Android Development Laptop Setup Guide

1. Purpose

This document describes how to prepare a laptop for Android application development using Android Studio, Kotlin, and modern development tools.[1][2]

2. Hardware Requirements

- **Processor**: 64-bit CPU with virtualization support (Intel VT-x or AMD-V); modern multi-core processors (Intel Core i5/i7 or AMD Ryzen 5/7) recommended.[1]
- **RAM**: Minimum 8 GB (16 GB recommended; 32 GB ideal for comfortable development with multiple emulators).[1][3]
- **Storage**: SSD with at least 32–50 GB free space for Android Studio, SDK, emulators, and project files.[1][2]
- **Display**: Minimum 1920×1080 resolution for comfortable layout editing and tool visibility.[1]
- **Operating System**: 64-bit Windows 10/11, recent macOS, or modern 64-bit Linux distribution.[1][2]

3. Base Software Installation

1. Update the operating system to the latest stable version, including security patches and drivers.[1]
2. Install a Java Development Kit (JDK); Android Studio bundles a compatible JDK, but JDK 17 or later is recommended for system-wide tools.[2]
3. Install Git from the official distribution for your operating system for version control integration.[4]
4. Verify installations:
   - `java -version` → confirms JDK installation
   - `git --version` → confirms Git installation[1]

4. Download and Install Android Studio

1. Download the latest Android Studio (Otter or newer) from the official Android Developers website.[1][5]
2. Run the installer and follow the setup wizard on your operating system.[1]

3. During initial launch, Android Studio will prompt to install the Android SDK, platform tools, and system images.
4. Accept the license agreements and allow the download of necessary components (this may take 10–30 minutes depending on internet speed).[1][2]

5. Configure Android SDK and Tools

1. Open Android Studio and navigate to **Tools → SDK Manager**.
2. Install the following components:
   - **SDK Platforms**: Android API levels 24–35 (covering most active devices)[1]
   - **SDK Tools**: Android SDK Build-Tools, Android Emulator, Android SDK Platform-Tools[1]
   - **Google APIs and Play Services**: For access to Google services in apps[1]
3. Verify the SDK location is set correctly (typically `~/Android/Sdk` on macOS/Linux or `%APPDATA%\Android\sdk` on Windows).[1]

6. Create and Configure Virtual Devices (AVD)

1. In Android Studio, open **Tools → Device Manager** or **AVD Manager**.
2. Create at least one Android Virtual Device:
   - Select a recent Android API level (API 31 or higher)[1]
   - Choose x86_64 or ARM64 architecture with hardware acceleration enabled[1]
   - Allocate sufficient RAM (2–4 GB) and storage (2–4 GB)[1]
3. Test the emulator by clicking **Play** and ensuring the Android home screen launches.[1]
4. Optionally, prepare a second AVD for testing older Android versions (API 24–28).[2]

7. Install and Configure Git

1. Verify Git installation: `git --version`
2. Configure user identity globally:

   git config --global user.name "Your Name"
   git config --global user.email "your.email@example.com"

3. Generate an SSH key for secure repository access:

   ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa

4. Add the public key to your Git hosting service (GitHub, GitVerse, Bitbucket, etc.).[4]
5. Test SSH connectivity: `ssh -T git@github.com` (or your Git server).[4]

8. Performance Tuning for Android Studio

1. **Memory Configuration**: Edit `studio.vmoptions` or use IDE settings to increase heap size:
   - For 16 GB RAM: set Xmx to 4096 MB
   - For 32 GB RAM: set Xmx to 6144 MB[1]
2. **Gradle Configuration**: In your project root, create or edit `gradle.properties`:

   ```
   org.gradle.jvmargs=-Xmx4096m
   org.gradle.parallel=true
   org.gradle.caching=true
   ```

3. **Enable Hardware Acceleration**: Ensure GPU acceleration is enabled in emulator settings and IDE graphics rendering.[1]
4. **Disable Unnecessary Plugins**: In **File → Settings → Plugins**, disable unused plugins to reduce IDE overhead.[1]

9. Create Your First Android Project

1. Launch Android Studio and click **New Project**.
2. Select a project template (e.g., Empty Activity or Compose-based template).[2]
3. Configure project settings:
   - **Name**: Project name (e.g., `MyFirstApp`)
   - **Package Name**: Reverse domain notation (e.g., `com.example.myfirstapp`)[2]
   - **Save Location**: Choose an SSD location with sufficient free space
   - **Language**: Select Kotlin (recommended for modern Android development)[2]
   - **Minimum SDK**: API 24 or higher (covers ~95% of active devices)[1]
4. Complete the wizard and allow Gradle to sync the project.
5. Run the app on a virtual device: click **Run** > **Run 'app'** and select your AVD.[1]

10. Connect a Physical Android Device

1. Enable **Developer Options** on your Android device:
   - Go to **Settings → About Phone**
   - Tap **Build Number** seven times
   - Return to Settings and open **Developer Options**[1]

2. Enable **USB Debugging** in Developer Options.[1]
3. Connect the device to your laptop via USB cable.
4. Accept the RSA fingerprint prompt on your device when prompted.
5. Verify the device appears in Android Studio's **Device Manager** or **Run Configurations**.[1]
6. Deploy and test your app directly on the physical device.[1]

11. Git Integration and Version Control

1. In your Android project, initialize Git: `git init` (or clone an existing repository).[4]
2. Configure remote repository:

   ```
   git remote add origin git@github.com:username/project.git
   ```
3. In Android Studio, enable VCS integration: **File → Settings → Version Control** and verify Git is configured.
4. Use Android Studio's built-in VCS tools to commit, branch, and push changes directly from the IDE.[4]
5. Create a `.gitignore` file for Android projects to exclude build artifacts, local configurations, and sensitive files:

   ```
   .gradle
   .idea
   local.properties
   *.apk
   build/
   ```

12. Testing and Debugging

1. **Unit Testing**: Use JUnit and Mockito for Java/Kotlin unit tests; run tests via `gradlew test`.[1]
2. **Instrumented Testing**: Use Espresso or Compose Testing for Android UI tests; run via `gradlew connectedAndroidTest`.[1]
3. **Debugging**: Set breakpoints in Kotlin code, run the app in debug mode, and use the Android Studio debugger to inspect variables and step through code.[1]
4. **Profiling**: Use **Profiler** tools in Android Studio to monitor CPU, memory, disk I/O, and network performance.[1]

13. Publishing and Build Variants

1. Create a **Keystore** for signing release builds (required for production):

keytool -genkey -v -keystore my-release-key.keystore -keyalg RSA -keysize 2048 -validity 10000

2. Configure release build signing in `build.gradle` or `build.gradle.kts`.[1]
3. Build a release APK or App Bundle: **Build → Build Bundle(s)/APK(s)** → select desired format.[1]
4. Test the signed build on a device before publishing to Google Play Store or other distribution channels.[1]

14. Continuous Learning and Resources

- **Official Android Documentation**: https://developer.android.com[1]
- **Kotlin Language Guide**: https://kotlinlang.org/docs[2]
- **Android Stack Overflow**: Search common issues and solutions[1]
- **GitHub**: Explore open-source Android projects for best practices and examples[4]
- **IDE Documentation**: Use Android Studio's built-in help (**Help → Android Studio Help**)

References

[1] Android Developers. (2025). Install Android Studio. Google. https://developer.android.com/studio/install

[2] Android Developers. (2025). Create Your First Android App in Kotlin. Google. https://developer.android.com/codelabs/basic-android-kotlin-compose-install-android-studio

[3] Stack Overflow Community. (2015). Minimum hardware requirement for Android Studio. https://stackoverflow.com/questions/28746027/minimum-hardware-requirement-for-android-studio

[4] JetBrains. (2025). Version Control with Git. IntelliJ IDEA Documentation. https://www.jetbrains.com/help/idea/version-control-integration.html

[5] Android Developers. (2025). Android Studio Otter | 2025.2.1 Release. Google. https://developer.android.com/studio/releases