Cómo instalar Gradle, herramienta de automatización de builds

Por **Alejandro Pérez García** - 4 diciembre, 2013

Creación: 03-12-2013

Índice de contenidos

1. Introducción
2. Entorno
3. Prerrequisitos
4. Descarga y desempaquetado
5. Variables de entorno
6. Construyendo nuestro primer proyecto Java
7. Conclusiones
8. Sobre el autor

1. Introducción

Gradle es una herramienta para automatizar el proceso de construcción de software, o lo que comúnmente se llama, *hacer una build*

Viene a ser un **Maven** moderno, donde podemos destacar las siguientes diferencias:

- Se escribe en un DSL de Groovy en lugar de en XML, por lo que queda mucho más conciso.
- Al ser Groovy un lenguaje de programación, tenemos mucha más flexibilidad para particularizar el proceso. Ojo porque esto puede ser un arma de doble filo, ya que si nos salimos demasiado del estándar no habrá nadie que entienda nuestro proceso, así que esto tratarlo con cariño.
- Soporta de manera sencilla el proceso de construcción de otros lenguajes que no sean Java, como Groovy, Scala, ...

Por supuesto, y al igual que Maven, tenemos gestión automática de las dependencias de nuestro proyecto, de hecho se conecta a los mismos repositorios de Maven o incluso de Ivi.

En este tutorial vamos a ver como instalarlo.

2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 15' (2.3 GHz Intel i7, 16GB 1600 Mhz DDR3, 500GB Flash Storage).
- NVIDIA GeForce G7 750M
- Sistema Operativo: Mac OS X Lion 10.9
- Java 1.7.0_45
- Maven 3.1.1

3. Prerrequisitos

Antes de ponernos a instalar Gradle tenemos que estar seguros de que tenemos instalada una **Java JDK 1.5 o superior**. Para comprobarlo basta con abrir un Terminal y ejecutar:

```
$ java -version
```

Si no tenemos instalada ninguna JDK o es demasiado antigua, podemos bajarnos la última en la página de Java de Oracle.

Lo bueno es que no hace falta que instalemos Groovy, de hecho, si ya tenemos instalada alguna versión de Groovy no tenemos que preocuparnos de posibles incompatibilidades, ya que Gradle tiene su propia versión, y la que tengamos en el sistema la va a ignorar por completo.

4. Descarga y desempaquetado

Lo primero es ir a la página principal de Gradle **http://www.gradle.org/** y descargar la última versión disponible.

En mi caso la 1.9. Esto nos descarga un fichero con el nombre: **gradle-1.9-all.zip**.

El *all* significa que trae tanto los binarios como el código fuente y documentación. Aunque si quisiéramos, podríamos bajarlo por separado.

Por ahora lo que vamos a hacer es descomprimirlo en el directorio de nuestra elección, por ejemplo en `/opt`, y como al descomprimirlo nos aparece un directorio cuyo nombre tiene el número de versión, es conveniente hacer un enlace simbólico para facilitar las futuras actualizaciones o la gestión de varias versiones instaladas de forma simultánea. Para ello,
y mediante el uso de un Terminal, nos situamos en el directorio donde lo hemos descomprimidos y ejecutamos el comando:
`$ ln -s gradle-1.9 gradle`

También vamos a hacer un enlace simbólico para poder ejecutar Gradle desde cualquier directorio. Para ello ejecutamos en un Terminal el comando: `$ cd /usr/local/bin/ ; sudo ln -s /opt/gradle/bin/gradle`

5. Variables de entorno

Ya tenemos los ficheros en nuestro disco duro, ahora vamos a definir algunas variables de entorno para encontrarlos adecuadamente. Estas variables de entorno las podemos definir, por ejemplo, en nuestro **`$HOME/.bash_profile`**

- **`GRADLE_HOME`** – Tenemos que definir esta nueva variable de entorno que indica donde hemos descomprimido el zip del apartado anterior. En nuestro caso sería:
  `export GRADLE_HOME=/opt/gradle`
- **`GRADLE_OPTS`** – Esta variable no es obligatoria, pero nos puede venir bien para pasarle parámetros a la JVM que va a ejecutar Gradle. Por ejemplo podríamos hacer:
  `export GRADLE_OPTS="-Xmx768m -XX:MaxPermSize=128m"`

Para ver que todo está correcto podemos abrir un Terminal y ejecutar: **`$ gradle --version`**

Deberíamos ver algo como:

```
Shell
1  ------------------------------------------------------------
2  Gradle 1.9
3  ------------------------------------------------------------
4
5  Build time:   2013-11-19 08:20:02 UTC
6  Build number: none
7  Revision:     7970ec3503b4f5767ee1c1c69f8b4186c4763e3d
8
9  Groovy:       1.8.6
10 Ant:          Apache Ant(TM) version 1.9.2 compiled on July 8 2013
11 Ivy:          2.2.0
12 JVM:          1.7.0_45 (Oracle Corporation 24.45-b08)
```

```
13    OS:           Mac OS X 10.9 x86_64
```

6. Construyendo nuestro primer proyecto Java

Aunque en Gradle podemos definir nuestras propias tareas, ya tenemos una serie de plugins que nos facilitan el trabajo.
De esta forma podemos encontrarnos con el plugin de Java que nos define un ciclo de construcción estándar y similar al de Maven.

Para ello vamos a definir el fichero **`build.gradle`**. Este fichero siempre es necesario,
estemos trabajando con Java o no, y en él es donde definiremos nuestro proceso de construcción de proyectos y tareas.

```
Shell
1  apply plugin: 'java'
2
3  version = '1.0-SNAPSHOT'
4
5  sourceCompatibility = 1.7
6
7  repositories {
8      mavenCentral()
9  }
10
11 dependencies {
12     compile group: 'commons-collections', name: 'commons-collections', version: '3.2'
13     testCompile group: 'junit', name: 'junit', version: '4.+'
14     testCompile group: 'org.hamcrest', name: 'hamcrest-library', version: '1.3'
15     testCompile group: 'org.mockito', name: 'mockito-core', version: '1.9.5'
16 }
17
18 jar {
19     manifest {
20         attributes 'Implementation-Title': 'Gradle adictosaltrabajo.com tutorial', 'Implementation-Version': version
21     }
22 }
```

Vemos como activamos el plugin de java (en la línea 1). Luego definimos la versión de nuestro proyecto (3) y con qué versión de Java queremos que sea compatible (5). Indicamos de dónde queremos sacar las dependencias (7-9), en el ejemplo de los repositorios oficiales de Maven. Luego las dependencias (11-16), donde cabe destacar como con un *+* en la versión de jUnit estamos indicando que queremos la última versión de la 4. Y finalmente (18-22) personalizamos un poco el MANIFEST del jar que estamos construyendo.

Si compilamos el proyecto con **`$ gradle build`**, deberíamos ver algo como:

```
Shell
1      :compileJava UP-TO-DATE
2      :processResources UP-TO-DATE
3      :classes UP-TO-DATE
4      :jar
5      :assemble
6      :compileTestJava UP-TO-DATE
7      :processTestResources UP-TO-DATE
8      :testClasses UP-TO-DATE
9      :test UP-TO-DATE
10     :check UP-TO-DATE
11     :build
12
13     BUILD SUCCESSFUL
14
15     Total time: 3.7 secs
```

Donde vemos cómo va pasando por todas las fases, dejando los binarios construidos en el directorio
`build`.

Si queréis saber todas las taréas que podéis invocar basta con hacer **`$ gradle tasks`**,
y veréis algo como:

```
Shell
1      :tasks
2
3      ------------------------------------------------------------
4      All tasks runnable from root project
5      ------------------------------------------------------------
6
7      Build tasks
8      -----------
9      assemble - Assembles the outputs of this project.
10     build - Assembles and tests this project.
11     buildDependents - Assembles and tests this project and all projects that depend on it.
12     buildNeeded - Assembles and tests this project and all projects it depends on.
13     classes - Assembles classes 'main'.
14     clean - Deletes the build directory.
15     jar - Assembles a jar archive containing the main classes.
16     testClasses - Assembles classes 'test'.
```

```
Build Setup tasks
-----------------
init - Initializes a new Gradle build. [incubating]
wrapper - Generates Gradle wrapper files. [incubating]

Documentation tasks
-------------------
javadoc - Generates Javadoc API documentation for the main source code.

Help tasks
----------
dependencies - Displays all dependencies declared in root project 'gradle'.
dependencyInsight - Displays the insight into a specific dependency in root project 'gradle'.
help - Displays a help message
projects - Displays the sub-projects of root project 'gradle'.
properties - Displays the properties of root project 'gradle'.
tasks - Displays the tasks runnable from root project 'gradle'.

Verification tasks
------------------
check - Runs all checks.
test - Runs the unit tests.

Rules
-----
Pattern: build: Assembles the artifacts of a configuration.
Pattern: upload: Assembles and uploads the artifacts belonging to a configuration.
Pattern: clean: Cleans the output files of a task.

To see all tasks and more detail, run with --all.

BUILD SUCCESSFUL

Total time: 2.587 secs
```

7. Conclusiones

Ya sabéis, sólo hemos atisbado la potencia de Gradle y ahora toca explorar la documentación para sacarle el máximo partido: proyectos multi-módulo, aplicaciones web, generación de documentación, repositorios privados, ejecución como demonio para mejorar tiempos de construcción ...

Además conviene insistir en lo importante que es que os intentéis ceñir siempre al estándar. De lo contrario acabaremos reinventando la rueda y la curva de aprendizaje de cualquiera que se incorpore a vuestro proyecto será muy alta. Para ello intentar siempre usar el ciclo de vida y los plugins estándar, y sólo hacer vuestras propias tareas cuando no os quede más remedio.

También os dejo aquí el mini proyecto que hemos usado para el tutorial.

8. Sobre el autor

Alejandro Pérez García, Ingeniero en Informática (especialidad de Ingeniería del Software) y Certified ScrumMaster

Socio fundador de Autentia (Desarrollo de software, Consultoría, Formación)

mailto:alejandropg@autentia.com

Autentia Real Business Solutions S.L. – "Soporte a Desarrollo"

http://www.autentia.com

Alejandro Pérez García

Alejandro es socio fundador de Autentia y nuestro experto en Java EE, Linux y optimización de aplicaciones empresariales.

Ingeniero en Informática y Certified ScrumMaster.

Seguir @alejandropgarci

Si te gusta lo que ves, puedes contratarle para darte ayuda con soporte experto, impartir cursos presenciales en tu empresa o para que realicemos tus proyectos como factoría (Madrid).

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación.