

## OPCIONES DE LÍNEA DE COMANDO GRADLE



El comando **gradle** se usa para ejecutar una compilación. Este comando acepta varias opciones de línea de comandos. Conocemos la opción **--quiet** (o **-q**) para reducir la salida de una compilación. Si usamos la opción **gradle --help** (o **-h** o **-?**), Vemos la lista completa de opciones:

```
C:\Users\bquevedof>gradle --help
```

La estructura base para usar los comandos es:

**gradle [option...] [task...]**

- ?, -h, --help** Muestra la ayuda del comando.
- a, --no-rebuild** No reconstruir dependencias del proyecto.
- b, --build-file** Especifica el archivo de compilación.
- C, --cache** Especifica cómo se deben almacenar en caché los scripts de compilación compilados. Los valores posibles son: 'rebuild' y 'on'. El valor predeterminado es 'on' [deprecated - Use '--rerun-tasks' or '--recompile-scripts' instead]
- c, --settings-file** Especifica el archivo de configuración.
- continue** Continúa la ejecución de la tarea después de una falla. [experimental]
- D, --system-prop** Establecer la propiedad del sistema de la JVM (por.ej. -Dmyprop=myvalue).
- d, --debug** Inicie sesión en modo de depuración (includes normal stacktrace).
- daemon** Uses the Gradle daemon to run the build. Starts the daemon if not running.
- foreground** Starts the Gradle daemon in the foreground. [experimental]
- g, --gradle-user-home** Especifica el directorio de inicio del usuario de gradle.
- gui** Lanza la interfaz gráfica de usuario de Gradle.
- I, --init-script** Especifica un script de inicialización.
- i, --info** Establecer nivel de registro a información.
- m, --dry-run** Ejecuta las compilaciones con todas las acciones de tareas deshabilitadas.
- no-color** No use color en la salida de la consola.

**--no-daemon** Do not use the Gradle daemon to run the build.

**--no-opt** Ignorar cualquier optimización de tareas. [deprecated - Use '--rerun-tasks' instead]

**--offline** La compilación debe funcionar sin acceder a los recursos de la red..

**-P, --project-prop** Establecer propiedad del proyecto para el script de compilación (ej. - Pmyprop=myvalue).

**-p, --project-dir** Especifica el directorio de inicio de Gradle. El valor predeterminado es el directorio actual.

**--profile** Los perfiles generan tiempo de ejecución y generan un informe en el <build\_dir>/reports/profile directory.

**--project-cache-dir** Especifica el directorio de caché específico del proyecto. El valor predeterminado es .gradle en el directorio raíz del proyecto.

**-q, --quiet** Solo errores de registro.

**--recompile-scripts** Forzar la compilación del script de compilación.

**--refresh** Actualiza el estado de los recursos del tipo o tipos especificados. Actualmente solo se admiten 'dependencias'. [deprecated - Use '--refresh-dependencies' instead.]

**--refresh-dependencies** Actualizar el estado de las dependencias.

**--rerun-tasks** Ignorar los resultados de la tarea previamente almacenados en caché.

**-S, --full-stacktrace** Imprima el seguimiento de pila completo (muy detallado) para todas las excepciones.

**-s, --stacktrace** Imprima el seguimiento de la pila para todas las excepciones.

**--stop** Detiene el Gradle daemon si se está ejecutando.

**-u, --no-search-upward** No busque en las carpetas principales un archivo settings.gradle.

**-v, --version** Información de la versión impresa.

**-x, --exclude-task** Especifique una tarea para excluir de la ejecución.

## TAREAS ESTANDAR DE GRADLE

En esta sección se expondrán algunas tareas por defecto de Gradle. Estas tareas serán útiles para el desarrollo de nuevas tareas personalizadas. Aquí encontrará mayor información: [API Documentation: Task](#)

Gradle build : Compilación y generación de artefactos

Gradle dependencies : Se muestra en consola todo el arbol de dependencias

Gradle clean: Se borran todas los artefactos generados del comando gradle build (reportes, jars, wars...)

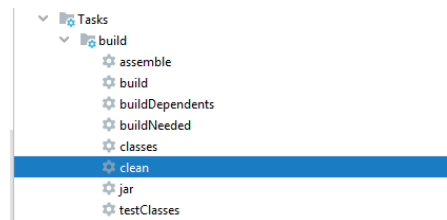
Para ejecutar nuestro proyecto utilizamos el siguiente comando en la consola del proyecto:

**gradle -Dtest.single=RunnerTags test aggregate**

aggregate: genera los informes agregados de Serenity a partir de los resultados de la prueba JSON producidos cuando ejecuta las pruebas de Serenity BDD.

**Nota:** para que funcione el comando aggregate es necesario tenerlo agregado [**apply plugin: 'net.serenity-bdd.aggregator'**] en el archivo **build.gradle**

**Tareas Comunes:** Algunas tareas que utilizamos constantemente en cualquier proyecto Gradle son:



**Assemble** =Traduce los archivos fuente del lenguaje ensamblador en archivos objeto.

**Build** = Tareas agregadas que realizan una compilación completa del proyecto.

**buildDependents** = Realiza una compilación completa del proyecto y todos los proyectos que dependen de él.

**buildNeeded**=Realiza una compilación completa del proyecto y todos los proyectos de los que depende.

**classes**= Esta es una tarea agregada que solo depende de otras tareas. Otros complementos pueden adjuntarle tareas de compilación adicionales.

**Clean** = Elimina el directorio de compilación del proyecto.

**Jar** = Ensambla el archivo JAR de producción, en función de las clases y los recursos adjuntos al conjunto de origen principal.

**testClasses** =Esta es una tarea agregada que solo depende de otras tareas. Otros complementos pueden adjuntarle tareas de compilación de prueba adicionales.

Para más información: [Task](#)

## EJECUTE PRUEBAS ESPECÍFICAS

A continuación se muestran otros ejemplos para ejecutar tu proyecto desde la consola.

SHELL

```
# You can view the test report by opening the HTML output file, located at
# ./build/reports/tests/test/index.html
#
# Test one class
gradle test --tests *LibraryTest
gradle test -Dtest.single=LibraryTest
# Test one method
gradle test --tests *LibraryTest.testSomeLibraryMethod
# Test two methods
gradle test --tests *LibraryTest.testFoo --tests *LibraryTest.testBar
# Test multiple items with a wildcard (*)
gradle test --tests *Lib*
gradle test -Dtest.single=L*Test
# Test a specific Package and prefix
gradle test --tests my.company*.Library*
```

## VOLVER A EJECUTAR PRUEBAS

Gradle no volverá a ejecutar una tarea si la entrada y la salida no han cambiado. Para volver a ejecutar las pruebas, proporcione un argumento o tarea adicional si la tarea de prueba está ACTUALIZADA

SHELL

```
# Deletes files created during test
gradle cleanTest test --tests *LibraryTest
# Or force the tasks to be re-run
gradle --rerun-tasks test --tests *LibraryTest
# Or clean the whole build before running the task
gradle clean test --tests *LibraryTest
```

## INTRODUCCIÓN A GRADLE WRAPPER

Gradle tiene la capacidad de agregar una envoltura a los proyectos. Esta envoltura alivia la necesidad de que todos los usuarios o sistemas de integración continua tengan instalado Gradle. También evita los problemas de versión en los que existe cierta incompatibilidad entre la versión que usa el proyecto y la que los usuarios han instalado. Lo hace instalando una versión de gradle localmente en el proyecto.

Los usuarios del proyecto simplemente ejecutan:

```
C:\Users\bquevedof>gradlew <task> # on Windows
```

```
C:\Users\bquevedof>./gradlew <task> # on *Nix or MacOSX_
```

Para configurar un proyecto para usar un contenedor, los desarrolladores:

Ejecutan

**gradle wrapper [--gradle-version 4.7]**

Donde **--gradle-version X** es opcional y si no se proporciona (o la tarea de envoltura no se incluye, como se muestra a continuación), la versión utilizada es la versión de gradle que se está utilizando.

Para forzar al proyecto a usar una versión específica, agregue lo siguiente a **build.gradle** :

```
task wrapper(type: Wrapper) {  
    gradleVersion = '4.7'  
}
```

Cuando se usa el comando **gradle wrapper** crea los siguientes archivos:

```
PruebaGradle/  
gradlew  
gradlew.bat  
gradle/wrapper/  
gradle-wrapper.jar  
gradle-wrapper.properties
```

Este equipo > Nuevo vol (D:) > PruebaGradle

Nombre
.gradle
gradle
gradlew
gradlew.bat

A continuación procedemos a explicar cada uno de los archivos creados por gradle wrapper

gradlew: es el shell script Unix que los usuarios pueden ejecutar para ejecutar tareas de Gradle

gradlew.bat: El bat script de los usuarios de Windows pueden ejecutar tareas de Gradle

gradle/wrapper/gradle-wrapper.jar: El JAR ejecutable del contenedor; aquí es donde reside el código contenedor

gradle/wrapper/gradle-wrapper.properties: Un archivo de propiedades para configurar el contenedor.

Ahora hablemos de lo que hace y lo que no hace gradle wrapper.

## **QUE HACE GRADLE WRAPPER**

Cuando ejecuta Gradle Wrapper, realiza las siguientes acciones:

1. Analiza los argumentos pasados a gradlew.
2. Instala la versión correcta de Gradle.
3. Invoca Gradle para ejecutar las tareas especificadas.

Tenga en cuenta que el wrapper solo acepta dos argumentos de configuración opcionales:

1. **-q** o **--quiet** para suprimir la salida
2. **-g** o **--gradle-user-home** para especificar un directorio de inicio alternativo para Gradle.

El paso de instalación de Gradle es el único que es particularmente interesante. Primero verificará su GRADLE\_USER\_HOME para la distribución de Gradle deseada. Si ya existe, el contenedor continúa invocando a Gradle. Si la distribución aún no existe, el contenedor descargará la distribución.

También es importante tener en cuenta que wrapper ignora por completo cualquier distribución de Gradle en su sistema. Wrapper descarga y descomprime las distribuciones de Gradle en su propio directorio separado y solo usará esas distribuciones para compilaciones. Wrapper utilizará cualquier configuración global de Gradle, como un archivo gradle.properties definido en su directorio de inicio de Gradle cuando ejecute tareas.

## **LO QUE EL GRADLE WRAPPER NO HACE**

El Gradle Wrapper en sí no ejecuta sus tareas. Todo lo que hace es asegurarse de que tiene la distribución de Gradle deseada y luego invocarla. Wrapper es una capa muy delgada para eliminar la necesidad de que los desarrolladores de redes administren las distribuciones ellos mismos.

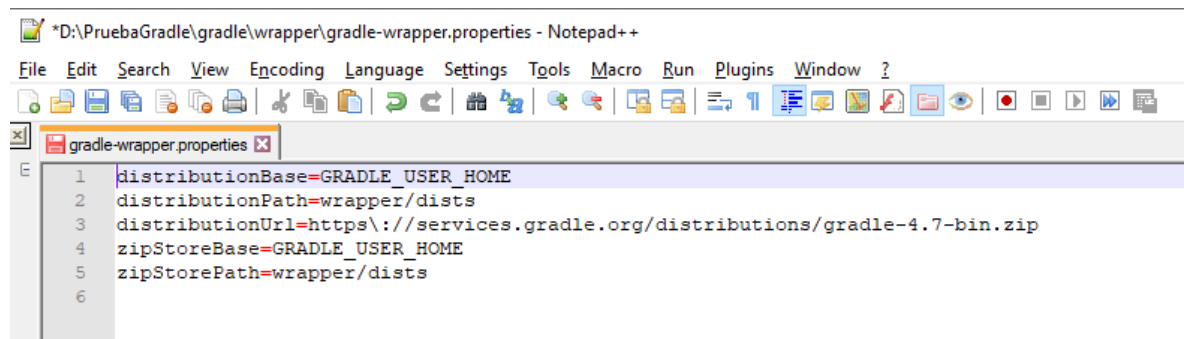
Esto significa que wrapper está efectivamente desacoplada por completo de Gradle. Un contenedor de 2014 puede construir un proyecto usando Gradle 4.0, y un contenedor instalado hoy puede construir un proyecto usando Gradle 2.0.

Si encuentra un problema con sus compilaciones, es extremadamente improbable que el contenedor tenga la culpa, ya que principalmente invoca a Gradle correctamente.

## **Configuración de Wrapper**

Mencioné anteriormente que uno de los archivos que el contenedor coloca en su proyecto es un archivo de configuración en gradle / wrapper / gradle-wrapper.properties.

Este archivo generalmente se ve así:



```
*D:\PruebaGradle\gradle\wrapper\gradle-wrapper.properties - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-4.7-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

**distributionBase:** especifica la ruta en la que el contenedor almacenará las distribuciones de Gradle. De manera predeterminada, GRADLE\_USER\_HOME.

**distributionPath:** almacenará las distribuciones de Gradle en / .gradle / wrapper / dists.

**distributionUrl:** es el que más importa. Esto es lo que especifica qué versión de Gradle desea usar para sus compilaciones y dónde descargarla.

**zipStoreBase y zipStorePath:** son muy similares debido a que estos especifican dónde almacenará el contenedor las distribuciones comprimidas que descarga.

Para más información: [Gradle Wrapper](#)

## GIT IGNORE

Es un archivo que sirve para decirle a Git qué archivos o directorios completos debe ignorar y no subir al repositorio de código.

## CONFIGURACIÓN DE GIT IGNORE

El archivo git.ignore debe agregarse en la creación del repositorio. De no ser así el desarrollador debe llevarlos al repositorio de Artifactory desde su proyecto local.

En el archivo .gitignore se debe validar que se estén ignorando los archivos .jar sin embargo, el gradle wrapper sube un jar que es necesario para la compilación automática del proyecto, por lo tanto únicamente este jar se debe poder subir al repositorio y esto se debe especificar en dicho archivo.

Para lo mencionado anteriormente se debe validar que el archivo .gitignore tenga las siguientes líneas:

```
### Java ###
# Compiled class file
*.class
# Log file
*.log
# BlueJ files
*.ctxt
# Mobile Tools for Java (J2ME)
.mtj.tmp/
# Package Files #
*.jar
*.war
*.ear
*.zip
*.tar.gz
*.rar
# virtual machine crash logs, see http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*
```

*¡Importante! que despues de las líneas de JAVA tenga las siguientes de GRADLE:*

```
### Gradle ###
.gradle
build/
# Ignore Gradle GUI config
gradle-app.setting
# Avoid ignoring Gradle wrapper jar file (.jar files are usually ignored)
!gradle-wrapper.jar
# Cache of project
.gradletasknamecache
```

NOTA: Tener en cuenta que esto es una plantilla de ejemplo, por ende en el gitignore debe ignorar cualquier tipo de archivo que no se deba subir, ejemplo de esto, los archivos que generan los IDE's o herramientas similares, como:

```
.settings
.classpath
.project
etc...
```