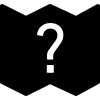
**MOBILE DEVELOPEMENT**

*YOUR DEV TOOLS GUIDE!*

# Flutter, Android Emulator, VS Code

* More info?*

* [Flutter Setup on Windows](https://docs.flutter.dev/get-started/install/windows/mobile)
* [YouTube Installation Guide](https://www.youtube.com/watch?v=ZSWfgxrxN0M)

#### Step 1: Install Flutter SDK

1. **Download and Install Flutter**:
   * Download the Flutter SDK from [Flutter’s official page](https://flutter.dev/docs/get-started/install) and extract it to C:\Flutter.
2. **Update Environment Variables**:
   * Add C:\Flutter\bin to your system **Path** under **Environment Variables**.
3. **Run Flutter Doctor**:
   * Open **Command Prompt** or **PowerShell** and run:
   * flutter doctor
   * This checks for missing dependencies and issues.

#### Step 2: Install Android Studio

1. **Install Android Studio**:
   * Download and install [Android Studio](https://developer.android.com/studio).
2. **Install SDK Tools**:
   * Open **SDK Manager** in Android Studio, go to **SDK Tools**, and install:
     + Android SDK Build Tools
     + Command-line Tools
     + Android Emulator
     + Android SDK Platform-Tools
3. **Accept Android Licenses**:
   * Run:
   * flutter doctor --android-licenses
   * Accept all licenses.

#### Step 3: Install VS Code and Create a Flutter Project

1. **Install VS Code**:
   * Download [VS Code](https://code.visualstudio.com/) and install the **Flutter** and **Dart** extensions from the Extensions tab.
2. **Create a Flutter Project**:
   * In the VS Code terminal, run:
   * flutter create <YourProjectName>

#### Step 4: Run Flutter on Chrome

1. **Open Your Project**:
   * Use the terminal to navigate to your project folder.
2. **Run Your Project**:
   * Run:
   * flutter run
   * Select **Chrome** as the target device.

#### Step 5: Run Flutter on an Android Emulator

1. **Create an Emulator**:
   * Open Android Studio, go to **Device Manager**, and create a new virtual device.
2. **Launch the Emulator**:
   * Start the device, then in VS Code, run:
   * flutter devices
   * Ensure the emulator appears, then run:
   * flutter run

#### Step 6: Hot Reload

1. **Make Edits**:
   * Modify lib/main.dart, then press r in the terminal to apply **Hot Reload**.

#### Troubleshooting

* **Path Issues**: Ensure Flutter and Android SDK paths are added to **Windows Path**.
* **Emulator Not Detected**: Make sure the emulator is running and listed by flutter devices.
* **Licensing Issues**: Re-run flutter doctor --android-licenses if prompted.

#### Conclusion

You’ve successfully set up Flutter, VS Code, and Android Emulator on Windows. You can now develop and test Flutter apps efficiently!

# Flutter & Android SDK

In Flutter development, the **Flutter SDK** and the **Android SDK** serve different purposes, even though they may interact when building apps for Android.

#### Flutter SDK

The **Flutter SDK** is a complete framework for building cross-platform applications (iOS, Android, web, desktop) using the Dart programming language. It includes:

* **Dart SDK**: Provides the Dart language tools, such as the compiler and runtime.
* **Flutter Framework**: Includes a collection of pre-built widgets, libraries, and APIs to build responsive and performant apps.
* **Flutter Tools**: Includes command-line tools (flutter) for managing the development process (e.g., creating projects, running apps, debugging).
* **Rendering Engine**: Uses Skia, a graphics engine, to render UIs directly on the screen.

The Flutter SDK allows you to build apps with a single codebase for multiple platforms, abstracting away platform-specific code.

#### Android SDK

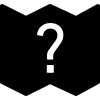
The **Android SDK** is a development kit for building native Android apps, using Java or Kotlin as the primary languages. It includes:

* **Platform-specific APIs**: Libraries and tools to access Android-specific features (e.g., notifications, location services).
* **Android Debug Bridge (ADB)**: A tool to interact with Android devices for debugging.
* **Emulator**: Allows developers to simulate Android devices for testing.
* **Gradle Build System**: Automates building and packaging Android apps.

# 

# Flutter & Android CLI

In the context of Flutter development, the **Flutter CLI** and the **Android CLI** (command-line interface) serve different purposes, although both are essential tools when developing Android apps with Flutter.

* More info?*

Flutter CLI commands: <https://docs.flutter.dev/reference/flutter-cli>

Android Debug Bridge commands: <https://docs.flutter.dev/reference/flutter-cli>

#### Flutter CLI

The **Flutter CLI** is a command-line tool that is part of the Flutter SDK. It is primarily designed to assist in developing cross-platform applications using Flutter.

|  |  |
| --- | --- |
| Flutter CLI | Description |
| **Development** |  |
| flutter create project\_name | Creates a new Flutter project with the default structure. |
| **flutter pub get** | Fetches the dependencies listed in the pubspec.yaml file. |
| **flutter pub upgrade** | update to the latest compatible versions of all the dependencies listed in the pubspec.yaml |
| flutter doctor | Checks your environment for missing dependencies or configuration issues. |
| flutter clean | Cleans the project by removing the build/ directory and caches. |
| Flutter devices | Display all available devices connected to your system |
| flutter run | Runs the Flutter app on the connected emulator or physical device.   * Option: -d flag followed by device ID - specify which device you want to launch the app on |
| **Testing & Debugging:** |  |
| flutter test | Runs all unit and widget tests. |
| flutter analyze | Analyzes the Dart code for potential issues or warnings. |
| **Building** |  |
| flutter build apk | Builds file for Android release. |
| flutter build ios | Builds file for IOS release. |
| **Devices** |  |
| flutter devices | Lists all connected devices (emulators and physical devices). |
| flutter emulators | Lists available emulators to launch |
| flutter emulators --launch emulator\_id | Launches a specific emulator using its ID |

#### Android CLI

The **Android CLI** refers to a set of command-line tools provided by the **Android SDK**. These tools are more focused on Android-specific development and include:

* **avdmanager**: Manages Android Virtual Devices (AVDs) for emulating Android devices.
* **adb**  Communicate with Android devices

|  |  |
| --- | --- |
| ANDROID CLI | Description |
| adb emu kill | Terminate the running Android emulator |

Note: you will need to add Android SDK to your windows environment variable

* + C:\Users\YOUR-USER\AppData\Local\Android\Sdk\platform-tools

# Save time and space with your Flutter projects

During this course, we will work on many small Flutter projects (30)

Creating a full flutter workspace for each project will consume space and take time to execute (download the Gradle dependencies …)

#### Solution

1. Create only 1 flutter workspace
2. Keep your flutter sources files (lib folder and assets) on separate folders
3. You can then switch from project to project by changing the lib and assets folder
   * flutter\_workspace/
   * ├── android/
   * ├── build/
   * ├── ios/
   * ├── ...
   * ├── lib/
   * ├── assets/
   * └── pubspec.yaml
   * flutter\_projects/
   * ├── project1/
   * │ ├── lib/
   * │ ├── assets/
   * │ └── pubspec.yaml
   * ├── project2/
   * │ ├── lib/
   * │ ├── assets/
   * │ └── pubspec.yaml
   * └── ...

#### Note on pubspcec

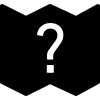
If dependencies are different from project to project, you also need to keep the pubspec.yaml

You can then run the following 3 commands to upgrade, download dependencies and run the project

* + flutter pub upgrade --major-versions
  + flutter pub get
  + flutter run

*What to commit / not to commit?*

When adding a Flutter project to GitHub, it's important to include **only the necessary files** while excluding platform-specific or generated files that aren't needed for collaboration.

* More info?*

<https://dart.dev/guides/libraries/private-files>

#### What to commit?

1. **Project Files**:
   * All the Dart files (lib/ directory) and any other files related to your app’s source code.
   * Your Flutter project configuration files:
     + pubspec.yaml: This file lists all the dependencies and assets used in your Flutter project.
     + pubspec.lock: Contains specific versions of your dependencies.
     + README.md: For project documentation and usage instructions.
     + LICENSE (if you’re applying an open-source license to the project).
     + .gitignore: The Git ignore file (explained below).
     + android/, ios/, and web/ directories (if needed, but without generated files).
2. **Assets**:
   * Any images, fonts, or other assets you’re using in your project (listed in pubspec.yaml).
3. **Configuration Files**:
   * android/app/build.gradle and ios/Runner.xcodeproj: You should include the essential configuration for Android and iOS builds, but avoid uploading the entire build folders.
4. **Tests**:
   * Include any test files from the test/ directory.

#### What not to commit?

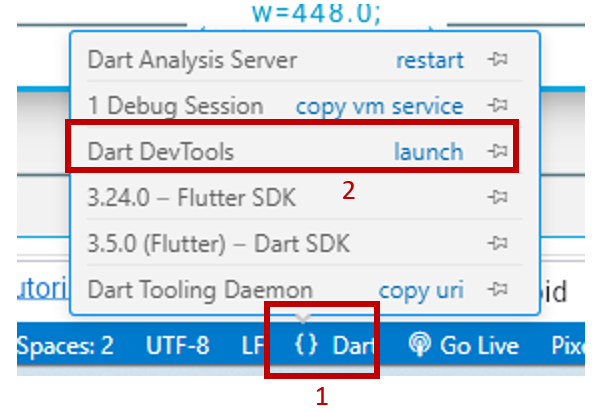
Here is an example of Git ignore file:

* + # Miscellaneous
  + \*.class
  + \*.log
  + \*.pyc
  + \*.swp
  + .DS\_Store
  + .atom/
  + .buildlog/
  + .history
  + .svn/
  + migrate\_working\_dir/
  + # Flutter/Dart/Pub related
  + \*\*/doc/api/
  + \*\*/ios/Flutter/.last\_build\_id
  + .dart\_tool/
  + .flutter-plugins
  + .flutter-plugins-dependencies
  + .pub-cache/
  + .pub/
  + /build/
  + # Symbolication related
  + app.\*.symbols
  + # Obfuscation related
  + app.\*.map.json
  + # Android Studio will place build artifacts here
  + /android/app/debug
  + /android/app/profile
  + /android/app/release

# Performance Tools

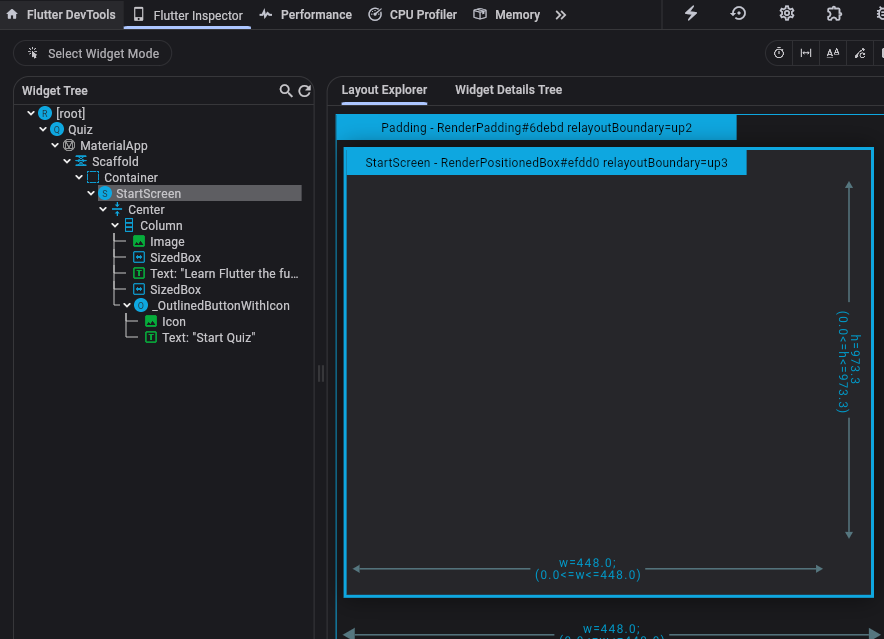
To learn about profiling mobile or desktop apps, [check out this link](https://docs.flutter.dev/tools/devtools).

* **Start** project
  + Open your Flutter project in VS Code.
* **Run** project
  + Go to the Run and Debug tab and select the device/emulator you want to run on.
* **Open** DevTools
  + In the bottom toolbar of VS Code, you should see the Dart: DevTools button.
  + Click on it, and a new tab will open in your browser with Flutter DevTools.
  + *Alternatively, you can open the command palette (Ctrl+Shift+P), then type and select Dart: Open DevTools*

**

*Flutter DevTools can be opened many ways*

* **Monitor** performance: Now, you can monitor and debug your Flutter app using the DevTools in the browser.



*Flutter DevTools can be displayed on browser or within VScode*

# Evaluate yourself on DevTools!!!

This checklist covers the fundamental tools skills you will need to master in Flutter development.

Mastering tools skills is **key to becoming a confident Flutter developer**. You'll be able to efficiently manage projects, write clean, **maintainable** code, and **debug** issues faster. Plus, **testing** and **building** apps for release ensures you're delivering high-quality work!

|  |  |  |
| --- | --- | --- |
| **Project Management** | | |
| Create or Update a Project | Create a new Flutter project.  Update an existing Flutter project when dependencies or configurations change. | 1 point |
| Build project | Use the flutter build command to compile and package the app.  Identify and update deprecated dependencies in pubspec.yaml. | 1 point |
| Check Missing Dependencies | Run flutter doctor to check if the development environment is properly set up (e.g., Android SDK, emulators).  Fix configuration issues as reported by flutter doctor. | 2 point |
| Fetch and Manage Dependencies | Use flutter pub get to fetch dependencies listed in pubspec.yaml.  Understand how to manage package versions and update dependencies. | 2 point |
| **Code Refactoring** | | |
| Extract Widget | Refactor the code by extracting code segments into separate widgets for better readability and reusability. | 1 point |
| Change a Widget Type | [Change an existing widget to another](https://www.youtube.com/watch?v=iqtuxvERRBw) type (e.g., from Container to Column) without breaking functionality. | 1 point |
| Wrap a Widget with Another Widget: | Use VS Code shortcuts to quickly wrap a widget inside another widget (e.g., wrap with Padding, Center, etc.). | 1 point |
| Find References in Code: | Use the “Find References” feature to locate where specific widgets, variables, or functions are used across the project | 1 point |
| **Execution** | | |
| Run a Flutter Project | Execute flutter run to start the app on a connected device or emulator | 1 point |
| Start / Stop an Emulator | Launch an Android emulator from within VS Code.  Stop an emulator or switch between different device profiles as needed. | 1 point |
| Run the Flutter App on a Connected Emulator or Device: | Use VS Code’s built-in controls to run the app on a connected physical device or emulator. | 2 point |
| **Debugging / Monitoring** | | |
| Hot Reload | Perform hot reload using VS Code to see immediate updates without rebuilding the entire app. | 1 point |
| Debug Code | Set breakpoints, step through the code, and inspect variables during runtime using VS Code’s debugging tools. | 1 point |
| Flutter DevTools | Use Flutter DevTools to inspect widget trees, monitor widget states, monitor performances | 1 point |
| Testing |  | 1 point |
| Unit Tests | Use the VS Code terminal or integrated tools to run unit tests and widget tests.  Write and execute tests using flutter test. | 1 point |
| **Build and Deployment** | | |
| Build for Android | Use flutter build APL or flutter build app bundle to build the app for Android release.  Configure app signing and release configurations in build.gradle. | 1 point |
| **Other** | | |
| Working with Git | Integrate Git for version control, commit changes, push to repositories, and collaborate with team members. | 1 point |
| Flutter DevTools Usage | Launch and use Flutter DevTools to inspect widget layouts, track performance, and analyze memory usage. | 1 point |
| **TOTAL** | | **… / 20 points** |

*What is your ToolDev score on 20 points?*