Setup guide for React Native development (macOS)

- Clone Gutenberg
  - Install node and npm
  - Do you have an older existing Gutenberg checkout?
- iOS
  - CocoaPods
    - System Ruby
    - Ruby Manager
  - Set up Xcode
  - react-native doctor
  - Run the demo app
- Android
  - Java Development Kit (JDK)
  - Set up Android Studio
  - Update Paths
  - Create a new device image
  - Run the demo app
- Unit Tests
- Integration Tests
  - iOS Integration Tests
  - Android Integration Tests

Are you interested in contributing to the native mobile editor? This guide is a detailed walk through designed to get you up and running!

Note that these instructions are primarily focused on the macOS environment. For other environments, the React Native quickstart documentation has helpful pointers and steps for getting

set up.

Clone Gutenberg

```
git clone git@github.com:WordPress/gutenberg.git
```

Install node and npm

If you're working in multiple JS projects, a node version manager may make sense. A manager will let you switch between different node and npm versions of your choosing.

We recommend nvm.

After installing nvm, run the following from the top-level directory of the cloned project:

```
nvm install 'lts/*'
nvm alias default 'lts/*' # sets this as the default when opening a new terminal
nvm use # switches to the project settings
```

Then install dependencies:

```
npm ci
```

Do you have an older existing Gutenberg checkout?

If you have an existing Gutenberg checkout be sure to fully clean `node_modules` and re-install dependencies.
This may help avoid errors in the future.

```
npm run distclean
npm ci
```

iOS

CocoaPods

CocoaPods is required to fetch React and third-party dependencies. The steps to install it vary depending on how Ruby is managed on your machine.

System Ruby

If you're using the default Ruby available with MacOS, you'll need to use the `sudo` command to install gems like Cocoapods:

```
sudo gem install cocoapods
```

Note, Mac M1 is not directly compatible with Cocoapods. If you encounter issues, try running these commands to install the ffi package, which will enable pods to be installed with the proper architecture:

```
sudo arch -x86_64 gem install ffi
arch -x86_64 pod install
```

Ruby Manager

It may not be necessary to manually install Cocoapods or the `ffi` package if you're using a Ruby Version manager. Please refer to your chosen manager's documentation for guidance.

`rbenv` is the recommended manager if you're running Gutenberg from within the WordPress iOS app (vs. only the demo app).

Set up Xcode

Install Xcode via the app store and then open it up:

- Accept the license agreement.
- Verify that `Xcode > Preferences > Locations > Command Line Tools` points to the current Xcode version.

Screenshot of Xcode command line tools settings.

react-native doctor

react-native doctor can be used to identify anything that's missing from your development environment. From your Gutenberg checkout, or relative to `/packages/react-native-editor` folder, run:

```
npx @react-native-community/cli doctor
```

Screenshot of react-native-community/cli doctor tool running in the terminal.

See if `doctor` can fix both "common" and "iOS" issues. (Don't worry if "Android" still has s at this stage, we'll get to those later!)

Run the demo app

Once all common and iOS issues are resolved, try:

```
npm run native start:reset #starts metro
```

In another terminal type:

```
npm run native ios
```

After waiting for everything to build, the demo app should be running from the iOS simulator:

Screenshot of the block editor in iOS simulator.

Android

Java Development Kit (JDK)

The JDK recommended in the React Native documentation is called Azul Zulu. It can be installed using Homebrew. To install it, run the following commands in a terminal after installing Homebrew:

```
brew tap homebrew/cask-versions
brew install --cask zulu11
```

If you already have a JDK installed on your system, it should be JDK 11 or newer.

Set up Android Studio

To compile the Android app, download Android Studio.

Next, open an existing project and select the Gutenberg folder you cloned.

From here, click on the cube icon that's highlighted in the following screenshot to access the SDK Manager. Another way to the SDK Manager is to navigate to `Tools > SDK Manager`:

Screenshot highlighting where the package manager button is located in Android Studio.

We can download SDK platforms, packages and other tools on this screen. Specific versions are hidden behind the "Show package details" checkbox, check it, since our build requires specific versions for E2E and development:

Screenshot of the package manager in Android Studio, highlighting the Show Package Details check

Check all related packages from build.gradle. Then click on "Apply" to download items. There may be other related dependencies from build.gradle files in node_modules.

If you don't want to dig through files, stack traces will complain of missing packages, but it does take quite a number of tries if you go through this route.

Screenshot of the build.gradle configuration file.
Screenshot of the package manager displaying SDK Platforms.
Screenshot of the package manager displaying SDK Tools.

Update Paths

Export the following env variables and update $PATH. We can normally add this to our `~/.zshrc` file if we're using zsh
in our terminal, or `~/.bash_profile` if the terminal is still using bash.

```
### Java that comes with Android Studio:
export JAVA_HOME=/Applications/Android\ Studio.app/Contents/jre/Contents/Home
### Android Home is configurable in Android Studio. Go to Preferences > System Settings >
Android SDK
export ANDROID_HOME=$HOME/Library/Android/sdk
export PATH=$PATH:$ANDROID_HOME/emulator
export PATH=$PATH:$ANDROID_HOME/tools
export PATH=$PATH:$ANDROID_HOME/tools/bin
export PATH=$PATH:$ANDROID_HOME/platform-tools
```

Save then source, or open a new terminal to pick up changes.

```
source ~/.zshrc
```

or

```
source ~/.bash_profile
```

If the SDK path can’t be found, you can verify its location by visiting Android Studio > Preferences > System Settings > Android SDK

Screenshot of where the SDK Path may be found in Android Studio.

Create a new device image

Next, let’s create a virtual device image. Click on the phone icon with an android to the bottom-right.

Screenshot of where to find the android device manager button.

This brings up the “Android Virtual Device Manager” or (AVD). Click on “Create Virtual Device”. Pick a phone type of your choice:

Screenshot of the Virtual Device Configuration setup.

Pick the target SDK version. This is the targetSdkVersion set in the build.gradle file.

Screenshot of picking a system image in the Android Device Manager workflow.

There are some advanced settings we can toggle, but these are optional. Click finish.

Run the demo app

Start metro:

```
npm run native start:reset
```

In another terminal run the following to launch the demo app in the Android emulator (if the emulator isn’t already running, it’ll also be launched as part of this command):

```
npm run native android
```

After a bit of a wait, we’ll see something like this:

Screenshot of a the block editor in Android Simulator.

Unit Tests

```
npm run test:native
```

Integration Tests

Appium has it own doctor tool. Run this with:

```
npx appium-doctor
```

Screenshot of the appium-doctor tool running in the terminal.

Resolve any required dependencies.

iOS Integration Tests

If we know we can run the iOS local environment without issue, E2Es for iOS are straightforward. Stop any running metro processes. This was launched previously with `npm run native start:reset`.

Then in terminal type:

```
npm run native test:e2e:ios:local
```

Passing a filename should also work to run a subset of tests:

```
npm run native test:e2e:ios:local gutenberg-editor-paragraph.test.js
```

If all things go well, it should look like:

Android Integration Tests

Create a new virtual device() that matches the device specified in packages/react-native-editor/**device-tests**/helpers/caps.js At the time of this writing, this would be a Pixel 3 XL image, using Android 9 (API 28).

Start the virtual device first. Go back to the AVD by clicking on the phone icon, then click the green play button.

A screenshot of how to start the Android Simulator.

Make sure no metro processes are running. This was launched previously with `npm run native start:reset`.

Then in a terminal run:

```
npm run native test:e2e:android:local
```

Passing a filename should also work to run a subset of tests:

```
npm run native test:e2e:android:local gutenberg-editor-paragraph.test.js
```

After a bit of a wait we should see:

A screenshot of block editor integration tests in Android Simulator.

□□□ #2
□ □□□ □□□ 2026-10-31 18:27:05 UTC
□ □□□ □□□ 2026-10-31 18:27:07 UTC