Docs

[User Manual](http://docs.google.com/userguide/userguide.html)

[Guides and Tutorials](https://guides.gradle.org)

[DSL Reference](http://docs.google.com/dsl/)

[Javadoc](http://docs.google.com/javadoc/)

[Release Notes](http://docs.google.com/release-notes.html)

[Forums](https://discuss.gradle.org/)

[Training](https://gradle.org/training/)

[Try Gradle Enterprise](https://gradle.com/enterprise)

[PDF](http://docs.google.com/userguide/userguide.pdf)

* [User Manual Home](http://docs.google.com/userguide/userguide.html)
* [Release Notes](http://docs.google.com/release-notes.html)
* [Installing Gradle](http://docs.google.com/userguide/installation.html)
* [Tutorials](https://guides.gradle.org/)

### Reference

* [Groovy DSL Reference](http://docs.google.com/dsl/)
* [Gradle API Javadoc](http://docs.google.com/javadoc/)
* [Core Plugins](http://docs.google.com/userguide/plugin_reference.html)
* [Gradle & Third-party Tools](http://docs.google.com/userguide/third_party_integration.html)

### Getting Started

* [Creating New Gradle Builds](https://guides.gradle.org/creating-new-gradle-builds/)
* [Creating Build Scans](https://guides.gradle.org/creating-build-scans/)
* [Migrating From Maven](https://guides.gradle.org/migrating-from-maven/)

### Running Gradle Builds

* [Command-Line Interface](http://docs.google.com/userguide/command_line_interface.html)
* [Customizing Execution](#gjdgxs)
  + [Configuring the Build Environment](http://docs.google.com/userguide/build_environment.html)
  + [Configuring the Gradle Daemon](http://docs.google.com/userguide/gradle_daemon.html)
  + [Initialization Scripts](http://docs.google.com/userguide/init_scripts.html)
* [Directory Layout](http://docs.google.com/userguide/directory_layout.html)
* [Executing Multi-Project Builds](http://docs.google.com/userguide/intro_multi_project_builds.html)
* [Gradle Wrapper](http://docs.google.com/userguide/gradle_wrapper.html)
* [Troubleshooting](http://docs.google.com/userguide/troubleshooting.html)
* [Using Build Scans](https://docs.gradle.com/build-scan-plugin)
* [Enabling and Configuring the Build Cache](http://docs.google.com/userguide/build_cache.html)
* [Integrating Separate Gradle Builds (Composite Builds)](http://docs.google.com/userguide/composite_builds.html)

### Authoring Gradle Builds

* [Fundamentals](#30j0zll)
  + [Introducing the Basics of Build Scripts](http://docs.google.com/userguide/tutorial_using_tasks.html)
  + [Working with Tasks](http://docs.google.com/userguide/more_about_tasks.html)
  + [Learning More About Build Scripts](http://docs.google.com/userguide/writing_build_scripts.html)
  + [Working with Files](http://docs.google.com/userguide/working_with_files.html)
  + [Creating Custom Task Types](http://docs.google.com/userguide/custom_tasks.html)
  + [Using Gradle Plugins](http://docs.google.com/userguide/plugins.html)
  + [The Standard Gradle Plugins](http://docs.google.com/userguide/standard_plugins.html)
  + [Understanding the Build Lifecycle](http://docs.google.com/userguide/build_lifecycle.html)
  + [Working with Logging](http://docs.google.com/userguide/logging.html)
  + [Configuring Multi-Project Builds](http://docs.google.com/userguide/multi_project_builds.html)
* [Best Practices](#1fob9te)
  + [Authoring Maintainable Build Scripts](http://docs.google.com/userguide/authoring_maintainable_build_scripts.html)
  + [Organizing Gradle Projects](http://docs.google.com/userguide/organizing_gradle_projects.html)
  + [Optimizing Build Performance](https://guides.gradle.org/performance/)
  + [Using the Build Cache](https://guides.gradle.org/using-build-cache/)
* [Dependency Management](#3znysh7)
  + [Introduction to Dependency Management](http://docs.google.com/userguide/introduction_dependency_management.html)
  + [Dependency Management Terminology](http://docs.google.com/userguide/dependency_management_terminology.html)
  + [Dependency Types](http://docs.google.com/userguide/dependency_types.html)
  + [Repository Types](http://docs.google.com/userguide/repository_types.html)
  + [Declaring Dependencies](http://docs.google.com/userguide/declaring_dependencies.html)
  + [Declaring Repositories](http://docs.google.com/userguide/declaring_repositories.html)
  + [Inspecting Dependencies](http://docs.google.com/userguide/inspecting_dependencies.html)
  + [Managing Dependency Configurations](http://docs.google.com/userguide/managing_dependency_configurations.html)
  + [Managing Transitive Dependencies](http://docs.google.com/userguide/managing_transitive_dependencies.html)
  + [Dependency Locking](http://docs.google.com/userguide/dependency_locking.html)
  + [Troubleshooting Dependency Resolution](http://docs.google.com/userguide/troubleshooting_dependency_resolution.html)
  + [Customizing Dependency Resolution Behavior](http://docs.google.com/userguide/customizing_dependency_resolution_behavior.html)
  + [Dependency Cache Internals](http://docs.google.com/userguide/dependency_cache.html)
  + [Working with Dependencies](http://docs.google.com/userguide/working_with_dependencies.html)
* [Publishing Artifacts](http://docs.google.com/userguide/artifact_management.html)
* [C++ Projects](#2et92p0)
  + [Building Native Software](http://docs.google.com/userguide/native_software.html)
  + [Software Model Concepts](http://docs.google.com/userguide/software_model_concepts.html)
  + [Rule-based Model Configuration](http://docs.google.com/userguide/software_model.html)
  + [Implementing Model Rules in a Plugin](http://docs.google.com/userguide/rule_source.html)
  + [Extending the Software Model](http://docs.google.com/userguide/software_model_extend.html)
* [Java Projects](#tyjcwt)
  + [Building Java & JVM projects](http://docs.google.com/userguide/building_java_projects.html)
  + [Testing Java & JVM projects](http://docs.google.com/userguide/java_testing.html)
* [Advanced Techniques](#3dy6vkm)
  + [Configuring Tasks Lazily](http://docs.google.com/userguide/lazy_configuration.html)
  + [Developing Parallel Tasks](https://guides.gradle.org/using-the-worker-api/)
  + [Testing Your Build with TestKit](http://docs.google.com/userguide/test_kit.html)
  + [Using Ant from Gradle](http://docs.google.com/userguide/ant.html)
* [Sample Gradle builds](#1t3h5sf)
  + [Groovy DSL Samples](https://github.com/gradle/gradle/tree/master/subprojects/docs/src/samples)
  + [Kotlin DSL Samples](https://github.com/gradle/kotlin-dsl/tree/master/samples)

### Extending Gradle

* [Writing Custom Plugins](http://docs.google.com/userguide/custom_plugins.html)
* [Plugin Development Guides](https://gradle.org/guides/?q=Plugin+Development)

[Edit this page](https://github.com/gradle/gradle/edit/master/subprojects/docs/src/docs/userguide/)

# Build Script Basics

Contents

[Projects and tasks](#4d34og8)

[Hello world](#2s8eyo1)

[A shortcut task definition](#17dp8vu)

[Build scripts are code](#3rdcrjn)

[Task dependencies](#26in1rg)

[Dynamic tasks](#lnxbz9)

[Manipulating existing tasks](#35nkun2)

[Shortcut notations](#1ksv4uv)

[Extra task properties](#44sinio)

[Using Ant Tasks](#2jxsxqh)

[Using methods](#z337ya)

[Default tasks](#3j2qqm3)

[Configure by DAG](#1y810tw)

[External dependencies for the build script](#4i7ojhp)

[Where to next?](#2xcytpi)

[Projects and tasks](#4d34og8)

Everything in Gradle sits on top of two basic concepts: *projects* and *tasks*.

Every Gradle build is made up of one or more *projects*. What a project represents depends on what it is that you are doing with Gradle. For example, a project might represent a library JAR or a web application. It might represent a distribution ZIP assembled from the JARs produced by other projects. A project does not necessarily represent a thing to be built. It might represent a thing to be done, such as deploying your application to staging or production environments. Don’t worry if this seems a little vague for now. Gradle’s build-by-convention support adds a more concrete definition for what a project is.

Each project is made up of one or more *tasks*. A task represents some atomic piece of work which a build performs. This might be compiling some classes, creating a JAR, generating Javadoc, or publishing some archives to a repository.

For now, we will look at defining some simple tasks in a build with one project. Later chapters will look at working with multiple projects and more about working with projects and tasks.

[Hello world](#2s8eyo1)

You run a Gradle build using the gradle command. The gradle command looks for a file called build.gradle in the current directory.[[1](#1ci93xb)] We call this build.gradle file a *build script*, although strictly speaking it is a build configuration script, as we will see later. The build script defines a project and its tasks.

To try this out, create the following build script named build.gradle.

[Example: Your first build script](#3whwml4)

**build.gradle**

task hello {  
 doLast {  
 println 'Hello world!'  
 }  
}

In a command-line shell, move to the containing directory and execute the build script with gradle -q hello:

| **💡** | What does -q do?  Most of the examples in this user guide are run with the -q command-line option. This suppresses Gradle’s log messages, so that only the output of the tasks is shown. This keeps the example output in this user guide a little clearer. You don’t need to use this option if you don’t want to. See [Logging](http://docs.google.com/logging.html#logging) for more details about the command-line options which affect Gradle’s output. |
| --- | --- |

[Example: Execution of a build script](#2bn6wsx)

**Output of** gradle -q hello

> gradle -q hello  
Hello world!

What’s going on here? This build script defines a single task, called hello, and adds an action to it. When you run gradle hello, Gradle executes the hello task, which in turn executes the action you’ve provided. The action is simply a closure containing some Groovy code to execute.

If you think this looks similar to Ant’s targets, you would be right. Gradle tasks are the equivalent to Ant targets, but as you will see, they are much more powerful. We have used a different terminology than Ant as we think the word *task* is more expressive than the word *target*. Unfortunately this introduces a terminology clash with Ant, as Ant calls its commands, such as javac or copy, tasks. So when we talk about tasks, we *always* mean Gradle tasks, which are the equivalent to Ant’s targets. If we talk about Ant tasks (Ant commands), we explicitly say *Ant task*.

[A shortcut task definition](#17dp8vu)

| **✨** | This functionality is deprecated and will be removed in Gradle 5.0 without replacement. Use the methods [Task.doFirst(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.Task.html#org.gradle.api.Task:doFirst(org.gradle.api.Action)) and [Task.doLast(org.gradle.api.Action)](http://docs.google.com/dsl/org.gradle.api.Task.html#org.gradle.api.Task:doLast(org.gradle.api.Action)) to define an action instead, as demonstrated by the rest of the examples in this chapter. |
| --- | --- |

There is a shorthand way to define a task like our hello task above, which is more concise.

[Example: A task definition shortcut](#qsh70q)

**build.gradle**

task hello << {  
 println 'Hello world!'  
}

Again, this defines a task called hello with a single closure to execute. The << operator is simply an alias for doLast.

[Build scripts are code](#3rdcrjn)

Gradle’s build scripts give you the full power of Groovy. As an appetizer, have a look at this:

[Example: Using Groovy in Gradle’s tasks](#3as4poj)

**build.gradle**

task upper {  
 doLast {  
 String someString = 'mY\_nAmE'  
 println "Original: " + someString  
 println "Upper case: " + someString.toUpperCase()  
 }  
}

**Output of** gradle -q upper

> gradle -q upper  
Original: mY\_nAmE  
Upper case: MY\_NAME

or

[Example: Using Groovy in Gradle’s tasks](#1pxezwc)

**build.gradle**

task count {  
 doLast {  
 4.times { print "$it " }  
 }  
}

**Output of** gradle -q count

> gradle -q count  
0 1 2 3

[Task dependencies](#26in1rg)

As you probably have guessed, you can declare tasks that depend on other tasks.

[Example: Declaration of task that depends on other task](#49x2ik5)

**build.gradle**

task hello {  
 doLast {  
 println 'Hello world!'  
 }  
}  
task intro(dependsOn: hello) {  
 doLast {  
 println "I'm Gradle"  
 }  
}

**Output of** gradle -q intro

> gradle -q intro  
Hello world!  
I'm Gradle

To add a dependency, the corresponding task does not need to exist.

[Example: Lazy dependsOn - the other task does not exist (yet)](#2p2csry)

**build.gradle**

task taskX(dependsOn: 'taskY') {  
 doLast {  
 println 'taskX'  
 }  
}  
task taskY {  
 doLast {  
 println 'taskY'  
 }  
}

**Output of** gradle -q taskX

> gradle -q taskX  
taskY  
taskX

The dependency of taskX to taskY is declared before taskY is defined. This is very important for multi-project builds. Task dependencies are discussed in more detail in [Adding dependencies to a task](http://docs.google.com/more_about_tasks.html#sec:adding_dependencies_to_tasks).

Please notice that you can’t use [shortcut notation](#1ksv4uv) when referring to a task that is not yet defined.

[Dynamic tasks](#lnxbz9)

The power of Groovy can be used for more than defining what a task does. For example, you can also use it to dynamically create tasks.

[Example: Dynamic creation of a task](#147n2zr)

**build.gradle**

4.times { counter ->  
 task "task$counter" {  
 doLast {  
 println "I'm task number $counter"  
 }  
 }  
}

**Output of** gradle -q task1

> gradle -q task1  
I'm task number 1

[Manipulating existing tasks](#35nkun2)

Once tasks are created they can be accessed via an *API*. For instance, you could use this to dynamically add dependencies to a task, at runtime. Ant doesn’t allow anything like this.

[Example: Accessing a task via API - adding a dependency](#3o7alnk)

**build.gradle**

4.times { counter ->  
 task "task$counter" {  
 doLast {  
 println "I'm task number $counter"  
 }  
 }  
}  
task0.dependsOn task2, task3

**Output of** gradle -q task0

> gradle -q task0  
I'm task number 2  
I'm task number 3  
I'm task number 0

Or you can add behavior to an existing task.

[Example: Accessing a task via API - adding behaviour](#23ckvvd)

**build.gradle**

task hello {  
 doLast {  
 println 'Hello Earth'  
 }  
}  
hello.doFirst {  
 println 'Hello Venus'  
}  
hello.doLast {  
 println 'Hello Mars'  
}  
hello {  
 doLast {  
 println 'Hello Jupiter'  
 }  
}

**Output of** gradle -q hello

> gradle -q hello  
Hello Venus  
Hello Earth  
Hello Mars  
Hello Jupiter

The calls doFirst and doLast can be executed multiple times. They add an action to the beginning or the end of the task’s actions list. When the task executes, the actions in the action list are executed in order.

[Shortcut notations](#1ksv4uv)

There is a convenient notation for accessing an *existing* task. Each task is available as a property of the build script:

[Example: Accessing task as a property of the build script](#ihv636)

**build.gradle**

task hello {  
 doLast {  
 println 'Hello world!'  
 }  
}  
hello.doLast {  
 println "Greetings from the $hello.name task."  
}

**Output of** gradle -q hello

> gradle -q hello  
Hello world!  
Greetings from the hello task.

This enables very readable code, especially when using the tasks provided by the plugins, like the compile task.

[Extra task properties](#44sinio)

You can add your own properties to a task. To add a property named myProperty, set ext.myProperty to an initial value. From that point on, the property can be read and set like a predefined task property.

[Example: Adding extra properties to a task](#32hioqz)

**build.gradle**

task myTask {  
 ext.myProperty = "myValue"  
}  
  
task printTaskProperties {  
 doLast {  
 println myTask.myProperty  
 }  
}

**Output of** gradle -q printTaskProperties

> gradle -q printTaskProperties  
myValue

Extra properties aren’t limited to tasks. You can read more about them in [Extra properties](http://docs.google.com/writing_build_scripts.html#sec:extra_properties).

[Using Ant Tasks](#2jxsxqh)

Ant tasks are first-class citizens in Gradle. Gradle provides excellent integration for Ant tasks by simply relying on Groovy. Groovy is shipped with the fantastic AntBuilder. Using Ant tasks from Gradle is as convenient and more powerful than using Ant tasks from a build.xml file. From the example below, you can learn how to execute Ant tasks and how to access Ant properties:

[Example: Using AntBuilder to execute ant.loadfile target](#1hmsyys)

**build.gradle**

task loadfile {  
 doLast {  
 def files = file('./antLoadfileResources').listFiles().sort()  
 files.each { File file ->  
 if (file.isFile()) {  
 ant.loadfile(srcFile: file, property: file.name)  
 println " \*\*\* $file.name \*\*\*"  
 println "${ant.properties[file.name]}"  
 }  
 }  
 }  
}

**Output of** gradle -q loadfile

> gradle -q loadfile  
 \*\*\* agile.manifesto.txt \*\*\*  
Individuals and interactions over processes and tools  
Working software over comprehensive documentation  
Customer collaboration over contract negotiation  
Responding to change over following a plan  
 \*\*\* gradle.manifesto.txt \*\*\*  
Make the impossible possible, make the possible easy and make the easy elegant.  
(inspired by Moshe Feldenkrais)

There is lots more you can do with Ant in your build scripts. You can find out more in [Ant](http://docs.google.com/ant.html#ant).

[Using methods](#z337ya)

Gradle scales in how you can organize your build logic. The first level of organizing your build logic for the example above, is extracting a method.

[Example: Using methods to organize your build logic](#41mghml)

**build.gradle**

task checksum {  
 doLast {  
 fileList('./antLoadfileResources').each { File file ->  
 ant.checksum(file: file, property: "cs\_$file.name")  
 println "$file.name Checksum: ${ant.properties["cs\_$file.name"]}"  
 }  
 }  
}  
  
task loadfile {  
 doLast {  
 fileList('./antLoadfileResources').each { File file ->  
 ant.loadfile(srcFile: file, property: file.name)  
 println "I'm fond of $file.name"  
 }  
 }  
}  
  
File[] fileList(String dir) {  
 file(dir).listFiles({file -> file.isFile() } as FileFilter).sort()  
}

**Output of** gradle -q loadfile

> gradle -q loadfile  
I'm fond of agile.manifesto.txt  
I'm fond of gradle.manifesto.txt

Later you will see that such methods can be shared among subprojects in multi-project builds. If your build logic becomes more complex, Gradle offers you other very convenient ways to organize it. We have devoted a whole chapter to this. See [Organizing Gradle Projects](http://docs.google.com/organizing_gradle_projects.html#organizing_gradle_projects).

[Default tasks](#3j2qqm3)

Gradle allows you to define one or more default tasks that are executed if no other tasks are specified.

[Example: Defining a default task](#2grqrue)

**build.gradle**

defaultTasks 'clean', 'run'  
  
task clean {  
 doLast {  
 println 'Default Cleaning!'  
 }  
}  
  
task run {  
 doLast {  
 println 'Default Running!'  
 }  
}  
  
task other {  
 doLast {  
 println "I'm not a default task!"  
 }  
}

**Output of** gradle -q

> gradle -q  
Default Cleaning!  
Default Running!

This is equivalent to running gradle clean run. In a multi-project build every subproject can have its own specific default tasks. If a subproject does not specify default tasks, the default tasks of the parent project are used (if defined).

[Configure by DAG](#1y810tw)

As we later describe in full detail (see [Build Lifecycle](http://docs.google.com/build_lifecycle.html#build_lifecycle)), Gradle has a configuration phase and an execution phase. After the configuration phase, Gradle knows all tasks that should be executed. Gradle offers you a hook to make use of this information. A use-case for this would be to check if the release task is among the tasks to be executed. Depending on this, you can assign different values to some variables.

In the following example, execution of the distribution and release tasks results in different value of the version variable.

[Example: Different outcomes of build depending on chosen tasks](#vx1227)

**build.gradle**

task distribution {  
 doLast {  
 println "We build the zip with version=$version"  
 }  
}  
  
task release(dependsOn: 'distribution') {  
 doLast {  
 println 'We release now'  
 }  
}  
  
gradle.taskGraph.whenReady {taskGraph ->  
 if (taskGraph.hasTask(release)) {  
 version = '1.0'  
 } else {  
 version = '1.0-SNAPSHOT'  
 }  
}

**Output of gradle -q distribution**

> gradle -q distribution  
We build the zip with version=1.0-SNAPSHOT

**Output of gradle -q release**

> gradle -q release  
We build the zip with version=1.0  
We release now

The important thing is that whenReady affects the release task *before* the release task is executed. This works even when the release task is not the *primary* task (i.e., the task passed to the gradle command).

[External dependencies for the build script](#4i7ojhp)

If your build script needs to use external libraries, you can add them to the script’s classpath in the build script itself. You do this using the buildscript() method, passing in a closure which declares the build script classpath.

[Example: Declaring external dependencies for the build script](#3fwokq0)

**build.gradle**

buildscript {  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath group: 'commons-codec', name: 'commons-codec', version: '1.2'  
 }  
}

The closure passed to the buildscript() method configures a [ScriptHandler](http://docs.google.com/javadoc/org/gradle/api/initialization/dsl/ScriptHandler.html) instance. You declare the build script classpath by adding dependencies to the classpath configuration. This is the same way you declare, for example, the Java compilation classpath. You can use any of the [dependency types](http://docs.google.com/dependency_types.html#dependency_types) except project dependencies.

Having declared the build script classpath, you can use the classes in your build script as you would any other classes on the classpath. The following example adds to the previous example, and uses classes from the build script classpath.

[Example: A build script with external dependencies](#1v1yuxt)

**build.gradle**

import org.apache.commons.codec.binary.Base64  
  
buildscript {  
 repositories {  
 mavenCentral()  
 }  
 dependencies {  
 classpath group: 'commons-codec', name: 'commons-codec', version: '1.2'  
 }  
}  
  
task encode {  
 doLast {  
 def byte[] encodedString = new Base64().encode('hello world\n'.getBytes())  
 println new String(encodedString)  
 }  
}

**Output of** gradle -q encode

> gradle -q encode  
aGVsbG8gd29ybGQK

For multi-project builds, the dependencies declared with a project’s buildscript() method are available to the build scripts of all its sub-projects.

Build script dependencies may be Gradle plugins. Please consult [Using Gradle Plugins](http://docs.google.com/plugins.html#plugins) for more information on Gradle plugins.

Every project automatically has a buildEnvironment task of type [BuildEnvironmentReportTask](http://docs.google.com/dsl/org.gradle.api.tasks.diagnostics.BuildEnvironmentReportTask.html) that can be invoked to report on the resolution of the build script dependencies.

[Where to next?](#2xcytpi)

In this chapter, we have had a first look at tasks. But this is not the end of the story for tasks. If you want to jump into more of the details, have a look at [More About Tasks](http://docs.google.com/more_about_tasks.html#more_about_tasks).

Otherwise, continue on to [the tutorials](http://docs.google.com/tutorial_java_projects.html#tutorial_java_projects) and [Dependency Management for Java Projects](http://docs.google.com/dependency_management_for_java_projects.html#dependency_management_for_java_projects).

[1](#4f1mdlm). There are command line switches to change this behavior. See [Command-Line Interface](http://docs.google.com/command_line_interface.html#command_line_interface))

Docs

* [User Manual](http://docs.google.com/userguide/userguide.html)
* [DSL Reference](http://docs.google.com/dsl/)
* [Release Notes](http://docs.google.com/release-notes.html)
* [Javadoc](http://docs.google.com/javadoc/)

News

* [Blog](https://blog.gradle.org/)
* [Newsletter](https://newsletter.gradle.com/)
* [Twitter](https://twitter.com/gradle)

Products

* [Build Scans](https://gradle.com/build-scans)
* [Build Cache](https://gradle.com/build-cache)
* [Enterprise Docs](https://gradle.com/enterprise/resources)

Get Help

* [Forums](https://discuss.gradle.org/c/help-discuss)
* [GitHub](https://github.com/gradle/)
* [Training](https://gradle.org/training/)
* [Services](https://gradle.org/services/)

Subscribe for important Gradle updates and news

Subscribe

By entering your email, you agree to our [Terms](https://gradle.org/terms/) and [Privacy Policy](https://gradle.org/privacy/), including receipt of emails. You can unsubscribe at any time.

© [Gradle Inc.](https://gradle.com) 2018 All rights reserved.

[Careers](https://gradle.com/careers) | [Privacy](https://gradle.org/privacy) | [Terms of Service](https://gradle.org/terms) | [Contact](https://gradle.org/contact/)