Gradle Essentials

Gradle is a modern, advanced build automation tool. It inherits the best elements of the past generation of build tools, but it also differs and innovates to bring terseness, elegance, simplicity, and the flexibility to build.

Starting with your first build file, this book will gently guide you through its topics in a step-by-step fashion. You will learn to compile, unit test and package Java projects, build web-applications that run on Servlet containers like Jetty, Tomcat. You will write custom tasks, declare dependencies and write the multi-project builds. You will also learn variety of topics such as migrating from Ant and Maven based projects, publishing artifacts, integration testing, code coverage, running Gradle on CI Servers like Jenkins and writing Gradle plugins. By the end, you'll build polyglot projects in Groovy and Scala with Gradle.

You will also learn the key aspects of Groovy that are essential to understand the Gradle build files. The unique approach taken in this book to explain Gradle DSL with help of Groovy and Gradle API will make Gradle DSL more accessible and intuitive for you.

Who this book is written for

This book is for Java and other JVM-based language developers who want to use Gradle or are already using Gradle on their projects. No prior knowledge of Gradle is required, but some familiarity with build-related terminologies and an understanding of the Java language would help.

What you will learn from this book

- Master the Gradle DSL by identifying the building blocks
- Learn just enough Groovy for Gradle
- Set up tests and reports for your projects to make them CI ready
- Create library, stand-alone, and web projects
- Craft multi-module projects quickly and efficiently
- Migrate existing projects to a modern Gradle build
- Extract common build logic into plugins
- Write builds for languages such as Java, Groovy, and Scala

Kunal Dabir
Abhinandan

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Running Your First Gradle Task'
- A synopsis of the book’s content
- More information on Gradle Essentials
Kunal Dabir has over 10 years of experience working with clients ranging from Fortune 500 companies to startups. Currently, he works as a Lead Consultant at ThoughtWorks. He is a Java user group’s co-organizer and speaks at various meet-ups and conferences.

While he is always eager to learn a new language, he usually codes in languages such as Groovy, Scala, JavaScript, CoffeeScript, Ruby, and Java. He frequently contributes to open source projects and also hosts his own projects on GitHub.

He has always been passionate about automating and scripting. From there, he got a knack for build tools. Apart from Gradle, he has spent a fair amount of time writing build scripts with tools such as Ant, Maven, Grunt, and Gulp. He was introduced to Gradle in 2011 while using Gaelyk. Since then, Gradle has become his tool of choice for build automation.

He can be found on Twitter and GitHub as @kdabir.
When I first came across Gradle in 2011, it was a young yet powerful tool. If I remember correctly, the version was 0.9. It was difficult for me to get started despite Gradle having adequate official documentation. What I missed the most was a guide that would just help me understand the core concepts first, without having to go through the entire documentation.

Gradle is a fantastic build tool. There is so much to learn about it that new users are often clueless about where to start. It is unwise to expect an application developer to go through the entire Gradle reference material just to understand the basics.

This book attempts to help a reader get started with Gradle by revealing the key concepts in a step-by-step manner. It introduces a more advanced topic succinctly. This book focuses on the practical usage of Gradle that a reader can immediately put to use on his or her project. This book strives to stay true to the spirit of 'essentials' by avoiding going into every possible feature and option that Gradle has to offer. Code samples for applications have been consciously kept very small in order to avoid distractions from application logic.

This book is a quick start guide for Gradle. If you are a Java developer already building your code with Ant or Maven and want to switch to Gradle, this book helps you to quickly understand the different concepts of Gradle. Even if you do not have exposure to other build tools such as Ant or Maven, you can start afresh on Gradle with the help of this book. It starts with the basics of Gradle and then gently moves to concepts such as multimodule projects, migration strategies, testing strategies, Continuous Integration, and code coverage with the help of Gradle.
What this book covers
This book can be roughly divided into three parts.

Section 1 includes Chapter 1, Running Your First Gradle Task, Chapter 2, Building Java Projects, and Chapter 3, Building a Web Application. This section introduces the basics of Gradle, with very simple examples, which helps readers to create build files for Java projects and Web applications. It gives a gentle start without involving any complex concepts.

Section 2 includes Chapter 4, Demystifying Build Scripts, and Chapter 5, Multiprojects Build. This section helps the reader to understand the underpinning of Gradle in more depth, still maintaining the 'essentials' aspect of this book. It also helps the reader to understand how to interpret and write scripts that conform to Gradle DSL.

Section 3 includes Chapter 6, The Real-world Project with Gradle, Chapter 7, Testing and Reporting with Gradle, Chapter 8, Organizing Build Logic and Plugins, and Chapter 9, Polyglot Projects. This section covers more real-world use cases that Gradle users come across. Some examples include migrating to Gradle from the existing build system, using Gradle on CI servers, maintaining code quality with Gradle, using Gradle to build project languages such as Groovy and Scala, and so on. These concepts mostly revolve around what various plugins have to offer and also allows the reader to create their own custom plugins.

Also, there are multiple places in all chapters where the reader can find tips, references, and other informative notes.

Chapter 1, Running Your First Gradle Task, starts with an introduction to Gradle and its installation, subsequently moving on to exploring the Gradle command-line interface, and finally running the first build file.

Chapter 2, Building Java Projects, explains topics such as building Java applications and libraries, unit testing with JUnit, reading test reports, and creating application distributions.

Chapter 3, Building a Web Application, deals with building and running Web applications. It also briefly introduces concepts such as dependencies, repositories, and configurations.

Chapter 4, Demystifying Build Scripts, starts with a primer to the Groovy syntax in the context of Gradle DSL. Then, it goes on to explain the backbone concepts of a Gradle build such as build phases, project API, and various topics related to Gradle tasks.
Chapter 5, *Multiprojects Build*, covers a few options to structure multiproject directories. Then, covers organization of a build logic, which is a multiproject build.

Chapter 6, *The Real-world Project with Gradle*, deals with one of the important problems faced by developers, that is, migrating their existing Ant and Maven scripts to Gradle. This chapter provides different strategies and examples, which guide developers to perform migration in a more simpler and manageable way. This chapter also gives an insight into the different ways of publishing artifacts with the help of Gradle and also how a developer can integrate Gradle with Continuous Integration workflow.

Chapter 7, *Testing and Reporting with Gradle*, deals with the integration of the TestNG framework with Gradle. Apart from unit testing with TestNG, it also deals with different strategies for integration testing, which the user can follow to execute integration tests separate from unit test cases. It also discusses about integrating Sonar with Gradle, which helps developers to analyze the quality of code on different parameters, and JaCoCo integration for code coverage analysis.

Chapter 8, *Organizing Build Logic and Plugins*, discusses one of the important building blocks of Gradle plugins, without which you will find this book incomplete. It discusses the needs of the plugin and the different ways in which developers can create a plugin based on the project size and complexities.

Chapter 9, *Polyglot Projects*, demonstrates how to use Gradle for projects that use languages apart from or in addition to Java; this chapter shows the examples of building Groovy and Scala projects.
Running Your First Gradle Task

We are embarking on a fast-paced ride to learn the Gradle Essentials. To take a gentle start, we will first install Gradle. Then, we will get friendly with the Gradle's command-line interface by looking at the usage of the `gradle` command. Also, by the end of this chapter, we would have run our first Gradle build script.

Building a software artifact is a complex process involving various activities such as compiling source code, running automated tests, packaging distributable files, and so on. These activities are further split into many steps, often dependent on the execution order, fetching dependent artifacts, resolving configuration variables, and so on. Executing all these activities manually is cumbersome and often error-prone. A good build automation tool helps us reduce the effort and time it takes to build correct artifacts in a repeatable manner.

Gradle is an advanced build automation tool that brings the best from various proven build tools and innovates on top of them. Gradle can be used to produce artifacts such as web applications, application libraries, documentation, static sites, mobile apps, command lines, and desktop applications. Gradle can be used to build projects based on various languages and technology stacks such as Java, C/C++, Android, Scala, Groovy, Play, Grails, and many more. As Java Virtual Machine (JVM) happens to be one of the first class supported platforms by Gradle, the examples in this book will mostly focus on building Java-based projects.

Gradle gives us full control over build just like Ant but without ever needing to repeat ourselves by providing intelligent defaults in the form of conventions. Gradle truly works by conventions over configuration, just like Maven. However, it never gets in our way when we need to deviate. Also this puts it in complete contrast with Maven. Gradle attempts to maintain the right balance between conventions and configurability.
Running Your First Gradle Task

The previous generation of build tools, such as Ant and Maven, chose XML to represent the build logic. While XML is human-readable, it is more of a machine-friendly format (easier to be read/written by programs). It is great for representing and exchanging hierarchical data, but when it comes to writing any logic, even the simplest logic can easily take hundreds of lines. On the other hand, a Gradle build can be configured using very human-friendly Groovy DSL. Groovy is a powerful, expressive, and low ceremony dynamic language and is a perfect fit for build scripts.

Gradle itself is a JVM application written in Java and Groovy. Since Gradle runs on the JVM, it runs the same way on Windows, Mac OS X and Linux. Gradle also boasts an advanced dependency resolution system and can resolve dependencies from the existing Maven and Ivy repositories or even a file system.

Over the years Gradle has matured into a very stable open source project with active contributors and commercial backing. The rich plugin ecosystem and vibrant community makes Gradle an excellent choice for a variety of projects. Gradle already has an impressive list of adopters, which includes tech giants such as Google Android, LinkedIn, Unity 3D, Netflix and many more. Open source libraries and frameworks such as Spring, Hibernate, and Grails are using Gradle to power their builds.

Installing Gradle

Before we move forward with running Gradle, we must have it installed on our machine. There are multiple ways through which Gradle can be installed and updated. We will first see a more manual way to install Gradle and then take a quick look at installing it via some commonly used package managers. We can choose any one method that fits the bill. Irrespective of the way we install Gradle, we must meet the following prerequisite.

Gradle needs Java Runtime Environment (JRE) 6 or Java Development Kit (JDK) 1.6 or higher. There is no other dependency. We recommend having JDK installed. To verify this, on the command line, we can check the Java version with the following command:

$ java -version
java version "1.8.0"
Java(TM) SE Runtime Environment (build 1.8.0-b132)
Java HotSpot(TM) 64-Bit Server VM (build 25.0-b70, mixed mode)
If we don't see the output more or less like the one shown in the preceding command, there is problem with our JDK installation.

The latest JDK can be downloaded from the following URL:

### Installing manually

If we want a finer control over the installation then this is a suitable route. This could be the case, when we cannot use the package managers, want very specific binaries to be downloaded and installed, or behind corporate firewalls where automatic downloading by package managers is not allowed. We need to download the Gradle binaries and make them available for use on the command line.

The latest Gradle distribution can be downloaded from http://www.gradle.org/downloads. As of writing the latest version is 2.9.

Gradle binary distribution comes in two flavors as follows:

- gradle-2.9-all.zip: This contains binaries, sources, and documentation
- gradle-2.9-bin.zip: This contains binaries only

We can download any of the above depending on what we need. Also, this is an OS-independent zip so the same zip can be extracted on Mac OS X, Windows, and Linux. The next section makes the Gradle command available on the command line. This section is dependent on the OS we use.

### Installing on Mac OS X and Linux

Let’s say we extracted the downloaded zip as ~/gradle-2.9/. Now, we just need to add the following two lines at the end of .bashrc/, .bash_profile/, or .zshrc, depending on the OS and the shell that we use:

```shell
export GRADLE_HOME=~/.gradle-2.9
export PATH=$PATH:$GRADLE_HOME/bin
```

Restart the terminal or source the modified file to have the change take effect.
Installing on Windows

Let's say we extracted the zip as `C:\gradle-2.9`, then perform the following steps:

1. Open the Start menu, right click on **Computer** and select **Properties**.
2. On **Advanced system settings**, select the **Advanced** tab, and then select **Environment Variables**.

   ![System Properties dialog box](image)

3. Click on **New**.
4. Create a **GRADLE_HOME** environment variable with the value `C:\gradle-2.9`.

---

**Downloading the example code**

You can download the example code files from your account at [http://www.packtpub.com](http://www.packtpub.com) for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit [http://www.packtpub.com/support](http://www.packtpub.com/support) and register to have the files e-mailed directly to you.
In future when we download the later version of Gradle, we would need to change on this value to point to the correct folder.

5. Edit (or add if not already there) the \PATH\ environment variable. At the end of its value, append ;\%GRADLE_HOME%\bin (add a semicolon if multiple path entries exist).

**Alternate methods of installing Gradle**

Although the manual installation gives absolute control over the installation process, various tasks such as downloading and extracting the right version, upgrading to the latest versions, uninstalling, and editing environment variables quickly become cumbersome and error-prone. That is why many people prefer package managers to control the whole process.
Installing via OS-specific package managers

While installing manually, as mentioned in the previous section, is very easy, we can make it super-easy by using a package manager.

Some Linux distributions like Ubuntu ship with their package manager, Mac OS X, Windows don’t have any package manager installed by default. However, luckily, there are multiple package managers available for both platforms. We will see the example of Homebrew on Mac and Chocolatey on Windows.

**Mac OS X**

Make sure we have Homebrew installed. If it is, installing Gradle is only a matter of using the following command:

```
$ brew install gradle
```

More details on Homebrew can be found at [http://brew.sh](http://brew.sh).

**Linux (Ubuntu)**

Using the built in package manager on Ubuntu, which is called Advanced Packaging Tool (APT), we can install Gradle with the following command:

```
$ sudo apt-get install gradle
```

**Windows**

If we have Chocolatey installed, installing Gradle is just a command away:

```
c:\> cinst gradle
```

More details on Chocolatey can be found at [https://chocolatey.org](https://chocolatey.org).

**Installing via SDKMAN**

SDKMAN stands for the Software Development Kit Manager. In its own words, the website describes it as: "SDKMAN! is a tool for managing parallel versions of multiple Software Development Kits on most Unix based systems."
The advantage SDKMAN has over other package managers is that we can have multiple Gradle versions installed on a system and select a different version for a given project. If we have it installed, all we need to do is run following command:

$ sdk install gradle

SDKMAN can be installed from http://sdkman.io/.

**Verifying the installation**

In whichever way we choose to install Gradle, it's a good idea to verify that if it's working before we move ahead. We can do this by simply checking for Gradle's version on the command line:

$ gradle --version

```
------------------------------------------------------------
Gradle 2.9
------------------------------------------------------------
Build time:   2015-11-17 07:02:17 UTC
Build number: none
Revision:     b463d7980c40d44c4657dc80025275b84a29e31f
Groovy:       2.4.4
Ant:          Apache Ant(TM) version 1.9.3 compiled on December 23 2013
JVM:          1.8.0_25 (Oracle Corporation 25.25-b02)
OS:           Mac OS X 10.10.5 x86_64
```

If we see output similar to the above, we have Gradle installed correctly on our machine.

We can use `-v` instead `--version` to get the same result.

**Setting JVM options**

Although it's not required most of the time, but if in case we need to set some global options for the JVM that Gradle will use, Gradle provides us a convenient way to do that. We can set the `GRADLE_OPTS` environment variable with acceptable flags to tune the JVM.
Gradle also honors the `JAVA_OPTS` environment variable. However, we need to be careful when setting it, as this affects the setting for all the Java programs on a machine. Setting that we want to keep common for all the Java apps should be done via this variable and those that only need to be applied to Gradle should be set via `GRADLE_OPTS`.

Some commonly used options are `-Xms` and `-Xmx`, which set the minimum and maximum heap size of the JVM.

### The Gradle command-line interface

Gradle, just like other build tools, is primarily run from a command line. That's why it is worth spending some time to get familiar with its command-line interface. Typically, a `gradle` command is issued from the root of a project directory with some tasks to be executed. Let's say we are in the `hello-gradle` directory, which is currently empty.

Gradle provides a very simple **command-line interface (CLI)**, which takes the following form:

```
gradle [options...] [tasks...]
```

As we can see, apart from the `gradle` command itself, everything else is optional. The **options** tweak the execution of the Gradle whereas **tasks**, which we will see in detail later, are the basic units of work. Options are common across all projects and are specific to Gradle but tasks may vary depending on the project in which the `gradle` command is being run.

There are some tasks that are available on all projects. One such task is `help`:

```
$ gradle help
:help
```

Welcome to Gradle 2.9.

To run a build, run `gradle <task>` ...

To see a list of available tasks, run `gradle tasks`
To see a list of command-line options, run gradle --help

To see more detail about a task, run gradle help --task <task>

BUILD SUCCESSFUL

Total time: 0.639 secs

Gradle is helping us out by telling us how to find all the available tasks and list all command-line options. Let’s first check what other tasks are currently available on our project. Remember we are still in the empty directory hello-gradle:

$ gradle tasks
:tasks

-------------------------------------------------------------
All tasks runnable from root project
-------------------------------------------------------------

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

Help tasks
--------
components - Displays the components produced by root project 'hello-gradle'. [incubating]
dependencies - Displays all dependencies declared in root project 'hello-gradle'.
dependencyInsight - Displays the insight into a specific dependency in root project 'hello-gradle'.
help - Displays a help message.
model - Displays the configuration model of root project 'hello-gradle'. [incubating]
projects - Displays the sub-projects of root project 'hello-gradle'.
properties - Displays the properties of root project 'hello-gradle'.
tasks - Displays the tasks runnable from root project 'hello-gradle'.
Running Your First Gradle Task

To see all tasks and more detail, run gradle tasks --all

To see more detail about a task, run gradle help --task <task>

BUILD SUCCESSFUL

Total time: 0.652 secs

This shows us some generic tasks that are available even without us adding any task to our project. We can try running all these tasks and see the output. We will see these tasks in details in the upcoming chapters.

The other useful command gradle help suggested us to check all the available options with the --help option.

![Help task is not the same as the --help option.]

When we run the gradle --help command, we get the following output:

$ gradle --help

USAGE: gradle [option...] [task...]

-?, -h, --help   Shows this help message.
-a, --no-rebuild Do not rebuild project dependencies.
-b, --build-file Specifies the build file.
...

(The output is truncated for brevity.)

The option has a long form such as --help and may have a short from such as -h. We have already used one option before, that is --version or -v, which prints information about the Gradle version. The following are some commonly used options; there are many more options, which can be seen using the gradle --help command:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-b, --build-file</td>
<td>This specifies a build file (default: build.gradle)</td>
</tr>
<tr>
<td>--continue</td>
<td>This continues task execution even after a task failure</td>
</tr>
<tr>
<td>-D, --system-prop</td>
<td>This sets the system property of the JVM</td>
</tr>
</tbody>
</table>
### Options Description

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d, --debug</td>
<td>This prints debug level logs</td>
</tr>
<tr>
<td>--gui</td>
<td>This starts Gradle GUI</td>
</tr>
<tr>
<td>-i, --info</td>
<td>This prints info level logs</td>
</tr>
<tr>
<td>-P, --project-prop</td>
<td>This adds a property to the project</td>
</tr>
<tr>
<td>-q, --quiet</td>
<td>This logs only errors</td>
</tr>
<tr>
<td>-s, --stacktrace</td>
<td>This prints stack traces for exceptions</td>
</tr>
<tr>
<td>-x, --exclude-task</td>
<td>This excludes a specific task</td>
</tr>
</tbody>
</table>

## The first Gradle build script

So we are now ready to get our feet wet and see our first Gradle script in action. Let's create a file called `build.gradle` in the `hello-gradle` directory. Unless the build file path is provided using the `--build-file` option, Gradle treats the current directory as a project root and tries to find the `build.gradle` file there. If we have used Ant or Maven earlier, we can relate this file with `build.xml` or `pom.xml`, respectively.

Now, open the `build.gradle` file and let's declare a task by adding the following line:

```gradle
    task helloWorld
```

We should be able to see this task on the command line as follows:

```bash
$ gradle tasks
... Other tasks
-------------
helloWorld
... 
```

Here, we have successfully created a task object called `helloWorld`. Tasks are first-class objects in Gradle, which means they have properties and methods on them. This gives us tremendous flexibility in terms of customizability and programmability of build.
However, this task actually does not do anything yet. So let's add some meaningful action to this task:

```groovy
task helloWorld << {
    println "Hello, World!"
}
```

Now from the command line, we can execute this task by issuing the following command:

```
$ gradle -q helloWorld
Hello, World!
```

Notice that we used the `-q` flag to reduce the verbosity in the output. When this task is run, we see the output that our task generates but nothing from Gradle unless it's an error.

Now, let's try to briefly understand the `build.gradle` file. The first line declares the tasks and starts the body of a code block that will be executed at the end. The left shift operator (`<<`) might feel oddly placed, but it is very important in this context. We will see in the later chapters what it exactly means. The second line is a Groovy statement that prints the given string to the console. Also, the third line ends the code block.

```
Groovy's println "Hello, World!" is equivalent to System.out.println("Hello, World!") in Java.
```

**Task name abbreviation**

While calling a gradle task from a command line, we can save a few keystrokes by typing only the characters that are enough to uniquely identify the task name. For example, the task `helloWorld` can be called using `gradle hw`. We can also use `helloW`, `hWorld`, or even `heWo`. However, if we just call `gradle h`, then the `help` task will be called.

This comes very handy when we need to frequently call long Gradle task names. For example, a task named `deployToProductionServer` can be invoked just by calling `gradle dTPS`, provided that this does not match any other task name abbreviation.
Gradle Daemon

While we are talking about frequently calling Gradle, it is a good time to know about a recommended technique to boost the performance of our builds. Gradle Daemon, a process that keeps running in the background, can speed up the builds significantly.

For a given gradle command invocation, we can specify the --daemon flag to enable the Daemon process. However, we should keep in mind that when we start the daemon, only the subsequent builds will be faster, but not the current one.

For example:

```bash
$ gradle helloWorld --daemon
Starting a new Gradle Daemon for this build (subsequent builds will be faster).
:helloWorld
Hello, World!

BUILD SUCCESSFUL

Total time: 2.899 secs
```

```bash
$ gradle helloWorld
:helloWorld
Hello, World!

BUILD SUCCESSFUL

Total time: 0.6 secs
```

In the preceding example, if we notice the time taken by two runs, the second one completed much faster, thanks to the Gradle Daemon.

We can also prevent a specific build invocation from utilizing a Daemon process by passing the --no-daemon flag.

There are various ways to enable or disable Gradle Daemon, which are documented at [https://docs.gradle.org/current/userguide/gradle_daemon.html](https://docs.gradle.org/current/userguide/gradle_daemon.html)
Running Your First Gradle Task

Gradle Wrapper
A Gradle Wrapper consists of a `gradlew` shell script for Linux/Mac OS X, a `gradlew.bat` batch script for Windows, and a few helper files. These files can be generated by running a gradle wrapper task and should be checked into the version control system (VCS) along with project sources. Instead of using the system-wide `gradle` command, we can run the builds via the wrapper script.

Some of the advantages of running builds via a wrapper script are as follows:

1. We don’t need to download and install Gradle manually. The wrapper script takes care of this.
2. It uses a specific version of Gradle that the project needs. This reduces the risk of breaking a project’s build because of incompatible Gradle versions. We can safely upgrade (or downgrade) the system-wide Gradle installation without affecting our projects.
3. It transparently enforces the same Gradle version for our project across all developers’ machines in the team.
4. This is extremely useful in Continuous Integration build environments, as we do not need to install/update Gradle on the servers.

Generating wrapper files
The Gradle wrapper task is already available to all Gradle projects. To generate the wrapper scripts and supporting files, just execute the following code from the command line:

```
$ gradle wrapper
```

While generating wrapper, we can specify the exact Gradle version as follows:

```
$ gradle wrapper --gradle-version 2.9
```

In this example, we are specifying the Gradle version to be used is 2.9. After running this command, we should check-in the generated files into VCS. We can customize the wrapper task to use a configured Gradle version, produce wrapper scripts with different names, change their locations, and so on.

Running a build via wrapper
For availing the benefits of a wrapper script, instead of using the `gradle` command, we need to call the wrapper script based on our OS.
On Mac OS X/Linux:

$ ./gradlew taskName

On Windows:

$ gradlew taskName

We can use the arguments and flags exactly in the same way as we pass to the gradle command.

Summary

In this chapter, we started with a brief introduction to Gradle. Then, we looked at manual installation and also installation via package managers. We also learned about Gradle's command-line interface. Also, finally, we wrote our first Gradle build script.

If you have followed the chapter until this point, you are all set to check out any Gradle-based project on your machine and execute builds. Also, you are equipped with the knowledge to write a very basic Gradle build script. Going forward, we will look at building Java-based projects with Gradle.
Where to buy this book

You can buy Gradle Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.