5. Repositories

Contents

[1. Introduction 1](#_Toc28363)

[2. What Is a Repository? 1](#_Toc16248)

[3. Repositories 2](#_Toc10709)

[4. Dependency Repository 3](#_Toc3264)

[5. Demo: Adding a Repository 3](#_Toc6244)

[6. Plugin Repository 5](#_Toc18685)

[7. Releases and Snapshots 5](#_Toc10905)

[8. Summary 5](#_Toc8637)

# 1. Introduction

=>slides: Pg. 1

In this module, we're going to cover repositories. Repositories or short repos are what Maven looks at to download code and other artifacts from so that we can use them inside of our application.

=>slides: Pg. 2

We're going to look at the difference between a dependency repo and a plugin repo, and we'll also see how to override those repos from their default location. Then we'll look at how you can tell Maven where it should look for releases versus snapshots. Projects don't always release snapshots at the same place that they'll release their final artifacts at.

# What Is a Repository?

=>slides: Pg. 3

We've covered local repositories in a previous module. We're going to briefly go over them again just to tie it all together with where it downloads your code from. Maven looks in your local repo first. It's where it stores everything. It looks locally to see if we have this code, and if it doesn't, it will go out and download that from a remote repository. When it downloads it, it will install it in your home directory /.m2. For the past few years, Windows has allowed Linux or Unix‑style paths, so these paths should work, whether you are on a Windows or Linux Unix‑type system. The home directory /m2/<groupId>/<artifactId> is these default location, and you can override it, but most people just generally leave it here, and I would recommend you doing so, too. If you upgrade Maven versions or other configuration changes, it's easy to forget that it was changed and can cause you some unnecessary headaches. In our local repository, we always store artifacts using the information that we provided for the artifact, group, and version ID. It's not something that we necessarily get to choose. Its these three things that are required to download anything. We don't get to choose it because it's defined by the people who created the artifact that we want to use. So, for example, we previously used commons‑lang, the org‑apache‑commons was the group ID, and the commons‑lang3 was the artifact ID, and 3 was the version number. The reason this is good is it avoids duplication by copying everything in our project and storing it in our source code control module. So our path looked something very similar to this. We had ~/.m2/repository/<groupId>/<artifactId>. This naming convention keeps us from storing those jars in our subversion or Git, Bitbucket repositories, whatever you're using for your SCM, and just to show you a kind of image of where this is all stored at, here it is on my local box that I'm using right now. You can see that I've got my user's directory, m2, and then my repository, and then it goes clear up the ladder into org, apache, commons, commons‑lang, 3.8.1.

# Repositories

=>slides: Pg. 4

Repositories are, very simply, just an HTTP‑accessible location that we can download files from. Very often they don't have any security wrapped around them, although for your internal repository you can definitely add it. Our default location is defined in the file or the super pom.xml. We've mentioned this before previously. The Super POM is located inside of our Maven installation. Generally, you don't need to go into it, the Super POM that is. In fact, I'll go out on a limb and say you should never really edit the Super POM because there's other ways to override it. Either go through a global parent project file or through your own project's pom.xml. So don't edit the one that comes with the default Maven installation. The default location that it's pointing to is repo.maven.apache.org. This has 95% of what you'll ever want to download, but it's that extra 5% that we're going to focus on now, because it's actually a very important part. The Maven repo that's out there by default has all of the open source projects. You're never going to upload your company's internal resource libraries to repo.maven.apache.org, because you probably don't want other people using that. We should note that multiple repositories are allowed and, in fact, often encouraged. You'll have a case where you'll have to download things from more than one repository, and that's fine, it's a common practice. To do so, we can use a corporate repository. A corporate repository is definitely outside the scope of this course, but you are encouraged to use one, though. A few years ago when Maven was getting really popular, nobody was using a corporate repo yet. Developers actually caused a denial‑of‑service attack on the main repo.maven.apache.org site. Two of the most common ones out there are Nexus, that's actually what's backing repo.maven.apache.org, and there's another one that is almost used is equally in some other projects called Artifactory. If I were starting from scratch, I would probably tell you to look at the Nexus repository, but Artifactory works well. I've used both on projects and been happy with both of them.

# Dependency Repository

=>slides: Pg. 5

A dependency repository is exactly what it sounds like, it's where we download all of our dependencies from. It can contain releases or snapshots or both, for that matter, but it's not uncommon to have them in separate repositories. It's very easy for us to define our dependency repository.

=>slides: Pg. 6

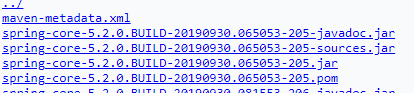
It's just a snippet of XML in the repository section of our POM. Here is an example of that code. It has to have an id, and in this example, we're using spring‑snapshot, and generally we want to give it a name, although it's not required. But just so it's more descriptive we'll use Spring Maven SNAPSHOT Repository for our example, and then the URL to our repository. Now usually organizations are pretty good about labeling their repository as a snapshot repository, and you can see on the end of this URL here, libs‑snapshot. You'll also notice that in this one we have snapshots enabled to true and releases enabled to false. If we were going to enable people to download both snapshots and releases from this repository, that would just say releases enabled true, pretty straightforward. Like I said, you can have multiple repos and it would be another element inside the repository's element to just define a separate one for releases versus snapshots. Let's take a look at a demo of how we can download an artifact from that repository.

# Demo: Adding a Repository



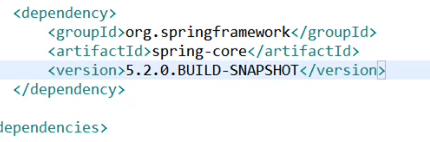
To demonstrate showing you repositories, I want to begin by showing you Spring's repo for snapshots. So we're going to start off by opening up a browser and going to repo.spring.io/libs‑snapshot/. This will pull up a repository, and you'll quickly see they're a basic website. There's nothing really that exciting about it. I'm going to navigate all the way down to org, which is about halfway down this page here, and we're going to use the groupId of our artifact to navigate into our Spring core artifact. So if we go to org,   
and then this will pull up another page, and we'll go to the Spring Framework, and I'm actually going to use these search on this page to find it quicker. Now notice before I click on here that there are three springframeworks here, and this is another thing I wanted to point out. Notice the misspellings. There's springframeWork with a capital W, springFramework with a capital F, and then all lowercase springframework. One of the bad things about artifacts, and it's a good thing and a bad thing, is that once we've created them, they're really not supposed to be deleted. And so these misspellings of springframeWork with a capital W and springFramework with a capital F, we should leave there for anything we build the future. Now this is a snapshot repository, so we really probably don't care, but that's why there are three renditions of that spelling. Let's click on springframework. Inside of here, we can scroll down

  
and we'll find spring‑core. If I click on spring‑core, and scroll down to the bottom, you'll see   
5.2.0.BUILD‑SNAPSHOT. Now, when I click on this, you're going to see something that you maybe weren't expecting.

  
Inside of here, there is not a file called BUILD‑SNAPSHOT. This is where I said the meaning of snapshot has an extra special something behind it. Maven knows to go look for the latest timestamp or increment inside that directory. So when we give it a groupId of org/springframework and an artifactId of spring‑core, and a version of whatever‑snapshot, it goes and looks in that directory for the most recent code there, every time we build. So if you do this with other files, if I back up a directory here and I go to one that's not listed as a snapshot anymore, the code inside of here will have specific build numbers on there.



So if I go inside of this RC1, well, this RC1 doesn't handle the same way snapshots are handled, and so inside of this directory, we just have RC1; we don't have these timestamped files. So you kind of get an idea of what that snapshot is doing. Inside of there, it has a BUILD‑ year, month, day, dot hour, minute, second, dash their build number. So there's been 402 builds of 5.2.0, and you get to see the timestamp that this was done on May 30th. So this is how snapshots work inside of Maven. Now, let's go demonstrate this inside of our project.

  
I'm going to go ahead and switch to our IDE, and inside of our IDE, I want to add that dependency, and I'm going to start off by saying dependency, and give it a groupId of org.springframework, and then an artifactId of spring‑core, and then our version is going to be 5.2.0.BUILD in all uppercase, SNAPSHOT, and save this. Now you'll notice, we immediately got a red error. You can see on line 33, we've got a red X, and the reason why is this is a snapshot, and we haven't pulled in that repository. So we're going to do two things here.



We're going to add a repository that goes ahead and allows us to download snapshots, because the main repo doesn't allow you to pull down snapshots. Now to download that dependency, because you can see we have that red X again. I need to add another repository, and I'm going to go down between my dependency's closing element and my opening build element and say repositories, and inside of this element, I have to give it the singular repository element. And in here I've got to give it an id, and for the id, I'll just do spring‑snapshot, and really, the id is important to you; it's not important to Maven, going out and pulling that down. The next thing we need to add is the URL, and I've gone ahead and grabbed that URL from our browser, and its http://repo.spring.io/libs‑snapshot. And before we close this out, I want to restrict this just to snapshots. So I'm going to say snapshots enabled, and for sake of screen real estate, I'm going to do this on the same line, and then I'm going to create a new line that says releases, enabled, and false. So this going to to restrict this to just snapshots. Now, I want to point out before I save this, it's going to automatically download that and adjust my project and will get rid of that red X on line 33. So I'm going to hit Ctrl+S and save it, and it's already building my application. You'll notice the red X went away. Okay, did it actually download it, though?

  
Well, we can go check that by going up to our Maven dependencies, opening it up, and if we scroll down, you'll see spring‑core 5.2.0.BUILD‑SNAPSHOT.jar, and you'll notice its path is pointing to our hard drive. You see the path exactly to where that file is stored. So if you wanted to go out there and verify it, you can actually navigate to that in your File Explorer, and see that it did in fact exist at that location. So pretty simple. We showed you that HTTP sites for repositories are just that. They're very simple websites that a dependency with a snapshot will not build, because most people don't upload their snapshots to the central Maven repo. And to add a repository, we can go ahead and just add a repositories element below our closing dependencies tag, but before our build tag, and add a single repository for snapshots and restrict it to just a snapshot release and no other final releases from that repository. Really simple to do, but a good way for you go download code that's possibly under development by a library that you're using.

# Plugin Repository

=>slides: Pg. 7

Plugin repositories are almost identical to a dependency repository, they just deal with plugins. We haven't covered plugins very much yet in this course, and that's actually the next module, but us far as where we go to look for them, it's in a plugin repository.

=>slides: Pg. 8

We'll only look for plugins usually in a separate repository. It doesn't have to be that way, it's just more of a code cleanliness thing. They are defined very similar to repositories, except that we have a different XML element that's the pluginRepositories element. As you can see here, it's instead of just a plain Repositories element. The same rules apply that we can have snapshots here, we can have releases here. In this particular example, we did it for a fake corporation, kind of like our own internal corporation. Sometimes people will build their own plugins for things like versioning or how they handle releases or things like that, and put them in an internal corporate repository.

# Releases and Snapshots

=>slides: Pg. 9

Releases and snapshots can come from the same repository as we had previously mentioned. Why would we not want projects to just upload everything to the central repo? Well, there's a release process not unlike deploying a mobile app to an app store, or something similar to that. It's a little bit difficult. We also want to make sure that we have everything final before we pushed it up there. Things like snapshots, milestones, release candidates, all of these are best not to deploy to the central repository just because it's a little bit more work and a little bit of a headache to do so. We'll often define a repository to handle just snapshots and host it off of our own corporate website. This was the case with the dependency repo that we looked at earlier from Spring. They don't want to put all of their snapshots up in repo.maven.apache.org.

=>slides: Pg. 10

And here's that repository again where we just get snapshots from the corporate repository of Spring. Snapshots are enabled, but releases are Disabled

# Summary

=>slides: Pg. 11

Dependency repositories and plugin repositories can be separate or the same repository. As we mentioned, projects will often not upload their snapshot code to central because of the headaches and sometimes frequent changes that their code will go through before they get it up to a final released project. Plugins are usually in the same repository as dependencies, but can be broken out. They often are broken out for corporate repositories, and if you've created your own plugin, should store it there. Also, as I mentioned, companies should look at using their own corporate repository to help lighten the load on the central repo. You're eventually going to build your own libraries that you want to have hosted internally and not be posting those up to the central repository.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*