Gradle Guides (https://guides.gradle.org)

Getting Started Creating New Gradle Builds

build passing

(https://travis-ci.org/gradle-guides/creating-new-gradle-builds)

Table of Contents

Creating a new build using Gradle means that you become a *build author*. You still need to follow the same path as a build user (https://guides.gradle.org/using-an-existing-gradle-build), but in addition, you will want to understand the Gradle Build Language and its API.

A build user needs to be able to identify the project type and how such a project is structured. In your case, you know in advance what project type you want but you still need to understand the conventions and project structure. In addition, you also have to understand the basic declarative elements and underlying model of the project type you want to build.

What you'll need

- About 5 minutes
- A text editor or IDE
- A Java Development Kit (JDK), version 1.7 or better
- A Gradle distribution (https://gradle.org/install), version 3.4 or better

Create a project

You can start a new project simply by creating a `build.gradle` file and writing build logic. When you are new to Gradle this can be a daunting task and an easier way is to get your project started via the `init` command. It will create a skeleton project with a Gradle build file based on the project type that you specify. Run the following to create a basic Java project:

```
$ mkdir test
$ cd test
$ gradle init --type java-library  1
```

1 The `init` plugin is documented in the Build Init plugin (https://docs.gradle.org/3.4/userguide/build_init_plugin.html) chapter of the user guide. Also have a look to find out what project types (https://docs.gradle.org/3.4/userguide/build_init_plugin.html#sec:build_init_types) are currently supported.

If you inspect the `test` directory you will see that it has created a number of files and directories.

Top-level directory layout after gradle init

```
├── build.gradle
├── gradle
├── gradlew
├── gradlew.bat
├── settings.gradle
└── src
```

Congratulations! You can now edit this the `build.gradle` files and the source files as a quick start to your project.

If you are specifically interested in building **Java** applications, read the Getting Started Building Java Applications (https://guides.gradle.org/creating-java-applications) guide. You should also read the Java Quickstart (https://docs.gradle.org/3.4/userguide/tutorial_java_projects.html) chapter of the user manual to obtain a wide understanding.

To learn more about bootstrapping other kinds of popular Gradle build, particularly if you want or need to do it by hand, follow these readings for the type of project you want:

Documentation for other popular project types

- **C**, **C***, ***Assembler***, ***Objective-c***, ***Objective-c**: Native builds (https://docs.gradle.org/3.4/userguide/native_software.html)

- **Scala**: Scala plugin (https://docs.gradle.org/3.4/userguide/scala_plugin.html)
- **Groovy**: Groovy quickstart (https://docs.gradle.org/3.4/userguide/tutorial_groovy_projects.html)
- **Android**: Android userguide (http://tools.android.com/tech-docs/new-build-system/user-guide)

Next Steps

Learn the key concepts

Gradle is a very powerful tool, which means you need a good understanding of the underlying model to create maintainable builds. Fortunately, there are just a few core concepts to understand:

- **Build phases**: read the first section of the build lifecycle chapter (https://docs.gradle.org/3.4/userguide/build_lifecycle.html) for a description of these.
- **Properties**: The project API and properties (https://docs.gradle.org/3.4/userguide/writing_build_scripts.html)
- **Tasks**: Tasks (https://docs.gradle.org/3.4/userguide/more_about_tasks.html)
- **File handling**: File handling (https://docs.gradle.org/3.4/userguide/working_with_files.html)
- **Dependency management**: Dependency management (https://docs.gradle.org/3.4/userguide/artifact_dependencies_tutorial.html)

Understand multi-project builds

It's not uncommon in multi-project builds to have subprojects of different types, particularly when you have a polyglot project or a separate project for documentation. If you don't see a need right now to create a multi-project build, then you can safely skip this step—it's easy to make the switch later on if necessary. But if you do want to split a project into multiple, dependent and independent modules, you will need to learn how to configure them and manage the inter-module relationships. That requires a much more in-depth understanding of the topic than was covered in the build user path. We recommend you read the entirety of the multi-project builds chapter (https://docs.gradle.org/3.4/userguide/multi_project_builds.html) of the user guide.

Learn basics of Groovy

You can make a fair bit of progress with Gradle without learning the language that build scripts are written in, but ultimately you'll limit yourself if you don't learn the basics of Groovy. Head to the Groovy website (http://groovy-lang.org/learn.html) to find out what learning resources are available or simply check

out the Groovy fundamentals and Groovy Closures (https://classroom.udacity.com/courses/ud867/lessons/3968239469/concepts/42963752880923) sections of the free Udacity course.

Last updated 2017-03-05 18:50:44 UTC