Gradle

> ⚠ **DOCUMENTO EN DESARROLLO** ⚠

Introducción

Gradle automatiza la construcción, prueba y implementación de software a partir de información en scripts de construcción.

Un proyecto Gradle es una pieza de software que se puede construir, como una aplicación o una biblioteca.

Las construcciones de proyectos únicos incluyen un solo proyecto llamado **proyecto raíz** o ***root project***. Las construcciones multiproyectos incluyen un proyecto raíz y cualquier número de subproyectos.

Los scripts de construcción le indican a Gradle los pasos necesarios para construir el proyecto. Cada proyecto puede incluir uno o varios scripts de construcción.

La **gestión de la dependencias** es una técnica automatizada para declarar y resolver los recursos externos requeridos por un proyecto. Cada proyecto típicamente incluye una serie de dependencias externas que Gradle resolverá durante la construcción.

Las **tareas** o ***tasks*** son una unidad de trabajo básica, como compilar código o ejecutar su prueba. Cada proyecto contiene una o más tareas definidas dentro de un script de compilación o un plugin.

Gradle tiene soporte para proyectos en Android, Java, Kotlin Multiplatform, Groovy, Scala, Javascript, y C/C++.

Instalación

Gradle requiere que esté instalado **Java Development Kit (JVM)** versión 8 o superior.

Es recomendable que esté definida la variable **JAVA_HOME** en las variables del sistema. Utilizamos esta variable para añadir al path el directorio *'bin'* de Java con la forma `%JAVA_HOME%\bin;`

```
# CMD
$ echo %JAVA_HOME%
$ echo %PATH%

# PowerShell
$ $env:java_home
$ Get-Item Env:JAVA_HOME
```

Comprobamos si Java está instalado correctamente ejecutando `java -version` en el terminal.

Una vez tenemos Java instalado correctamente se instala Gradle en el sistema. En el caso de Windows, se descarga el fichero .zip de Gradle y se descomprime en la ubicación deseada.

Al igual que Java, se añade la variable **GRADLE_HOME** a las variables del sistema para que apunte al directorio donde se ha descomprimido. Se utiliza esta variable para añadir al path el directorio *'bin'* de Gradle con la forma `%GRADLE_HOME%\bin;`.

Comprobamos si se ha instalado correctamente ejecutando `gradle -v` en un terminal.

Por defecto, el directorio raíz del usuario se encuentra en '~/.gradle o C:\Users\USERNAME\.gradle' y almacena las propiedades de configuración globales, los scripts de inicialización, ficheros de caché y ficheros de registro.

Se puede establecer con la variable de entorno **GRADLE_USER_HOME**.

- Más información

Inicialización de un proyecto

La forma de inicializar un proyecto con Gradle o añadir Gradle a un proyecto existente es mediante la tarea **init** de Gradle. Soporta la creación de nuevas construcciones de Gradle de varios tipos, así como la conversión de las construcciones existentes de Apache Maven a Gradle.

```
# Ejemplo de comando para inicializar un proyecto
$ gradle init \
  --type java-application \
  --dsl kotlin \
  --test-framework junit-jupiter \
  --package my.project \
  --project-name my-project  \
  --no-split-project  \
  --java-version 17
```

La forma más simple y recomendada de usar la tarea es ejecutar `gradle init` desde una consola interactiva. Gradle irá mostrando diferentes opciones y el usuario deberá ir seleccionando aquellas opciones que más se ajusten al proyecto:

```
# Modo interactivo
$ gradle init
```

Hay varias opciones de línea de comandos disponibles para la tarea de inicialización. Para mostrar las opciones disponibles se utiliza la tarea **"help"** de Gradle (disponible también para todas las tareas):

```
# Mostrar ayuda de la tarea 'init'
gradle help --task init
```

Esta tarea sirve para diferentes tipos de compilación de proyectos. Estos se enumeran a continuación:

- pom - Converts an existing Apache Maven build to Gradle
- basic - A basic, empty, Gradle build
- java-application - A command-line application implemented in Java
- java-gradle-plugin - A Gradle plugin implemented in Java
- java-library - A Java library
- kotlin-application - A command-line application implemented in Kotlin/JVM
- kotlin-gradle-plugin - A Gradle plugin implemented in Kotlin/JVM
- kotlin-library - A Kotlin/JVM library
- groovy-application - A command-line application implemented in Groovy
- groovy-gradle-plugin - A Gradle plugin implemented in Groovy
- groovy-library - A Groovy library
- scala-application - A Scala application
- scala-library - A Scala library
- cpp-application - A command-line application implemented in C++
- cpp-library - A C++ library

Gestión de proyectos múltiples

Gradle es capaz de gestionar tanto proyectos monolíticos, es decir, proyectos con un único proyecto raíz como proyectos compuestos por varios subproyectos o submódulos.

En estos casos, el fichero `settings.gradle` se ubica en la raíz del proyecto mientras que cada subproyecto se ubica en su propia carpeta y cuenta con su propio fichero `build.gradle`.

Para visualizar la lista de proyectos en formato árbol, Gradle dispone de la tarea `projects`:

```
gradle projects
```

Debido a que cada subproyecto puede tener su propia lista de tareas, hay dos formas de invocar tareas en proyectos compuestos.

Una forma es invocar las tareas por su **nombre simple**. Si estamos en el directorio raíz, al invocar una tarea por nombre invocaremos todas las tareas de los subproyectos que tengan ese nombre. Si estamos ubicados en un subproyecto, únicamente se invocan las tareas de ese subproyecto (y de otros subproyectos que puedan depender de éste) que tengan ese nombre.

Otra forma es invocar las tareas por su **nombre calificado** o *qualified name*. Este nombre se compone de `:{subproyecto}:{tarea}`.

> ⚠ El primer `':'` no es un separador com tal sino que se corresponde con el **nombre del proyecto raíz**.

En proyectos que se componen de varios subproyectos, es normal que alguno de los subproyectos tenga una dependencia con otro de los subproyectos.

Para construir un proyecto con subproyectos tenemos varias tareas:

```
# La tarea 'build' compila, prueba y valida un único proyecto
$ gradle build

# Monta y prueba este proyecto y todos los proyectos que dependen de él.
$ gradle :subproject:buildDependents

# Ensambla y prueba este proyecto y todos los proyectos de los que depende.
$ gradle :subproject:buildNeeded
```

Ciclo de vida de construcción

TODO

Fichero de configuración

El archivo de configuración `settings.gradle` en Groovy o `settings.gradle.kts` en Kotlin es el punto de entrada de cada proyecto Gradle.

El objetivo principal del archivo de configuración es añadir subproyectos a su compilación:

- Para proyectos únicos, el fichero de configuración es **opcional**.
- Para proyectos múltiples, el fichero de configuración es **obligatorio** y declara todos los supbroyectos.

Este fichero se encuentra en la raíz del proyecto y acepta los lenguajes Groovy DSL y Kotlin DSL.

Un ejemplo de fichero de configuración con tres subproyectos:

```
rootProject.name = 'root-project'

include('sub-project-a')
include('sub-project-b')
include('sub-project-c')
```

- Más información

Fichero de construcción

TODO

Gestión de dependencias

TODO

Gestión de plugins

TODO

Tareas

Gradle Wrapper Reference

La forma recomendada de ejecutar cualquier construcción de Gradle es con la ayuda del **Wrapper** de Gradle.

El **Wrapper** es un script que invoca una versión declarada de Gradle, descargándolo en la carpeta del usuario si es necesario.

En los casos en que hay diversos desarrolladores trabajando en un mismo proyecto, si se utiliza el comando `gradle`, cada desarrollador realizará la compilación del proyecto con su propia versión de Gradle que tenga instalada en su sistema.

En cambio, al usar el **Wrapper** mediante el comando `.\gradlew` se estandariza la versión utilizada, ya que al estar incluido en el control de versiones todos los desarrolladores utilizarán la misma versión de Gradle.

Para añadir el **Wrapper** a un proyecto es necesario que esté instalado Gradle. Sin embargo, una vez añadido a la raíz del proyecto, cualquier otro desarrollador que se descargue el proyecto podrá utilizar este **Wrapper** sin necesidad de tener instalado Gradle en su equipo.

Adding the Gradle Wrapper

Para añadir el **Wrapper** a un proyecto (si aún no se ha añadido) se requiere que haya una versión de Gradle instalada en la máquina y por tanto que sea accesible vía línea de comandos. Nos situaremos en la raíz del proyecto y ejecutaremos la tarea correspondiente:

```
gradle wrapper
```

Esta tarea genera una carpeta con los ficheros `gradle-wrapper.jar` y `gradle-wrapper.properties`. El fichero de propiedades contendrá la URL de la versión de Gradle en uso, además de otra información.

> ⚠ Es recomendable añadir estos ficheros al **control de versiones** para compartirlos con el resto de desarrolladores del proyecto.

Upgrading the Gradle Wrapper

Para actualizar la versión del **Wrapper** se puede modificar la versión directamente en el fichero `gradle-wrapper.properties` aunque la forma recomendada es utilizar la tarea correspondiente:

```
# Actualiza el Wrapper a la última versión
$ ./gradlew wrapper --gradle-version latest

# Actualizar una versión específica
$ ./gradlew wrapper --gradle-version 8.4

# Comprobar la versión
$ ./gradlew --version
```

- Más información

Command-Line Interface Reference

La interfaz de línea de comandos es el método principal de interactuar con Gradle.

Se recomienda el uso del **Wrapper** de Gradle por las ventajas que aporta haciendo uso de `./gradlew` en Linux/macOS o `gradlew.bat` en Windows dentro de la raíz del proyecto. Sin embargo, podemos usar la versión de Gradle instalada en el sistema desde cualquier ubicación con:

- `gradle [taskName...] [--option-name...]`

Las opciones se permiten tanto delante como detrás de la tarea:

- `gradle [--option-name...] [taskName...]`

En el caso de especificar múltiples tareas, se deben separar con un espacio:

- `gradle [taskName1 taskName2...] [--option-name...]`

Ejemplos de comandos:

```
# Mostrar la versión de Gradle
$ gradle --version

# Mostrar la ayuda
$ gradle --help

# Mostrar la ayuda de una tarea
$ gradle help --task {tarea}

# Mostrar la lista de tareas desde la raíz del proyecto
$ gradle tasks # normal
$ ./gradlew tasks # usando el wrapper

# Mostrar la lista detallada de tareas
$ gradle tasks --all # normal
$ ./gradlew tasks --all # usando el wrapper
```

- Más información

Referencias

- https://gradle.org/
- https://www.baeldung.com/gradle-guide
- https://gradlehero.com/
- https://www.john-cd.com/cheatsheets/Java/Gradle/

Licencia