Controles básicos

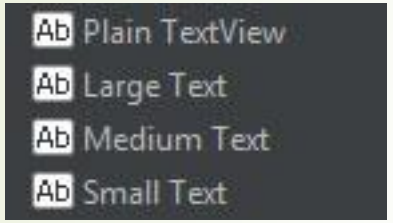
Controles básicos

- **Propiedades genéricas** para casi todos los controles:
  - **android:textColor**
    - Define el color del texto del control, su valor será un valor hexadecimal para expresar el color deseado.
  - **android:textSize**
    - Determina el tamaño del texto.
  - **android:typeface**
    - Establece los estilos de fuente como son “normal”, “sans”, “serif”, “monospace”.
  - **android:textStyle**
    - Define el estilo usado en el texto, si va en negrita (“bold”) o cursiva (“italic”).

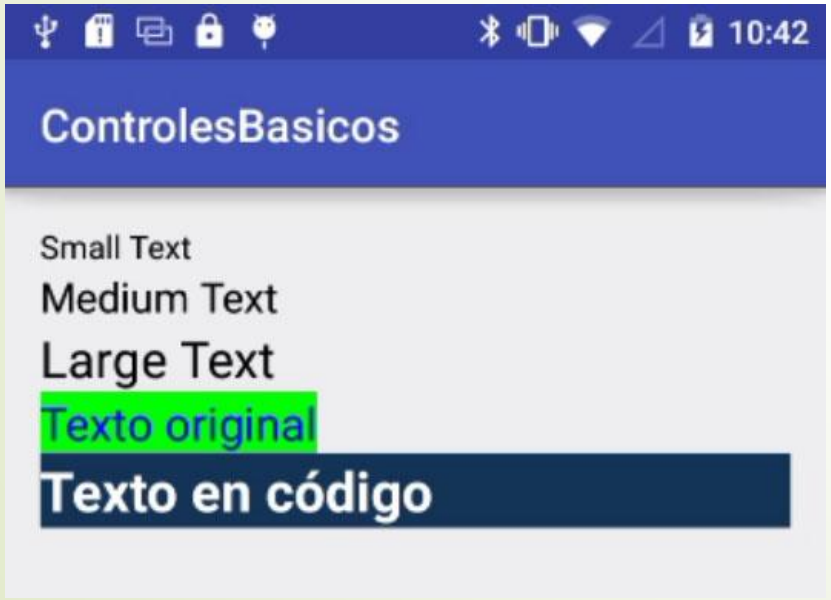
TextView

- Estos controles son muy usados, son etiquetas capaces de mostrar textos al usuario. Su propiedad principal es **android:text** en la que le establecemos el texto a mostrar. Esta misma propiedad en código sería **setText()**. Además de esta propiedad, se puede cambiar el formato de texto usando las siguientes propiedades:
  - **android:background**
    - Color de fondo
  - **android:textColor**
    - Color de texto
  - **android:textSize**
    - Tamaño de la fuente
  - **android:typeface**
    - Estilo del texto

TextView




```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Texto original"
    android:id="@+id/textView4"
    android:textColor="#0000FF"
    android:textSize="20dp"
    android:background="#00FF00"/>
```

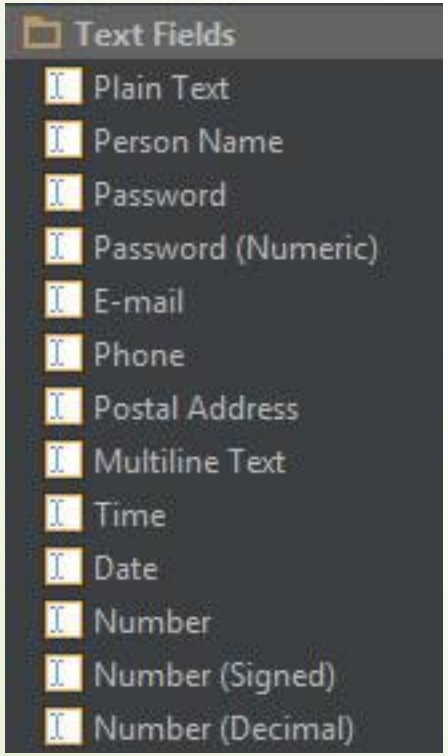



```
TextView textView5 = (TextView) findViewById(R.id.textView5);
textView5.setText("Texto en código");
textView5.setTextSize(24);
textView5.setTextColor(Color.parseColor("#FFFFFFFF"));
textView5.setBackgroundColor(Color.parseColor("#FF123456"));
textView5.setTypeface(Typeface.DEFAULT_BOLD);
```

EditText

- Es el control que se usa para la edición de texto por parte del usuario, en diseño la propiedad más sobresaliente es, como en el caso anterior, **android.text**, que establece el texto que contiene.
- En cambio, cuando lo usamos realmente lo interesante es el texto introducido por el usuario, así que en nuestro código nos interesará mucho la propiedad **getText()**, para recuperar el texto de una etiqueta. También podemos poner texto mediante **setText()**.

EditText




```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="text"
    android:id="@+id/editText" />
```

EditText

- Para obtener el texto usaríamos el método getText()

```
EditText editText = (EditText) findViewById(R.id.editText);
String texto = editText.getText().toString();
```

- En el código anterior hemos hecho un cambio de formato usando el método **toString()** sobre el resultado de **getText()**. El método **getText() NO devuelve una cadena** (String), sino un objeto de tipo Editable (tipo Spanned, algo así como una cadena de caracteres en la que podemos insertar etiquetas). Es decir, el componente EditText permite editar texto plano y texto enriquecido o con formato, por eso hemos tenido que usar un método para cambiar la cadena perdiendo el formato enriquecido.

EditText

- Para poder obtener el texto con el formato correspondiente, podemos usar la clase Html de Android, que dispone de los métodos para convertir objeto de tipo Spanned en su representación HTML equivalente.

```
EditText editText = (EditText) findViewById(R.id.editText);
String texto = editText.getText().toString();
String textoHtml = Html.toHtml(editText.getText());
```

```
texto = "Esto es una prueba"
textoHtml = "<p dir="ltr">Esto es una prueba</p>\n"
```

- También es posible realizar la operación opuesta, es decir, establecer en un cuadro de texto (EditText) o en una etiqueta (TextView) un texto en formato HTML. Para ello, se utiliza el método Html.fromHtml(String) así:

```
editText.setText(Html.fromHtml("<p>Esto es una <b>prueba</b>.</p>"));
```

Esto es una **prueba**.

EditText

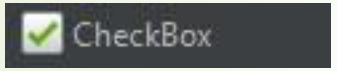
- La propiedad android:hint, sirve para poner un texto de fondo inicial con poca intensidad de color, el cual no se puede tratar con getText()/setText.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="phone"
    android:hint="Teléfono"
    android:ems="10"
    android:id="@+id/editText3" />
```

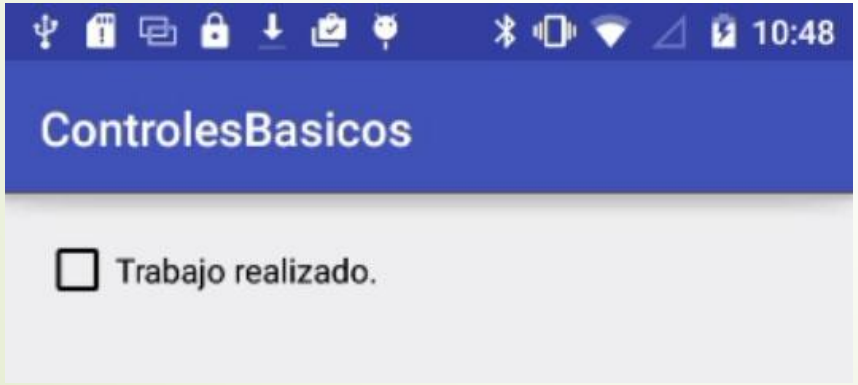
CheckBox

- El control típico para indicar al usuario que debe marcar o desmarcar una opción, tiene dos estados que son marcado o no marcado. Esto se comprueba mediante la propiedad **isChecked()** que nos devolverá un valor verdadero/falso. También podemos usar **setChecked(boolean)** para establecer un estado en concreto. Respecto al a personalización de estilo del componente, podemos emplear casi todas las opciones del componente TextView comentadas anteriormente

CheckBox




```
<CheckBox
    android:id="@+id/checkbox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/trabajoRealizado"/>
```



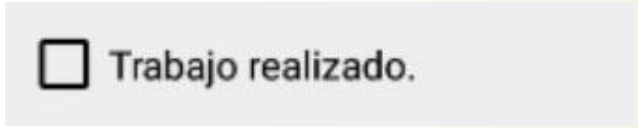
CheckBox

- En cuanto a los posibles eventos interesantes que puede lanzar este componente, el más interesante es onCheckedChanged que notifica que la selección ha cambiado. Usaremos el siguiente código para añadir un evento y por lo tanto funcionalidad al control.

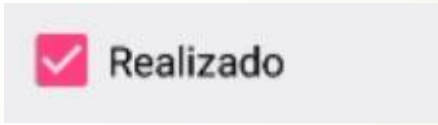
```
CheckBox check = (CheckBox) findViewById(R.id.checkbox);
CompoundButton.OnCheckedChangeListener eventoCheck = new CheckBox.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton checkView, boolean isChecked) {
        if(isChecked){
            checkView.setText("Realizado");
        }else{
            checkView.setText("No realizado");
        }
    }
};

check.setOnCheckedChangeListener(eventoCheck);
```

CheckBox

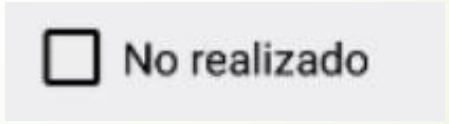
- CompoundButton es el componente CheckBox en sí, con lo cual puedo acceder a tributos del mismo como su texto getText(),etc.
  - Sin tocar el control



  - Control marcado



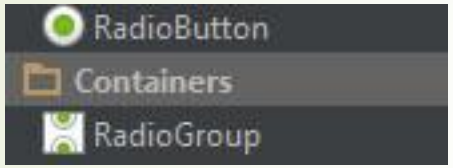
  - Control desmarcado

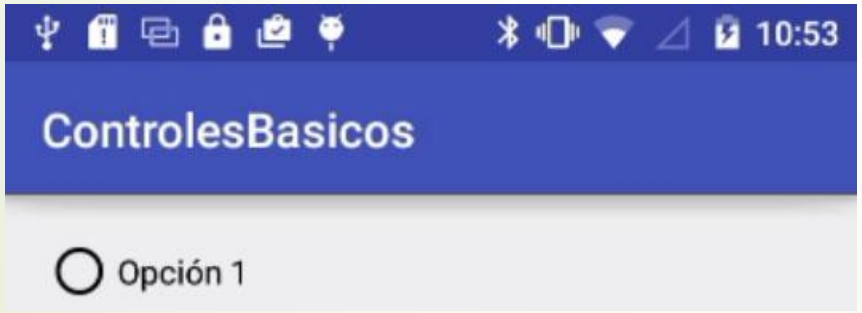


RadioButton

- Tiene una función similar al **CheckBox** pero en este caso se usa de forma colectiva englobando varios **RadioButton** dentro de un grupo llamado **RadioGroup** para que uno y solo uno de los **RadioButton**, y por tanto de las opciones, esté marcada.

RadioButton






```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/radioGroup"
    android:orientation="vertical">

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 1"
        android:id="@+id/radio" />

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 1"
        android:id="@+id/radio2" />

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Opción 1"
        android:id="@+id/radio3" />

</RadioGroup>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Medium Text"
    android:id="@+id/textView"/>
```

RadioButton

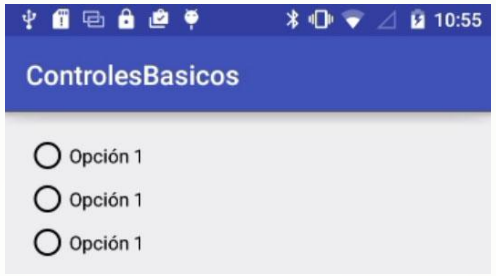
- En primer lugar, hemos establecido la **orientación** (vertical u horizontal) como hicimos con el componente LinearLayout. Después, hemos añadido todos los componentes RadioButton necesarios, indicando su ID mediante la propiedad **android:id** y su texto mediante la propiedad **android:text**

RadioButton

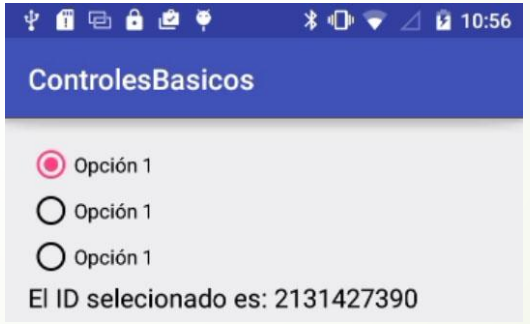
```
//Obtenemos el grupo
RadioGroup grupo = (RadioGroup) findViewById(R.id.radioGroup);

//Establece la marca en un radio determinado
grupo.check(R.id.radio2);

//Elimina la marca de todas las opciones
grupo.clearCheck();

//Nos devolverá el ID del radio marcado o -1 si no hay ninguno
int id = grupo.getCheckedRadioButtonId();

//Obtenemos la etiqueta de texto
final TextView txt = (TextView) findViewById(R.id.textView);

RadioGroup.OnCheckedChangeListener evento = new RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup radioGroup, int checkedId) {
        txt.setText("El ID seleccionado es: "+ checkedId);
    }
};

//Asignamos el evento creado al grupo
grupo.setOnCheckedChangeListener(evento);
```

RadioButton

- Estado inicial



- Marcada un opción



- Desde XML podemos indicar que un RadioButton esté seleccionado a priori con la propiedad **android:checked="true"**.

Objetos Spannable

- Este tipo de objetos son muy importantes ya que básicamente nos permiten manipular los textos, enriqueciéndolos y aplicando estilos. Los spans más importantes de los que disponemos son:
  - **ForeGroundspan**

    Define el color del texto
  - **TypefaceSpan**
    - Determina el tipo de letra del texto
  - **StyleSpan**
    - Define el estilo usado en el texto
  - **AbsoluteStyleSpan**
    - Establece el tamaño del texto

Objetos Spannable

- Esto se usa conjuntamente con los controles básicos explicados para interactuar con ellos. Si miramos los códigos de ejemplo al recuperar un texto con la propiedad **getText()** está escrito a continuación **toString()**, esto es debido a que **getText()** nos devuelve un objeto **Editable** que hereda de **Spannable** y se hace necesario convertirlo en una cadena de texto normal.

Objetos Spannable

- Ejemplo:

```
<TextView
    android:id ="@+id/txtPrueba"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FFFFFFFF"
    android:textSize="20dp"
    android:typeface="normal"
    android:text=""
    android:background="#FF123456"/>
```

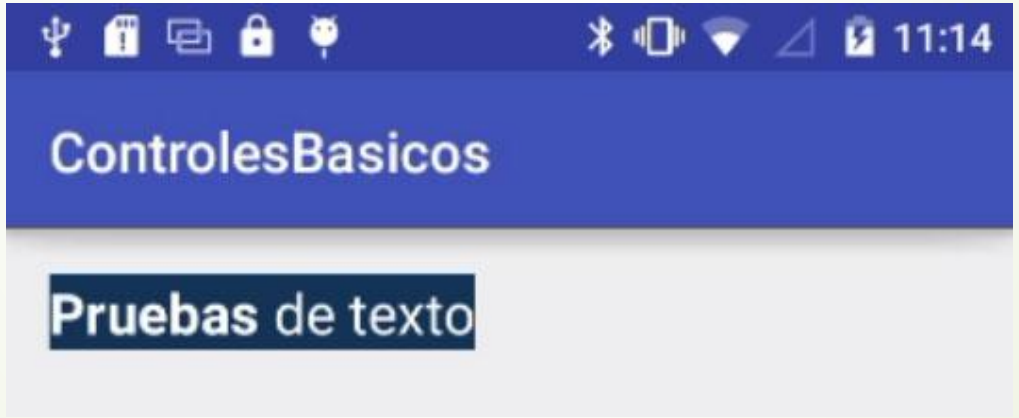
Objetos Spannable

- Ejemplo:

```
//Creamos un objeto Editable
Editable str = Editable.Factory.getInstance().newEditable("Pruebas de texto");

//Establecemos que la palabra "Pruebas" sea negrita
str.setSpan(new StyleSpan(Typeface.BOLD),0,7, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);

//Obtenemos la etiqueta de pruebas
TextView etiqueta = (TextView) findViewById(R.id.txtPrueba);

etiqueta.setText(str);
```

Objetos Spannable

- Ejemplo:



Botones

- Hay cuatro tipos
  - Button:
  - ToggleButton
  - Switch
  - ImageButton

Botones

- Button
  - Lo destacable del botón es la creación del evento **onClick()**, a continuación se muestra un ejemplo de como implementar un evento que recoge el número de pulsaciones.

Botones

- Button

```xml
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:layout_weight="1"
    android:gravity="center">

    <Button
        android:id="@+id/testButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Botón"/>

    <TextView
        android:id="@+id/txt1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""/>

</LinearLayout>
```

```java
final TextView txt = (TextView) findViewById(R.id.txt1);

Button boton = (Button) findViewById(R.id.testButton);

View.OnClickListener eventoBoton = new View.OnClickListener() {
    int vecesPulsado = 0;
    @Override
    public void onClick(View view) {
            vecesPulsado++;
        txt.setText("Se ha pulsado el botón "+ vecesPulsado+ " vez/veces");
    }
};

boton.setOnClickListener(eventoBoton);
```

Botones

- Button
  - También se puede hacer de manera más sencilla a través del código XML, asignándole al atributo **onClick** del botón el nombre de un método (ejemplo cuenta) y luego en el código Java de la actividad que alberga dicho botón, crear un método llamado igual que el nombre que hemos puesto en el atributo **onClick** del botón, y pasándole como parámetro un objeto View.

Botones

- Button

```
<Button
    android:id="@+id/testButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="cuenta"
    android:text="Botón"/>
```

```
int vecesPulsado = 0;
public void cuenta(View view) {
    vecesPulsado++;
    txt.setText("Se ha pulsado el botón "+ vecesPulsado+ " vez/veces");
}
```

Botones

- Button
  - Al ejecutarlo

  - Tras pulsar cinco veces el botón

Botones

- ToggleButton
  - Destacamos en este control las propiedades de texto referentes a los dos estados posibles, por defecto son **ON** y **OFF** pero mediante las propiedades **android:textOn** y **android:textOff** podemos personalizar el control con cualquier texto. También destacamos que este tipo de botón dispone de la propiedad **isChecked()** para comprobar en qué estado se encuentra al igual que los **ChekBox** Vistos con anterioridad.

Botones

- ToggleButton

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:layout_weight="1"
    android:gravity="center">

    <ToggleButton
        android:id="@+id/toggleButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOn="Encendido"
        android:textOff="Apagado"/>

    <TextView
        android:id="@+id/txtOn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""/>

    <TextView
        android:id="@+id/txtOff"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""/>

</LinearLayout>
```

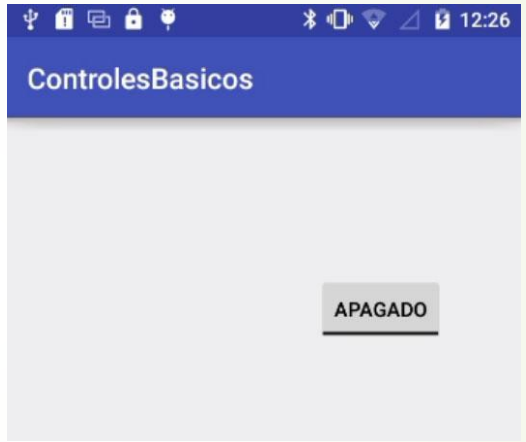
```
final TextView txtON = (TextView) findViewById(R.id.txtOn);
final TextView txtOFF = (TextView) findViewById(R.id.txtOff);

final ToggleButton botonToggle = (ToggleButton) findViewById(R.id.toggleButton1);

View.OnClickListener eventoToggle = new View.OnClickListener() {
    int vecesOn = 0;
    int vecesOff = 0;
    @Override
    public void onClick(View view) {
        if(botonToggle.isChecked()){
            vecesOn++;
            txtON.setText("El botón ha estado encendido "+ vecesOn+ " vez/veces");
        }else{
            vecesOff++;
            txtOFF.setText("El botón ha estado apagado "+ vecesOff+ " vez/veces");
        }
    }
};

botonToggle.setOnClickListener(eventoToggle);
```

Botones

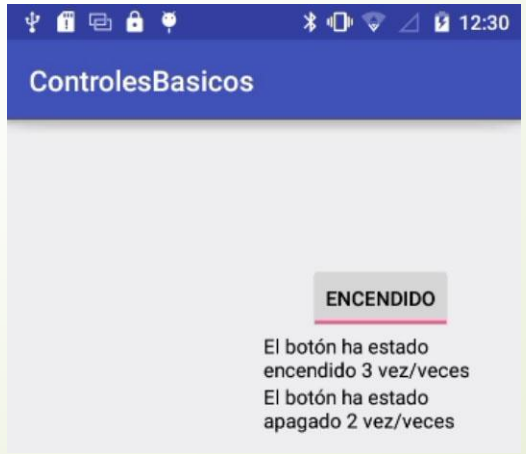
- ToggleButton
  - También se puede utilizar la manera antes explicada que es más sencilla, mediante el atributo **onClick**.

Botones

- ToggleButton
  - Al ejecutarlo



  - Tras pulsar varias veces el botón

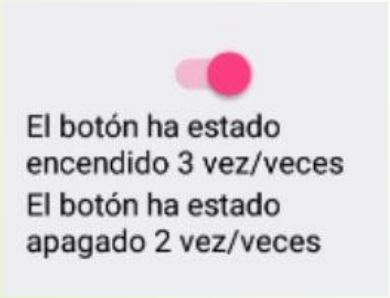


Botones

- Switch
  - El control switch es muy similar al ToggleButton, donde tan sólo cambia su aspecto visual, que en vez de mostrar un estado u otro sobre el mismo espacio, se muestra en forma de deslizador o interruptor. Su sería completamente igual al ya comentado en el ToggleButton

Botones

- Switch




```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:layout_weight="1"
    android:gravity="center">

    <Switch
        android:id="@+id/bswitch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOn="on"
        android:textOff="off"/>

    <TextView
        android:id="@+id/txtOnSwitch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""/>

    <TextView
        android:id="@+id/txtOffSwitch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""/>

</LinearLayout>
```

Botones

- ImageButton
  - Funciona de la misma manera que el control Button por lo que todo lo visto en ese control le es aplicable, salvo que dispone de una propiedad más que permite establecer una imagen, **android:src** señalado al recurso de la imagen que deseamos mostrar.
  - Normalmente, indicamos esta propiedad usando el descriptor de alguna imagen que hayamos copiado en la carpeta /res/drawable. Así, por ejemplo, en nuestro caso hemos incluido la imagen "stop.png", a la que hacemos referencia en **"@drawable/stop"**. Los botones disponen de eventos que se pueden capturar. El más común es el evento **onClick**.
  - Cabe destacar que se puede conseguir el mismo efecto usando un control **Button** con la propiedad **android:background** apuntado al mismo recurso.

Botones

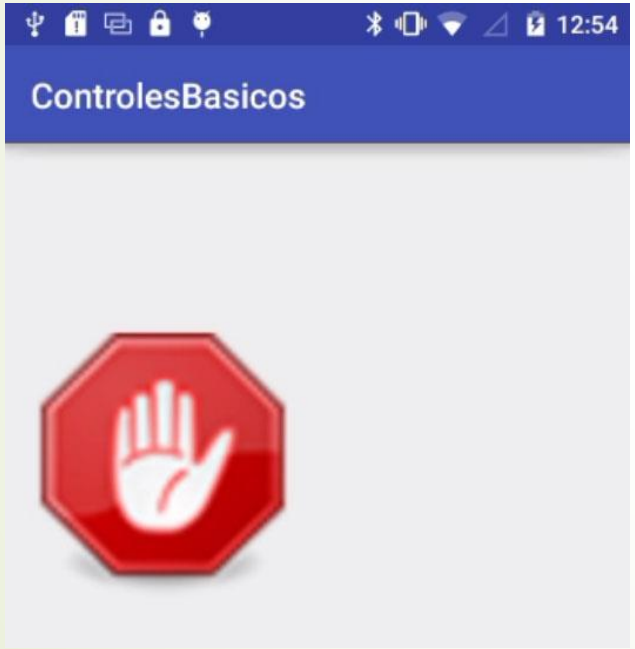
- ImageButton

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:layout_weight="1"
    android:gravity="center">

    <ImageButton
        android:id="@+id/imageButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/stop"/>
</LinearLayout>
```

ImageView

- Por último hablaremos de este control, que como su nombre indica sirve para visualizar una imagen. Al igual que el **ImageButton** lo destacable es la propiedad **android:src** que es donde es establecerá la imagen que se muestra. De nuevo, lo usual es indicar como origen de la imagen el identificador de un recurso de la carpeta /res/drawable. Además de esta propiedad, existen otras, como las destinadas a establecer el tamaño máximo que puede ocupar la imagen: **android:maxWidth** y **android:maxHeight**.

ImageView




```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="151dp"
    android:layout_height="336dp"
    android:src="@drawable/stop"/>
```

ImageView

- En la lógica de la aplicación, podemos establecer la imagen mediante el método **setImageResource (int idImage)**.

```
ImageView img = (ImageView) findViewById(R.id.imageView);
img.setImageResource(R.drawable.stop);
```

- https://github.com/rononunsys/ControlesBasicos.git