GRADLE

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Gradle is an open source, advanced general purpose build management system. It is built on ANT, Maven, and lvy repositories. It supports Groovy based Domain Specific Language (DSL) over XML.

This tutorial explains how you can use Gradle as a build automation tool for Java as well as Groovy projects.

Audience

This tutorial is designed for those software professionals who would like to learn the basics of Gradle as a build tool in simple and easy steps.

Prerequisites

Gradle is a Groovy-based build automation tool. So, it will certainly help if you have some prior exposure to Groovy. In addition, you should have working knowledge of Java.

Disclaimer & Copyright

Table of Contents

1. GRADLE –OVERVIEW

ANT and Maven shared considerable success in the Java marketplace. ANT was the first build tool released in 2000 and it is developed based on procedural programming idea. Later, it was improved with an ability to accept plug-ins and dependency management over the network with the help on Apache-Ivy. The main drawback was XML as a format to write build scripts. XML being hierarchical is not good for procedural programming and tends to become unmanageably big.

Maven was introduced in 2004. It comes with a lot of improvement than ANT. It changes its structure and it continues using XML for writing build specifications. Maven relies on the conventions and is able to download the dependencies over the network. The main benefit of Maven is its life cycle. While following the same life cycle for multiple projects continuously, this comes a cost of flexibility. Maven also faces some problems in dependency management. It does not handle well conflicts between versions of the same library, and complex customized build scripts are actually harder to write in Maven than in ANT.

Finally, Gradle came into the picture in 2012. Gradle carries some efficient features from both the tools.

Features of Gradle

Following is the list of features that Gradle provides.

- **Declarative builds and build-by-convention:** Gradle is available with separate Domain Specific Language (DSL) based on Groovy language. Gradle provides declarative language elements. The elements also provide build-by-convention support for Java, Groovy, OSGi, Web and Scala.

- **Language for dependency based programming:** The declarative language lies on top of a general purpose task graph, which you can fully leverage in your build.

- **Structure your build:** Gradle allows you to apply common design principles to your build. It gives you a perfect structure for build, so that you can design well-structured and easily maintained, comprehensible build.

- **Deep API:** Using this API, you can monitor and customize its configuration and execution behavior to its core.

- **Gradle scales:** Gradle can easily increase productivity, from simple and single project builds to huge enterprise multi-project builds.

- **Multi-project builds:** Gradle supports multi-project builds and also partial builds. If you build a subproject, Gradle takes care of building all the subprojects that it depends on.

- **Different ways to manage your builds:** Gradle supports different strategies to manage your dependencies.

- **First build integration tool:** Gradle completely supports ANT tasks, Maven and lvy repository infrastructure for publishing and retrieving dependencies. It also provides a converter for turning a Maven pom.xml to Gradle script.

- **Ease of migration:** Gradle can easily adapt to any structure you have. Therefore, you can always develop your Gradle build in the same branch where you can build live script.

- **Gradle Wrapper:** Gradle Wrapper allows you to execute Gradle builds on machines where Gradle is not installed. This is useful for continuous integration of servers.

- **Free open source:** Gradle is an open source project, and licensed under the Apache Software License (ASL).

- **Groovy:** Gradle's build script is written in Groovy. The whole design of Gradle is oriented towards being used as a language, not as a rigid framework. Groovy allows you to write your own script with some abstractions. The entire Gradle API is designed in Groovy language.

Why Groovy?

The complete Gradle API is designed using Groovy language. This is an advantage of an internal DSL over XML. Gradle is general purpose build tool at its core; its main focus is Java projects. In such projects, the team members will be very familiar with Java and it is better that a build should be as transparent as possible to all team members.

Languages like Python, Groovy or Ruby are better for build framework. Why Groovy was chosen is, because it offers by far the greatest transparency for people using Java. The base syntax of Groovy is same as Java. Groovy provides much more on top of that.

2. GRADLE – INSTALLATION

Gradle is a build tool, based on Java. There are some prerequisites that needs to be installed before installing the Gradle framework.

Prerequisites

JDK and Groovy are the prerequisites for Gradle installation.

- Gradle requires JDK version 6 or later to be installed in your system. It uses the JDK libraries which is installed and sets to the JAVA_HOME environmental variable.

- Gradle carries its own Groovy library, therefore, we do no need to install Groovy explicitly. If it is installed, that is ignored by Gradle.

Following are the steps to install Gradle in your system.

Step 1: Verify JAVA Installation

First of all, you need to have Java Software Development Kit (SDK) installed on your system. To verify this, execute **Java –version** command in any of the platform you are working on.

In Windows

Execute the following command to verify Java installation. I have installed JDK 1.8 in my system.

```
C:\> java - version
```

If the command is executed successfully, you will get the following output.

```
java version "1.8.0_66"
Java(TM) SE Runtime Environment (build 1.8.0_66-b18)
Java HotSpot(TM) 64-Bit Server VM (build 25.66-b18, mixed mode)
```

In Linux

Execute the following command to verify Java installation. I have installed JDK 1.8 in my system.

```
$ java - version
```

If the command is executed successfully, you will get the following output.

```
java version "1.8.0_66"
Java(TM) SE Runtime Environment (build 1.8.0_66-b18)
Java HotSpot(TM) 64-Bit Server VM (build 25.66-b18, mixed mode)
```

We assume the readers of this tutorial have Java SDK version 1.8.0_66 installed on their system.

Step 2: Download Gradle Build File

Download the latest version of Gradle from the Download Gradle link. In the reference page, click on the **Complete Distribution** link. This step is common for any platform. For this you will get the complete distribution file into your Downloads folder.

Step 3: Set Up Environment for Gradle

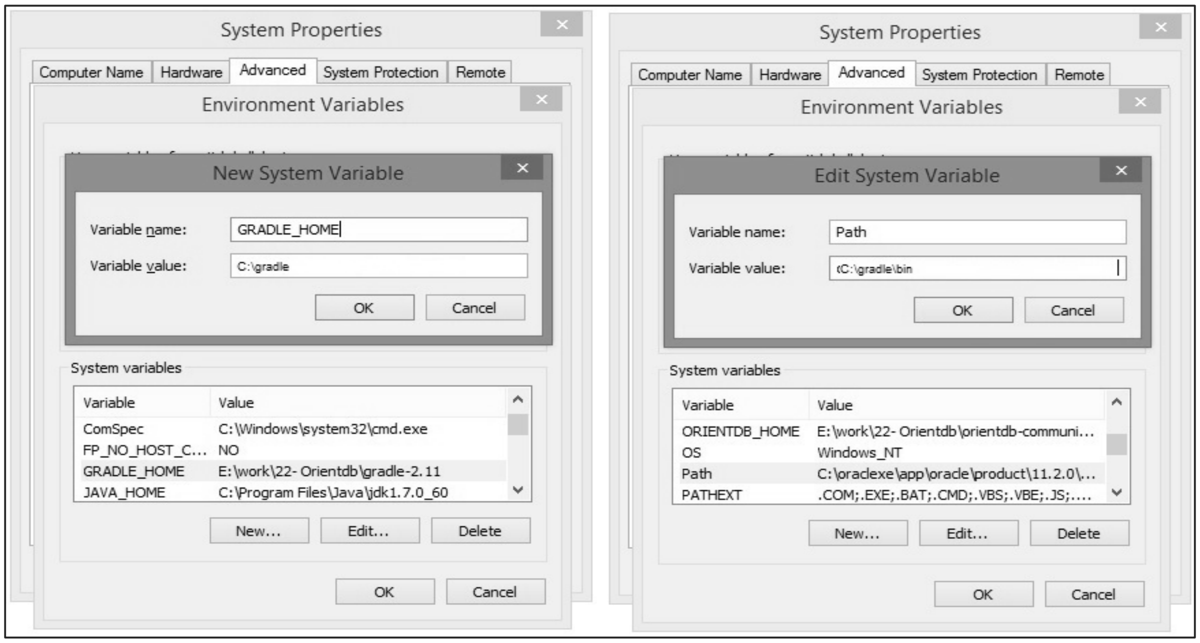
Setting up the environment means we have to extract the distribution file and copy the library files into proper location. Setting up **GRADLE_HOME** and **PATH** environmental variables.

This step is platform dependent.

In Windows

Extract the downloaded zip file named **gradle-2.11-all.zip** and copy the distribution files from **Downloads\gradle-2.11\** to **C:\gradle\** location.

Later, add the **C:\gradle** and **C:\gradle\bin** directories to the **GRADLE_HOME** and **PATH** system variables. Right-click on My Computer -> Click properties -> Advanced system settings -> Environment variables. There you will find a dialog box for creating and editing system variables. Click 'New' button for creating GRADLE_HOME variable (follow the left side screenshot). Click 'Edit' for editing the existing Path system variable (follow the right side screenshot). The process is shown in the following screenshots.



New System Variable

Variable name: GRADLE_HOME

Variable value: C:\gradle

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
GRADLE_HOME	E:\work\22- Orientdb\gradle-2.11
JAVA_HOME	C:\Program Files\Java\jdk1.7.0_60

Edit System Variable

Variable name: Path

Variable value: ;C:\gradle\bin

Variable	Value
ORIENTDB_HOME	E:\work\22- Orientdb\orientdb-communi...
OS	Windows_NT
Path	C:\oraclexe\app\oracle\product\11.2.0\...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....

In Linux

Extract the downloaded zip file named **gradle-2.11-all.zip** then you will find an extracted file named **gradle-2.11**.

You can use the following to move the distribution files from **Downloads/gradle-2.11/** to **/opt/gradle/** location. Execute this operation from the Downloads directory.

```
$ sudo mv gradle-2.11 /opt/gradle
```

Edit the ~/.bashrc file and paste the following content to it and save it.

```
export ORIENT_HOME = /opt/gradle

export PATH = $PATH:
```

Execute the following command to execute **~/.bashrc** file.

```
$ source ~/.bashrc
```

Step 4: Verify the Gradle Installation

In Windows

You can execute the following command in command prompt.

```
C:\> gradle –v
```

Output: You will find the Gradle version.

```
------------------------------------------------------------
Gradle 2.11
------------------------------------------------------------
Build time:   2016-02-08 07:59:16 UTC
Build number: none
Revision:     584db1c7c90bdd1de1d1c4c51271c665bfcba978

Groovy:       2.4.4
Ant:          Apache Ant(TM) version 1.9.3 compiled on December 23 2013
JVM:          1.7.0_60 (Oracle Corporation 24.60-b09)
OS:           Windows 8.1 6.3 amd64
```

In Linux

You can execute the following command in terminal.

```
$ gradle –v
```

Output: You will find the Gradle version.

```
------------------------------------------------------------
Gradle 2.11
------------------------------------------------------------


Build time:   2016-02-08 07:59:16 UTC
Build number: none
Revision:     584db1c7c90bdd1de1d1c4c51271c665bfcba978

Groovy:       2.4.4
Ant:          Apache Ant(TM) version 1.9.3 compiled on December 23 2013
JVM:          1.7.0_60 (Oracle Corporation 24.60-b09)
OS:           Linux 3.13.0-74-generic amd64
```

3. GRADLE – BUILD SCRIPT

Gradle builds a script file for handling two things; one is **projects** and another one is **tasks**. Every Gradle build represents one or more projects. A project represents a library JAR or a web application or it might represent a ZIP that assembled from the JARs produced by other projects. In simple words, a project is made up of different tasks. A task means a piece of work which a build performs. A task might be compiling some classes, creating a JAR, generating Javadoc, or publishing some archives to a repository.

Gradle uses **Groovy language** for writing scripts.

Writing Build Script

Gradle provides a Domain Specific Language (DSL), for describing builds. This uses the Groovy language to make it easier to describe a build. Each build script of Gradle is encoded using UTF-8, saved offline and named as build.gradle.

build.gradle

We are describing about tasks and projects by using a Groovy script. You can run a Gradle build using the Gradle command. This command looks for a file called **build.gradle**. Take a look at the following example which represents a small script that prints **tutorialspoint**. Copy and save the following script into a file named **build.gradle**. This build script defines a task name hello, which is used to print tutorialspoint string.

```
task hello {

    doLast {

        println 'tutorialspoint'

    }

}
```

Execute the following command in the command prompt. It executes the above script. You should execute this, where the build.gradle file is stored.

```
C:\> gradle –q hello
```

If the command is executed successfully, you will get the following output.

```
tutorialspoint
```

If you think task works similar to ANT's target, then that's right - Gradle task is equivalent to ANT target.

You can simplify this hello task by specifying a shortcut (represents a symbol **<<**) to the **doLast** statement. If you add this shortcut to the above task **hello** it will look like the following script.

```
task hello << {
  println 'tutorialspoint'
}
```

You can execute the above script using **gradle –q hello** command.

The Gradle script mainly uses two real Objects; one is Project Object and another one is Script Object.

Project Object: Each script describes about one or multiple projects. While in the execution, this script configures the Project Object. You can call some methods and use property in your build script which are delegated to the Project Object.

Script Object: Gradle takes script code into classes, which implements Script Interface and then executes. This means that of all the properties and methods declared by the script interface are available in your script.

The following table defines the list of **standard project properties**. All these properties are available in your build script.

Sr. No.	Name	Type	Default Value
1	project	Project	The Project instance
2	name	String	The name of the project directory.
3	path	String	The absolute path of the project.
4	description	String	A description for the project.
5	projectDir	File	The directory containing the build script.
6	buildDir	File	projectDir/build
7	group	Object	Unspecified
8	version	Object	Unspecified
9	ant	AntBuilder	An AntBuilder instance

How Gradle Uses Groovy

Gradle build scripts use the full length Groovy API. As a startup, take a look at the following examples. The following example explains about converting a string to upper case.

Copy and save the following code into **build.gradle** file.

```
task upper << {
    String expString = 'TUTORIALS point'
    println "Original: " + expString
    println "Upper case: " + expString.toUpperCase()}
```

Execute the following command in the command prompt. It executes the above given script. You should execute this, where the build.gradle file is stored.

```
C:\> gradle –q upper
```

If the command is executed successfully, you will get the following output.

```
Original: TUTORIALS point
Upper case: TUTORIALS POINT
```

The following example explains about printing the value of an implicit parameter ($it) for 4 times.

Copy and save the following code into **build.gradle** file.

```
task count << {
    4.times { print "$it " }
}
```

Execute the following command in the command prompt. It executes the above given script. You should execute this, where the build.gradle file is stored.

```
$ gradle –q count
```

If the command is executed successfully, you will get the following output.

```
0 1 2 3
```

Groovy language provides plenty of features. Following are some important features.

Groovy JDK Methods

Groovy adds lots of useful methods to the standard Java classes. For example, Iterable API from JDK implements an **each()** method which iterates over the elements of the Iterable Interface.

Copy and save the following code into **build.gradle** file.

```
task groovyJDK << {
String myName = "Marc";
myName.each() { println "${it}" };
}
```

Execute the following command in the command prompt. It executes the above given script. You should execute this, where the build.gradle file is stored.

```
C:\> gradle –q groovyJDK
```

If the command is executed successfully, you will get the following output.

```
M
a
r
c
```

Property Accessors

You can automatically accesses appropriate getter and setter methods of a particular property by specifying its reference.

The following snippet defines the syntaxes of getter and setter methods of a property **buildDir**.

```
// Using a getter method
println project.buildDir
println getProject().getBuildDir()

// Using a setter method
project.buildDir = 'target'
getProject().setBuildDir('target')
```

Optional Parentheses on Method Calls

Groovy contains a special feature in methods calling that is the parentheses are optional for method calling. This feature applies to Gradle scripting as well.

Take a look at the following syntax. That defines a method calling **systemProperty** of **test** object.

```
test.systemProperty 'some.prop', 'value'
test.systemProperty('some.prop', 'value')
```

Closure as the Last Parameter of the Method

Gradle DSL uses closures in many places. Where the last parameter of a method is a closure, you can place the closure after the method call.

The following snippet defines the syntaxes Closures use as repositories() method parameters.

```
repositories {
    println "in a closure"
}
repositories() { println "in a closure" }
repositories({ println "in a closure" })
```