



android

*An Open Handset Alliance Project*

# Installation & Outils

O.Legrand  
G. Seront

# Installation

- Pour développer des applications de type Android, il faut installer Android Studio (AS):
  - <http://developer.android.com/sdk/installing/index.html>
- Voir guide sur eCampus
- AS comprend l'Android SDK
- Eclipse n'est **plus** supporté!



# Environnements de développement

- Android Studio comprend:
  - IDE (basé sur IntelliJ)
  - Android SDK tools
  - Android 7.0 (Lollipop) platform
  - Emulateur
  - Gradle (outils de build)



# Emulateur

- Les applications peuvent être testées à l'aide d'un émulateur
- Utilisable via AS ou en mode commande
- Pour le lancer en mode commande :
  - *emulator -avd <android virtual device>*
- Un <Android Virtual Device> est un fichier de configuration d'un émulateur
- Pour le créer en mode commande :
  - *android avd*



# Android Debug Bridge (ADB)

- Application client-server permettant :
  - d'installer une application sur un émulateur ou un appareil mobile
  - d'y copier des fichiers
  - de le configurer
  - d'exécuter des commandes Linux
  - d'accéder à son sgbd (SQLite)
- Accessible via Android Studio ou en ligne de commandes



# Android Debug Bridge (ADB)

- Il comprend 3 éléments :
  - un client
    - qui tourne sur la machine de développement
    - permet de lancer des commandes
  - un serveur
    - qui tourne sur la machine de développement
    - gère les communications entre le(s) client(s) adb et le daemon tournant sur l'émulateur ou l'appareil
  - un daemon
    - qui tourne sur l'émulateur ou l'appareil mobile



# Android Debug Bridge (ADB)

- Quand un client adb est lancé
  - il cherche si un server adb tourne en local
  - le lance si pas trouvé
  - communique avec le serveur (via le port TCP 5037)
- Le serveur adb
  - recherche les émulateurs et appareils mobiles
    - scanne les ports impair de 5555 à 5585
  - communique avec le daemon de chaque émulateur/appareil mobile



# Android Debug Bridge (ADB)

- Un client adb peut accéder via le serveur à un ou plusieurs émulateurs/appareils
- Plusieurs clients adb peuvent accéder à un même émulateur/appareil





# Android Debug Bridge (ADB)

- L'outil adb se trouve en
  - `c:/.../android.../platform-tools/`
- Quelques commandes ADB :
  - *adb version*
  - *adb devices*
    - affiche les émulateurs/appareils disponibles
  - *adb install MonApplication.apk*
    - installe l'application dans le répertoire /data/app
  - *adb push*
    - copie des fichiers du pc sur l'émulateur/appareil



# Android Debug Bridge (ADB)

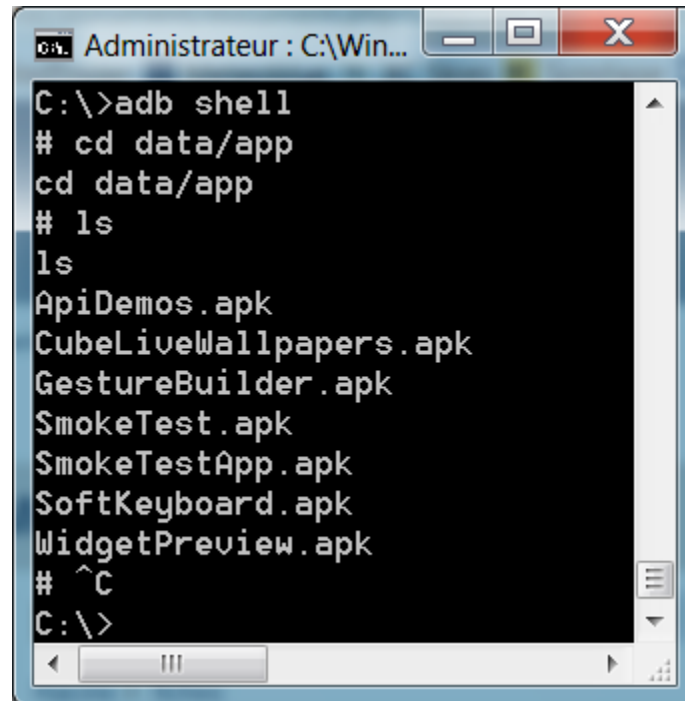
- *adb pull*
  - copie des fichiers de l'émulateur/appareil vers le pc
- *adb logcat*
  - affiche le contenu des buffers de l'émulateur/appareil
- *adb kill-server*
  - arrête le serveur
- *adb start-server*
  - relance le serveur



# Android Debug Bridge (ADB)

## – *adb shell*

- exécution de commandes Linux sur l'émulateur/appareil
- exemple : consultation du contenu d'un répertoire:



```

C:\>adb shell
# cd data/app
cd data/app
# ls
ls
ApiDemos.apk
CubeLiveWallpapers.apk
GestureBuilder.apk
SmokeTest.apk
SmokeTestApp.apk
SoftKeyboard.apk
WidgetPreview.apk
# ^C
C:\>
  
```



# adb logcat

- *adb logcat* permet l'affiche des messages contenus dans les buffers de l'émulateur.
- Chaque message possède :
  - une priorité
  - un *Tag*
  - un numéro de process
  - une description du message,...
- Exemples:
  - I/ActivityManager( 585): Starting activity: Intent { action=android.intent.action...}
  - D/dalvikvm( 113): GC freed 119 objects / 12845 bytes in 225ms...



# adb logcat (priorités)

- Les priorités :
  - *V — Verbose (lowest priority)*
  - *D — Debug*
  - *I — Info*
  - *W — Warning*
  - *E — Error*
  - *F — Fatal*
  - *S — Silent (highest priority, nothing printed)*
  - *WTF — What a terrible failure*



# adb logcat (*Tag*)

- Le *Tag* renseigne le composant système à l'origine du message
- Exemples :
  - *AndroidRuntime*
  - *dalvikvm*
  - *ActivityManager*
  - *DataBase*
  - *MapsActivity*
  - *WifiService*
  - *MonTag...*



# adb logcat : les filtres

- Les messages peuvent être filtrés;
- En renseignant : *Tag:priorité*
- Exemples:
  - *adb logcat \*:W*
    - affiche tous les messages de priorités  $\geq$  *Warning (W, E, F, WTF)*
  - *adb logcat \*:E*
    - affiche tous les messages de priorités  $\geq$  *Error (E, F, WTF)*
  - *adb logcat dalvikvm:\**
    - *affiche les messages générés par la machine virtuelle*



# adb logcat : les filtres

- Exemples (suite):
  - *adb logcat **AndroidRuntime:E***
    - *affiche tous les messages d'erreurs générés à l'exécution*
  - *adb logcat **MonTag:V***
    - *affiche tous les messages de type **MonTag***
  - *adb logcat **AndroidRuntime:E MonTag:V \*:S***
    - *combinaison de plusieurs filtres*
    - *très utile pour débayer*





# adb logcat : les filtres

- L'ajout du filtre `*:S` assure que seuls les messages des *Tags* renseignés dans le(s) filtre(s) seront affichés
  - `adb logcat MonTag:V *:S`
- Pour effacer tous les messages du log par défaut
  - `adb logcat -c`
- Il existe plusieurs logs : *radio, events, main* (par défaut)
- Pour afficher les messages du log *radio* :
  - `adb logcat -b radio`



# Classe *Log*

- La classe *android.util.Log* permet de générer ses propres messages à l'exécution
- Exemples dans le code source:
  - *Log.i("MonTag", "message de type information...");*
  - *Log.w("MonTag", "message de type warning...");*
- Consultation des messages à la console :



The screenshot shows a Windows command prompt window with the title bar "C:\WINDOWS\system32\cmd.exe - adb logcat MonTag:V \*:S". The command prompt displays the following text:

```
C:\>adb logcat MonTag:U *:S
I/MonTag ( 230): message de type information...
W/MonTag ( 230): message de type warning...
```

The window has a standard Windows interface with a title bar, maximize, minimize, and close buttons, and a scrollbar on the right side.

