Getting Started Using an Existing Gradle Build

build passing (https://travis-ci.org/gradle-guides/using-an-existing-gradle-build)

Table of Contents

You are a *build user* in our terminology. You want to be able to run the tool either from the command line or an IDE (or both). Follow these steps and you'll have Gradle up and running in no time. You'll also learn how to explore a Gradle build and find out what tasks you can execute and what they do.

What you'll need

- About 5 minutes
- A text editor or IDE
- A Java Development Kit (JDK), version 1.7 or better
- (Possibly) A Gradle distribution (https://gradle.org/install), version 3.3 or better

Step 1: Install Java

Gradle is a Java-based tool, which means you need a Java Virtual Machine (JVM) installed on your computer to use it. Gradle is happy with either a Java Runtime Environment (JRE) or a full Java Development Kit (JDK). You may already have a JRE installed if your browser has loaded a Java Applet in the past. You'll soon discover whether you need to install a JVM yourself when you attempt to run Gradle.

Gradle will work with version 7 and above of Java (1.7+ in JDK versioning). If you don't yet have either a JRE or JDK installed, we recommend you download JDK 8 (http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) and then follow Oracle's instructions on installing it (https://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html).

Step 2: Does the project use the Gradle wrapper?

Look in the root of the project for a `gradlew` file. If it exists, you can use the Gradle wrapper to execute the build without installing Gradle. The user guide has a short section describing how to use it (https://docs.gradle.org/current/userguide/gradle_wrapper.html#using_wrapper_scripts). In essence, it automatically downloads and uses the appropriate version of Gradle for that particular build.

If that file doesn't exist, you'll need to install Gradle and there are various approaches (https://gradle.org/install).

If you do come across a build without a wrapper, politely ask the owner to generate it or (if you have permission to commit the changes) generate it yourself (https://docs.gradle.org/current/userguide/gradle_wrapper.html#sec:wrapper_generation). We strongly recommend using the wrapper for Gradle builds to ensure the appropriate version of Gradle is always used to build your project, which iis important for reproducibility. It also means people don't need to install Gradle just to execute your build.

Step 3: Executing the build

There are two main ways to interact with Gradle: through the command line or via an IDE. Fortunately, they're not mutually exclusive, so you can take a mix and match approach. Ideally, you should configure your IDE to use the Gradle wrapper whenever it's available.

Another important aspect of executing Gradle builds is what we call the *Gradle daemon*. This is designed to improve the startup and execution speed of builds significantly. It's off by default, but we recommend enabling it globally on developer machines via the techniques described in the user guide (https://docs.gradle.org/current/userguide/gradle_daemon.html).

Using the command line

The user guide explains in depth (https://docs.gradle.org/current/userguide/tutorial_gradle_command_line.html) how to use the `gradle` command (or the wrapper, `gradlew`) to list tasks, execute them, and query for other build information, such as which projects are managed by the build.

Using an IDE

Many IDEs (certainly the ones in the Java ecosystem) allow you to import projects via their Gradle build files. This is the preferred approach as it means you can do everything you need to from the IDE, such as refresh the dependencies and view the available tasks.

You can also use Gradle plugins to customize the IDE integration. The user guide explains how the Eclipse (https://docs.gradle.org/current/userguide/eclipse_plugin.html) and IDEA (https://docs.gradle.org/current/userguide/idea_plugin.html) plugins can be configured to provide extra IDE customization such as custom run configurations and default VCS integration.

Step 4: Identify the type of project

Gradle can be used to build anything, such as Java projects, C++ projects, documentation and more. Even so, most Gradle builds are one of the core types: *Java*, *native*, or *Android*. Each of these has a set of conventions and concepts specific to that type, such as where to put your source files, what tasks you can execute and what binaries the build produces. It's important to understand these build-specific features so that you can easily find your way around a project and its build.

So what type of project are you working with? To find out, you just need to ask yourself a few questions:

- Run the command gradle projects. Does it list more than the root project? If so, start by learning about multi-project builds (https://docs.gradle.org/current/userguide/intro_multi_project_builds.html). Note that IDEs will automatically import subprojects as modules (IDEA) or projects (Buildship).
- Look in each project or subproject build file (`*.gradle`) for a line of the form:

GROOVY

```
apply plugin: '[lang]'
```

or a similar entry in the plugins block:

GROOVY

```
plugins {
   id: '[lang]'
}
```

where `[lang]` is one of the strings mentioned in the next entry.

There are three language groupings, which correspond to Java (or more correctly the JVM), native, and Android projects. Find out more about the grouping your project or projects fall into based on :

- `java` , `groovy` , `scala` : you have a JVM project (https://docs.gradle.org/current/userguide/tutorial_java_projects.html). The user guide also has more details about each of the language plugins if you want a bit more depth.
- `c` , `cpp` , `assembler` , `objective-c` , `objective-cpp` : you have a native build (https://docs.gradle.org/current/userguide/build_init_plugin.html#sec:build_init_types).
- `android` : you have an Android project (http://tools.android.com/tech-docs/new-build-system/user-guide).

Some projects won't fall into any of these core categories, in which case you'll need to do a bit more research to find out which, if any, of the plugins introduce special conventions. Most plugins will have at least some basic documentation explaining how they work. You can also run `gradle tasks` to learn more about the build, or `gradle help --task [taskName]` to find out what a task does and what command-line options it supports.

Step 5: Troubleshooting

You should now know enough to navigate around the project, run tests, find out what libraries a project depends on, and execute tasks. If you do happen to run into difficulties, check out the troubleshooting section (https://docs.gradle.org/current/userguide/troubleshooting.html) of the user guide for advice. If that doesn't help, you can always ask questions on the Gradle forums (https://discuss.gradle.org/).