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/)

# Command-Line Interface

Contents

[Executing tasks](#4d34og8)

[Common tasks](#2s8eyo1)

[Project reporting](#17dp8vu)

[Command-line completion](#3rdcrjn)

[Debugging options](#26in1rg)

[Performance options](#lnxbz9)

[Logging options](#35nkun2)

[Execution options](#1ksv4uv)

[Environment options](#44sinio)

[Bootstrapping new projects](#2jxsxqh)

[Continuous Build](#z337ya)

The command-line interface is one of the primary methods of interacting with Gradle. The following serves as a reference of executing and customizing Gradle use of a command-line or when writing scripts or configuring continuous integration.

Use of the [Gradle Wrapper](http://docs.google.com/gradle_wrapper.html#gradle_wrapper) is highly encouraged. You should substitute ./gradlew or gradlew.bat for gradle in all following examples when using the Wrapper.

Executing Gradle on the command-line conforms to the following structure. Options are allowed before and after task names.

gradle [taskName...] [--option-name...]

If multiple tasks are specified, they should be separated with a space.

Options that accept values can be specified with or without = between the option and argument; however, use of = is recommended.

--console=plain

Options that enable behavior have long-form options with inverses specified with --no-. The following are opposites.

--build-cache  
--no-build-cache

Many long-form options, have short option equivalents. The following are equivalent:

--help  
-h

| **✨** | Many command-line flags can be specified in gradle.properties to avoid needing to be typed. See the [configuring build environment guide](http://docs.google.com/build_environment.html#sec:gradle_configuration_properties) for details. |
| --- | --- |

The following sections describe use of the Gradle command-line interface, grouped roughly by user goal. Some plugins also add their own command line options, for example [--tests for Java test filtering](http://docs.google.com/java_testing.html#test_filtering). For more information on exposing command line options for your own tasks, see [Declaring and using command-line options](http://docs.google.com/custom_tasks.html#sec:declaring_and_using_command_line_options).

[Executing tasks](#4d34og8)

You can run a task and all of its [dependencies](http://docs.google.com/tutorial_using_tasks.html#sec:task_dependencies).

❯ gradle myTask

You can learn about what projects and tasks are available in the [project reporting section](#17dp8vu).

[Executing tasks in multi-project builds](#3j2qqm3)

In a [multi-project build](http://docs.google.com/intro_multi_project_builds.html#intro_multi_project_builds), subproject tasks can be executed with ":" separating subproject name and task name. The following are equivalent *when run from the root project*.

❯ gradle :mySubproject:taskName  
❯ gradle mySubproject:taskName

You can also run a task for all subprojects using the task name only. For example, this will run the "test" task for all subprojects when invoked from the root project directory.

❯ gradle test

When invoking Gradle from within a subproject, the project name should be omitted:

❯ cd mySubproject  
❯ gradle taskName

| **✨** | When executing the Gradle Wrapper from subprojects, one must reference gradlew relatively. For example: ../gradlew taskName. The community [gdub project](http://www.gdub.rocks/) aims to make this more convenient. |
| --- | --- |

[Executing multiple tasks](#1y810tw)

You can also specify multiple tasks. For example, the following will execute the test and deploy tasks in the order that they are listed on the command-line and will also execute the dependencies for each task.

❯ gradle test deploy

[Excluding tasks from execution](#4i7ojhp)

You can exclude a task from being executed using the -x or --exclude-task command-line option and providing the name of the task to exclude.



*Figure 1. Example Task Graph*

[Example: Excluding tasks](#2xcytpi)

**Output of** gradle dist --exclude-task test

> gradle dist --exclude-task test  
  
> Task :compile  
compiling source  
  
> Task :dist  
building the distribution  
  
BUILD SUCCESSFUL in 0s  
2 actionable tasks: 2 executed

You can see that the test task is not executed, even though it is a dependency of the dist task. The test task’s dependencies such as compileTest are not executed either. Those dependencies of test that are required by another task, such as compile, are still executed.

[Forcing tasks to execute](#1ci93xb)

You can force Gradle to execute all tasks ignoring [up-to-date checks](http://docs.google.com/more_about_tasks.html#sec:up_to_date_checks) using the --rerun-tasks option:

❯ gradle test --rerun-tasks

This will force test and *all* task dependencies of test to execute. It’s a little like running gradle clean test, but without the build’s generated output being deleted.

[Continuing the build when a failure occurs](#3whwml4)

By default, Gradle will abort execution and fail the build as soon as any task fails. This allows the build to complete sooner, but hides other failures that would have occurred. In order to discover as many failures as possible in a single build execution, you can use the --continue option.

❯ gradle test --continue

When executed with --continue, Gradle will execute *every* task to be executed where all of the dependencies for that task completed without failure, instead of stopping as soon as the first failure is encountered. Each of the encountered failures will be reported at the end of the build.

If a task fails, any subsequent tasks that were depending on it will not be executed. For example, tests will not run if there is a compilation failure in the code under test; because the test task will depend on the compilation task (either directly or indirectly).

[Task name abbreviation](#2bn6wsx)

When you specify tasks on the command-line, you don’t have to provide the full name of the task. You only need to provide enough of the task name to uniquely identify the task. For example, it’s likely gradle che is enough for Gradle to identify the check task.

You can also abbreviate each word in a camel case task name. For example, you can execute task compileTest by running gradle compTest or even gradle cT.

[Example: Abbreviated camel case task name](#qsh70q)

**Output of** gradle cT

> gradle cT  
  
> Task :compile  
compiling source  
  
> Task :compileTest  
compiling unit tests  
  
BUILD SUCCESSFUL in 0s  
2 actionable tasks: 2 executed

You can also use these abbreviations with the -x command-line option.

[Common tasks](#2s8eyo1)

The following are task conventions applied by built-in and most major Gradle plugins.

[Computing all outputs](#3as4poj)

It is common in Gradle builds for the build task to designate assembling all outputs and running all checks.

❯ gradle build

[Running applications](#1pxezwc)

It is common for applications to be run with the run task, which assembles the application and executes some script or binary.

❯ gradle run

[Running all checks](#49x2ik5)

It is common for *all* verification tasks, including tests and linting, to be executed using the check task.

❯ gradle check

[Cleaning outputs](#2p2csry)

You can delete the contents of the build directory using the clean task, though doing so will cause pre-computed outputs to be lost, causing significant additional build time for the subsequent task execution.

❯ gradle clean

[Project reporting](#17dp8vu)

Gradle provides several built-in tasks which show particular details of your build. This can be useful for understanding the structure and dependencies of your build, and for debugging problems.

You can get basic help about available reporting options using gradle help.

[Listing projects](#147n2zr)

Running gradle projects gives you a list of the sub-projects of the selected project, displayed in a hierarchy.

❯ gradle projects

You also get a project report within build scans. Learn more about [creating build scans](https://guides.gradle.org/creating-build-scans/).

[Listing tasks](#3o7alnk)

Running gradle tasks gives you a list of the main tasks of the selected project. This report shows the default tasks for the project, if any, and a description for each task.

❯ gradle tasks

By default, this report shows only those tasks which have been assigned to a task group. You can obtain more information in the task listing using the --all option.

❯ gradle tasks --all

[Show task usage details](#23ckvvd)

Running gradle help --task someTask gives you detailed information about a specific task.

[Example: Obtaining detailed help for tasks](#ihv636)

**Output of** gradle -q help --task libs

> gradle -q help --task libs  
Detailed task information for libs  
  
Paths  
 :api:libs  
 :webapp:libs  
  
Type  
 Task (org.gradle.api.Task)  
  
Description  
 Builds the JAR  
  
Group  
 build

This information includes the full task path, the task type, possible command line options and the description of the given task.

[Reporting dependencies](#32hioqz)

Build scans give a full, visual report of what dependencies exist on which configurations, transitive dependencies, and dependency version selection.

❯ gradle myTask --scan

This will give you a link to a web-based report, where you can find dependency information like this.



Learn more in [Inspecting Dependencies](http://docs.google.com/inspecting_dependencies.html#inspecting_dependencies).

[Listing project dependencies](#1hmsyys)

Running gradle dependencies gives you a list of the dependencies of the selected project, broken down by configuration. For each configuration, the direct and transitive dependencies of that configuration are shown in a tree. Below is an example of this report:

❯ gradle dependencies

Concrete examples of build scripts and output available in the [Inspecting Dependencies](http://docs.google.com/inspecting_dependencies.html#inspecting_dependencies).

Running gradle buildEnvironment visualises the buildscript dependencies of the selected project, similarly to how gradle dependencies visualizes the dependencies of the software being built.

❯ gradle buildEnvironment

Running gradle dependencyInsight gives you an insight into a particular dependency (or dependencies) that match specified input.

❯ gradle dependencyInsight

Since a dependency report can get large, it can be useful to restrict the report to a particular configuration. This is achieved with the optional --configuration parameter:

[Listing project properties](#41mghml)

Running gradle properties gives you a list of the properties of the selected project.

[Example: Information about properties](#2grqrue)

**Output of gradle -q api:properties**

> gradle -q api:properties  
  
------------------------------------------------------------  
Project :api - The shared API for the application  
------------------------------------------------------------  
  
allprojects: [project ':api']  
ant: org.gradle.api.internal.project.DefaultAntBuilder@12345  
antBuilderFactory: org.gradle.api.internal.project.DefaultAntBuilderFactory@12345  
artifacts: org.gradle.api.internal.artifacts.dsl.DefaultArtifactHandler\_Decorated@12345  
asDynamicObject: DynamicObject for project ':api'  
baseClassLoaderScope: org.gradle.api.internal.initialization.DefaultClassLoaderScope@12345

[Software Model reports](#vx1227)

You can get a hierarchical view of elements for [software model](http://docs.google.com/software_model.html#software_model) projects using the model task:

❯ gradle model

Learn more about [the model report](http://docs.google.com/software_model.html#model-report) in the software model documentation.

[Command-line completion](#3rdcrjn)

Gradle provides bash and zsh tab completion support for tasks, options, and Gradle properties through [gradle-completion](https://github.com/gradle/gradle-completion), installed separately.

[Debugging options](#26in1rg)

-?, -h, --help

Shows a help message with all available CLI options.

-v, --version

Prints Gradle, Groovy, Ant, JVM, and operating system version information.

-S, --full-stacktrace

Print out the full (very verbose) stacktrace for any exceptions. See also [logging options](#35nkun2).

-s, --stacktrace

Print out the stacktrace also for user exceptions (e.g. compile error). See also [logging options](#35nkun2).

--scan

Create a [build scan](https://gradle.com/build-scans) with fine-grained information about all aspects of your Gradle build.

-Dorg.gradle.debug=true

Debug Gradle client (non-Daemon) process. Gradle will wait for you to attach a debugger at localhost:5005 by default.

-Dorg.gradle.daemon.debug=true

Debug [Gradle Daemon](http://docs.google.com/gradle_daemon.html#gradle_daemon) process.

[Performance options](#lnxbz9)

Try these options when optimizing build performance. Learn more about [improving performance of Gradle builds here](https://guides.gradle.org/performance/).

Many of these options can be specified in gradle.properties so command-line flags are not necessary. See the [configuring build environment guide](http://docs.google.com/build_environment.html#sec:gradle_configuration_properties).

--build-cache, --no-build-cache

Toggles the [Gradle build cache](http://docs.google.com/build_cache.html#build_cache). Gradle will try to reuse outputs from previous builds. *Default is off*.

--configure-on-demand, --no-configure-on-demand

Toggles [Configure-on-demand](http://docs.google.com/multi_project_builds.html#sec:configuration_on_demand). Only relevant projects are configured in this build run. *Default is off*.

--max-workers

Sets maximum number of workers that Gradle may use. *Default is number of processors*.

--parallel, --no-parallel

Build projects in parallel. For limitations of this option, see [Parallel Project Execution](http://docs.google.com/multi_project_builds.html#sec:parallel_execution). *Default is off*.

--profile

Generates a high-level performance report in the $buildDir/reports/profile directory. --scan is preferred.

--scan

Generate a build scan with detailed performance diagnostics.



[Gradle daemon options](#3fwokq0)

You can manage the [Gradle Daemon](http://docs.google.com/gradle_daemon.html#gradle_daemon) through the following command line options.

--daemon, --no-daemon

Use the [Gradle Daemon](http://docs.google.com/gradle_daemon.html#gradle_daemon) to run the build. Starts the daemon if not running or existing daemon busy. *Default is on*.

--foreground

Starts the Gradle Daemon in a foreground process.

--status (Standalone command)

Run gradle --status to list running and recently stopped Gradle daemons. Only displays daemons of the same Gradle version.

--stop (Standalone command)

Run gradle --stop to stop all Gradle Daemons of the same version.

-Dorg.gradle.daemon.idletimeout=(number of milliseconds)

Gradle Daemon will stop itself after this number of milliseconds of idle time. *Default is 10800000* (3 hours).

[Logging options](#35nkun2)

[Setting log level](#1v1yuxt)

You can customize the verbosity of Gradle logging with the following options, ordered from least verbose to most verbose. Learn more in the [logging documentation](http://docs.google.com/logging.html#logging).

-Dorg.gradle.logging.level=(quiet,warn,lifecycle,info,debug)

Set logging level via Gradle properties.

-q, --quiet

Log errors only.

-w, --warn

Set log level to warn.

-i, --info

Set log level to info.

-d, --debug

Log in debug mode (includes normal stacktrace).

Lifecycle is the default log level.

[Customizing log format](#4f1mdlm)

You can control the use of rich output (colors and font variants) by specifying the "console" mode in the following ways:

-Dorg.gradle.console=(auto,plain,rich,verbose)

Specify console mode via Gradle properties. Different modes described immediately below.

--console=(auto,plain,rich,verbose)

Specifies which type of console output to generate.

Set to plain to generate plain text only. This option disables all color and other rich output in the console output. This is the default when Gradle is *not* attached to a terminal.

Set to auto (the default) to enable color and other rich output in the console output when the build process is attached to a console, or to generate plain text only when not attached to a console. *This is the default when Gradle is attached to a terminal.*

Set to rich to enable color and other rich output in the console output, regardless of whether the build process is not attached to a console. When not attached to a console, the build output will use ANSI control characters to generate the rich output.

Set to verbose to enable color and other rich output like the rich, but output task names and outcomes at the lifecycle log level, as is done by default in Gradle 3.5 and earlier.

[Showing or hiding warnings](#2u6wntf)

By default, Gradle won’t display all warnings (e.g. deprecation warnings). Instead, Gradle will collect them and render a summary at the end of the build like:

Deprecated Gradle features were used in this build, making it incompatible with Gradle 5.0.

You can control the verbosity of warnings on the console with the following options:

-Dorg.gradle.warning.mode=(all,none,summary)

Specify warning mode via [Gradle properties](http://docs.google.com/build_environment.html#sec:gradle_configuration_properties). Different modes described immediately below.

--warning-mode=(all,none,summary)

Specifies how to log warnings. Default is summary.

Set to all to log all warnings.

Set to summary to suppress all warnings and log a summary at the end of the build.

Set to none to suppress all warnings, including the summary at the end of the build.

[Rich Console](#19c6y18)

Gradle’s rich console displays extra information while builds are running.



Features:

* Progress bar and timer visually describe overall status
* Parallel work-in-progress lines below describe what is happening now
* Colors and fonts are used to highlight important output and errors

[Execution options](#1ksv4uv)

The following options affect how builds are executed, by changing what is built or how dependencies are resolved.

--include-build

Run the build as a composite, including the specified build. See [Composite Builds](http://docs.google.com/composite_builds.html#composite_builds).

--offline

Specifies that the build should operate without accessing network resources. Learn more about [options to override dependency caching](http://docs.google.com/troubleshooting_dependency_resolution.html#sec:controlling_dependency_caching_command_line).

--refresh-dependencies

Refresh the state of dependencies. Learn more about how to use this in the [dependency management docs](http://docs.google.com/troubleshooting_dependency_resolution.html#sec:controlling_dependency_caching_command_line).

--dry-run

Run Gradle with all task actions disabled. Use this to show which task would have executed.

--write-locks

Indicates that all resolved configurations that are *lockable* should have their lock state persisted. Learn more about this in [dependency locking](http://docs.google.com/dependency_locking.html#dependency_locking).

--update-locks <group:name>[,<group:name>]\*

Indicates that versions for the specified modules have to be updated in the lock file. This flag also implies --write-locks. Learn more about this in [dependency locking](http://docs.google.com/dependency_locking.html#dependency_locking).

[Environment options](#44sinio)

You can customize many aspects about where build scripts, settings, caches, and so on through the options below. Learn more about customizing your [build environment](http://docs.google.com/build_environment.html#build_environment).

-b, --build-file

Specifies the build file. For example: gradle --build-file=foo.gradle. The default is build.gradle, then build.gradle.kts, then myProjectName.gradle.

-c, --settings-file

Specifies the settings file. For example: gradle --settings-file=somewhere/else/settings.gradle

-g, --gradle-user-home

Specifies the Gradle user home directory. The default is the .gradle directory in the user’s home directory.

-p, --project-dir

Specifies the start directory for Gradle. Defaults to current directory.

--project-cache-dir

Specifies the project-specific cache directory. Default value is .gradle in the root project directory.

-u, --no-search-upward (deprecated)

Don’t search in parent directories for a settings.gradle file.

-D, --system-prop

Sets a system property of the JVM, for example -Dmyprop=myvalue. See [System Properties](http://docs.google.com/build_environment.html#sec:gradle_system_properties).

-I, --init-script

Specifies an initialization script. See [Init Scripts](http://docs.google.com/init_scripts.html#init_scripts).

-P, --project-prop

Sets a project property of the root project, for example -Pmyprop=myvalue. See [System Properties](http://docs.google.com/build_environment.html#sec:project_properties).

-Dorg.gradle.jvmargs

Set JVM arguments.

-Dorg.gradle.java.home

Set JDK home dir.

[Bootstrapping new projects](#2jxsxqh)

[Creating new Gradle builds](#3tbugp1)

Use the built-in gradle init task to create a new Gradle builds, with new or existing projects.

❯ gradle init

Most of the time you’ll want to specify a project type. Available types include basic (default), java-library, java-application, and more. See [init plugin documentation](http://docs.google.com/build_init_plugin.html#build_init_plugin) for details.

❯ gradle init --type java-library

[Standardize and provision Gradle](#28h4qwu)

The built-in gradle wrapper task generates a script, gradlew, that invokes a declared version of Gradle, downloading it beforehand if necessary.

❯ gradle wrapper --gradle-version=4.4

You can also specify --distribution-type=(bin|all), --gradle-distribution-url, --gradle-distribution-sha256-sum in addition to --gradle-version. Full details on how to use these options are documented in the [Gradle wrapper section](http://docs.google.com/gradle_wrapper.html#gradle_wrapper).

[Continuous Build](#z337ya)

Continuous Build allows you to automatically re-execute the requested tasks when task inputs change.

For example, you can continuously run the test task and all dependent tasks by running:

❯ gradle test --continuous

Gradle will behave as if you ran gradle test after a change to sources or tests that contribute to the requested tasks. This means that unrelated changes (such as changes to build scripts) will not trigger a rebuild. In order to incorporate build logic changes, the continuous build must be restarted manually.

[Terminating Continuous Build](#nmf14n)

If Gradle is attached to an interactive input source, such as a terminal, the continuous build can be exited by pressing CTRL-D (On Microsoft Windows, it is required to also press ENTER or RETURN after CTRL-D). If Gradle is not attached to an interactive input source (e.g. is running as part of a script), the build process must be terminated (e.g. using the kill command or similar). If the build is being executed via the Tooling API, the build can be cancelled using the Tooling API’s cancellation mechanism.

[Limitations and quirks](#37m2jsg)

| **✨** | Continuous build is an [incubating](http://docs.google.com/feature_lifecycle.html#feature_lifecycle) feature. |
| --- | --- |

There are several issues to be aware with the current implementation of continuous build. These are likely to be addressed in future Gradle releases.

[Build cycles](#1mrcu09)

Gradle starts watching for changes just before a task executes. If a task modifies its own inputs while executing, Gradle will detect the change and trigger a new build. If every time the task executes, the inputs are modified again, the build will be triggered again. This isn’t unique to continuous build. A task that modifies its own inputs will never be considered up-to-date when run "normally" without continuous build.

If your build enters a build cycle like this, you can track down the task by looking at the list of files reported changed by Gradle. After identifying the file(s) that are changed during each build, you should look for a task that has that file as an input. In some cases, it may be obvious (e.g., a Java file is compiled with compileJava). In other cases, you can use --info logging to find the task that is out-of-date due to the identified files.

[Restrictions with Java 9](#46r0co2)

Due to class access restrictions related to Java 9, Gradle cannot set some operating system specific options, which means that:

* On macOS, Gradle will poll for file changes every 10 seconds instead of every 2 seconds.
* On Windows, Gradle must use individual file watches (like on Linux/Mac OS), which may cause continuous build to no longer work on very large projects.

[Performance and stability](#2lwamvv)

The JDK file watching facility relies on inefficient file system polling on macOS (see: [JDK-7133447](https://bugs.openjdk.java.net/browse/JDK-7133447)). This can significantly delay notification of changes on large projects with many source files.

Additionally, the watching mechanism may deadlock under *heavy* load on macOS (see: [JDK-8079620](https://bugs.openjdk.java.net/browse/JDK-8079620)). This will manifest as Gradle appearing not to notice file changes. If you suspect this is occurring, exit continuous build and start again.

On Linux, OpenJDK’s implementation of the file watch service can sometimes miss file system events (see: [JDK-8145981](https://bugs.openjdk.java.net/browse/JDK-8145981)).

[Changes to symbolic links](#111kx3o)

* Creating or removing symbolic link to files will initiate a build.
* Modifying the target of a symbolic link will not cause a rebuild.
* Creating or removing symbolic links to directories will not cause rebuilds.
* Creating new files in the target directory of a symbolic link will not cause a rebuild.
* Deleting the target directory will not cause a rebuild.

[Changes to build logic are not considered](#3l18frh)

The current implementation does not recalculate the build model on subsequent builds. This means that changes to task configuration, or any other change to the build model, are effectively ignored.

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/)