



UV : L052 – Rapport

Travaux Pratiques

TP 1 : mise en place de l'environnement de
développement

1. Mise en place de l'environnement de développement

L'objectif est de faire une prise en main de l'outil Git, configurer et mettre en place l'environnement de développement.

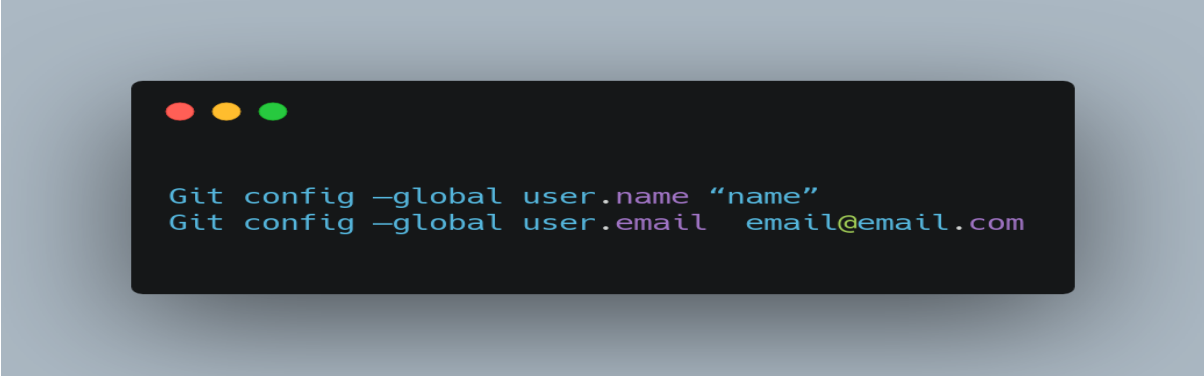
1.1. Installation et configuration de git

- Installation

Nous avons installé git sur notre machine en suivant les instructions, depuis le site de téléchargement de git : <https://git-scm.com/downloads>.

- Configuration

L'installation faite, nous avons procédé à la configuration de git avec nos informations d'identification sur github :

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays two lines of text: `Git config --global user.name "name"` and `Git config --global user.email email@email.com`.

```
Git config --global user.name "name"
Git config --global user.email email@email.com
```

Ensuite nous avons clone le dépôt git du projet sur github afin d'avoir une copie du code source sur notre machine.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays a single line of text: `Git clone https://github.com/gxfab/L052_A2020`.

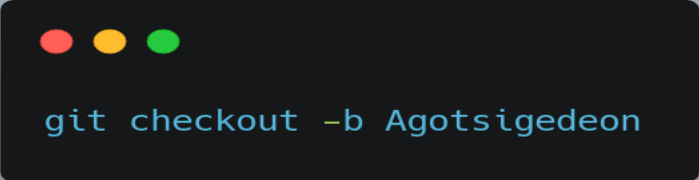
```
Git clone https://github.com/gxfab/L052_A2020
```

Pour le suivi de notre travail, nous avons créé une branche git spécifique :




```
Git branch Agotsigedeon
```

Pour se positionner sur notre branche :



```
git checkout -b Agotsigedeon
```

Après avoir expliqué l'organisation de travail dans le fichier TP1.txt, on met à jour notre dépôt distant sur github.



```
Git add .  
Git commit -m "Mise à jour du fichier TP1.txt. "  
Git push -u origin Agotsigedeon.
```

Git étant configuré et l'accès au répertoire de travail étant fait, nous passons à l'étape de développement de l'application android.

1.2. Développement application android

L'objectif de cette partie est de pouvoir installer et prendre en main l'environnement de travail pour le développement android et ensuite développer une application "Hello world " de navigation entre différents activity.

- Installation

L'installation d'Android studio (IDE de développement android) est faite en suivant le guide officiel : <https://developer.android.com/studio/install>

- Réalisation du projet

Le projet consiste à créer deux activity android et ensuite passer de l'une à l'autre quand on clique sur un bouton. Nous développerons en JAVA.

Après avoir créé un "projet vide" avec l'assistant d'android studio, nous avons créé l'activity principale et son layout associé :

```
package com.example.helloworld;

import ...

public class MainActivity extends AppCompatActivity {

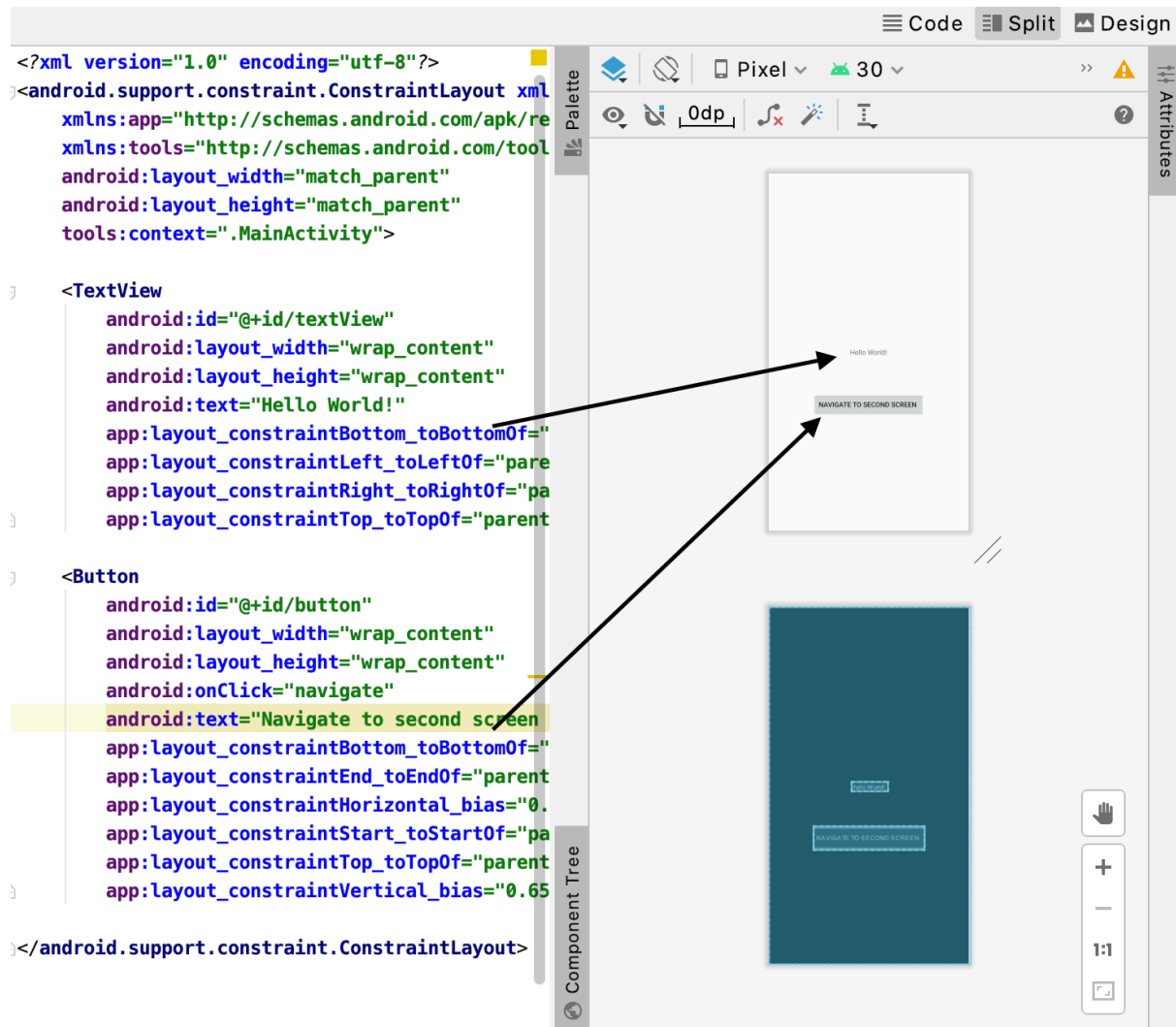
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void navigate(View view) {
        Intent intent = new Intent(packageContext: MainActivity.this, HelloActivity.class);
        startActivity(intent);
    }
}
```

Studio performance could be improved
Increasing the maximum heap size from...

Code source MainActivity

Il est associé à l'activity un fichier xml permettant de construire l'interface : on remarque la présence d'un texte « Hello Word » et d'un bouton intitulé : navigation to second screen.



Code source activity_main.xml

Ensuite nous, allons créer la seconde activity : HelloActivity , voici ci-dessous le code source de HelloActivity.java et son fichier xml activity_hello.xml décrivant l'interface

```
package com.example.helloworld;

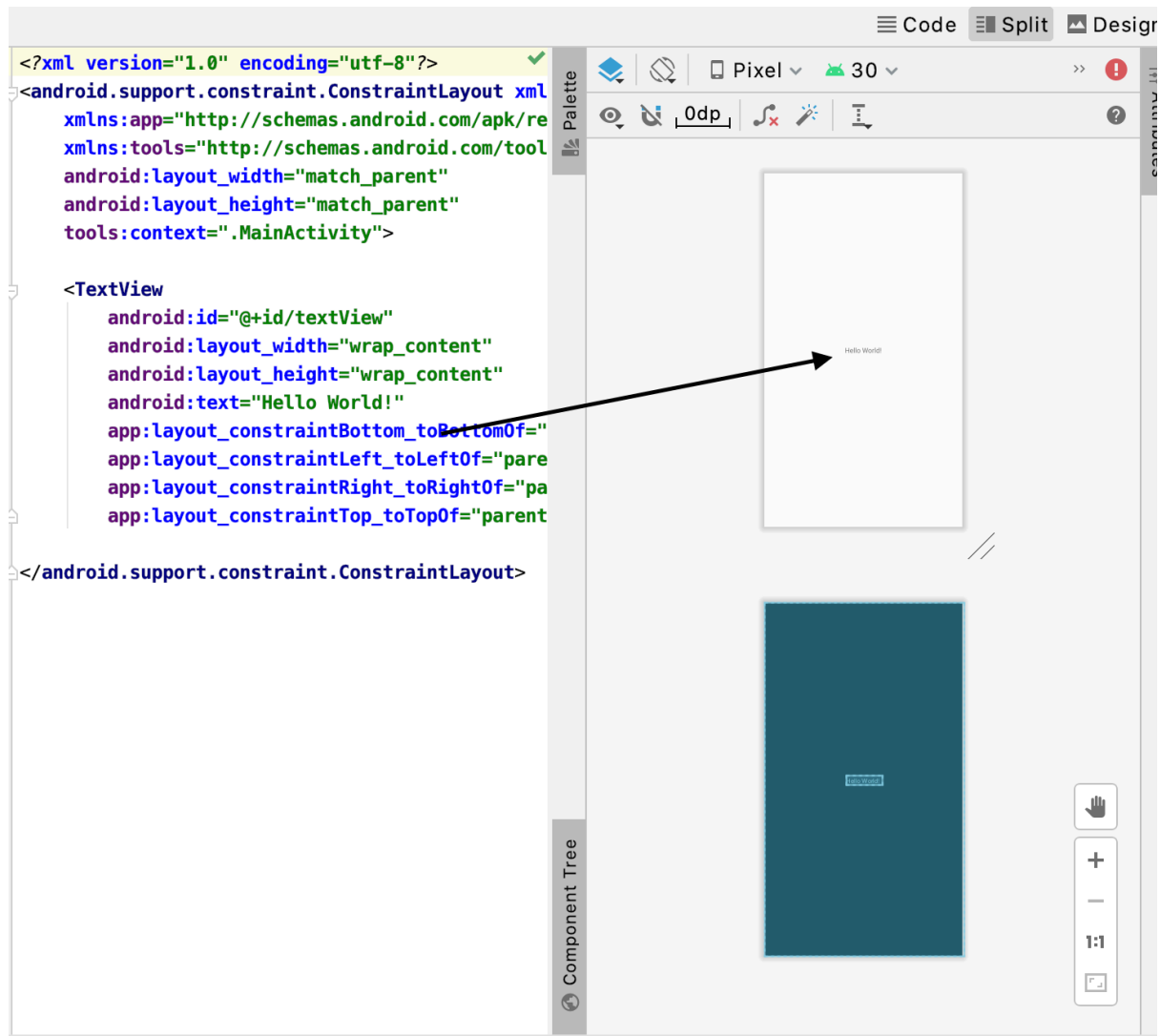
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

public class HelloActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello);
    }
}
```

Code source HelloActivity

Nous avons déposé le composant textView dans le layout lié à HelloActivity affichant :
Hello Word



Code source activity_hello.xml

Le fichier `string.xml` contient la valeur des textes affichés sur nos composants, nous permettant de la modification de nos intitulés :

```
<resources>
<string name="app_name">Main Activity</string>
<string name="hello_activity_name">Hello World Activity</string>
</resources>
```

Pour pouvoir passer du Main Activity vers le Hello Activity, nous avons créé la méthode navigation qui lors d'un événement `onClick` du bouton spécifié dans `activity_main.xml`

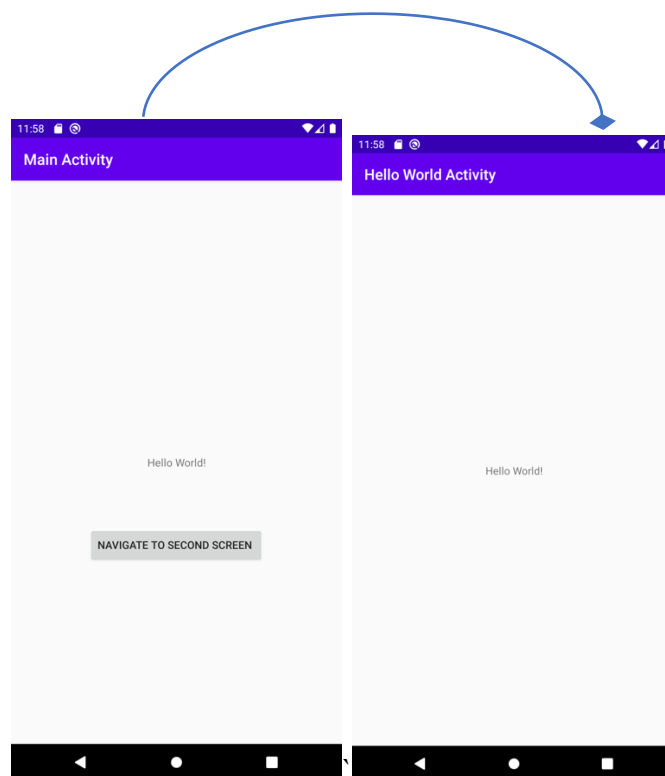

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="navigate"
    android:text="Navigate to second screen "
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.657" />

public void navigate(View view) {
    Intent intent = new Intent( packageContext: MainActivity.this, HelloActivity.class);
    startActivity(intent);
}
```

Code source méthode navigate et button de navigation

1.3. Démo

Nous avons pu obtenu le résultat suivant :



Demo TP1

TP 2 : manipulation d'un Kernel Linux

1. Préparation de l'environnement et configuration du noyau

L'objectif de ce TP est de configurer un noyau Android avec des modifications et générer par la suite une image.

1.1. Préparation de l'environnement

Afin de pouvoir récupérer le code source du noyau hikey-linaro, nous avons installé l'indispensable outil, repo :

```
sudo apt-get install repo
```

1.2. Récupération du code source

Repo étant installé, nous pouvons l'utiliser pour récupérer le code source du noyau hikey-linaro dans le dossier android-kernel, préalablement créé.

```
mkdir ~/android-kernel  
cd ~/android-kernel  
repo init -u https://android.googlesource.com/kernel/manifest -b  
hikey-linaro-android-4.19  
repo sync
```

1.3. Spécification de l'architecture du système

Pour pouvoir configurer le noyau récupéré, nous avons besoin de spécifier l'architecture système utilisé, dans notre cadre, c'est l'arm64.

```
export ARCH=arm64
```

1.4. Sauvegarde de la configuration par défaut

Étant donné, que nous allons charger la configuration relative pour une carte ranchu64, nous allons dans un premier temps charger la configuration par défaut du noyau hikey-linaro, qui se trouve dans le dossier hikey-linaro et la sauvegarder dans notre dossier config, sous le nom de « defconfig_default »

```
cd ~/android-kernel/hikey-linaro  
make defconfig  
make savedefconfig  
mkdir ~/config
```

```
cp defconfig ~/config/defconfig_default
```

1.5. Chargement de la configuration pour la carte ranchu64

La configuration par défaut étant sauvegardée, dans un second temps, nous avons cherché l'emplacement du fichier config de la carte ranchu64, qui se trouve dans le dossier *arch/mips/configs/generic*. Nous copions ce fichier dans le dossier *arch/arm64/configs* afin de le charger.

```
make clean
cp arch/mips/configs/generic/board-ranchu.config arch/arm64/configs/
make board-ranchu.config
make savedefconfig
```

Après avoir chargé et sauvegardé le fichier config relative à la carte ranchu64 avec les commandes ci-hauts, nous le sauvegardons dans le dossier config sous le nom de « defconfig_ranchu64 »

1.6. Différences entre les deux configurations (par défaut et ranchu64)

```
cp defconfig ~/config/defconfig_ranchu64
```

Pour observer la différence entre le fichier de configuration par défaut (defconfig_default) du noyau et du fichier de configuration relative à la carte ranchu64 (defconfig_ranchu64), nous avons utilisé l'outil de comparaison de l'IDE Visual studio code.

Nous avons enregistré une vidéo que vous pouvez visualiser sur ce lien (<https://youtu.be/drKN3LU02dU>) qui montre les différences que nous avons pu observer.

2. Modification de la configuration du noyau

Il nous a été demandé de modifier certaines configurations du noyau comme :

- Compatibilité pour la carte ARMv8 Versatile mais aussi Qualcomm et Realtek
- Activation du NFC et du protocole NFC HC
- Activation de l'option Frequency Scaling de la CPU
- Activation du support de l'HDMI CEC et activation du support LED

Pour le faire, nous avons utilisé l'outil graphique xconfig, nous permettant en un premier temps, de cocher les configurations à ajouter et de décocher les options superflues comme :

- Plan 9 Resource Sharing
- Carte son
- Bluetooth

```
make xconfig
```

Après avoir fait les modifications on ferme la fenêtre de l'outil, on enregistre les modifications et on le copie vers notre dossier config sous le nom de « ranchu64_optimise »

```
make savedefconfig  
cp defconfig ~/config/defconfig_ranchu64_optimise
```

3. Bonus : images du noyau

Pour le bonus, nous avons créé les images respectives des noyaux par défaut et optimisé. Pour pouvoir le faire, nous avons téléchargé la toolchain de compilation du noyau et l'avons exporté comme une variable globale du système.

```
mkdir ~/toolchain  
cd ~/toolchain  
wget  
https://releases.linaro.org/components/toolchain/binaries/latest-  
7/aarch64-linux-gnu/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-  
gnu.tar.xz  
tar -xf gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz
```

Avec la toolchain, nous pouvons ainsi générer le noyau.

```
make -j8
```

Les images se trouvent dans le dossier `arch/arm64/boot/`

Nous avons observé la différence de taille pour ces deux images.

Nous avons créé un fichier script contenant nos scripts et un fichier TP2.txt contenant toutes nos commandes.

Nous avons envoyé les configurations, les images, le script et le fichier TP2.txt sur notre dépôt git.

TP 3 : Création d'un device Android