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/)

# Repository Types

Contents

[Flat directory repository](#4d34og8)

[Maven Central repository](#2s8eyo1)

[JCenter Maven repository](#17dp8vu)

[Google Maven repository](#3rdcrjn)

[Local Maven repository](#26in1rg)

[Custom Maven repositories](#lnxbz9)

[Custom Ivy repositories](#35nkun2)

[Supported metadata sources](#1ksv4uv)

[Supported repository transport protocols](#44sinio)

[Flat directory repository](#4d34og8)

Some projects might prefer to store dependencies on a shared drive or as part of the project source code instead of a binary repository product. If you want to use a (flat) filesystem directory as a repository, simply type:

[Example: Flat repository resolver](#2jxsxqh)

**build.gradle**

repositories {  
 flatDir {  
 dirs 'lib'  
 }  
 flatDir {  
 dirs 'lib1', 'lib2'  
 }  
}

This adds repositories which look into one or more directories for finding dependencies. Note that this type of repository does not support any meta-data formats like Ivy XML or Maven POM files. Instead, Gradle will dynamically generate a module descriptor (without any dependency information) based on the presence of artifacts. However, as Gradle prefers to use modules whose descriptor has been created from real meta-data rather than being generated, flat directory repositories cannot be used to override artifacts with real meta-data from other repositories. For example, if Gradle finds only jmxri-1.2.1.jar in a flat directory repository, but jmxri-1.2.1.pom in another repository that supports meta-data, it will use the second repository to provide the module.

For the use case of overriding remote artifacts with local ones consider using an Ivy or Maven repository instead whose URL points to a local directory. If you only work with flat directory repositories you don’t need to set all attributes of a dependency.

[Maven Central repository](#2s8eyo1)

Maven Central is a popular repository hosting open source libraries for consumption by Java projects.

To declare the [central Maven repository](https://repo.maven.apache.org/maven2/) for your build add this to your script:

[Example: Adding central Maven repository](#z337ya)

**build.gradle**

repositories {  
 mavenCentral()  
}

[JCenter Maven repository](#17dp8vu)

[Bintray](http://bintray.com)'s JCenter is an up-to-date collection of all popular Maven OSS artifacts, including artifacts published directly to Bintray.

To declare the [JCenter Maven repository](https://jcenter.bintray.com) add this to your build script:

[Example: Adding Bintray’s JCenter Maven repository](#3j2qqm3)

**build.gradle**

repositories {  
 jcenter()  
}

[Google Maven repository](#3rdcrjn)

The Google repository hosts Android-specific artifacts including the Android SDK. For usage examples, see the [relevant documentation](https://developer.android.com/studio/build/dependencies.html#google-maven).

To declare the [Google Maven repository](https://maven.google.com/) add this to your build script:

[Example: Adding Google Maven repository](#1y810tw)

**build.gradle**

repositories {  
 google()  
}

[Local Maven repository](#26in1rg)

Gradle can consume dependencies available in the [local Maven repository](https://maven.apache.org/guides/introduction/introduction-to-repositories.html). Declaring this repository is beneficial for teams that publish to the local Maven repository with one project and consume the artifacts by Gradle in another project.

| **✨** | Gradle stores resolved dependencies in [its own cache](http://docs.google.com/dependency_cache.html#dependency_cache). A build does not need to declare the local Maven repository even if you resolve dependencies from a Maven-based, remote repository. |
| --- | --- |

To declare the local Maven cache as a repository add this to your build script:

[Example: Adding the local Maven cache as a repository](#4i7ojhp)

**build.gradle**

repositories {  
 mavenLocal()  
}

Gradle uses the same logic as Maven to identify the location of your local Maven cache. If a local repository location is defined in a settings.xml, this location will be used. The settings.xml in *USER\_HOME*/.m2 takes precedence over the settings.xml in *M2\_HOME*/conf. If no settings.xml is available, Gradle uses the default location *USER\_HOME*/.m2/repository.

[Custom Maven repositories](#lnxbz9)

Many organizations host dependencies in an in-house Maven repository only accessible within the company’s network. Gradle can declare Maven repositories by URL.

For adding a custom Maven repository you can do:

[Example: Adding custom Maven repository](#2xcytpi)

**build.gradle**

repositories {  
 maven {  
 url "http://repo.mycompany.com/maven2"  
 }  
}

Sometimes a repository will have the POMs published to one location, and the JARs and other artifacts published at another location. To define such a repository, you can do:

[Example: Adding additional Maven repositories for JAR files](#1ci93xb)

**build.gradle**

repositories {  
 maven {  
 // Look for POMs and artifacts, such as JARs, here  
 url "http://repo2.mycompany.com/maven2"  
 // Look for artifacts here if not found at the above location  
 artifactUrls "http://repo.mycompany.com/jars"  
 artifactUrls "http://repo.mycompany.com/jars2"  
 }  
}

Gradle will look at the first URL for the POM and the JAR. If the JAR can’t be found there, the artifact URLs are used to look for JARs.

See [Configuring HTTP authentication schemes](#3whwml4) for authentication options.

[Custom Ivy repositories](#35nkun2)

Organizations might decide to host dependencies in an in-house Ivy repository. Gradle can declare Ivy repositories by URL.

[Defining an Ivy repository with a standard layout](#2bn6wsx)

To declare an Ivy repository using the standard layout no additional customization is needed. You just declare the URL.

[Example: Ivy repository](#qsh70q)

**build.gradle**

repositories {  
 ivy {  
 url "http://repo.mycompany.com/repo"  
 }  
}

[Defining a named layout for an Ivy repository](#3as4poj)

You can specify that your repository conforms to the Ivy or Maven default layout by using a named layout.

[Example: Ivy repository with named layout](#1pxezwc)

**build.gradle**

repositories {  
 ivy {  
 url "http://repo.mycompany.com/repo"  
 layout "maven"  
 }  
}

Valid named layout values are 'gradle' (the default), 'maven', 'ivy' and 'pattern'. See [IvyArtifactRepository.layout(java.lang.String, groovy.lang.Closure)](http://docs.google.com/dsl/org.gradle.api.artifacts.repositories.IvyArtifactRepository.html#org.gradle.api.artifacts.repositories.IvyArtifactRepository:layout(java.lang.String,%20groovy.lang.Closure)) in the API documentation for details of these named layouts.

[Defining custom pattern layout for an Ivy repository](#49x2ik5)

To define an Ivy repository with a non-standard layout, you can define a 'pattern' layout for the repository:

[Example: Ivy repository with pattern layout](#2p2csry)

**build.gradle**

repositories {  
 ivy {  
 url "http://repo.mycompany.com/repo"  
 layout "pattern", {  
 artifact "[module]/[revision]/[type]/[artifact].[ext]"  
 }  
 }  
}

To define an Ivy repository which fetches Ivy files and artifacts from different locations, you can define separate patterns to use to locate the Ivy files and artifacts:

Each artifact or ivy specified for a repository adds an *additional* pattern to use. The patterns are used in the order that they are defined.

[Example: Ivy repository with multiple custom patterns](#147n2zr)

**build.gradle**

repositories {  
 ivy {  
 url "http://repo.mycompany.com/repo"  
 layout "pattern", {  
 artifact "3rd-party-artifacts/[organisation]/[module]/[revision]/[artifact]-[revision].[ext]"  
 artifact "company-artifacts/[organisation]/[module]/[revision]/[artifact]-[revision].[ext]"  
 ivy "ivy-files/[organisation]/[module]/[revision]/ivy.xml"  
 }  
 }  
}

Optionally, a repository with pattern layout can have its 'organisation' part laid out in Maven style, with forward slashes replacing dots as separators. For example, the organisation my.company would then be represented as my/company.

[Example: Ivy repository with Maven compatible layout](#3o7alnk)

**build.gradle**

repositories {  
 ivy {  
 url "http://repo.mycompany.com/repo"  
 layout "pattern", {  
 artifact "[organisation]/[module]/[revision]/[artifact]-[revision].[ext]"  
 m2compatible = true  
 }  
 }  
}

[Accessing password-protected Ivy repositories](#23ckvvd)

You can specify credentials for Ivy repositories secured by basic authentication.

[Example: Ivy repository with authentication](#ihv636)

**build.gradle**

repositories {  
 ivy {  
 url "http://repo.mycompany.com"  
 credentials {  
 username "user"  
 password "password"  
 }  
 }  
}

[Supported metadata sources](#1ksv4uv)

When searching for a module in a repository, Gradle, by default, checks for supported metadata file formats in that repository. In a Maven repository, Gradle looks for a .pom file, in an ivy repository it looks for an ivy.xml file and in a flat directory repository it looks directly for .jar files as it does not expect any metadata. Starting with 5.0, Gradle also looks for .module (Gradle module metadata) files.

However, if you define a customized repository you might want to configure this behavior. For example, you can define a Maven repository without .pom files but only jars. To do so, you can configure *metadata sources* for any repository.

[Example: Maven repository that supports artifacts without metadata](#32hioqz)

**build.gradle**

repositories {  
 maven {  
 url "http://repo.mycompany.com/repo"  
 metadataSources {  
 mavenPom()  
 artifact()  
 }  
 }  
}

You can specify multiple sources to tell Gradle to keep looking if a file was not found. In that case, the order of checking for sources is predefined.

The following metadata sources are supported:

Table 1. Repository transport protocols

| **Metadata source** | **Description** | **Order** | **Maven** | **Ivy / flat dir** |
| --- | --- | --- | --- | --- |
| gradleMetadata() | Look for Gradle .module files | 1st | yes | yes |
| mavenPom() | Look for Maven .pom files | 2nd | yes | yes |
| ivyDescriptor() | Look for ivy.xml files | 2nd | no | yes |
| artifact() | Look directly for artifact | 3rd | yes | yes |

| **✨** | The defaults for Ivy and Maven repositories change with Gradle 5.0. Before 5.0, artifact() was included in the defaults. Leading to some inefficiency when modules are missing completely. To restore this behavior, for example, for Maven central you can use mavenCentral { mavenPom(); artifact() }. In a similar way, you can opt into the new behavior in older Gradle verisions using mavenCentral { mavenPom() } |
| --- | --- |

[Supported repository transport protocols](#44sinio)

Maven and Ivy repositories support the use of various transport protocols. At the moment the following protocols are supported:

Table 2. Repository transport protocols

| **Type** | **Credential types** |
| --- | --- |
| file | none |
| http | username/password |
| https | username/password |
| sftp | username/password |
| s3 | access key/secret key/session token or Environment variables |
| gcs | [default application credentials](https://developers.google.com/identity/protocols/application-default-credentials) sourced from well known files, Environment variables etc. |

| **✨** | Username and password should never be checked in plain text into version control as part of your build file. You can store the credentials in a local gradle.properties file and use one of the open source Gradle plugins for encrypting and consuming credentials e.g. the [credentials plugin](https://plugins.gradle.org/plugin/nu.studer.credentials). |
| --- | --- |

The transport protocol is part of the URL definition for a repository. The following build script demonstrates how to create a HTTP-based Maven and Ivy repository:

[Example: Declaring a Maven and Ivy repository](#1hmsyys)

**build.gradle**

repositories {  
 maven {  
 url "http://repo.mycompany.com/maven2"  
 }  
  
 ivy {  
 url "http://repo.mycompany.com/repo"  
 }  
}

The following example shows how to declare SFTP repositories:

[Example: Using the SFTP protocol for a repository](#41mghml)

**build.gradle**

repositories {  
 maven {  
 url "sftp://repo.mycompany.com:22/maven2"  
 credentials {  
 username "user"  
 password "password"  
 }  
 }  
  
 ivy {  
 url "sftp://repo.mycompany.com:22/repo"  
 credentials {  
 username "user"  
 password "password"  
 }  
 }  
}

When using an AWS S3 backed repository you need to authenticate using [AwsCredentials](http://docs.google.com/dsl/org.gradle.api.credentials.AwsCredentials.html), providing access-key and a private-key. The following example shows how to declare a S3 backed repository and providing AWS credentials:

[Example: Declaring a S3 backed Maven and Ivy repository](#2grqrue)

**build.gradle**

repositories {  
 maven {  
 url "s3://myCompanyBucket/maven2"  
 credentials(AwsCredentials) {  
 accessKey "someKey"  
 secretKey "someSecret"  
 // optional  
 sessionToken "someSTSToken"  
 }  
 }  
  
 ivy {  
 url "s3://myCompanyBucket/ivyrepo"  
 credentials(AwsCredentials) {  
 accessKey "someKey"  
 secretKey "someSecret"  
 // optional  
 sessionToken "someSTSToken"  
 }  
 }  
}

You can also delegate all credentials to the AWS sdk by using the AwsImAuthentication. The following example shows how:

[Example: Declaring a S3 backed Maven and Ivy repository using IAM](#vx1227)

**build.gradle**

repositories {  
 maven {  
 url "s3://myCompanyBucket/maven2"  
 authentication {  
 awsIm(AwsImAuthentication) // load from EC2 role or env var  
 }  
 }  
  
 ivy {  
 url "s3://myCompanyBucket/ivyrepo"  
 authentication {  
 awsIm(AwsImAuthentication)  
 }  
 }  
}

When using a Google Cloud Storage backed repository default application credentials will be used with no further configuration required:

[Example: Declaring a Google Cloud Storage backed Maven and Ivy repository using default application credentials](#3fwokq0)

**build.gradle**

repositories {  
 maven {  
 url "gcs://myCompanyBucket/maven2"  
 }  
  
 ivy {  
 url "gcs://myCompanyBucket/ivyrepo"  
 }  
}

[S3 configuration properties](#1v1yuxt)

The following system properties can be used to configure the interactions with s3 repositories:

org.gradle.s3.endpoint

Used to override the AWS S3 endpoint when using a non AWS, S3 API compatible, storage service.

org.gradle.s3.maxErrorRetry

Specifies the maximum number of times to retry a request in the event that the S3 server responds with a HTTP 5xx status code. When not specified a default value of 3 is used.

[S3 URL formats](#4f1mdlm)

S3 URL’s are 'virtual-hosted-style' and must be in the following format

s3://<bucketName>[.<regionSpecificEndpoint>]/<s3Key>

e.g. s3://myBucket.s3.eu-central-1.amazonaws.com/maven/release

* myBucket is the AWS S3 bucket name.
* s3.eu-central-1.amazonaws.com is the *optional* [region specific endpoint](http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region).
* /maven/release is the AWS S3 key (unique identifier for an object within a bucket)

[S3 proxy settings](#2u6wntf)

A proxy for S3 can be configured using the following system properties:

* https.proxyHost
* https.proxyPort
* https.proxyUser
* https.proxyPassword
* http.nonProxyHosts

If the 'org.gradle.s3.endpoint' property has been specified with a http (not https) URI the following system proxy settings can be used:

* http.proxyHost
* http.proxyPort
* http.proxyUser
* http.proxyPassword
* http.nonProxyHosts

[AWS S3 V4 Signatures (AWS4-HMAC-SHA256)](#19c6y18)

Some of the AWS S3 regions (eu-central-1 - Frankfurt) require that all HTTP requests are signed in accordance with AWS’s [signature version 4](http://docs.aws.amazon.com/general/latest/gr/signature-version-4.html). It is recommended to specify S3 URL’s containing the region specific endpoint when using buckets that require V4 signatures. e.g.

s3://somebucket.s3.eu-central-1.amazonaws.com/maven/release

| **✨** | When a region-specific endpoint is not specified for buckets requiring V4 Signatures, Gradle will use the default AWS region (us-east-1) and the following warning will appear on the console:  Attempting to re-send the request to …​. with AWS V4 authentication. To avoid this warning in the future, use region-specific endpoint to access buckets located in regions that require V4 signing.  Failing to specify the region-specific endpoint for buckets requiring V4 signatures means:   * 3 round-trips to AWS, as opposed to one, for every file upload and download. * Depending on location - increased network latencies and slower builds. * Increased likelihood of transmission failures. |
| --- | --- |

[AWS S3 Cross Account Access](#3tbugp1)

Some organizations may have multiple AWS accounts, e.g. one for each team. The AWS account of the bucket owner is often different from the artifact publisher and consumers. The bucket owner needs to be able to grant the consumers access otherwise the artifacts will only be usable by the publisher’s account. This is done by adding the bucket-owner-full-control [Canned ACL](https://docs.aws.amazon.com/AmazonS3/latest/dev/acl-overview.html#canned-acl) to the uploaded objects. Gradle will do this in every upload. Make sure the publisher has the required IAM permission, PutObjectAcl (and PutObjectVersionAcl if bucket versioning is enabled), either directly or via an assumed IAM Role (depending on your case). You can read more at [AWS S3 Access Permissions](https://docs.aws.amazon.com/AmazonS3/latest/dev/s3-access-control.html).

[Google Cloud Storage configuration properties](#28h4qwu)

The following system properties can be used to configure the interactions with [Google Cloud Storage](https://cloud.google.com/storage/) repositories:

org.gradle.gcs.endpoint

Used to override the Google Cloud Storage endpoint when using a non-Google Cloud Platform, Google Cloud Storage API compatible, storage service.

org.gradle.gcs.servicePath

Used to override the Google Cloud Storage root service path which the Google Cloud Storage client builds requests from, defaults to /.

[Google Cloud Storage URL formats](#nmf14n)

Google Cloud Storage URL’s are 'virtual-hosted-style' and must be in the following format gcs://<bucketName>/<objectKey>

e.g. gcs://myBucket/maven/release

* myBucket is the Google Cloud Storage bucket name.
* /maven/release is the Google Cloud Storage key (unique identifier for an object within a bucket)

[Configuring HTTP authentication schemes](#3whwml4)

When configuring a repository using HTTP or HTTPS transport protocols, multiple authentication schemes are available. By default, Gradle will attempt to use all schemes that are supported by the Apache HttpClient library, [documented here](http://hc.apache.org/httpcomponents-client-ga/tutorial/html/authentication.html#d5e625). In some cases, it may be preferable to explicitly specify which authentication schemes should be used when exchanging credentials with a remote server. When explicitly declared, only those schemes are used when authenticating to a remote repository.

You can specify credentials for Maven repositories secured by basic authentication using api:org.gradle.api.credentials.PasswordCredentials[].

[Example: Accessing password-protected Maven repository](#37m2jsg)

**build.gradle**

repositories {  
 maven {  
 url "http://repo.mycompany.com/maven2"  
 credentials {  
 username "user"  
 password "password"  
 }  
 }  
}

| **✨** | The code for this example can be found at samples/userguide/artifacts/defineRepository in the ‘-all’ distribution of Gradle. |
| --- | --- |

The following example show how to configure a repository to use only api:org.gradle.authentication.http.DigestAuthentication[]:

[Example: Configure repository to use only digest authentication](#1mrcu09)

**build.gradle**

repositories {  
 maven {  
 url 'https://repo.mycompany.com/maven2'  
 credentials {  
 username "user"  
 password "password"  
 }  
 authentication {  
 digest(DigestAuthentication)  
 }  
 }  
}

Currently supported authentication schemes are:

[BasicAuthentication](http://docs.google.com/javadoc/org/gradle/authentication/http/BasicAuthentication.html)

Basic access authentication over HTTP. When using this scheme, credentials are sent preemptively.

[DigestAuthentication](http://docs.google.com/javadoc/org/gradle/authentication/http/DigestAuthentication.html)

Digest access authentication over HTTP.

[HttpHeaderAuthentication](http://docs.google.com/javadoc/org/gradle/authentication/http/HttpHeaderAuthentication.html)

Authentication based on any custom HTTP header, e.g. private tokens, OAuth tokens, etc.

[Using preemptive authentication](#46r0co2)

Gradle’s default behavior is to only submit credentials when a server responds with an authentication challenge in the form of a HTTP 401 response. In some cases, the server will respond with a different code (ex. for repositories hosted on GitHub a 404 is returned) causing dependency resolution to fail. To get around this behavior, credentials may be sent to the server preemptively. To enable preemptive authentication simply configure your repository to explicitly use the [BasicAuthentication](http://docs.google.com/javadoc/org/gradle/authentication/http/BasicAuthentication.html) scheme:

[Example: Configure repository to use preemptive authentication](#2lwamvv)

**build.gradle**

repositories {  
 maven {  
 url 'https://repo.mycompany.com/maven2'  
 credentials {  
 username "user"  
 password "password"  
 }  
 authentication {  
 basic(BasicAuthentication)  
 }  
 }  
}

[Using HTTP header authentication](#111kx3o)

You can specify any HTTP header for secured Maven repositories requiring token, OAuth2 or other HTTP header based authentication using api:org.gradle.api.credentials.HttpHeaderCredentials[] with api:org.gradle.authentication.http.HttpHeaderAuthentication[].

[Example: Accessing header-protected Maven repository](#3l18frh)

**build.gradle**

repositories {  
 maven {  
 url "http://repo.mycompany.com/maven2"  
 credentials(HttpHeaderCredentials) {  
 name = "Private-Token"  
 value = "TOKEN"  
 }  
 authentication {  
 header(HttpHeaderAuthentication)  
 }  
 }  
}

| **✨** | The code for this example can be found at samples/userguide/artifacts/defineRepository in the ‘-all’ distribution of Gradle. |
| --- | --- |

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/)