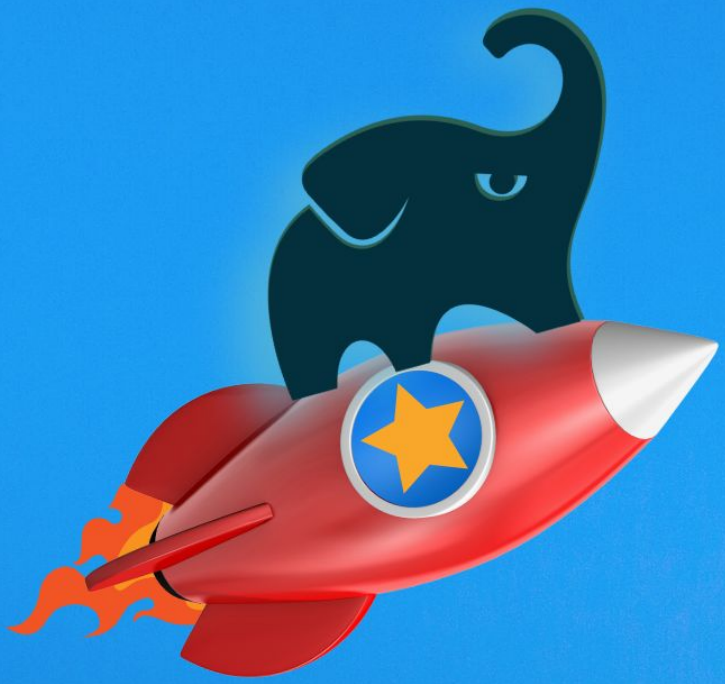
Get Going with Gradle

Getting started with Gradle just got much easier



Course Notes

Get Going with Gradle Course Notes

Contents

Below are the notes to accompany the beginner Gradle course, *Get Going with Gradle*.

1. Installing Gradle

1.1. Windows Gradle Installation

- open the Windows command prompt (open the start menu, type *cmd*, then hit enter)
- validate that Java is installed

  ```
  java -version
  ```

  it should print out details of your current Java installation. Don't worry if you've got a different Java version installed because Gradle supports versions 8 and above.

- go to https://gradle.org/releases
- scroll down and choose the most recent Gradle release. At the time of recording this was 6.8.
- choose the *binary-only* option. Click the link to download the Gradle zip file to your computer.
- create a Gradle directory on your hard drive, and extract the zip file into that directory. To do this open file explorer (open the start menu, type *file*, then hit enter).
- navigate to your hard drive root directory then right click, go to *New* > *Folder*, then enter the name *Gradle*.
- navigate to where you downloaded the Gradle zip file, copy the zip file, then paste it in the new Gradle directory
- right click the zip file and select *Extract all* to extract the zip file using Windows. Or use 7zip which is faster than Windows.
- you should now have an additional directory. Go into that directory, go into the *bin* directory, and copy the path from the address bar at the top of the File explorer and keep it safe for later on
- now we need to configure your *PATH* variable so that you can run Gradle commands from wherever you are in the command prompt
- go to the start menu, type *environment*, then hit enter when *Edit the system environment*

variables appears.

- on the dialog that appears, click *Environment Variables*, then under *System variables* double click the *Path* variable
- click *New*, then paste in the location of the Gradle *bin* directory which you copied earlier
- hit enter, select *OK*, then *OK* again, then *OK* again
- close the Windows command prompt, and open a new one (Start menu, type *cmd*, hit enter)
- now we're going to validate our Gradle installation:

  ```
  gradle --version
  ```

  you should see some output showing that a specific version of Gradle is installed

If you're a Windows user, well done, that's all you need to do before moving onto the next lesson.

1.2. Linux Gradle Installation

- validate that Java is installed:

  ```
  java -version
  ```

  it should print out details of your current Java installation. Don't worry if you've got a different Java version installed because Gradle supports versions 8 and above.

- download the latest version of Gradle using the curl command:

  ```
  curl https://downloads.gradle-dn.com/distributions/gradle-6.8-bin.zip --output ~/gradle.zip
  ```

- unzip the file using the unzip command:

  ```
  sudo unzip -d /opt/gradle gradle.zip
  ```

  if you're prompted for your password, enter it because we're running this command as the root user

- look at the contents of the Gradle installation using the *ls* command:

  ```
  ls /opt/gradle/gradle-6.8
  ```

  you should see some files and directories

- setup the *PATH* environment variable:

  ```
  echo 'export PATH="$PATH:/opt/gradle/gradle-6.8/bin"' >> ~/.bashrc
  ```

- close the terminal and open a new one
- show that Gradle has been successfully installed.

  ```
  gradle --version
  ```

- you should see some output showing that a specific version of Gradle is installed

If you're a Linux user, well done, that's all you need to do before moving onto the next lesson.

1.3. Other installation options

Mac

```
brew install gradle
```

SDKMAN!

```
sdk install gradle 6.8
```

2. Creating your first Gradle project (practical)

2.1. Setting up your Gradle project

Open up a terminal and navigate to your home directory.

Create a directory for the project:

```
mkdir get-going-with-gradle
```

Navigate into the directory:

```
cd get-going-with-gradle
```

Create a Gradle project:

```
gradle init
```

Select type of project to generate:

```
1
```

Select build script DSL:

```
1
```

Project name:

```
<enter>
```

2.2. Interact with the new project

Try interacting with the project:

`./gradlew help` (Linux/Mac)

`gradlew.bat help` (Windows)

List available tasks:

`./gradlew tasks` (Linux/Mac)

`gradlew.bat tasks` (Windows)

Look at the different files in the project:

`ls -l` (Linux/Mac)

`dir` (Windows)

2.3. Explore the project files

Look inside *build.gradle*:

`cat build.gradle` (Linux/Mac)

`type build.gradle` (Windows)

Look inside *settings.gradle*:

`cat settings.gradle` (Linux/Mac)

`type settings.gradle` (Windows)

Look at any hidden files or directories:

`ls -la` (Linux/Mac)

`dir/a` (Windows)

Look inside the *.gitignore*:

`cat .gitignore` (Linux/Mac)

`type .gitignore` (Windows)

2.4. Commit the project into version control

Initialise this directory as a Git repository:

```
git init
```

Get all the files ready to be committed:

```
git add .
```

See what files are staged for commit:

```
git status
```

Commit everything:

```
git commit -m "Initialise project"
```

Good work! You've just created your first Gradle project and committed it into version control.

3. Building a Java project with Gradle (practical)

Open the project you built in the *Creating your first Gradle project* lesson. Use a command-line, text editor, or an IDE as you prefer.

3.1. Adding the Java code

Add a directory *src/main/java* for the main application code.

Create directories for the Java package *com/tomgregory/languageapp*.

Create a new file *SayHello.java* and open it for editing.

Paste in the following Java code:

```
package com.tomgregory.languageapp;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;

public class SayHello {
    public static void main(String[] args) throws IOException {
        String language = args[0];

        InputStream resourceStream = SayHello.class.getClassLoader
().getResourceAsStream(language + ".txt");
        assert resourceStream != null;
        BufferedReader bufferedInputStream = new BufferedReader(new InputStreamReader
(resourceStream, StandardCharsets.UTF_8));

        System.out.println(bufferedInputStream.readLine());
    }
}
```

3.2. Adding the resources

Create a new directory under *src/main* called *resources*.

Add a file *en.txt* with contents *Hello!*.

Add a file *es.txt* with contents *Hola!*.

3.3. Building the application

Open *build.gradle* and delete the comment.

Insert this plugins configuration:

```
plugins {
    id 'java'
}
```

See what Gradle tasks we have available:

`./gradlew tasks` (Linux/Mac)

`gradlew.bat tasks` (Windows)

All the tasks listed under *Build tasks* have been added to the project by the Java plugin.

To see all Gradle tasks, including *compileJava* and *processResources* run:

`./gradlew tasks --all` (Linux/Mac)

`gradlew.bat tasks --all` (Windows)

These tasks appear under *Other tasks*.

Compile the Java classes:

`./gradlew compileJava` (Linux/Mac)

`gradlew.bat compileJava` (Windows)

Look for output in *build/classes/java/main*.

Process resources:

`./gradlew processResources` (Linux/Mac)

`gradlew.bat processResources` (Windows)

Look for output in *build/resources/main*.

Generate a jar file:

`./gradlew jar` (Linux/Mac)

`gradlew.bat jar` (Windows)

Look for the jar file in *build/libs*.

3.4. Running the application

Run the following Java command:

`java -jar build/libs/get-going-with-gradle.jar en` (Linux/Mac/Windows)

You'll see an error which says *no main manifest attribute*.

Add this jar configuration to the end of *build.gradle*:

```
jar {
    manifest {
        attributes('Main-Class': 'com.tomgregory.languageapp.SayHello')
    }
}
```

Build the jar file again:

`./gradlew jar` (Linux/Mac)

`gradlew.bat jar` (Windows)

Rerun the Java command:

`java -jar build/libs/get-going-with-gradle.jar en` (Linux/Mac/Windows)

You should see the text *Hello!* output.

Run the same Java command but replace *en* with *es* for Spanish

`java -jar build/libs/get-going-with-gradle.jar es` (Linux/Mac/Windows)

You should see the text *Hola!* output.

Awesome! You've just built your first Java application with Gradle.

3.5. Testing the application

Tests live in *src/test/java*, so under *src* create a new directory *test/java*.

Within this directory create the same package structure. Create directories *com/tomgregory/languageapp*.

Add a new file *SayHelloTest.java* and open it for editing.

Paste in the following Java code:

```
package com.tomgregory.languageapp;

import org.junit.jupiter.api.Test;
```

```
import java.io.IOException;

public class SayHelloTest {
    @Test
    public void testSayHello() throws IOException {
        SayHello.main(new String[]{"en"});
    }
}
```

Try running the tests:

`./gradlew test` (Linux/Mac)

`gradlew.bat test` (Windows)

You'll get an error here saying that it can't find the *org.junit.jupiter.api* package.

Add this dependency configuration block to *build.gradle*, after the plugins:

```
dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter:5.6.3'
}
```

Add this repositories configuration block, putting it just before the dependencies:

```
repositories {
    mavenCentral()
}
```

Rerun the test command:

`./gradlew test` (Linux/Mac)

`gradlew.bat test` (Windows)

The build should be successful.

Open the test report in a browser, located in *build/reports/tests/test/index.html*.

You'll see 0 tests have been run.

Add this test configuration block below the dependencies block:

```
test {
    useJUnitPlatform()
}
```

This configures tests to use the Junit 5 platform.

Run the tests again:

`./gradlew test` (Linux/Mac)

`gradlew.bat test` (Windows)

Open the test report again in a browser, located in *build/reports/tests/test/index.html*.

You should now see that 1 test was run, with a 100% success rate.

3.6. Commit your changes

Get all the files ready to be committed:

```
git add .
```

See what files are staged for commit:

```
git status
```

Commit everything:

```
git commit -m "Create Java project for language app."
```

Good work! You've just built, run, and tested your first Java project with Gradle!

4. Quiz

Take the quiz to deepen your new Gradle knowledge.

Questions have a single answer, unless specified. Answers and explanations provided afterwards.

4.1. Questions

Question 1

Which format CAN'T you use to define a Gradle build script?

1. Groovy
2. Kotlin
3. XML

Question 2

In which scenario should you use a local Gradle installation?

1. To run a Gradle task

2. To initialise a Gradle project
3. To clean your Gradle project

Question 3

You're working on a Windows machine, and have been asked to make some changes to a Java project built with Gradle. You've checked out the project, and now wish to build it and run tests within a Windows command prompt.

Which command should you run, choosing the option which will most likely result in a successful build?

1. `gradlew.bat build`
2. `./gradlew build`
3. `gradle build`

Question 4

Which repositories does Gradle support for downloading dependencies? (select multiple)

1. Maven central
2. JCenter
3. Google
4. A custom Maven repository

Question 5

Where is the Gradle project name configured?

1. *build.gradle*
2. *gradlew*
3. *gradlew.bat*
4. *settings.gradle*

Question 6

You're helping out a colleague who's created a Gradle project from scratch, without using the setup wizard. He's not sure which directories should go into version control and which shouldn't.

Which of the following directories and files SHOULD NOT go into version control? (select multiple)

1. *.gradle* (directory)
2. *build* (directory)
3. *build.gradle*
4. *gradle* (directory)

5. *gradlew*
6. *gradlew.bat*
7. *settings.gradle*

Question 7

You've been asked to make some changes to a very old Java project, which is built using Gradle. You checkout the project, and run the following command:

```
gradle build
```

You see a lot of errors. You're confused because your colleague shows the same project building on his machine. You've got the latest version of Gradle installed.

What's the most likely cause of this issue?

1. Your local Gradle version is incompatible with the version the project expects
2. Your Gradle installation is broken
3. The project's build is broken

Question 8

You're creating a Gradle build for an existing Java project. You've noticed that when you run `./gradlew assemble`, it says *BUILD SUCCESSFUL* but no compiled classes get generated in the *build* directory. You're confused because you have lots of Java classes located in *src/java*.

Which of the following options would fix this problem?

1. Apply the Java plugin in *build.gradle*
2. Run `./gradlew build` instead of `./gradlew assemble`
3. Move the Java classes into *src/main/java*
4. Run `gradle init` to reset the project

Question 9

You're working on a small Java application which relies on the *commons-lang3* Java library. You've configured the dependency like this in your *build.gradle* file:

```
dependencies {
    testImplementation 'org.apache.commons:commons-lang3:3.11'
}
```

You want to generate a jar file for the application, but when you run `./gradlew assemble` you're seeing the following error:

```
error: package org.apache.commons.lang3 does not exist
```

What should you do to resolve this problem?

1. Run `./gradlew test` instead.
2. Update the dependency to look like this:

   ```
   implementation 'org.apache.commons:commons-lang3:3.11'
   ```

3. Use a different library, since *commons-lang3* is obviously not fit for purpose.

Question 10

You're adding some tests to a project built using Gradle, and see the following error when you run `./gradlew test`:

```
error: package org.junit.jupiter.api does not exist
```

How should you fix this?

1. Add a *junit-jupiter* dependency to *settings.gradle*
2. Add a *junit-jupiter* dependency to *build.gradle*
3. Configure the Java plugin in *build.gradle*
4. Run the test again. It was probably a one-off.

Question 11

In a Gradle project there are two tasks, *taskA* and *taskB*.

If *taskA* depends on *taskB*, what would be the behaviour when running the following command?

```
./gradlew taskA
```

1. *taskB* will run before *taskA*
2. *taskA* will run before *taskB*
3. *taskA* and *taskB* will run simultaneously

Question 12

You're working on a large code project for a media website, built using Gradle.

You make some code changes and now want to generate the jar file. You're in a hurry and don't want to run all the tests, which are known to be slow.

Which Gradle task should you run to generate the jar file in the shortest time possible?

1. *build*

2. *check* or *test*
3. *assemble* or *jar*
4. *classes*

4.2. Answers

Question 1

Answer: 3

Gradle doesn't support XML format for its builds, but does support Groovy and Kotlin. In fact, not using XML is one advantage of Gradle over other build tools such as Maven.

Question 2

Answer: 2

A local Gradle installation should only be used for initialising a fresh Gradle project.

Question 3

Answer: 1

When running Gradle tasks you should always use the provided Gradle wrapper script. On Windows, this script is *gradlew.bat*, so *gradlew.bat* build is the correct answer.

- *./gradlew build* is incorrect as this is specific to Linux or Mac environments
- *gradle build* might work if you have a local Gradle installation, but running *gradlew.bat* is always the better option to ensure you're using the correct version of Gradle for the project. Hence, this answer is incorrect.

Question 4

Answers: 1, 2, 3 & 4

That's right, Gradle supports all these types of repositories for downloading dependencies.

Question 5

Answer: 4

You can configure the project name in *settings.gradle* like this:

```
rootProject.name = 'get-going-with-gradle'
```

Question 6

Answers: 1 & 2

Everything listed except the *.gradle* directory and the *build* directory should be committed into version control.

- the *.gradle* directory is a local cache used by Gradle during your build
- the *build* directory is reserved for any locally built artifacts. It changes based on what Gradle tasks you run, so it doesn't make sense to commit it into version control.

Question 7

Answer: 1

A Gradle project should come bundled up with the Gradle wrapper. You should always use the wrapper script to run Gradle tasks against the project.

In this case it would have been better to run:

```
./gradlew build
```

This would ensure that the version of Gradle you're using to build the project is the same one specified by the project itself.

The problem you're seeing is most likely caused because your new version of Gradle doesn't support the old version of Gradle used by the project.;

Question 8

Answer: 3

The Java plugin expects Java classes to exist in the *src/main/java* directory. Since the code is currently located in *src/java*, this explains why no classes are being compiled. Moving the classes to *src/main/java* is the correct fix.

- applying the Java plugin is incorrect since the fact that running `./gradlew assemble` was successful implies it's already been applied
- running `./gradlew build` is incorrect since that will just run the *test* task as well, which won't fix the issue
- running `gradle init` is incorrect as that won't fix the incorrect location of the Java classes. Besides, once you've already setup a Gradle project `gradle init` will have no effect.

Question 9

Answer: 2

If your application relies on the *commons-lang3* library, then it must be present on the Java classpath during compilation. Currently the dependency is declared on the *testImplementation* dependency configuration, which means it can only be used within test classes.

Moving the dependency to the *implementation* configuration is the correct solution, as it means the library will be on the Java classpath when compiling both main and test classes.

- running the *test* task is incorrect as this won't help with the compilation issue within the main application classes
- using a different library is incorrect, since the failure is related to a misconfigured Gradle build and not the library itself

Question 10

Answer: 2

This error implies a failure during test compilation, caused by a missing dependency for *junit-jupiter*.

Dependencies are configured in *build.gradle*, so you'd want to add a dependency like this:

```
dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter:5.6.3'
}
```

Question 11

Answer: 1

If *taskA* depends on *taskB*, then *taskA* cannot be run until *taskB* has run. Therefore, *taskB* runs first, followed by *taskA*.

Question 12

Answer: 3

The correct answer is *assemble* or *jar*, as these tasks are the most specific task for what you want to achieve. This is the quickest way to get the jar file built.

- *build* is incorrect because although it would build the jar file, it would also run the tests which would slow you down
- *check* or *test* is incorrect because this would also run the tests which would slow you down. Also, it wouldn't actually build the jar file
- *classes* is incorrect because this task compiles your main code and processes your main resources, but doesn't build the jar file

Thank you for checking out *Get Going with Gradle*!

To really master Gradle, why not sign up to the awesome Gradle Hero course?

Or head to tomgregory.com to explore other topics.

> Hopefully we'll meet again very soon.
>
> *Tom Gregory*