Capítulo 3

Ferramentas de Linha de Comando Android

Mesmo que uma IDE, como o Eclipse (entre outros), possa abstrair os passos demonstrados nesta seção, é interessante que seja entendido o processo de criação de uma aplicação a partir das ferramentas disponibilizadas pelo Android SDK.

Dentre outros benefícios que possam ser obtidos com este conhecimento, destacam-se a possibilidade de entender melhor os erros que ocorrem durante o processo de desenvolvimento e a possibilidade de criação de scripts que possibilitem uma maior produtividade no desenvolvimento.

Primeiramente será demonstrado como cada ferramenta disponibilizada pelo Android SDK poderá auxiliar na construção de uma aplicação e posteriormente será visto como o Ant poderá auxiliar a melhorar este processo.

Para iniciar é necessário haver as seguinte variáveis de ambiente JAVA_HOME e ANDROID_HOME, onde a primeira aponta para o diretório de instalação do JDK e o segundo aponta para o diretório do SDK do Android.

O procedimento para isto depende do sistema operacional e pode ser feito diretamente nos terminais. Segue alguns exemplos:

```
#Para Windows
export JAVA_HOME="C:\Arquivos de Programas\Java\jdk"
export ANDROID_HOME="C:\Arquivos de Programas\AndroidSDK"

#Para Linux
export JAVA_HOME=/usr/lib/jvm/java-7-oracle
export ANDROID_HOME=/usr/local/android-sdk-linux

#Para OSX
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.7.0_75.jdk/Contents/Home
export ANDROID_HOME=/usr/local/Cellar/android-sdk/
```

3.1 Principais Ferramentas

adb adb logcat android emulator

3.2 Criação uma aplicação do zero

Antes de construir uma aplicação é necessário conhecer quais são os dispositivos e APIs disponíveis. Para isto pode-se adotar o seguinte comando:

```
#No windows
#Para listar api's
%ANDROID_HOME%\tools\android list target
#Para listar discos
%ANDROID_HOME%\tools\android list avd
#Para listar tipos de dispositivos
%ANDROID_HOME%\tools\android list device

#No linux e osx
#Para listar api's
$ANDROID_HOME/tools/android list target
#Para listar discos
$ANDROID_HOME/tools/android list avd
#Para listar tipos de dispositivos
$ANDROID_HOME/tools/android list device
```

Para construir um novo projeto pode-se adotar a ferramenta **android** utilizando o seguinte comando:

```
#No windows
%ANDROID_HOME%\tools\android create project --target android-8
--name MyAndroidApp  --path ./MyAndroidApp --activity MainActivity
--package br.edu.ifpb.pdm.example

#No linux e osx
$ANDROID_HOME/tools/android create project \
--target android-8 \
--name MyAndroidApp \
--path ./MyAndroidApp \
--activity MainActivity \
--package br.edu.ifpb.pdm.example
```

Adicionalmente, para obter uma melhor organização dos arquivos, pode-se criar duas pastas adicionais: java-bin e release. O primeiro será usado para armazenar os arquivos resultantes da compilação dos arquivos java, usando a ferramenta javac, e o segundo será usado para armazenar os pacotes que serão implantados nos dispostivos android.

O projeto criado com este comando possui um display, por padrão, que pode ser testado no dispositivo, mas antes disto é necessário criar a classe R que representa os recursos desta aplicação. Isto é feito utilizando o seguinte comando:

```
#No windows
%ANDROID_HOME%\build-tools\22.0.1\aapt package -v -f -m
-S res -J src -M AndroidManifest.xml
-I %ANDROID_HOME%/platforms/android-8/android.jar
```

```
#No linux e osx
$ANDROID_HOME/build-tools/22.0.1/aapt package -v -f -m \
-S res \
-J src \
-M AndroidManifest.xml \
-I $ANDROID_HOME/platforms/android-8/android.jar
```

Agora, basta seguir o fluxo apresentado na figura 2.2, fazendo a compilação de códigos java, depois compilando para dex e finalmente empacontando o projeto.

O comando abaixo realiza a compilação do código em Java:

```
#No windows
%JAVA_HOME%\bin\javac -verbose -d java-bin
-classpath $ANDROID_HOME/platforms/android-8/android.jar;java-bin
-sourcepath src src/br/edu/ifpb/pdm/example/MainActivity.java


#No linux e osx
$JAVA_HOME/bin/javac -verbose -d java-bin \
-classpath $ANDROID_HOME/platforms/android-8/android.jar:java-bin \
-sourcepath src src/br/edu/ifpb/pdm/example/MainActivity.java
```

O comando abaixo realiza a compilação do código em DEX:

```
#No windows
%ANDROID_HOME%\build-tools\22.0.1\dx --dex --verbose
--output=bin/classes.dex java-bin libs


#No linux e osx
$ANDROID_HOME/build-tools/22.0.1/dx --dex --verbose \
--output=bin/classes.dex java-bin libs
```

O comando abaixo realiza o empacotamento dos códigos e cria um arquivo .apk:

```
#No windows
$ANDROID_HOME/build-tools/22.0.1/aapt package -v -f -m
-S res -M AndroidManifest.xml
-I $ANDROID_HOME/platforms/android-8/android.jar
-F release/MyAndroidApp.unsigned.apk bin

#No linux e osx
$ANDROID_HOME/build-tools/22.0.1/aapt package -v -f -m \
-S res \
-M AndroidManifest.xml \
-I $ANDROID_HOME/platforms/android-8/android.jar \
-F release/MyAndroidApp.unsigned.apk \
bin
```

Por que? Agora, de posse de um pacote .apk, será necessário fazer a assinatura do pacote, resultando em um pacote .apk assinado. Para tanto deve-se adotar o mesmo procedimento de assinatura de pacote .jar, com a ferramenta jarsigner, conforme apresentado abaixo:

```
#No windows
%JAVA_HOME%\bin\jarsigner -verbose -keystore -sigalg SHA1withRSA
-digestalg SHA1 C:\User\Ari\.android\debug.keystore -storepass android
-keypass android -signedjar release\MyAndroidApp.signed.apk
release\MyAndroidApp.unsigned.apk androiddebugkey

#No linux e osx
$JAVA_HOME/bin/jarsigner \
-verbose \
-keystore ~/.android/debug.keystore \
-storepass android \
-keypass android \
-signedjar release/MyAndroidApp.signed.apk \
-sigalg SHA1withRSA \
-digestalg SHA1 \
release/MyAndroidApp.unsigned.apk \
androiddebugkey
```

É possível criar um certificado próprio e para isto basta seguinte os passos descritos abaixo:

```
keytool -genkey -v -keystore release/androidtest.keystore -alias \
AndroidTestKey  -keyalg RSA -keysize 2048 -validity 10000
#
#resultado
#
#Enter keystore password: store123456
#Re-enter new password: store123456
#What is your first and last name?
#  [Unknown]:  Ari
#What is the name of your organizational unit?
#  [Unknown]:  AG
#What is the name of your organization?
#  [Unknown]:  AG
#What is the name of your City or Locality?
#  [Unknown]:  Cajazeiras/PB
#What is the name of your State or Province?
#  [Unknown]:  BR
#What is the two-letter country code for this unit?
#  [Unknown]:  BR
#Is CN=Ari, OU=AG, O=AG, L=Cajazeiras/PB, ST=BR, C=BR correct?
#  [no]:  yes
#
#Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA)
#with a validity of 10,000 days for: CN=Ari, OU=AG, O=AG, L=Cajazeiras/PB,
#ST=BR, C=BR Enter key password for <AndroidTestKey>
#        (RETURN if same as keystore password):  key123456
#Re-enter new password: key123456
#[Storing release/androidtest.keystore]
```

Depois disto basta colocar o arquivo keystore gerado no diretório .android que, normalmente, fica da pasta do usuário do sistema operacional, e fazer a assinatura do pacote, conforme o comando apresentado abaixo:

```
#No windows
%JAVA_HOME%\bin\jarsigner -verbose -keystore
release/androidtest.keystore -storepass store123456
-keypass key123456 -sigalg SHA1withRSA -digestalg SHA1
-signedjar release/MyAndroidApp.signed.apk
release/MyAndroidApp.unsigned.apk AndroidTestKey

#No linux e osx
$JAVA_HOME/bin/jarsigner \
-verbose \
-keystore release/androidtest.keystore \
-storepass store123456 \
-keypass key123456 \
-sigalg MD5withRSA \
-digestalg SHA1 \
-signedjar release/MyAndroidApp.signed.apk \
release/MyAndroidApp.unsigned.apk \
AndroidTestKey
```

Agora basta instanciar o servidor de debug e o emulador e depois realizar a instalação do arquivo .apk. Os comandos para realizar esta tarefa são:

```
#No windows - para instanciar o servidor
%ANDROID_HOME%\platform-tools\adb start-server
%ANDROID_HOME%\platform-tools\adb install release\MyAndroidApp.signed.apk

#No linux e osx
%ANDROID_HOME%/platform-tools/adb start-server
%ANDROID_HOME%/platform-tools/adb install release/MyAndroidApp.signed.apk
```