same bin directory and are therefore compatible. (Some versions of Windows support a similar where command.)

$ 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 or where 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](https://dart.dev/tools/dart-vm) page.

#### **Run flutter doctor**

From a console window that has the Flutter directory in the path (see above), run the following command to see if there are any platform dependencies you need to complete the setup:

C:\src\flutter>flutter doctor

This command checks your environment and displays a report of the status of your Flutter installation. 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 D:\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, you can run the flutter doctor command again to verify that you’ve set everything up correctly.

**Note:** If flutter doctor returns that either the Flutter plugin or Dart plugin of Android Studio are not installed, move on to [Set up an editor](https://flutter.dev/docs/get-started/editor?tab=androidstudio) to resolve this issue.

**Warning:** The flutter tool uses Google Analytics to anonymously report feature usage statistics and basic [crash reports](https://github.com/flutter/flutter/wiki/Flutter-CLI-crash-reporting). 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 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](https://policies.google.com/privacy) describes how data is handled in this service.

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

### **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](https://developer.android.com/studio) from <https://developer.android.com/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.

#### **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) or higher.

1. Enable **Developer options** and **USB debugging** on your device. Detailed instructions are available in the [Android documentation](https://developer.android.com/studio/debug/dev-options) (link here for more information <https://developer.android.com/studio/debug/dev-options>).
2. Windows-only: Install the [Google USB Driver](https://developer.android.com/studio/run/win-usb) (link here for more information <https://developer.android.com/studio/run/win-usb>).
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\_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](https://developer.android.com/studio/run/emulator-acceleration) (link here for more information <https://developer.android.com/studio/run/emulator-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 is recommended.
5. Under Emulated Performance, select **Hardware - GLES 2.0** to enable [hardware acceleration](https://developer.android.com/studio/run/emulator-acceleration) (link here for more information <https://developer.android.com/studio/run/emulator-acceleration>).
6. Verify the AVD configuration is correct, and select **Finish**.

For details on the above steps, see [Managing AVDs](https://developer.android.com/studio/run/managing-avds) (link here for more information <https://developer.android.com/studio/run/managing-avds>).

1. 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**

Flutter has early support for building web applications using the beta channel of Flutter. To add support for web development, follow [these instructions](https://flutter.dev/docs/get-started/web) (<https://flutter.dev/docs/get-started/web>) when you’ve completed the setup above.

* MacOS install:

## **Case2: MacOS operating system**

### **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

### **Get the Flutter SDK**

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

<https://flutter.dev/docs/development/tools/sdk/releases>

For other release channels, and older builds, see the [SDK releases](https://flutter.dev/docs/development/tools/sdk/releases) page.

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

$ cd ~/development

$ unzip ~/Downloads/flutter\_macos\_1.22.3-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](https://github.com/flutter/flutter) on GitHub with the following command:

$ git clone <https://github.com/flutter/flutter.git>

You can also change branches or tags as needed. For example, to get just the stable version:

$ git clone [https://github.com/flutter/flutter.git -b stable --depth 1](https://github.com/flutter/flutter.git%20-b%20stable%20--depth%201)

1. 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](https://flutter.dev/docs/get-started/install/macos#update-your-path).

1. 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

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](https://flutter.dev/docs/development/tools/sdk/upgrading).

#### **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

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

**✗ 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](https://github.com/flutter/flutter/wiki/Flutter-CLI-crash-reporting). 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 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](https://policies.google.com/privacy) 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](https://flutter.dev/docs/get-started/install/macos#get-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. Typing echo $SHELL in your Terminal tells you which shell you’re using. If you’re using Bash, edit $HOME/.bash\_profile or $HOME/.bashrc. If you’re using Z shell, edit $HOME/.zshrc. If you’re 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"

1. Run source $HOME/.<rc file> to refresh the current window, or open a new terminal window to automatically source the file.
2. 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 (on macOS, linux, and chrome OS), tells you whether the flutter and dart commands originate from the same bin directory and are therefore compatible. (Some versions of Windows support a similar where command.)

$ 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 or where 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](https://dart.dev/tools/dart-vm) 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](https://developer.apple.com/xcode/) or the [Mac App Store](https://itunes.apple.com/us/app/xcode/id497799835)).
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.

1. 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

1. 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.
2. Depending on your development machine’s screen size, simulated high-screen-density iOS devices might overflow your screen. 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

1. A my\_app directory is created, containing Flutter’s starter app. Enter this directory:

$ cd my\_app

1. 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 Developer 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](https://flutter.dev/docs/development/packages-and-plugins/developing-packages#types) with native iOS code. [Install and set up CocoaPods](https://guides.cocoapods.org/using/getting-started.html#installation) by running the following commands:

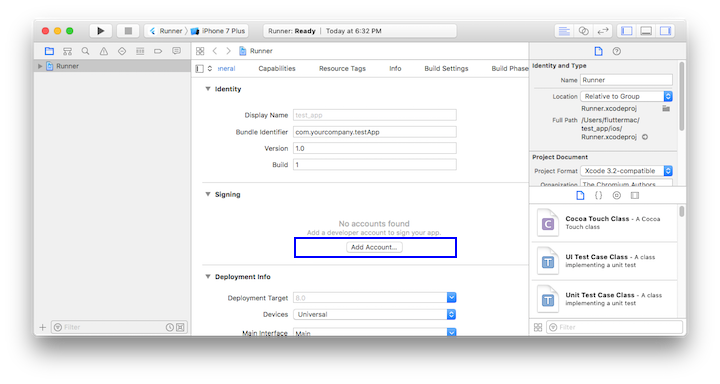
$ sudo gem install cocoapods

**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.

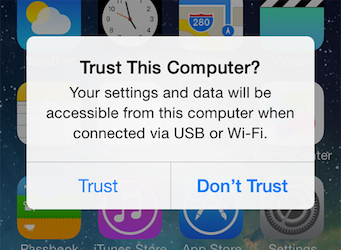
1. Follow the Xcode signing flow to provision your project:
   1. Open the default Xcode workspace in your project by running open ios/Runner.xcworkspace in a terminal window from your Flutter project directory.
   2. Select the device you intend to deploy to in the device drop-down menu next to the run button.
   3. Select the Runner project in the left navigation panel.
   4. 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](https://developer.apple.com/support/compare-memberships).

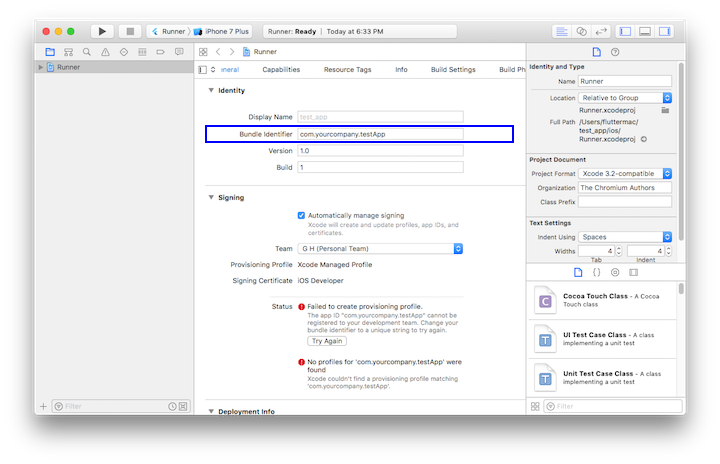


* + - The first time you use an attached physical device for iOS development, you need to trust both your Mac and the Development Certificate on that device. Select Trust in the dialog prompt when first connecting the iOS device to your Mac.



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



1. 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](https://developer.android.com/studio) from <https://developer.android.com/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.

#### **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) or higher.

1. Enable **Developer options** and **USB debugging** on your device. Detailed instructions are available in the [Android documentation](https://developer.android.com/studio/debug/dev-options) (link here for more information <https://developer.android.com/studio/debug/dev-options>).
2. Windows-only: Install the [Google USB Driver](https://developer.android.com/studio/run/win-usb) (link here for more information <https://developer.android.com/studio/run/win-usb>).
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\_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](https://developer.android.com/studio/run/emulator-acceleration) (link here for more information <https://developer.android.com/studio/run/emulator-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 is recommended.
5. Under Emulated Performance, select **Hardware - GLES 2.0** to enable [hardware acceleration](https://developer.android.com/studio/run/emulator-acceleration) (link here for more information <https://developer.android.com/studio/run/emulator-acceleration>).
6. Verify the AVD configuration is correct, and select **Finish**.

For details on the above steps, see [Managing AVDs](https://developer.android.com/studio/run/managing-avds) (link here for more information <https://developer.android.com/studio/run/managing-avds>).

1. 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**

Flutter has early support for building web applications using the beta channel of Flutter. To add support for web development, follow [these instructions](https://flutter.dev/docs/get-started/web) (<https://flutter.dev/docs/get-started/web>) when you’ve completed the setup above.

* Linux install:

## **Case3: Linux operating system**

### **System requirements**

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

* **Operating Systems**: Linux (64-bit)
* **Disk Space**: 600 MB (does not include disk space for IDE/tools).
* **Tools**: Flutter depends on these command-line tools being available in your environment.
  + bash
  + curl
  + file
  + git 2.x
  + mkdir
  + rm
  + unzip
  + which
  + xz-utils
  + zip
* **Shared libraries**: Flutter test command depends on this library being available in your environment.
  + libGLU.so.1 - provided by mesa packages such as libglu1-mesa on Ubuntu/Debian and mesa-libGLU on Fedora.

### **Get the Flutter SDK**

On Linux, you have two ways you can install Flutter.

#### **Install Flutter using snapd**

The easiest way to install Flutter on Linux is by using snapd. For more information, see [Installing snapd](https://snapcraft.io/docs/installing-snapd).

Once you have snapd, you can [install Flutter using the Snap Store](https://snapcraft.io/flutter), or at the command line:

$ sudo snap install flutter –classic

#### **Install Flutter manually**

If you don’t have snapd, or can’t use it, you can install Flutter using the following steps.

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

<https://flutter.dev/docs/development/tools/sdk/releases>

For other release channels, and older builds, see the [SDK releases](https://flutter.dev/docs/development/tools/sdk/releases) page.

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

$ cd ~/development

$ tar xf ~/Downloads/flutter\_linux\_1.22.3-stable.tar.xz

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](https://github.com/flutter/flutter) on GitHub with the following command:

$ git clone https://github.com/flutter/flutter.git

You can also change branches or tags as needed. For example, to get just the stable version:

$ git clone https://github.com/flutter/flutter.git -b stable --depth 1

1. 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](https://flutter.dev/docs/get-started/install/linux#update-your-path).

1. 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

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](https://flutter.dev/docs/development/tools/sdk/upgrading).

#### **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

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

**✗ 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](https://github.com/flutter/flutter/wiki/Flutter-CLI-crash-reporting). 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 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](https://policies.google.com/privacy) 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](https://flutter.dev/docs/get-started/install/linux#get-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, Linux uses the Bash shell by default, so edit $HOME/.bashrc. 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"

1. Run source $HOME/.<rc file> to refresh the current window, or open a new terminal window to automatically source the file.
2. 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 (on macOS, linux, and chrome OS), tells you whether the flutter and dart commands originate from the same bin directory and are therefore compatible. (Some versions of Windows support a similar where command.)

$ 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 or where 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](https://dart.dev/tools/dart-vm) page.

#### **Update path directly**

In some cases, your distribution may not permanently acquire the path when using the above directions. When this occurs, you can change the environment variables file directly. These instructions require administrator privileges:

1. Determine the directory where you placed the Flutter SDK.
2. Locate the etc directory at the root of the system, and open the profile file with root privileges.

$ sudo nano /etc/profile

1. Update the PATH string with the location of your Flutter SDK directory.

if [ "`id -u`" -eq 0 ]; then

PATH="..."

else

PATH="/usr/local/bin:...:[PATH\_TO\_FLUTTER\_GIT\_DIRECTORY]/flutter/bin"

fi

export PATH

1. End the current session or reboot your system.
2. Once you start a new session, verify that the flutter command is available by running:

$ which flutter

For more details on setting the path in Bash, see [this StackExchange question](https://unix.stackexchange.com/questions/26047/how-to-correctly-add-a-path-to-path). For information on setting the path in Z shell, see [this StackOverflow question](https://stackoverflow.com/questions/11530090/adding-a-new-entry-to-the-path-variable-in-zsh).

### **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](https://developer.android.com/studio) from <https://developer.android.com/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.

#### **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) or higher.

1. Enable **Developer options** and **USB debugging** on your device. Detailed instructions are available in the [Android documentation](https://developer.android.com/studio/debug/dev-options) (link here for more information <https://developer.android.com/studio/debug/dev-options>).
2. Windows-only: Install the [Google USB Driver](https://developer.android.com/studio/run/win-usb) (link here for more information <https://developer.android.com/studio/run/win-usb>).
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\_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](https://developer.android.com/studio/run/emulator-acceleration) (link here for more information <https://developer.android.com/studio/run/emulator-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 is recommended.
5. Under Emulated Performance, select **Hardware - GLES 2.0** to enable [hardware acceleration](https://developer.android.com/studio/run/emulator-acceleration) (link here for more information <https://developer.android.com/studio/run/emulator-acceleration>).
6. Verify the AVD configuration is correct, and select **Finish**.

For details on the above steps, see [Managing AVDs](https://developer.android.com/studio/run/managing-avds) (link here for more information <https://developer.android.com/studio/run/managing-avds>).

1. 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**

Flutter has early support for building web applications using the beta channel of Flutter. To add support for web development, follow [these instructions](https://flutter.dev/docs/get-started/web) (<https://flutter.dev/docs/get-started/web>) when you’ve completed the setup above.

* Chrome OS install:

## **Case4: Chrome OS operating system**

### **System requirements**

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

* **Operating Systems**: Chrome OS (64-bit) with [Linux (Beta)](https://support.google.com/chromebook/answer/9145439) turned on
* **Disk Space**: 600 MB (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
  + xz-utils
* **Shared libraries**: Flutter test command depends on this library being available in your environment.
  + libGLU.so.1 - provided by mesa packages such as libglu1-mesa on Ubuntu/Debian

### **Get the Flutter SDK**

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

<https://flutter.dev/docs/development/tools/sdk/releases>

For other release channels, and older builds, see the [SDK releases](https://flutter.dev/docs/development/tools/sdk/releases) page.

1. In the Files app, drag-and-drop the downloaded file from “Downloads” to “Linux Files” to access Flutter from your Linux container.
2. Extract the file in the desired location, for example:

$ cd ~/development

$ tar xf ~/Downloads/flutter\_linux\_1.22.3-stable.tar.xz

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](https://github.com/flutter/flutter) on GitHub with the following command:

$ git clone https://github.com/flutter/flutter.git

You can also change branches or tags as needed. For example, to get just the stable version:

$ git clone https://github.com/flutter/flutter.git -b stable --depth 1

1. 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](https://flutter.dev/docs/get-started/install/chromeos#update-your-path).

1. 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

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](https://flutter.dev/docs/development/tools/sdk/upgrading).

#### **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

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

**✗ 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](https://github.com/flutter/flutter/wiki/Flutter-CLI-crash-reporting). 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 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](https://policies.google.com/privacy) 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](https://flutter.dev/docs/get-started/install/chromeos#get-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, Linux uses the Bash shell by default, so edit $HOME/.bashrc. 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"

1. Run source $HOME/.<rc file> to refresh the current window, or open a new terminal window to automatically source the file.
2. 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 (on macOS, linux, and chrome OS), tells you whether the flutter and dart commands originate from the same bin directory and are therefore compatible. (Some versions of Windows support a similar where command.)

$ 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 or where 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](https://dart.dev/tools/dart-vm) page.

#### **Update path directly**

In some cases, your distribution may not permanently acquire the path when using the above directions. When this occurs, you can change the environment variables file directly. These instructions require administrator privileges:

1. Determine the directory where you placed the Flutter SDK.
2. Locate the etc directory at the root of the system, and open the profile file with root privileges.

$ sudo nano /etc/profile

1. Update the PATH string with the location of your Flutter SDK directory.

if [ "`id -u`" -eq 0 ]; then

PATH="..."

else

PATH="/usr/local/bin:...:[PATH\_TO\_FLUTTER\_GIT\_DIRECTORY]/flutter/bin"

fi

export PATH

1. End the current session or reboot your system.
2. Once you start a new session, verify that the flutter command is available by running:

$ which flutter

For more details on setting the path in Bash, see [this StackExchange question](https://unix.stackexchange.com/questions/26047/how-to-correctly-add-a-path-to-path) (<https://unix.stackexchange.com/questions/26047/how-to-correctly-add-a-path-to-path>). For information on setting the path in Z shell, see [this StackOverflow question](https://stackoverflow.com/questions/11530090/adding-a-new-entry-to-the-path-variable-in-zsh) (<https://stackoverflow.com/questions/11530090/adding-a-new-entry-to-the-path-variable-in-zsh>).

### **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](https://developer.android.com/studio) from <https://developer.android.com/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. Accept Android licenses.

$ flutter doctor --android-licenses

#### **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) or higher.

1. Enable **Developer options** and **USB debugging** on your device. Detailed instructions are available in the [Android documentation](https://developer.android.com/studio/debug/dev-options) (link here for more information <https://developer.android.com/studio/debug/dev-options>).
2. Using a USB cable, plug your phone into your computer. On your Chromebook, you may see a notification for “USB device detected”. Click on “Connect to Linux” If prompted on your Android device, authorize your computer to access your device.
3. 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\_SDK\_ROOT environment variable to that installation directory.

#### **Deploy to your Chromebook**

With the latest version of Chrome OS, you no longer need to put your device into developer mode to push apps to your Chrome OS device.

1. [Enable ADB](https://support.google.com/chromebook/answer/9770692) (link here for mor information <https://support.google.com/chromebook/answer/9770692>) in Settings. Note that this will require you to reboot your device once.
2. In the Terminal, run flutter devices. If prompted, authorize access to the Android container. Verify that flutter devices lists your Chrome OS device as a recognized device.

### **Flutter & Chrome OS tips & tricks**

For the current versions of Chrome OS, only certain ports from Crostini are exposed to the rest of the environments. Here’s an example of how to launch Flutter DevTools for an Android app with ports that will work:

$ flutter pub global run devtools -p 8000

$ cd path/to/your/app

$ flutter run --observatory-port**=**8080

Then, navigate to http://127.0.0.1:8000/# in your Chrome browser and enter the URL to your application. The last flutter run command you just ran should output a URL similar to the format of http://127.0.0.1:8080/auth\_code=/. Use this URL and select “Connect” to start the Flutter DevTools for your Android app.

##### **Flutter Chrome OS lint analysis**

The Flutter team is adding Chrome OS specific Lint Analysis checks that are available to make sure that the app that you’re building is going to work well on Chrome OS. It looks for things like required hardware in your Android Manifest that aren’t available on Chrome OS devices, permissions that imply requests for unsupported hardware, as well as other properties or code that would bring a lesser experience on these devices.

To activate these, you need to create a new analysis\_options.yaml file in your project folder to include these options. (If you have an existing analysis\_options.yaml file, you can update it)

include: package:flutter/analysis\_options\_user.yaml

analyzer:

optional-checks:

chrome-os-manifest-checks

To run these from the command line, use the following command:

$ flutter analyze

Sample output for this command might look like:

Analyzing ...

warning • This hardware feature is not supported on Chrome OS •

android/app/src/main/AndroidManifest.xml:4:33 • unsupported\_chrome\_os\_hardware

This functionality is still under development, but check back for instructions on how you can make this functionality work with your Chrome OS targeted Flutter app.

# Step 3: Set up your preferred editor

You can build apps with Flutter using any text editor combined with our command-line tools. However, we recommend using one of our editor plugins for an even better experience. These plugins provide you with code completion, syntax highlighting, widget editing assists, run & debug support, and more.

Follow the steps below to add an editor plugin for Android Studio, IntelliJ, VS Code, or Emacs. If you want to use a different editor, that’s OK, skip ahead to the [next step: Test drive](https://flutter.dev/docs/get-started/test-drive).

* [Android Studio and IntelliJ](https://flutter.dev/docs/get-started/editor#androidstudio)
* [Visual Studio Code](https://flutter.dev/docs/get-started/editor#vscode)
* [Emacs](https://flutter.dev/docs/get-started/editor#emacs)

## **Install Android Studio**

Android Studio offers a complete, integrated IDE experience for Flutter.

* [Android Studio](https://developer.android.com/studio), version 3.0 or later

Alternatively, you can also use IntelliJ:

* [IntelliJ IDEA Community](https://www.jetbrains.com/idea/download/) (<https://www.jetbrains.com/idea/download/#section=windows>), version 2017.1 or later
* [IntelliJ IDEA Ultimate](https://www.jetbrains.com/idea/download/) (<https://www.jetbrains.com/idea/download/#section=windows>), version 2017.1 or later

## **Install the Flutter and Dart plugins**

To install these:

1. Start Android Studio.
2. Open plugin preferences (**Configure > Plugins** as of v3.6.3.0 or later).
3. Select the Flutter plugin and click **Install**.
4. Click **Yes** when prompted to install the Dart plugin.
5. Click **Restart** when prompted.

This page describes how to create a new Flutter app from templates, run it, and experience “hot reload” after you make changes to the app.

Select your development tool of choice for writing, building, and running Flutter apps.

* [Android Studio and IntelliJ](https://flutter.dev/docs/get-started/test-drive?tab=androidstudio#androidstudio)
* [Visual Studio Code](https://flutter.dev/docs/get-started/test-drive?tab=androidstudio#vscode)
* [Terminal & editor](https://flutter.dev/docs/get-started/test-drive?tab=androidstudio#terminal)