

# Tema 0 - Entorno de desarrollo

Descargar estos apuntes [pdf](#) o [html](#)

## Índice

- [Introducción](#)
- ▼ [Instalación del entorno de desarrollo](#)
  - [Enlaces de interés](#)
  - [Descripción de los pasos de Instalación](#)
  - [¿Dónde encontrar las cosas instaladas?](#)
- ▼ [Introducción a Gradle](#)
  - [Enlaces de interés](#)
  - [¿Qué es Gradle?](#)
- [Creando y ejecutando el primer proyecto](#)
- ▼ [Ahondando un poco en el uso de Gradle](#)
  - [Añadiendo al PATH del sistema la versión de Gradle](#)
  - [Ejecutando tareas de Gradle desde consola](#)

# Introducción

En este primer tema vamos a tener la primera toma de contacto con el entorno de desarrollo (IDE) que se utilizará a lo largo de todo el curso, Android Studio, mediante su instalación.

Es importante tener en cuenta las siguientes recomendaciones a la hora de comenzar el desarrollo de una App:

- Debido a la cantidad de marcas y modelos diferentes que se pueden encontrar en el mercado, en el campo de Android, se debe buscar la máxima compatibilidad entre todos ellos.
- Es importante crear un buen diseño de la aplicación para hacer un uso adecuado de recursos, básicamente, liberar recursos cuando estos no sean necesarios y no consumir más recursos de los debidos. Es recomendable hacer que la aplicación funcione bien en un modelo de gama baja, ya que esto implica que en un modelo de gama media/alta funcionará mejor.

**JetBrains** es la empresa que desarrolla **IntelliJ IDEA**, el IDE en el que se basa **Android Studio**. Además, son los principales impulsores del lenguaje Kotlin junto a Google.

IntelliJ IDEA es un IDE de propósito general, que soporta Java, Groovy, Scala, Python, Ruby y Kotlin, entre otros lenguajes. Android Studio es un IDE específico para el desarrollo de aplicaciones Android, que está basado en IntelliJ IDEA, como hemos comentado, pero que incluye herramientas específicas para el desarrollo de aplicaciones Android. Como son: un emulador de dispositivos Android, un sistema de gestión de dependencias, un sistema de generación de APKs, etc.


En el [Stack Overflow Developer Survey](#) de uso de IDEs en 2023, podemos ver que **IntelliJ IDEA** es el IDE más usado por los desarrolladores de Java, por delante de Eclipse y NetBeans. Además, **Android Studio** es el IDE más usado para el desarrollo de aplicaciones Android una vez se abandonó el uso de Eclipse.

# Instalación del entorno de desarrollo


## Enlaces de interés

- [Cómo instalar Android Studio](#)
- [Cómo configurar Android Studio](#)
- [Android Estudio \(Vídeo DevExpert\)](#)
- [Android Estudio 2024 \(Vídeo Inglés\)](#)
- [Crear el primer proyecto en Android Studio](#)
- [Introducción a Android Studio](#)

## Descripción de los pasos de Instalación

 **Tip:** En [este codelab](#) puedes seguir una guía oficial paso a paso para instalar Android Studio y el SDK de Android en diferentes sistemas operativos.

Cada versión de Android Studio viene designada por un código de versión con un animal. Durante el desarrollo de estos apuntes la versión de Android Studio es **Koala**.

 **Nota:** Para evitar capturar excesivas imágenes de la instalación que quedarían desactualizadas rápidamente, en la siguiente guía se tratará de explicar los pasos a seguir para instalar Android Studio en Windows. Pero la instalación en otros sistemas operativos es muy similar.

Los pasos para instalar Android Studio son los siguientes:

1. Descargar la última versión estable de **Android Studio** desde la página oficial de [Android Studio](#).
2. Seleccionar la instalación **'Standard'**
3. Seleccionar los componentes a instalar. Por defecto, se instalarán los componentes necesarios para el desarrollo de aplicaciones Android.

Ya hemos instalado vamos a configurar algunas cosas al ejecutarlo por primera vez.

1. En la opción **'Customize'** podemos encontrar un enlace denominado **'All Settings'** donde nos aparecerán todas las opciones de configuración del IDE.
  1. En **'Apperence & Behavior → Apperance'** Seleccionaremos el tema y el tipo de fuente.
  2. En **'New UI (Beta)'** Seleccionamos **Enable New Editor UI**
  3. En **'Languages & Frameworks → Android SDK'**

1. En la pestaña **SDK Platforms**, seleccionaremos la versión de Android que queremos instalar. Seleccionaremos el API **34** así como las versiones **SandboxPreview**.
2. En la pestaña **SDK Tools**, seleccionaremos la versión de **Android SDK Build-Tools**, **Android Emulator**, **Android SDK Platform-Tools** y **Android SDK Command-line Tools**.
4. En **'Appearance & Behavior' → 'Keymap'** Seleccionaremos el perfil de teclado de **Visual Studio**
5. En **'Editor' → 'General'** Seleccionaremos  
*Change font size with Ctrl+Mouse Wheel y ALL editors* para poder controlar el tamaño del fuente fácilmente.
6. En **'Editor' → 'Font'** Seleccionaremos **JetBrains Mono** como fuente y **Size 20.0** para que sea más legible por los profesores que tienen presbicia 😊.
7. En **'Editor' → 'Live Edit'** seleccionaremos **Live Edit** junto con **Push Edits Automatically** de esta manera cuando ejecutemos la aplicación en el emulador o dispositivo físico, los cambios que hagamos en el código se verán reflejados en el emulador o dispositivo físico sin necesidad de volver a ejecutar la aplicación.
8. En **'Build, Execution, Deployment' → 'Build Tools' → 'Gradle'**  
Gestionaremos la configuración de Gradle, en este caso, dónde va a instalar las versiones de Gradle, etc. En nuestro caso, dejaremos la configuración por defecto que es


**C:\Users\alumno\.gradle**

🔴 **Nota:** Gradle se ejecuta sobre Java y la JRE. Por defecto, Android Studio, instala una versión del JDK Embebida propia, que será la que use Gradle en los proyectos. Este **Embedded JDK** que viene con Android Studio está ubicado en  
C:\Program Files\Android\Android Studio\jbr y que se corresponderá con la versión LTS de Java última en ese momento. En nuestro caso la versión 17. Más adelante cuando creamos un proyecto podríamos indicarle que use otra versión de Java que tengamos instalada en el equipo, pero no será necesario.

El resto de opciones las dejaremos por defecto y las iremos modificando o viendo a lo largo del curso.


2. En la opción **'Projects'** podemos encontrar un enlace denominado **'More Actions'** donde nos aparecerán varias opciones
  1. **SDK Manager** (Nos llevará a la configuración del SDK que ya hemos realizado)
  2. **Virtual Device manager** que nos permitirá crear un **Smartphone virtual** donde crear nuestro programas. Pulsaremos el botón **'Create Device'** arriba a la izquierda. Este nos abrirá un diálogo de configuración del dispositivo.
    1. Seleccionaremos el tipo de dispositivo **'Phone'** y el modelo **'Pixel 3a'** para tener todos el mismo.

2. Seleccionaremos la versión de Android en nuestro caso el **API 34 (UpsideDownCake)** y el tipo de CPU x86\_64

 **Nota:** Si no tenemos la imagen de esta versión descargada procederá a hacerlo automáticamente y esto puede tardar algún tiempo.

3. Por último nos pedirá darle un nombre al dispositivo. En nuestro caso '**Pixel 3a API 34**' y pulsaremos el botón '**Finish**'

Pulsaremos '▶' para ejecutar el emulador y comprobar que funciona correctamente. La primera vez puede tardar un poco.

 **Aviso:** Dependiendo de la versión de Windows y si tenemos activo algún tipo de aceleración de hardware puede que no funcione el emulador o nos falte algún tipo de librería. Googlea el error para tratar de solucionarlo. Puedes ejecutar el emulador desde consola con el comando:

```
C:\Users\alumno\AppData\Local\Android\Sdk\tools\emulator.exe -avd Pixel_3a_API_34
```

Para obtener información más concreta sobre el error.

# ¿Dónde encontrar las cosas instaladas?

En este caso vamos a poner la ubicaciones de las cosas instaladas en Windows que son los que usaremos en el aula. En otros sistemas operativos la ruta será diferente. Podemos destacar pues ...

- **Android Studio:** `C:\Program Files\Android\Android Studio`

Se trata de la ubicación del IDE de Android Studio. En esta carpeta encontraremos el ejecutable para iniciar el IDE y sus plug-ins.

- **Android SDK:** `C:\Users\alumno\AppData\Local\Android\Sdk`

Se trata de la ubicación del SDK de Android que ha instalado el usuario actual. **Otro usuario podría usar otras versiones.**

- En la carpeta `platforms` están los SDK con las APIs de las versiones **34** de Android por ejemplo.
- En la carpeta `build-tools` las herramientas de compilación.
- En la carpeta `sources` los fuentes del SDK para depuración.
- En la carpeta `emulator` el AVD o Android Virtual Device.
- En la carpeta `system-images` las imágenes para los diferentes dispositivos virtuales android.
- etc.

- **Gradle:** `C:\Users\alumno\.gradle`

Se trata de la ubicación de las herramientas de compilación y las versiones de Gradle que se han instalado. Hablaremos de ello más adelante.

- **Personalización y opciones configuradas por el usuario:** `C:\Users\alumno\.android`

Se trata de la ubicación por defecto donde se guardarán los proyectos de Android Studio. Podemos cambiarla en la configuración.

Cabe destacar que en la carpeta `avd` se guardan las configuraciones de los dispositivos virtuales que hemos creado. En nuestro caso el **Pixel 3a API 34**. Ojo aquí no está ni el emulador, ni la imagen de Android, ni el SDK, ni nada. Solo la configuración del dispositivo virtual.

# Introducción a Gradle

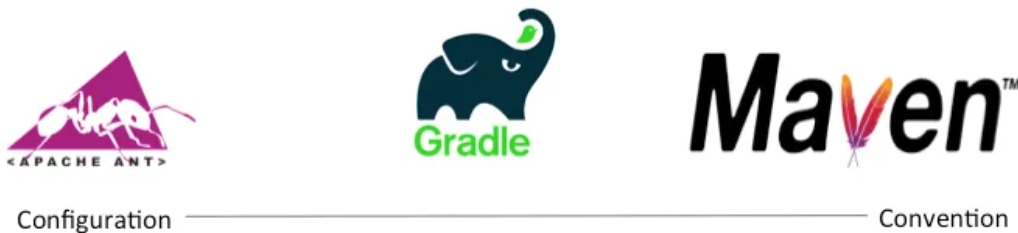
## Enlaces de interés

- [Video tutorial Gradle \(Vídeo DevExpert\)](#)
- [Gradle y el sistema de compilación de Android](#)

## ¿Qué es Gradle?

Ya que el ecosistema de Java y su JRE es muy extenso y tiene muchos 'actores' dispondremos de diferentes **gestores de proyectos** como [Ant](#), [Maven](#) y [Gradle](#). Sin embargo, el elegido para crear proyecto de Kotlin suele ser Gradle. Es una herramienta **CLI** (de la interfaz de línea de comandos) similar a las que usamos con C# como **dotnet** o con Javascript con **npm + gulp**, etc

**Gradle** es una herramienta de [automatización de la construcción de código](#) que bebe de las aportaciones que han realizado herramientas como **Ant** y **Maven**, pero intenta llevarlo todo un paso mas allá, mezclando características de ambos.




Para empezar se apoya en lenguajes sobre JRE como [Groovy](#) y el propio **Kotlin** usados como un **DSL (Domain Specific Language)** y que facilitan el trabajo con un lenguaje sencillo y claro a la hora de construir el build, comparado con Maven que usa **XML** que es muy verboso y tiene limitaciones como la definición de variables.

✦ **Nota:** Cabe destacar que Gradle forma parte de la [Kotlin Foundation](#) por lo que Kotlin es el lenguaje de DSL soportado **por defecto** desde 2023.

Por otro lado, dispone de una gran flexibilidad que permite trabajar con otros lenguajes y no solo **Java**.

**Gradle** nos gestiona automáticamente las dependencias del proyecto. Tradicionalmente, para usar una biblioteca de Android en un proyecto, había que descargar el código fuente, agregarlo al proyecto y pasar a la compilación. Gradle automatiza este proceso, ya que con agregar una sola

línea al archivo `build.gradle.kts` del módulo, se encargará de descargar la biblioteca compilada de un repositorio público y la agregará al proyecto.

 **Nota:** A continuación vamos a crear nuestro primer proyecto en gradle. Para ello vamos a usar **una plantilla** del IDE de Android Studio. Sin embargo, también podemos crear un proyecto de gradle desde consola usando el propio Gradle. Esta última opción la veremos más adelante al crear un proyecto de Kotlin vacío.



# Creando y ejecutando el primer proyecto

Usaremos el IDE Android Studio y tras realizar la configuración, para crear un proyecto nuevo pulsaremos en **'New Project'**. Nos aparecerá un diálogo donde configuraremos el proyecto.

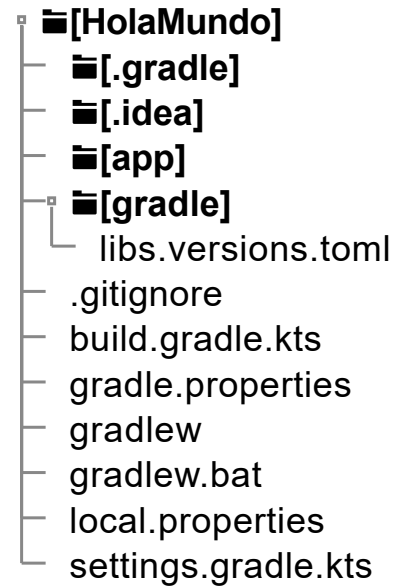
1. Seleccionaremos el template **'Empty Activity'** y pulsaremos **'Next'**
2. En el siguiente diálogo:
  1. Le daremos un nombre al proyecto. En nuestro caso **HolaMundo**
  2. El nombre del paquete podría ser **com.pmdm.holamundo**
  3. En ubicación podemos seleccionar el escritorio **C:\Users\alumno\Desktop\HolaMundo**.

✦ **Nota:** Fíjate que la carpeta con el nombre del Workspace se crea automáticamente y coincide con la del proyecto.
4. En SDK mínimo podemos seleccionar la versión **28** que significará que aunque el SDK de desarrollo usemos una versión posterior, el programa será compatible con dispositivos con la versión 28 o superior.
5. Por último en Build configuration dejaremos la opción por defecto **Kotlin DSL**. Esto significa que el archivo de configuración del proyecto será un archivo **build.gradle.kts** en lugar de **build.gradle** que es el que usa por defecto **Groovy**.

✦ **Nota:** Aunque el archivo de configuración sea un archivo **build.gradle.kts** podemos usar **Groovy** en lugar de **Kotlin**. Sin embargo, Kotlin es el lenguaje de DSL soportado en **por defecto** desde 2023.

Internamente Android Studio creará una carpeta con el nombre del proyecto que se corresponderá con el Workspace. En ella, se creará una plantilla de proyecto Gradle que contendrá a groso modo lo siguiente:

1. **.gradle** : Carpeta que contiene los archivos de configuración de Gradle. No deberemos borrarla nunca.
2. **.idea** : Carpeta que contiene los archivos de configuración del proyecto en el IDE Android Studio. La crea Android estudio al abrir un proyecto de Gradle. **Se podría borrar** sobre todo si queremos abrir un proyecto antiguo con una versión de Android Studio más reciente.
3. **app** : Módulo de la aplicación. Contiene el código fuente, recursos, etc. de nuestra aplicación. Hablaremos de su contenido más adelante.
4. **gradle** : Carpeta que contiene una **instalación de la versión de Gradle** que usa el proyecto. De esta manera si no está instalada al compilar el proyecto por primera vez se instalaría. Dentro tiene el fichero **libs.versions.toml** que contiene las versiones de las librerías que usa Gradle y que trataremos más adelante.
5. **.gitignore** : Archivo que contiene los archivos y carpetas que no queremos que se suban al repositorio Git. En nuestro caso, no queremos que se suban los archivos de configuración del IDE Android Studio, ni los archivos de configuración de Gradle, ni los archivos de compilación, ni los archivos de configuración de Git, etc.
6. **gradle.properties** : Archivo de configuración de Gradle que no debemos tocar.
7. **gradlew y gradlew.bat** : Script de Gradle que nos permite ejecutar tareas de Gradle desde consola. Lo usará internamente el Android Studio para realizar los diferentes procesos de compilación, generación de apk, instalación, ejecución, etc.
8. **local.properties** : Archivo de configuración de Gradle que le indica al Plugin su plugin para Android donde está ubicado el SDK de Android. No debemos tocarlo. El IDE Android Studio lo gestiona por nosotros si hay alguna inconsistencia.
9. **settings.gradle.kts** : Archivo de configuración de Gradle donde se indica el **nombre del proyecto**, los **repositorios de dónde descargar las dependencias** y los **módulos que componen el proyecto**. En nuestro caso solo tenemos el módulo **app**.
10. **libs.versions.toml** : Un archivo **TOML** (Tom's Obvious, Minimal Language) en Gradle actúa como un catálogo centralizado para gestionar las dependencias de tu proyecto. Piensa en él como una lista de compras bien organizada donde defines qué bibliotecas externas necesita



tu proyecto, junto con sus versiones específicas.

Como **ventajas principales** podemos destacar:

- **Centralización**: Todas las dependencias se gestionan en un solo lugar, facilitando su actualización y mantenimiento.
- **Legibilidad**: El formato TOML es sencillo de leer y entender.  
Reutilización: Puedes compartir el catálogo TOML entre diferentes proyectos.
- **Evita Conflictos**: Gradle puede ayudarte a detectar y resolver conflictos de versiones entre dependencias.

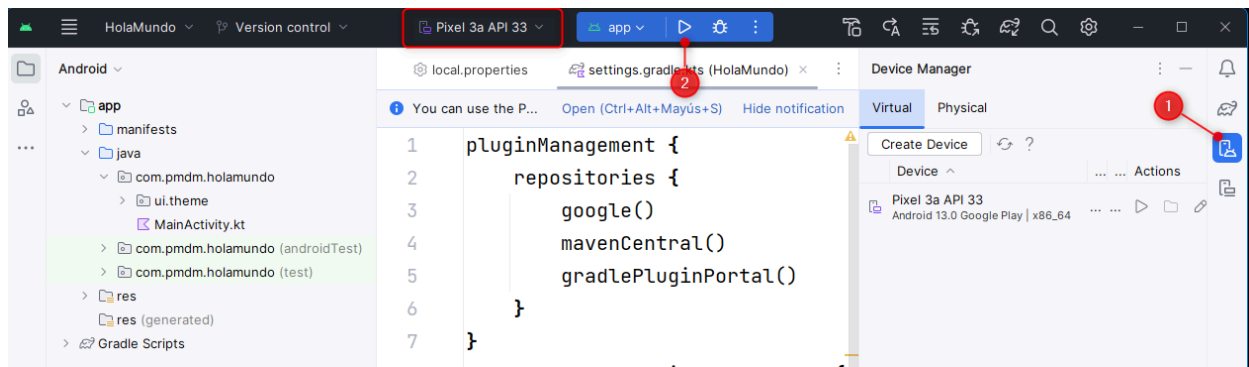
La **estructura de un archivo** `libs.versions.toml` se divide en tres secciones principales:

- **[versions]** : Aquí declaras las versiones específicas de las bibliotecas que usarás.  
Por ejemplo, la **versión** las librerías de Google sobre la JRE de Kotlin  
`guava = "32.1.3-jre"` .
- **[libraries]** : Defines alias para las coordenadas de tus dependencias.  
Por ejemplo, `guava = { module = "com.google.guava:guava", version.ref = "guava" }`  
donde el módulo `com.google.guava:guava` se identifica con el alias `guava` y se refiere a la versión `guava` definida en la sección **[versions]** .
  - `com.google.guava` : El **group ID**, que identifica la organización o proyecto responsable de la biblioteca.
  - `guava` : El **artifact ID**, que es el nombre específico de esta biblioteca dentro del proyecto.
- **[plugins]** : Aquí defines los plugins de Gradle que usarás en tu proyecto. Por ejemplo, la versión del lenguaje Kotlin sobre la JVM que vamos a usar  
`jvm = { id = "org.jetbrains.kotlin.jvm", version = "1.9.22" }` .

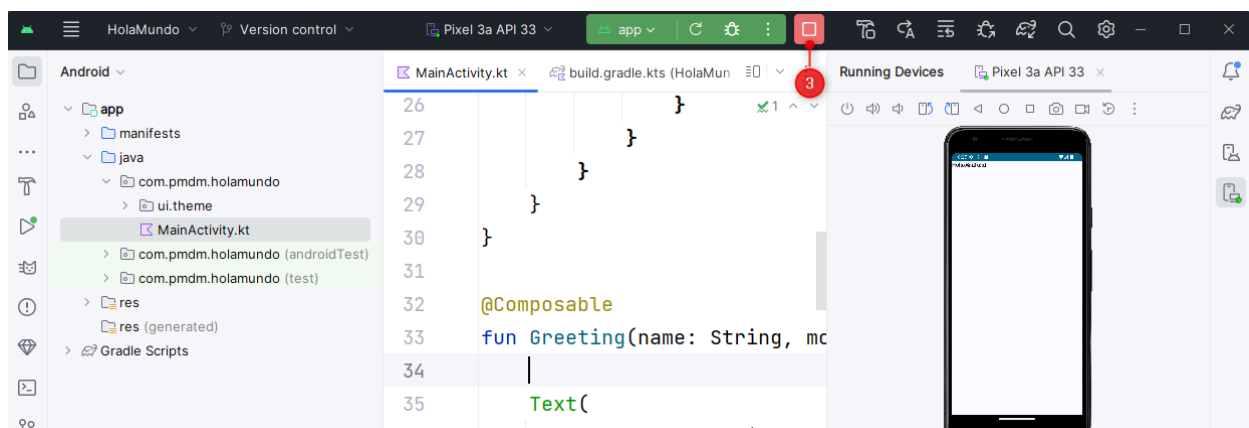
11. `build.gradle.kts` : Archivo de configuración de Gradle donde le indicamos que **plugins** y **dependencias** que queremos usar **en un módulo concreto de los que forman nuestro proyecto** y hemos definido en el archivo `libs.versions.toml` . Nosotros normalmente solo vamos a tener un módulo que es `app` .

```
plugins {  
    // Plugin jvm definido en la sección [plugins] de libs.versions.toml  
    alias(libs.plugins.jvm)  
    // Plugin que indica a Gradle que es una aplicación de consola sobre la JVM  
    application  
}  
...  
dependencies {  
    // Dependencia de la librería guava definida en la sección [libraries] de libs.versions.toml  
    implementation(libs.guava)  
}
```

Bueno ya conocemos un poco la estructura de un proyecto Gradle. Ahora vamos a ejecutarlo.



Para ello nos aseguraremos de que hemos creado bien el dispositivo virtual [1] y posteriormente pulsaremos en el botón de ejecutar [2]. Tras unos segundos se debería ejecutar la aplicación en el dispositivo virtual.



Podremos parar la ejecución pulsando en el botón de parar [3].

# Ahondando un poco en el uso de Gradle

Ya hemos comentado que Android Studio va a ejecutar por nosotros las tareas de gradle. Sin embargo, es conveniente conocer un poco el funcionamiento básico para poder solucionar problemas que puedan surgir.

En primer lugar vamos a añadir a la variable de entorno `PATH` la ruta donde se encuentra el ejecutable de Gradle.

## Añadiendo al PATH del sistema la versión de Gradle

Las versiones de Gradle que nos instalará Android Studio en Windows **al ejecutar nuestro primer proyecto** se guardará en la siguiente ubicación: `C:\Users\alumno\.gradle\wrapper\dists`

Aunque AndroidStudio instala su propia versión del JDK, es conveniente tener [JRE](#) instalado en el equipo y las variables de entorno `JAVA_HOME` y `JRE_HOME` establecidas para que el compilador de Kotlin pueda encontrarlo al ejecutar en la consola.

Por ejemplo, si hemos instalado el JDK de [Adoptium](#) sería:

```
JAVA_HOME=C:\Program Files\Eclipse Adoptium\jdk-17.0.6.10-hotspot\  
JRE_HOME=C:\Program Files\Eclipse Adoptium\jdk-17.0.6.10-hotspot\
```

o si queremos usar el JDK que trae la propia instalación de Android Studio sería:


```
JAVA_HOME=C:\Program Files\Android\Android Studio\jbr  
JRE_HOME=C:\Program Files\Android\Android Studio\jbr
```

Una vez instalado, elegiremos una de las versiones (para estos ejemplos hemos elegido la versión **8.7**) por tanto dentro de la carpeta `dists` iremos a:

```
...\gradle-8.7-bin\<hash de la versión>\gradle-8.7\bin
```

donde se encontrará el ejecutable que queremos añadir a `PATH`.

Para [cambiar el valor de la variable de sistema](#) `Path` ejecutaremos **Panel de Control → Editar las variables del sistema** y añadiremos toda la ruta a la variable de entorno `Path` para nuestro sistema o usuario.

 **Nota:** Podríamos [descargarlo](#) nosotros directamente e instalarlo donde queramos. Pero en nuestro caso no lo vamos a hacer y vamos a usar la versión que nos instala **Android**

## Ejecutando tareas de Gradle desde consola

En primer lugar vamos a ver cómo se ejecuta una tarea de Gradle desde consola. Para ello abriremos una consola de comandos y nos situaremos en la carpeta del proyecto. En nuestro caso `C:\Users\alumno\Desktop\HolaMundo` . Una vez allí ejecutaremos el siguiente comando:

📌 **Nota:** También podremos ejecutarlo desde la consola de Android Studio abriendo un terminal con `Ctrl+Alt+1` o desde el el menú `View → Tool Windows → Terminal` .

Para ver todas las tareas disponibles ejecutaremos:

```
PS C:\Users\alumno\Desktop\HolaMundo> ./gradlew tasks --all
```

Por ejemplo si ejecutamos la tarea...

```
PS C:\Users\alumno\Desktop\HolaMundo> ./gradlew app:packageRelease
```

gradle haría las siguientes tareas:

1. Comprobar si está instalada la versión de Gradle que usa el proyecto. Si no lo está la instalaría en el directorio `C:\Users\alumno\.gradle\wrapper\dists` .
2. Ver si está compilada la última versión de la aplicación. Si no lo está la compilaría.
3. Generaría la versión **release** de la apk de la aplicación en el directorio `C:\Users\alumno\Desktop\HolaMundo\app\build\outputs\apk\release` .

También podemos ver las tareas disponibles desde el propio IDE pulsando el botón de gradle [1] después la opción `Execute Gradle task` [2] y finalmente escribiendo el nombre de la tarea en el diálogo donde se nos muestran todas ellas [3].

