How to make Android apps without IDE from command line

Authmane Terki Follow
Nov 26, 2017 · 5 min read

A HelloWorld without Android Studio

Update: I’ve made a new course that explain how you can avoid Android Studio and Gradle, but still use IntelliJ iDE:

How to do Android development faster without Gradle
IntelliJ IDE, but not Gradle
medium.com

Hello everybody!

In this tutorial, I will show you how you can build/compile an APK (an Android app) from your java code using terminal (on Linux) without IDE or in other words without Android Studio. At the end, I will also show you a script to automate the process. In this example, I will use Android API 19 (4.4 Kitkat) to make a simple HelloWorld. I want to say that I will do this tutorial without android command which is deprecated.

1. Install Java

First, you need to install java, in my case, I install the headless version because I don’t use graphics (only command line):

```
sudo apt-get install openjdk-8-jdk-headless
```

2. Install all SDK tools

Then download the last SDK tools of Android which you can find here:

Download Android Studio and SDK Tools | Android Studio
Download the official Android IDE and developer tools to build apps for Android phones, tablets, wearables, TVs, and…
developer.android.com

Or simply do:

```
wget https://dl.google.com/android/repository/sdk-tools-linux-3859397.zip
```

I recommend to unzip it in the /opt directory inside another directory that we will call “android-sdk”:

```
mkdir -p /opt/android-sdk
unzip sdk-tools-linux-3859397.zip -d /opt/android-sdk
```

Now, we have to install platform tools (which contain ADB), an Android API and build tools.

In fact, if you are on Debian, you can avoid installing platform-tools package and only install ADB like that:

```
sudo apt-get install adb
```

3. Code the application

In this example, I want to compile a simple HelloWorld. So, first, we need to make a project directory:

```
mkdir HelloAndroid
cd HelloAndroid
```

Then we have to make the files tree:

```
mkdir -p src/com/example/helloandroid
mkdir obj
mkdir bin
mkdir -p res/layout
mkdir res/values
mkdir res/drawable
```

If you use exernal libraries (.jar files), also make a folder for them:

```
mkdir libs
```

You have an example here:

How to use JavaMail on Android (without Gradle)

Hello guys!

medium.com

Make the file src/com/example/helloandroid/MainActivity.java and put that inside:

```java
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Make the strings.xml file in the res/values folder. It contains all the text that your application uses:

```
<resources>
   <string name="app_name">A Hello Android</string>
   <string name="hello_msg">Hello Android!</string>
   <string name="menu_settings">Settings</string>
   <string name="title_activity_main">MainActivity</string>
</resources>
```

The activity_main.xml is a layout file which have to be in res/layout:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android" xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:text="@string/hello_msg"
      tools:context=".MainActivity" />
</RelativeLayout>
```

You also have to add the file AndroidManifest.xml at the root:

```
<?xml version='1.0'?>
<manifest xmlns:a='http://schemas.android.com/apk/res/android' package='com.example.helloandroid' a:versionCode='0' a:versionName='0'>
    <application a:label='A Hello Android'>
        <activity a:name='com.example.helloandroid.MainActivity'>
             <intent-filter>
```

```
            <intent-filter>
                <category a:name='android.intent.category.LAUNCHER'/>
                <action a:name='android.intent.action.MAIN'/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

4. Build the code

Now, I recommend to store the project path in a variable:

```
export PROJ=path/to/HelloAndroid
```

First, we need generate the R.java file which is necessary for our code:

```
cd /opt/android-sdk/build-tools/26.0.1/
./aapt package -f -m -J $PROJ/src -M $PROJ/AndroidManifest.xml -S $PROJ/res -I /opt/android-sdk/platforms/android-19/android.jar
```

- -m instructs aapt to create directories under the location specified by -J
- -J specifies where the output goes. Saying -J src will create a file like src/com/example/helloandroid/R.java
- -S specifies where is the res directory with the drawables, layouts, etc.
- -I tells aapt where the android.jar is. You can find yours in a location like android-sdk/platforms/android-<API level>/android.jar

Now, we have to compile the .java files:

```
cd /path/to/AndroidHello
javac -d obj -classpath src -bootclasspath /opt/android-sdk/platforms/android-19/android.jar src/com/example/helloandroid/*.java
```

If you have use an external, add it the classpath:

```
javac -d obj -classpath "src:libs/<your-lib>.jar" -bootclasspath /opt/android-sdk/platforms/android-19/android.jar src/com/example/helloandroid/*.java
```

The compiled .class files are in obj folder, but Android can’t read them. We have to translate them in a file called “classes.dex” which will be read by the dalvik Android runtime:

```
cd /opt/android-sdk/build-tools/26.0.1/
./dx --dex --output=$PROJ/bin/classes.dex $PROJ/obj
```

But if you use external libraries, do rather:

```
./dx --dex --output=$PROJ/bin/classes.dex $PROJ/*.jar $PROJ/obj
```

If you have the error `UNEXPECTED TOP-LEVEL EXCEPTION`, it can be because you use old build tools and DX try to translate java 1.7 rather than 1.8. To solve the problem, you have to specify 1.7 java version in the previous javac command:

```
cd /path/to/AndroidHello
javac -d obj -source 1.7 -target 1.7 -classpath src -bootclasspath /opt/android-sdk/platforms/android-19/android.jar
src/com/example/helloandroid/*.java
```

The -source option specify the java version of your source files. Note that we can use previous versions of Java even we use OpenJDK 8 (or 1.8).

We can now put everything in an APK:

```
./aapt package -f -m -F $PROJ/bin/hello.unaligned.apk -M $PROJ/AndroidManifest.xml -S $PROJ/res -I /opt/android-
sdk/platforms/android-19/android.jar
cp $PROJ/bin/classes.dex .
./aapt add $PROJ/bin/hello.unaligned.apk classes.dex
```

Be aware: until now, we used three AAPT commands, the first and the second one are similar but they don't do the same. You have to copy the classes.dex file at the root of project like above! Otherwise, AAPT won't put this file at right place in the APK archive (because an APK is like a .zip file).

The generated package can't be installed by Android because it's unaligned and unsigned.

If you want, you can check the content of the package like this:

```
./aapt list $PROJ/bin/hello.unaligned.apk
```

5. Sign the package

To do so, we firstly create a new keystore with the command keytool given by Java:

```
keytool -genkeypair -validity 365 -keystore mykey.keystore -keyalg RSA -keysize 2048
```

Just answer the questions and put a password.

You can sign an APK like this:

```
./apksigner sign --ks mykey.keystore $PROJ/bin/hello.apk
```

Note that apksigner only exist since Build Tools 24.0.3.

6. Align the package

It’s as simple as that:

```
./zipalign -f 4 $PROJ/bin/hello.unaligned.apk $PROJ/bin/hello.apk
```

Alignment increase the performance of the application and may reduce memory use.

7. Test the application

To test the application, connect your smartphone with a USB cable and use ADB:

```
adb install $PROJ/bin/hello.apk
adb shell am start -n com.example.helloandroid/.MainActivity
```

But before run this command, I recommend to run this one:

```
adb logcat
```

If there is an error during installation or running, you see it with that command.

8. Make a script

If you don’t want to run all these steps every time you would like to compile your app, make a script! Here’s mine:

```
#!/bin/bash

set -e

AAPT="/path/to/android-sdk/build-tools/23.0.3/aapt"
DX="/path/to/android-sdk/build-tools/23.0.3/dx"
ZIPALIGN="/path/to/android-sdk/build-tools/23.0.3/zipalign"
APKSIGNER="/path/to/android-sdk/build-tools/26.0.1/apksigner" # /!\ version 26
PLATFORM="/path/to/android-sdk/platforms/android-19/android.jar"

echo "Cleaning..."
rm -rf obj/*
rm -rf src/com/example/helloandroid/R.java

echo "Generating R.java file..."
$AAPT package -f -m -J src -M AndroidManifest.xml -S res -I $PLATFORM

echo "Compiling..."
javac -d obj -classpath src -bootclasspath $PLATFORM -source 1.7 -target 1.7 src/com/example/helloandroid/MainActivity.java
javac -d obj -classpath src -bootclasspath $PLATFORM -source 1.7 -target 1.7 src/com/example/helloandroid/R.java

```

```
echo "Translating in Dalvik bytecode..."
$DX --dex --output=classes.dex obj

echo "Making APK..."
$AAPT package -f -m -F bin/hello.unaligned.apk -M AndroidManifest.xml -S res -I $PLATFORM
$AAPT add bin/hello.unaligned.apk classes.dex

echo "Aligning and signing APK..."
$APKSIGNER sign --ks mykey.keystore bin/hello.unaligned.apk
$ZIPALIGN -f 4 bin/hello.unaligned.apk bin/hello.apk

if [ "$1" == "test" ]; then
	echo "Launching..."
	adb install -r bin/hello.apk
	adb shell am start -n com.example.helloandroid/.MainActivity
fi
```

build.sh hosted with ❤ by **GitHub** **view raw**

To use it, run:

```
cd /path/to/AndroidHello
./build.sh test
```

Notes

- You can remove “test” if you just want to compile without testing.
- This script **only compile and run the app** on the phone. But I can also make a script to automatically generate a new project like this one. I think I have a good idea to do so, but **I need to know if you are interested**. If it’s the case, please leave a comment or send me an e-mail.
- I can also complete the script for external libraries. Likewise, let me know if you want this.

If you have any questions, don’t hesitate to ask them below or by e-mail ;-)! **EDIT:** Well I’m very busy actually…