This is unreleased documentation for React Native **Next** version.

For up-to-date documentation, see the **latest version** (0.66).

Version: Next

Setting up the development environment

This page will help you install and build your first React Native app.

If you are new to mobile development, the easiest way to get started is with Expo CLI. Expo is a set of tools built around React Native and, while it has many features, the most relevant feature for us right now is that it can get you writing a React Native app within minutes. You will only need a recent version of Node.js and a phone or emulator. If you'd like to try out React Native directly in your web browser before installing any tools, you can try out Snack.

If you are already familiar with mobile development, you may want to use React Native CLI. It requires Xcode or Android Studio to get started. If you already have one of these tools installed, you should be able to get up and running within a few minutes. If they are not installed, you should expect to spend about an hour installing and configuring them.

| Expo CLI Quickstart | **React Native CLI Quickstart** |

Follow these instructions if you need to build native code in your project. For example, if you are integrating React Native into an existing application, or if you "ejected" from Expo, you'll need this section.

The instructions are a bit different depending on your development operating system, and whether you want to start developing for iOS or Android. If you want to develop for both Android and iOS, that's fine - you can pick one to start with, since the setup is a bit different.

Development OS

macOS   **Windows**   Linux

Target OS

Android   iOS

Installing dependencies

You will need Node, the React Native command line interface, a JDK, and Android Studio.

While you can use any editor of your choice to develop your app, you will need to install Android Studio in order to set up the necessary tooling to build your React Native app for Android.

Node, JDK

We recommend installing Node via Chocolatey, a popular package manager for Windows.

It is recommended to use an LTS version of Node. If you want to be able to switch between different versions, you might want to install Node via nvm-windows, a Node version manager for Windows.

React Native also requires Java SE Development Kit (JDK), which can be installed using Chocolatey as well.

Open an Administrator Command Prompt (right click Command Prompt and select "Run as Administrator"), then run the following command:

```
choco install -y nodejs-lts openjdk8
```

If you have already installed Node on your system, make sure it is Node LTS 12 or newer. If you already have a JDK on your system, make sure it is version 8 or newer.

> You can find additional installation options on Node's Downloads page.

> If you're using the latest version of Java Development Kit, you'll need to change the Gradle version of your project so it can recognize the JDK. You can do that by going to `{project root folder}\android\gradle\wrapper\gradle-wrapper.properties`

and changing the `distributionUrl` value to upgrade the Gradle version. You can
check out here the lastest releases of Gradle.

Android development environment

Setting up your development environment can be somewhat tedious if you're new to
Android development. If you're already familiar with Android development, there are a few
things you may need to configure. In either case, please make sure to carefully follow the
next few steps.

1. Install Android Studio

Download and install Android Studio. While on Android Studio installation wizard, make
sure the boxes next to all of the following items are checked:

- `Android SDK`
- `Android SDK Platform`
- `Android Virtual Device`
- If you are not already using Hyper-V: `Performance (Intel ® HAXM)` (See here for AMD
  or Hyper-V)

Then, click "Next" to install all of these components.

> If the checkboxes are grayed out, you will have a chance to install these
> components later on.

Once setup has finalized and you're presented with the Welcome screen, proceed to the
next step.

2. Install the Android SDK

Android Studio installs the latest Android SDK by default. Building a React Native app with
native code, however, requires the `Android 10 (Q)` SDK in particular. Additional Android
SDKs can be installed through the SDK Manager in Android Studio.

To do that, open Android Studio, click on "Configure" button and select "SDK Manager".
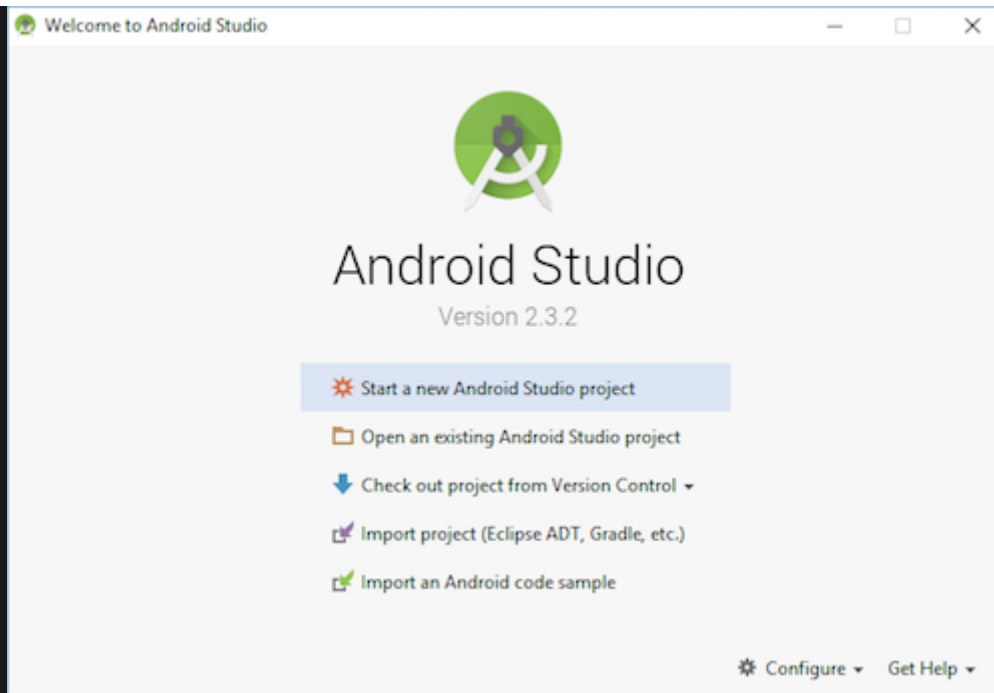
> The SDK Manager can also be found within the Android Studio "Preferences" dialog, under **Appearance & Behavior** → **System Settings** → **Android SDK**.

Select the "SDK Platforms" tab from within the SDK Manager, then check the box next to "Show Package Details" in the bottom right corner. Look for and expand the `Android 10 (Q)` entry, then make sure the following items are checked:

- `Android SDK Platform 29`
- `Intel x86 Atom_64 System Image` or `Google APIs Intel x86 Atom System Image`

Next, select the "SDK Tools" tab and check the box next to "Show Package Details" here as well. Look for and expand the "Android SDK Build-Tools" entry, then make sure that `29.0.2` is selected.
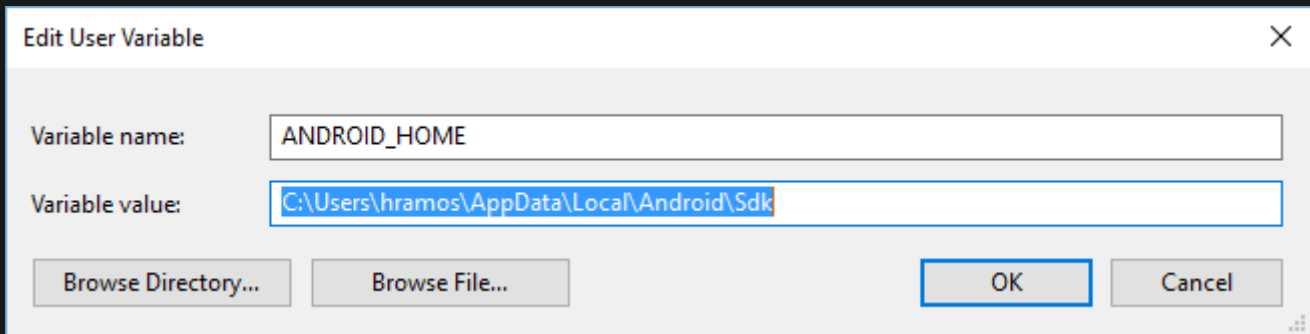
Finally, click "Apply" to download and install the Android SDK and related build tools.

3. Configure the ANDROID_HOME environment variable

The React Native tools require some environment variables to be set up in order to build apps with native code.

1. Open the **Windows Control Panel.**
2. Click on **User Accounts,** then click **User Accounts** again
3. Click on **Change my environment variables**

4. Click on **New...** to create a new `ANDROID_HOME` user variable that points to the path to your Android SDK:

Edit User Variable   ✕

Variable name: ANDROID_HOME

Variable value: C:\Users\hramos\AppData\Local\Android\Sdk

   Browse Directory...   Browse File...   OK   Cancel

The SDK is installed, by default, at the following location:

```
%LOCALAPPDATA%\Android\Sdk
```

You can find the actual location of the SDK in the Android Studio "Settings" dialog, under **Appearance & Behavior → System Settings → Android SDK**.

Open a new Command Prompt window to ensure the new environment variable is loaded before proceeding to the next step.

1. Open powershell

2. Copy and paste **Get-ChildItem -Path Env:** into powershell

3. Verify `ANDROID_HOME` has been added

4. Add platform-tools to Path

1. Open the **Windows Control Panel.**

2. Click on **User Accounts,** then click **User Accounts** again

3. Click on **Change my environment variables**

4. Select the **Path** variable.

5. Click **Edit.**

6. Click **New** and add the path to platform-tools to the list.

The default location for this folder is:

```
%LOCALAPPDATA%\Android\Sdk\platform-tools
```

React Native Command Line Interface

React Native has a built-in command line interface. Rather than install and manage a specific version of the CLI globally, we recommend you access the current version at runtime using `npx`, which ships with Node.js. With `npx react-native <command>`, the current stable version of the CLI will be downloaded and executed at the time the command is run.

Creating a new application

> If you previously installed a global `react-native-cli` package, please remove it as it may cause unexpected issues.

React Native has a built-in command line interface, which you can use to generate a new project. You can access it without installing anything globally using `npx`, which ships with Node.js. Let's create a new React Native project called "AwesomeProject":

```
npx react-native init AwesomeProject
```

This is not necessary if you are integrating React Native into an existing application, if you "ejected" from Expo, or if you're adding Android support to an existing React Native project (see Integration with Existing Apps). You can also use a third-party CLI to init your React Native app, such as Ignite CLI.

[Optional] Using a specific version or template

If you want to start a new project with a specific React Native version, you can use the `--version` argument:

```
npx react-native init AwesomeProject --version X.XX.X
```

You can also start a project with a custom React Native template, like TypeScript, with `--template` argument:

```
npx react-native init AwesomeTSProject --template react-native-template-typescript
```

Preparing the Android device

You will need an Android device to run your React Native Android app. This can be either a physical Android device, or more commonly, you can use an Android Virtual Device which allows you to emulate an Android device on your computer.

Either way, you will need to prepare the device to run Android apps for development.

Using a physical device

If you have a physical Android device, you can use it for development in place of an AVD by plugging it in to your computer using a USB cable and following the instructions here.

Using a virtual device

If you use Android Studio to open `./AwesomeProject/android`, you can see the list of available Android Virtual Devices (AVDs) by opening the "AVD Manager" from within Android Studio. Look for an icon that looks like this:

If you have recently installed Android Studio, you will likely need to create a new AVD. Select "Create Virtual Device...", then pick any Phone from the list and click "Next", then select the **Q** API Level 29 image.

> If you don't have HAXM installed, click on "Install HAXM" or follow these instructions to set it up, then go back to the AVD Manager.

Click "Next" then "Finish" to create your AVD. At this point you should be able to click on the green triangle button next to your AVD to launch it, then proceed to the next step.

Running your React Native application

Step 1: Start Metro

First, you will need to start Metro, the JavaScript bundler that ships with React Native. Metro "takes in an entry file and various options, and returns a single JavaScript file that includes all your code and its dependencies."—Metro Docs

To start Metro, run `npx react-native start` inside your React Native project folder:

```
npx react-native start
```

`react-native start` starts Metro Bundler.

> If you use the Yarn package manager, you can use `yarn` instead of `npx` when running React Native commands inside an existing project.

> If you're familiar with web development, Metro is a lot like webpack—for React Native apps. Unlike Kotlin or Java, JavaScript isn't compiled—and neither is React Native. Bundling isn't the same as compiling, but it can help improve startup performance and translate some platform-specific JavaScript into more widely supported JavaScript.

Step 2: Start your application

Let Metro Bundler run in its own terminal. Open a new terminal inside your React Native project folder. Run the following:

```
npx react-native run-android
```

If everything is set up correctly, you should see your new app running in your Android emulator shortly.
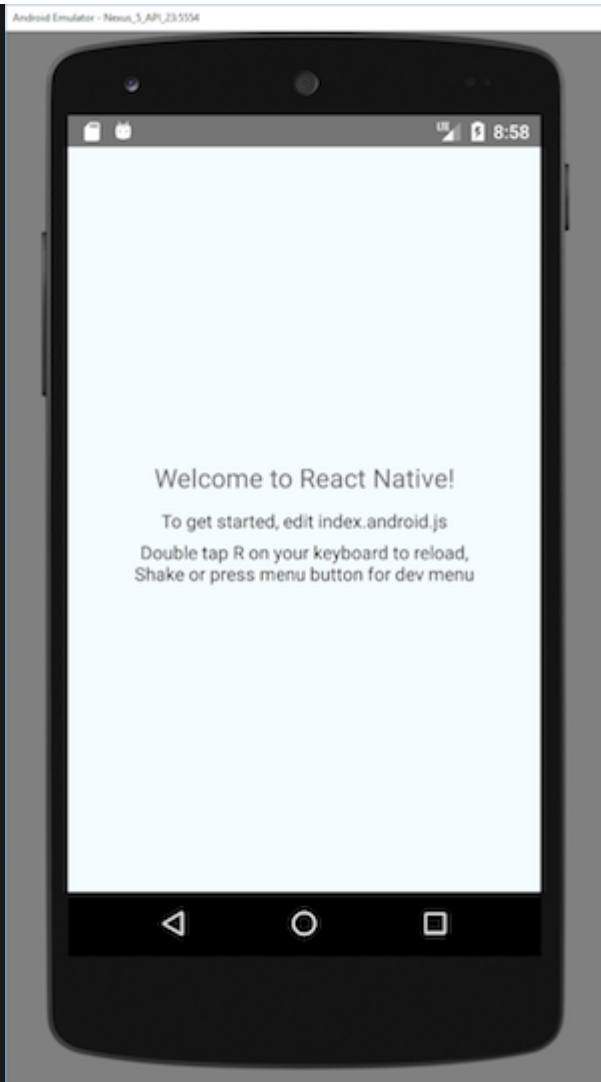
`npx react-native run-android` is one way to run your app - you can also run it directly from within Android Studio.

> If you can't get this to work, see the Troubleshooting page.

Modifying your app

Now that you have successfully run the app, let's modify it.

- Open `App.js` in your text editor of choice and edit some lines.
- Press the `R` key twice or select `Reload` from the Developer Menu (`Ctrl + M`) to see your changes!

That's it!

Congratulations! You've successfully run and modified your first React Native app.

Now what?

- If you want to add this new React Native code to an existing application, check out the Integration guide.

If you're curious to learn more about React Native, check out the Introduction to React Native.

Is this page useful? 👍 👎

✏️ Edit this page

Last updated on 7/22/2021