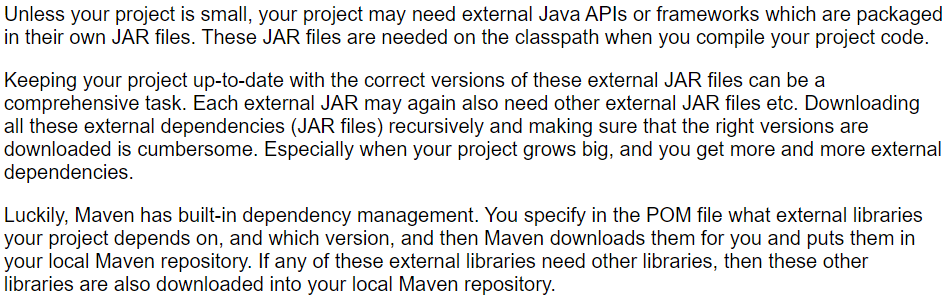
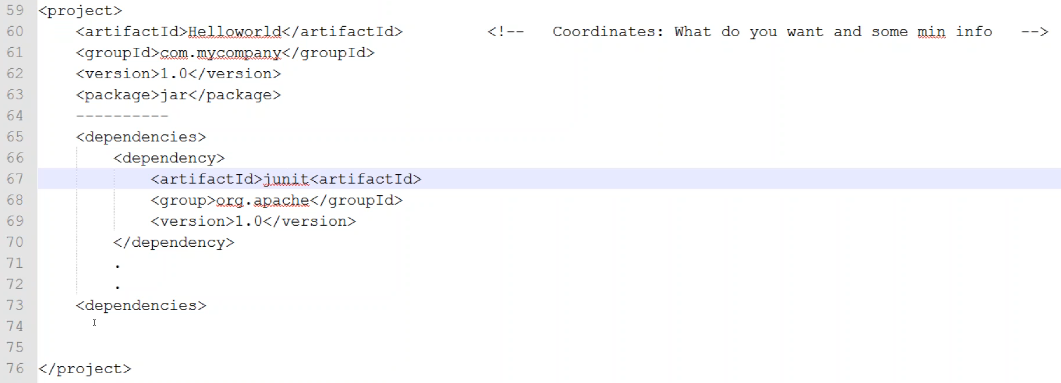
**Dependencies:**



* If our project depends on other project, maven can grab or download those dependencies and use them to compile the code, run the rest and even bundle these dependencies into war files
* Maven has a repository where it puts all the artefacts and plugins
* If a project needs a repository from open source, those can be pulled from below repositories and can be pushed also
* under dependency, we need to mention artifactid, groupid, version as well
* dependency is the second section

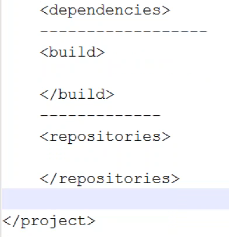




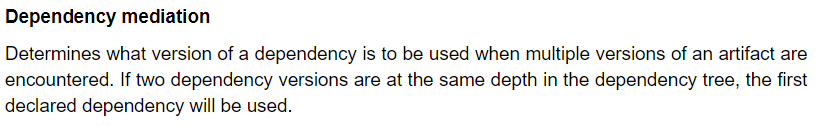


We can add the dependencies as above from other projects

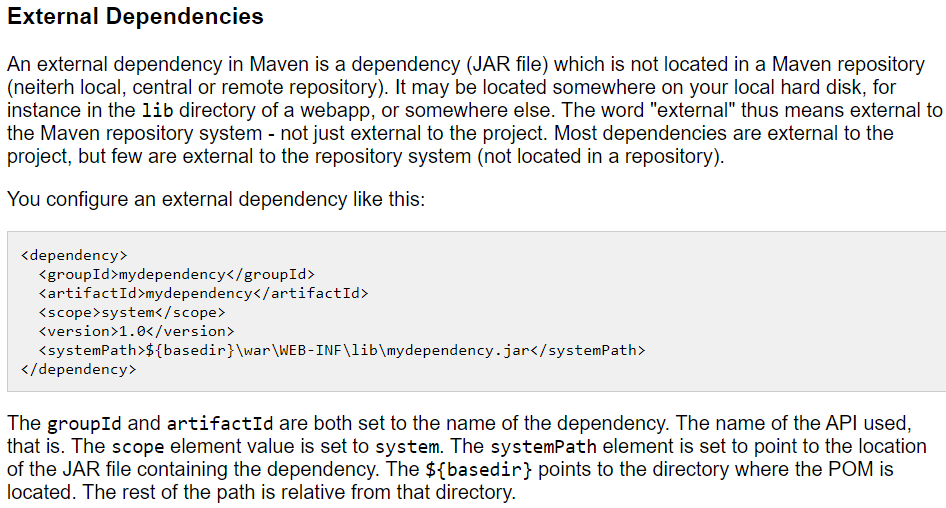
**<model version>** is the section introduced in maven 3.3 which tells the version of pom.xml we are using

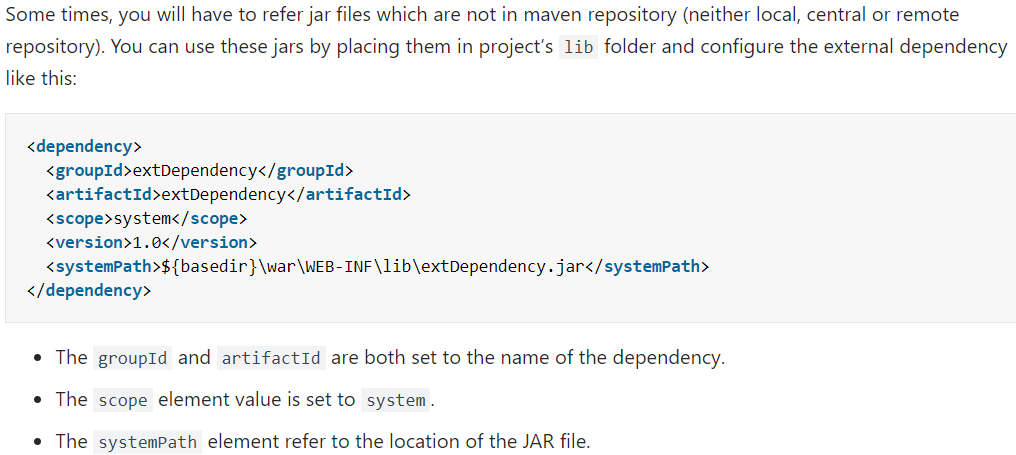


**Types of dependencies:**

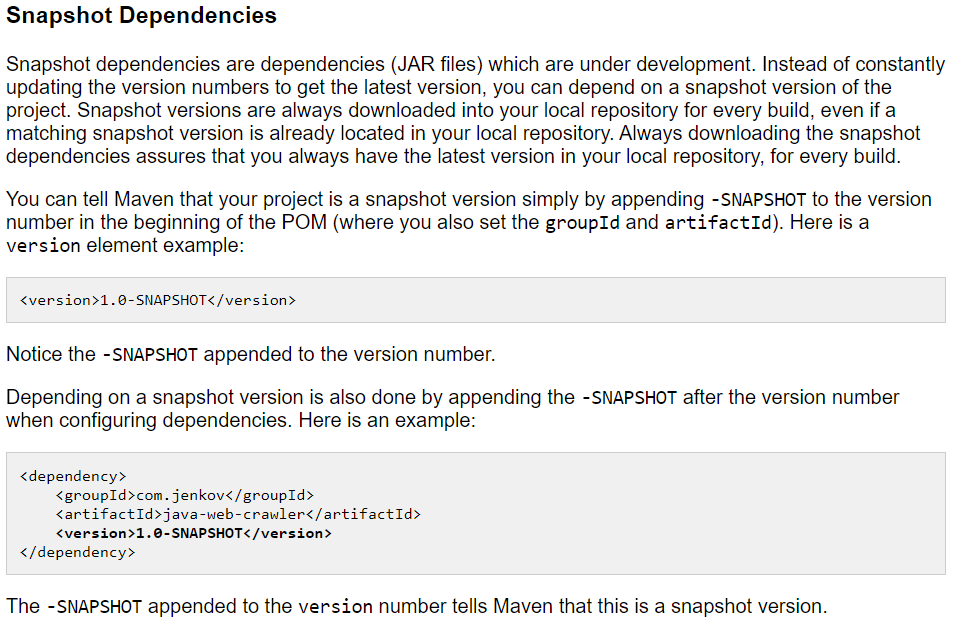


**External dependencies:**

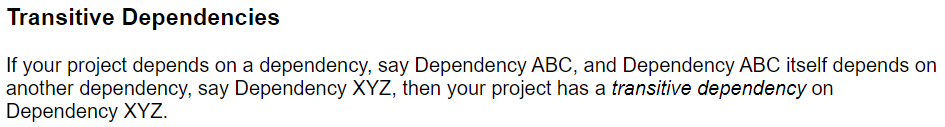


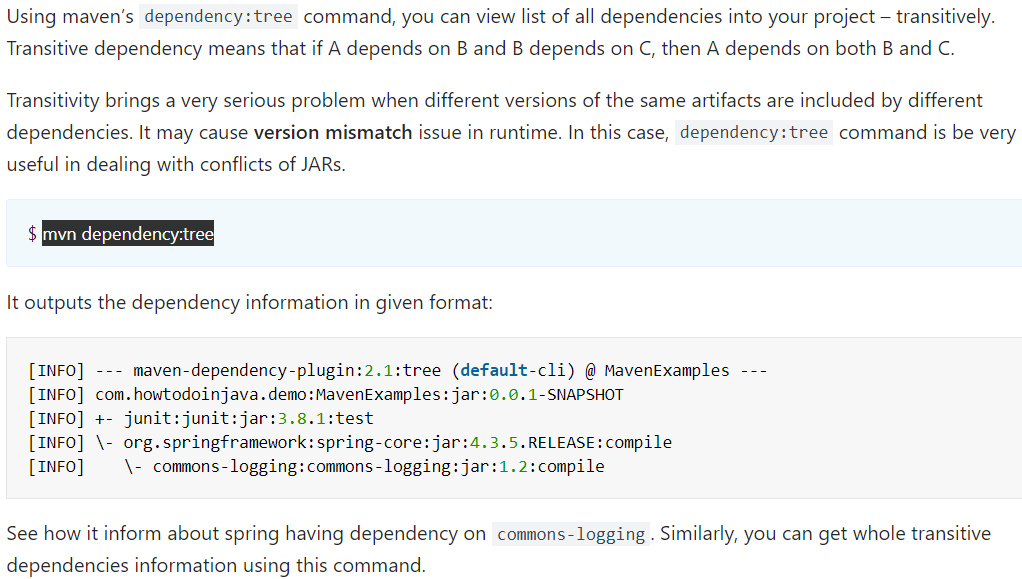


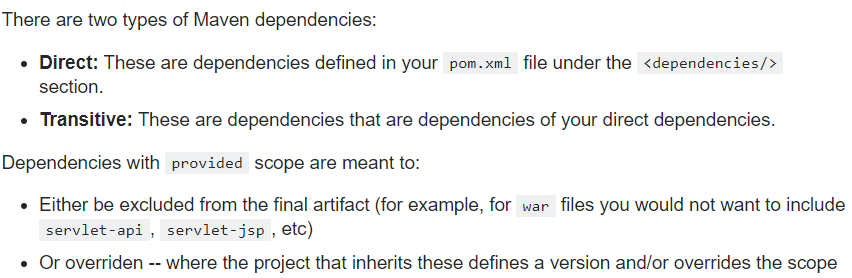
**Snapshot dependencies:**



**Transitive dependencies:**

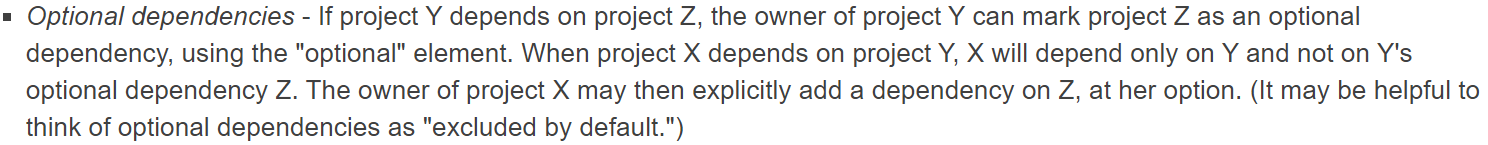


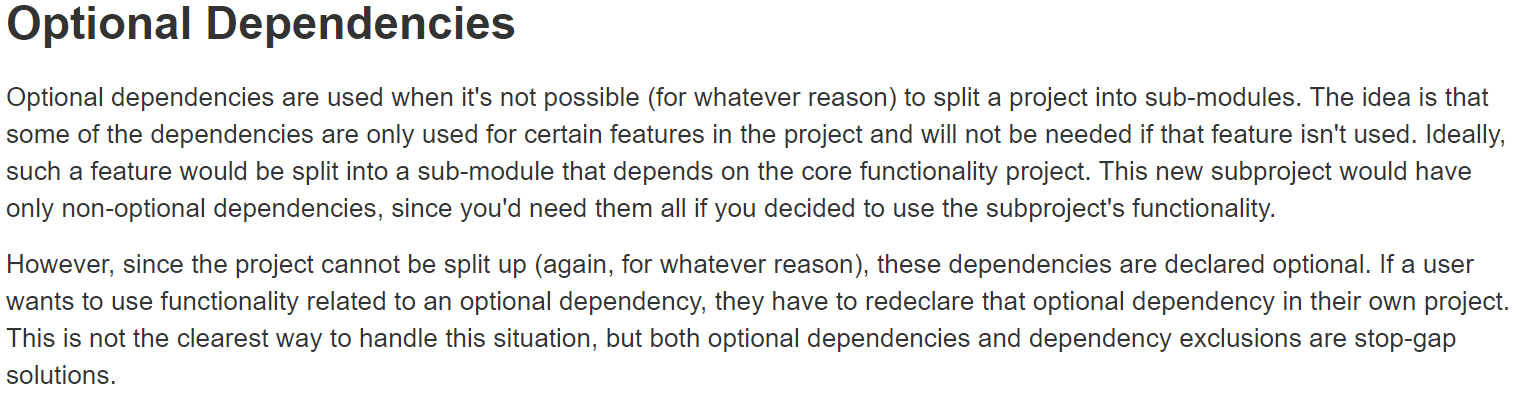


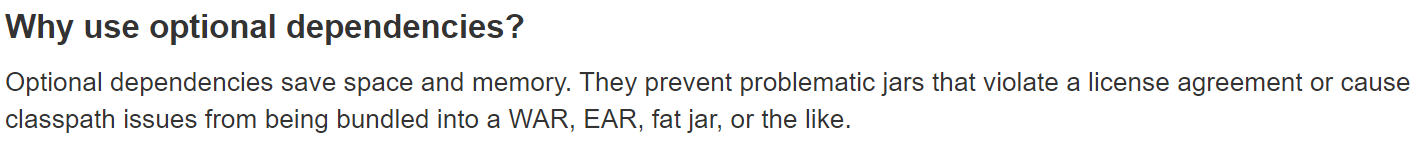


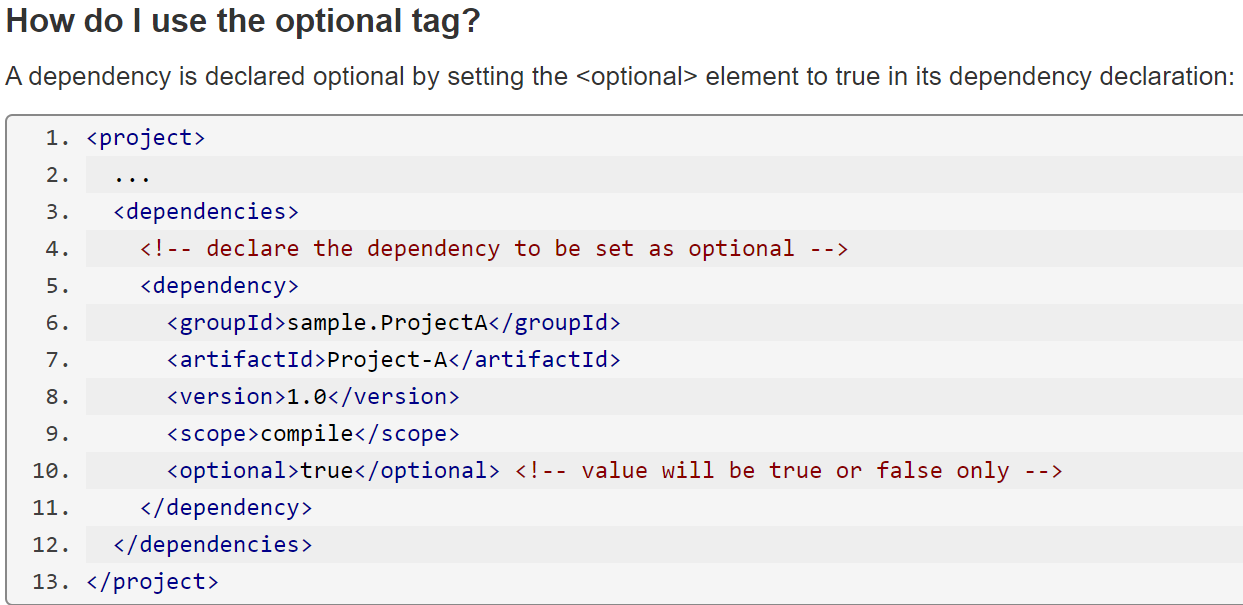
**Maven by default takes the transitive dependencies as well while building the project**

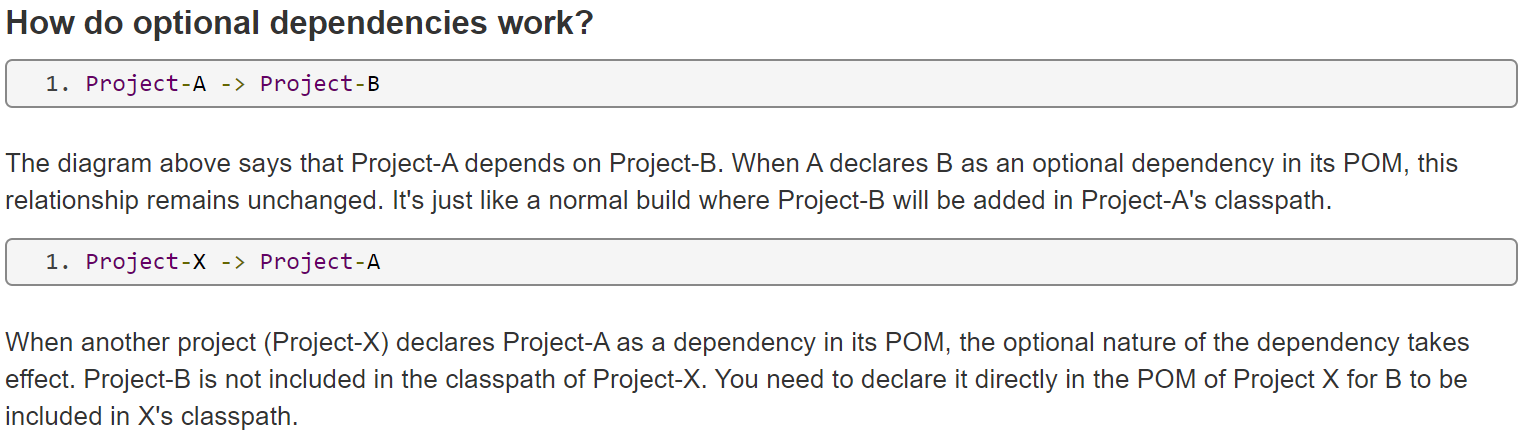
**Optional dependencies:**



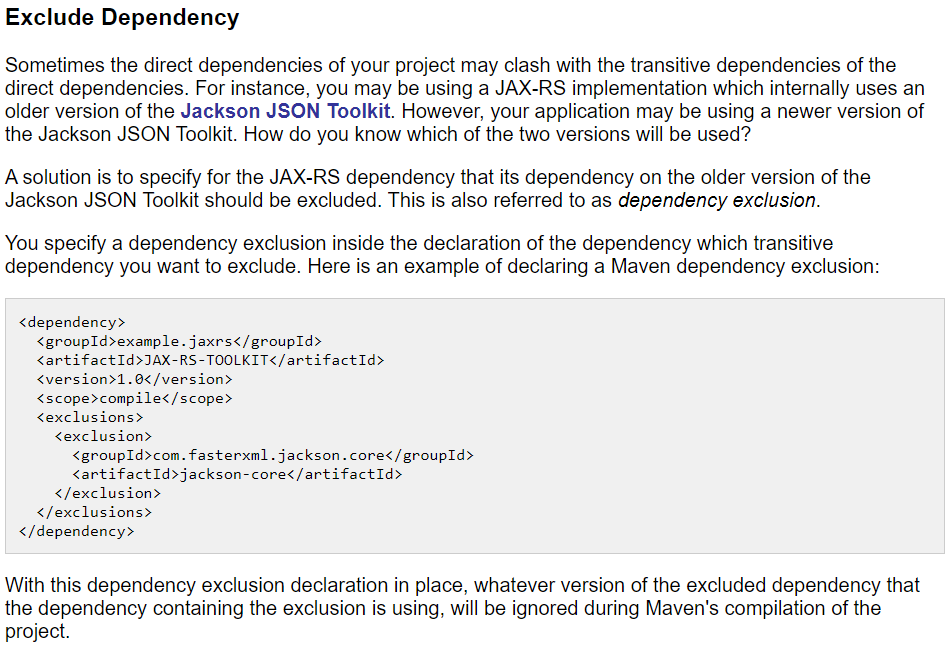


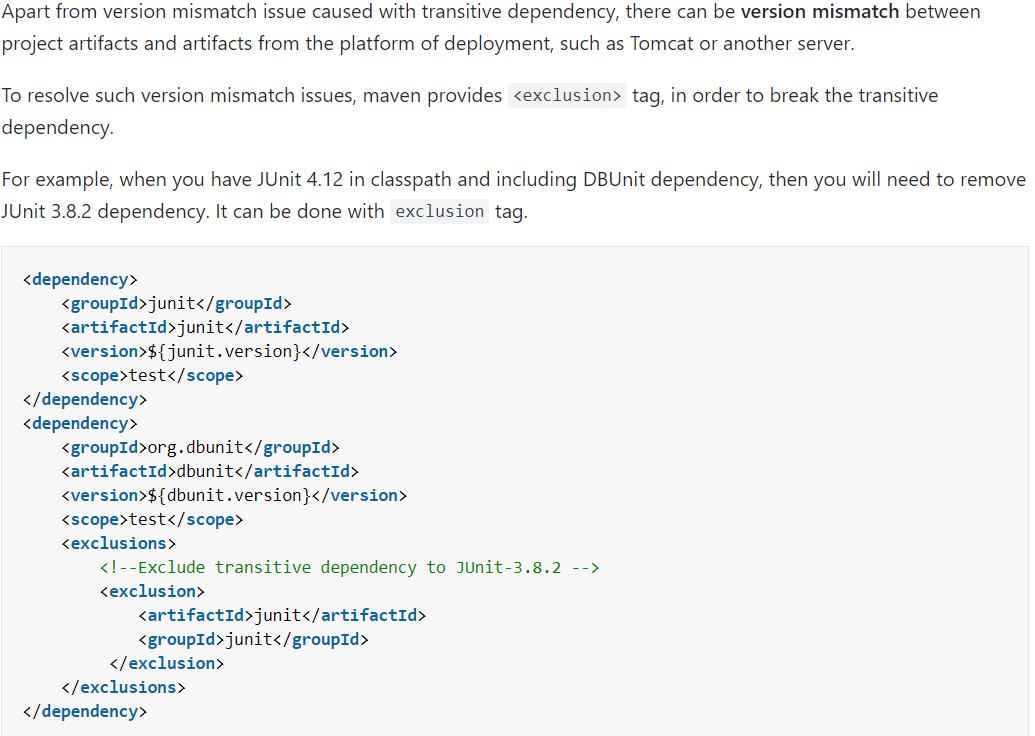


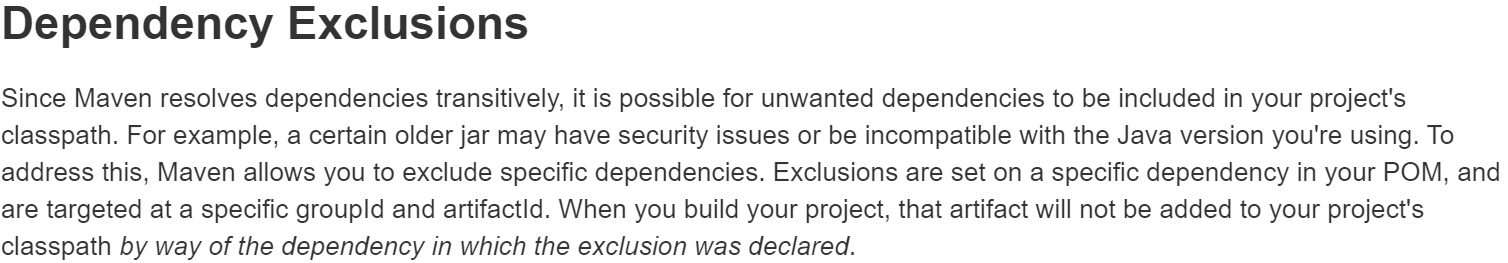




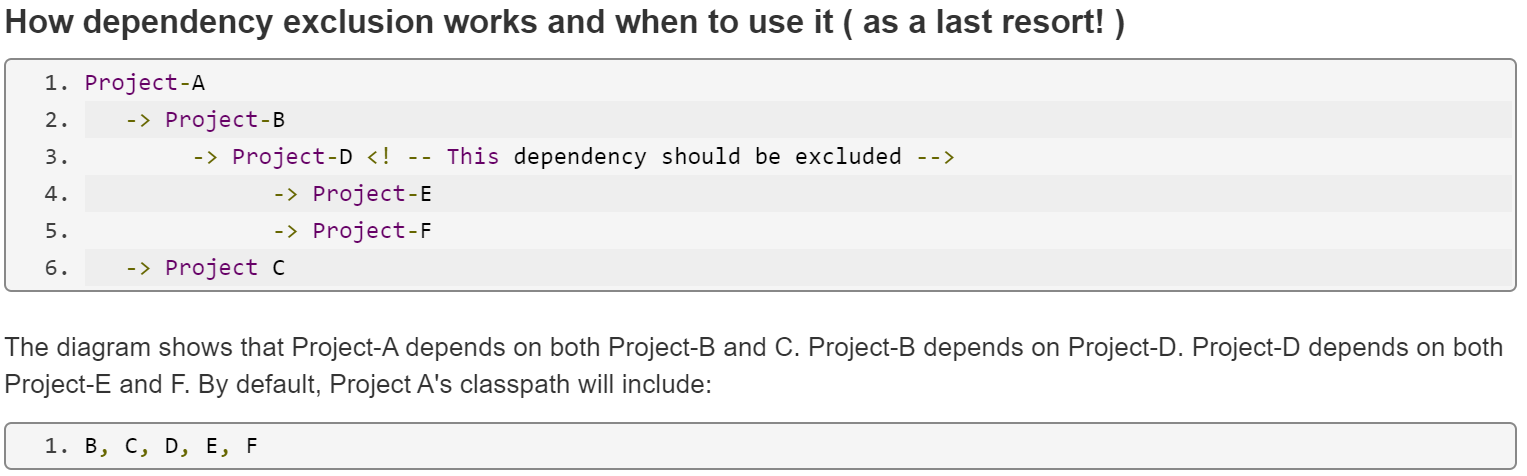
**Exclude dependencies:**

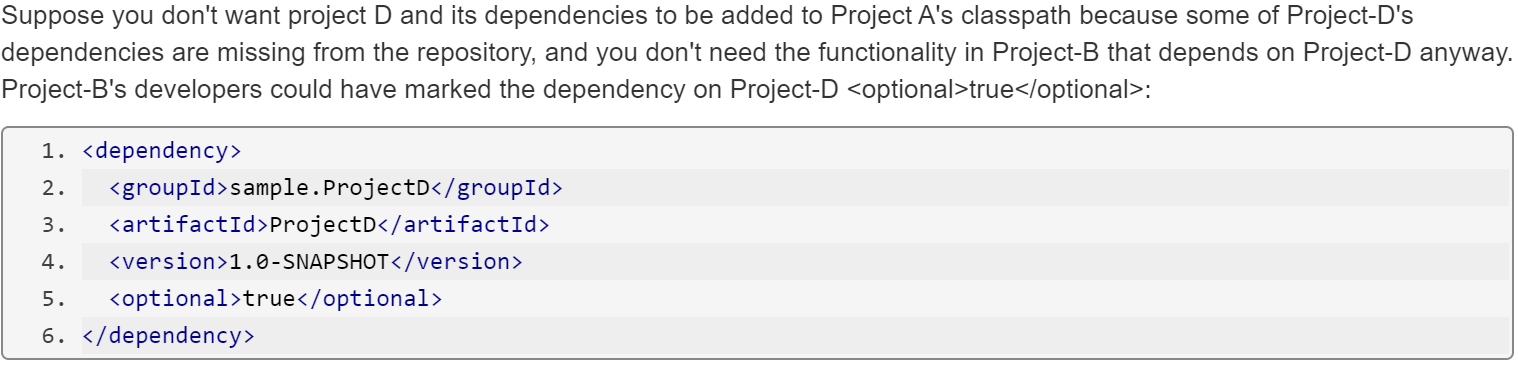


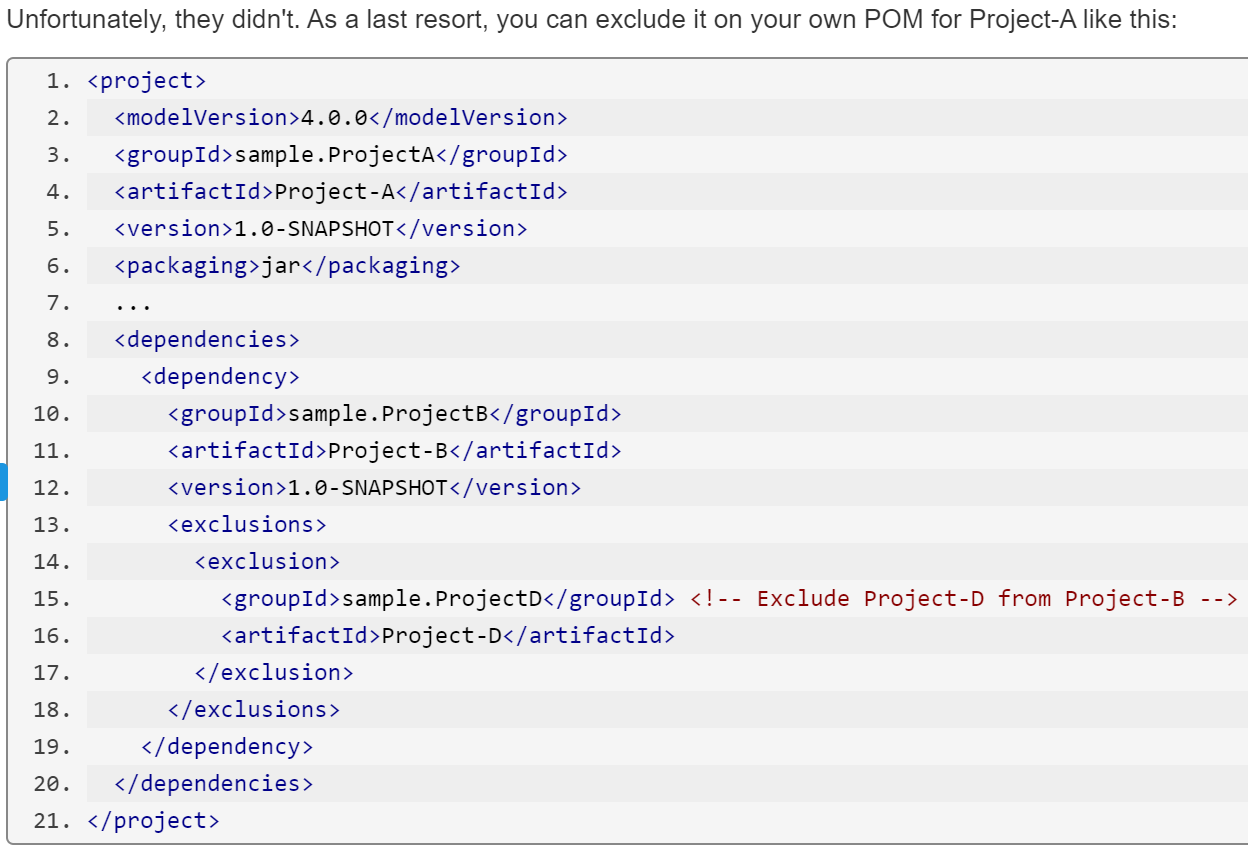


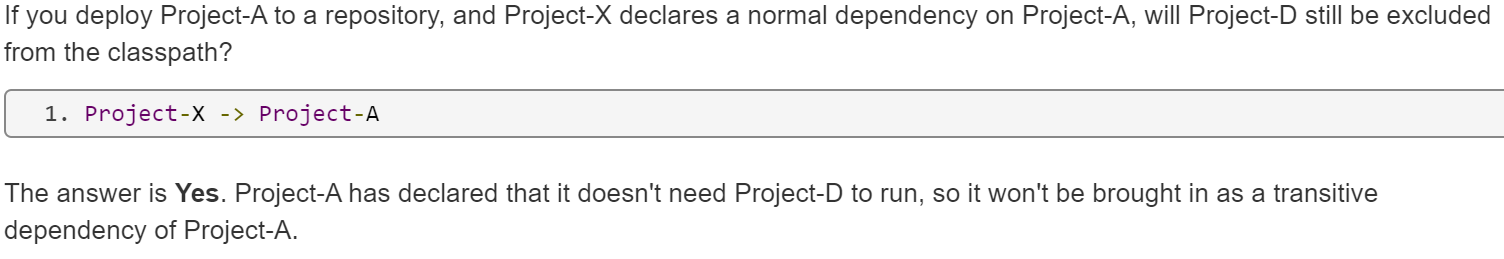


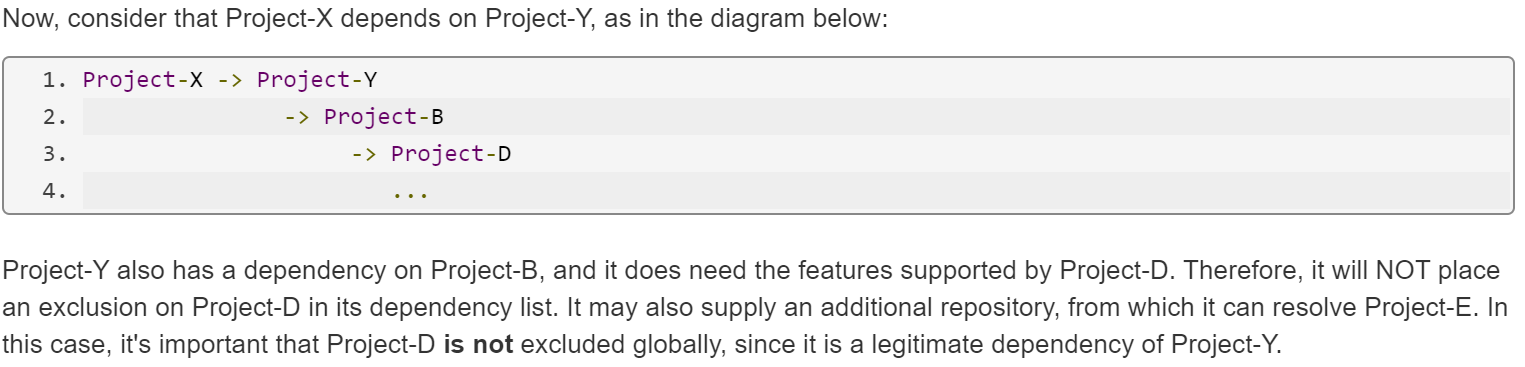


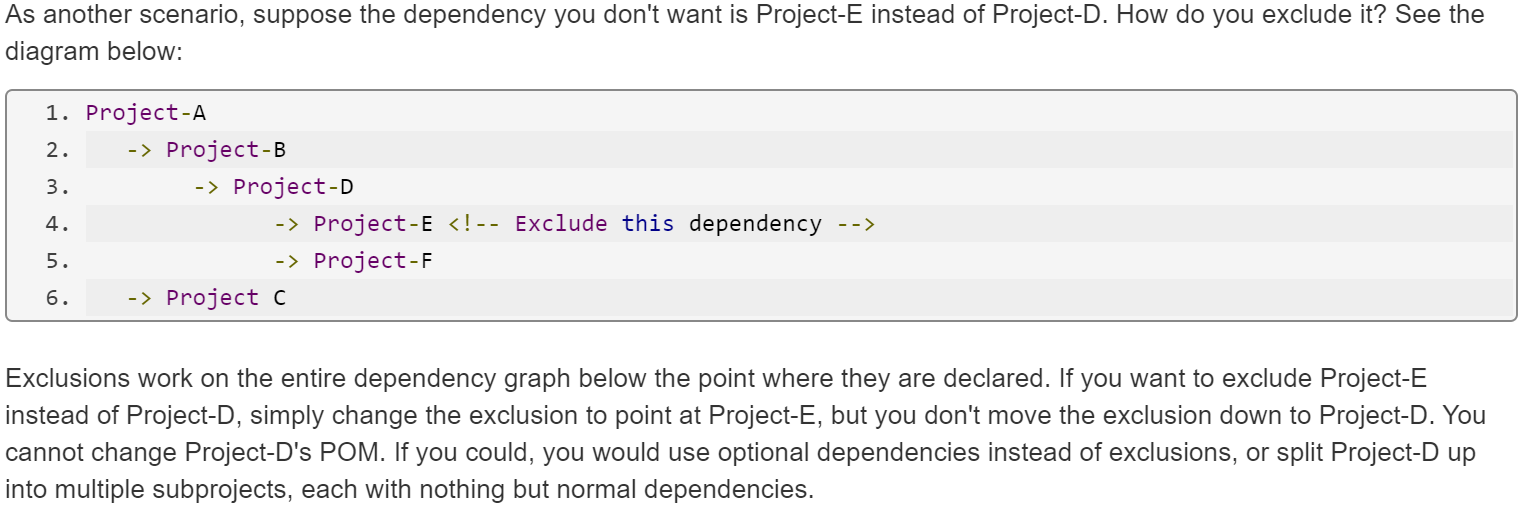










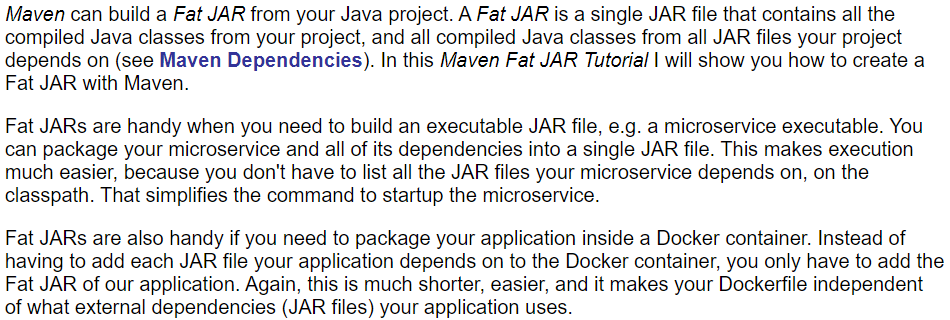




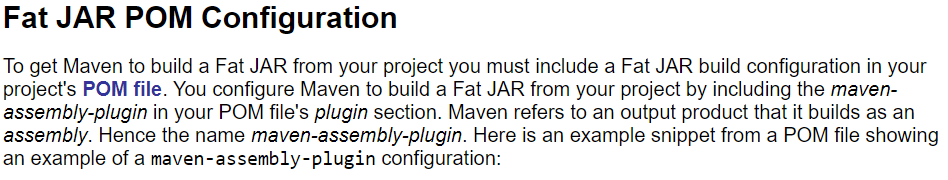
**Even the transitive dependency again added to the pom.xml as a direct dependency, maven will just ignore that as it is declared already, and it won’t bundle that again and create a bigger artefact file**

**But, it may reduce the maven build execution time if we don’t specify the transitive dependency again as a direct dependency**

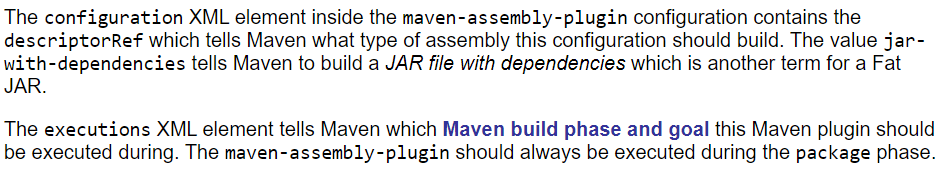
**Fat Jar:**



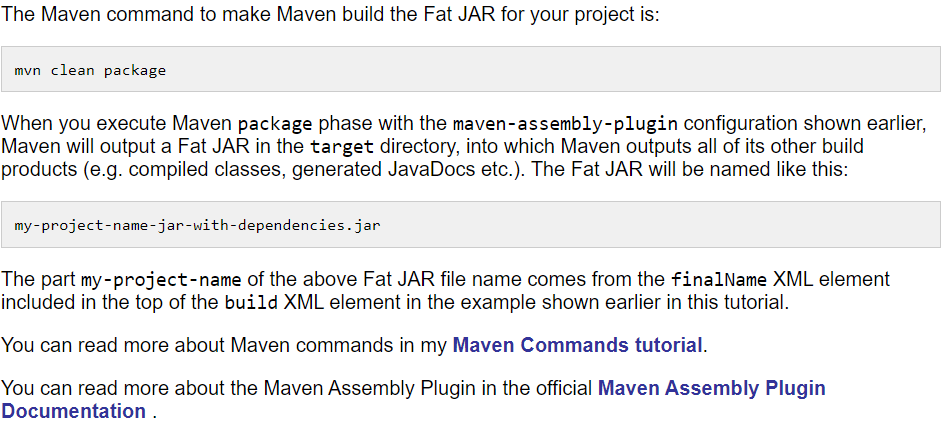
**POM configuration:**



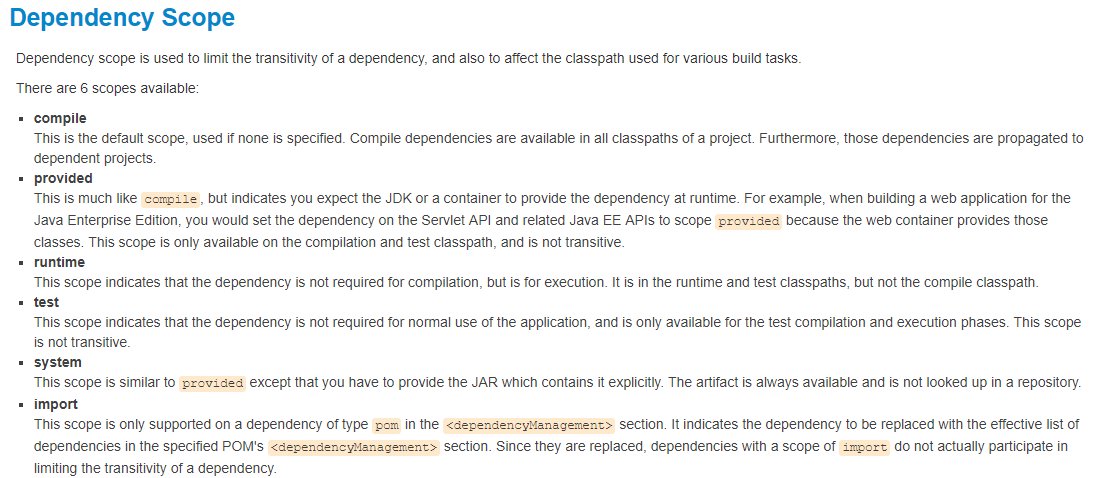


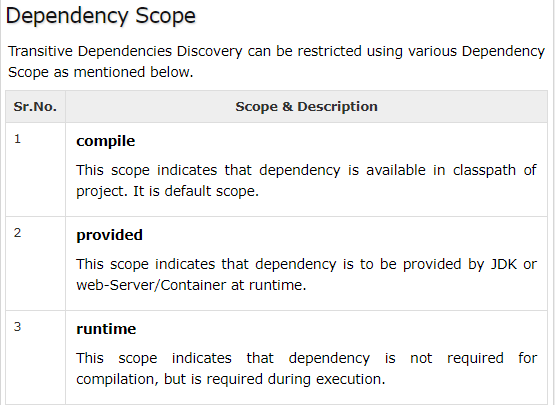


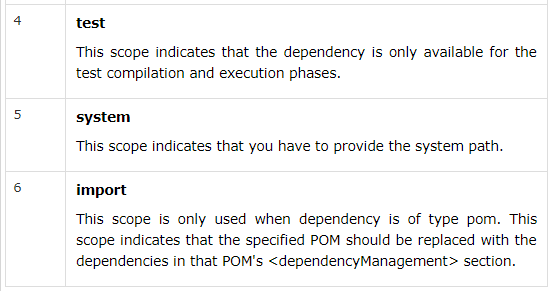
**Build fat jar:**



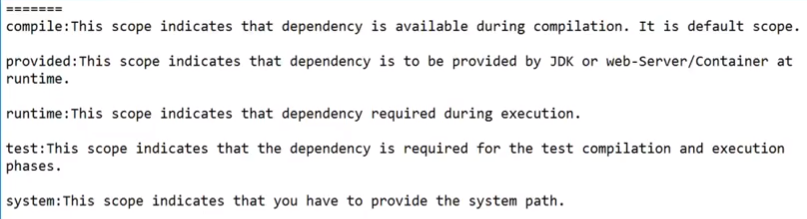
**Dependency scope:**



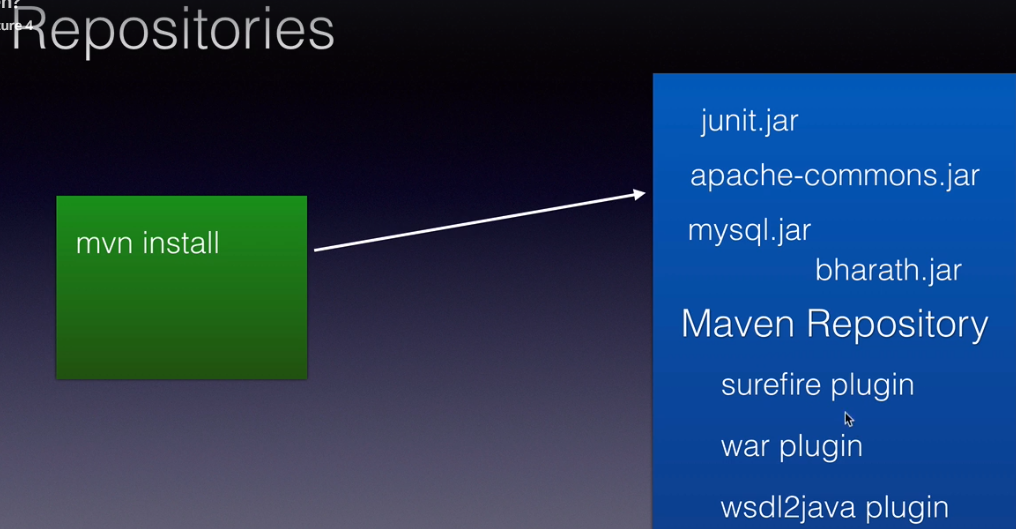




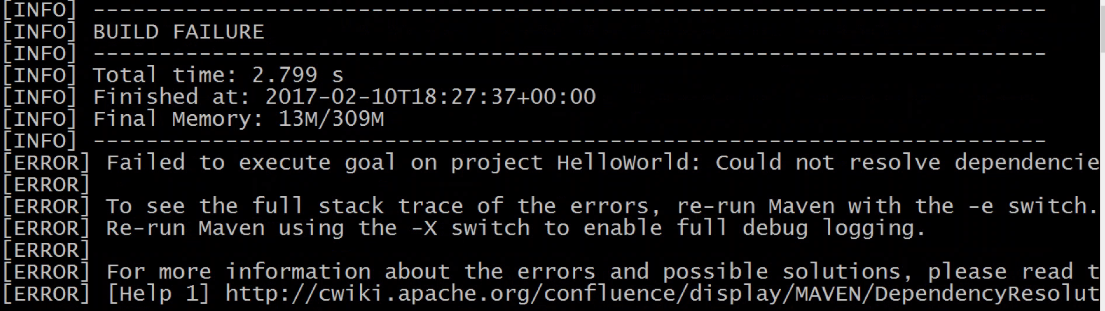
* For example, if we want to add junit dependency and don’t want that to involve in compile source code but only for test source file. Then we need to add scope as test
* Compile is the default scope
* If we give runtime, then the dependency will be available at only run time



**Repositories:**



* These repositories can be on the internet, there is a public maven repository where all the open source projects are available
* Many companies use their own repository which runs on their servers. If any other teams want it, they can pull it and use it
* Maven by default downloads the libraries under user home location
* First time build is little slow because it downloads the libraries
* .M2 is called the local repository
* If a developer B have dependency on the helloworld program developed by developer A
* Maven, first checks in user repository and then on apache repo. if it can’t find it, it will throw the below error



* And the solution for this is, developer B can ask the library from A and paste it on user repository
* If the developer is developing the library continuously and there are 100 more developers in the project, it will be very difficult to handle
* That’s why maven has binary repository concept
* It is just like an apache repo in our own server
* So, developer A instead of giving the library to other developers, he can push it to remote repo after developing and other developers can pull it
* The location of remote repo is gets stored in pom.xml, so the pom.xml takes care of downloading them
* Pom.xml first looks into local repository and second, it looks in public repository and third, it looks in remote repository
* We need to mention about the remote repository in pom.xml
* This is called a binary repository

**There are 3 types of repositories in maven:**

1. **Local repository (.M2)** – this one gets creates automatically
2. **Central/public repository** – it is already created with default libraries
3. **Remote repository** – this we create within organisation to share the internal libraries

We can setup the remote repo with the help of archifactory/nexus/archiva …

**Install phase:**

* With this simple command, we can compile, run the test and create package
* Go to the project folder, we can find a new directory called target, maven creates this
* Under this, it puts all the compiled classes under the classes folder
* It also packages the compiled classes into jar file
* If we run “mvn -install” command, it will deploy the package into local repository
* If we are working on some other new project and needs dependency of this project, we no need to go to central or remote repository. It can pull it from local repository. That is the advantage of storing the packages in local repo
* And one more advantage is, we can push it also
* Below is the repository location



**Deploy phase:**

* This push the library from .m2 to remote repository
* Once the work completed, the data pushed from workspace to .m2 is called install phase and pushing the libraries from .m2 to remote repo is called deploy phase
* If we run deploy, it will run install phase also, as it runs the prior phases
* We need to mention in repositories section about the remote repo

**There are 2 types of remote repos**

1. Snapshot
2. Release

* Snapshot is a development progress build
* Release is ready to release build
* Any time before the session is called snapshot. The partial completed jar files called snapshot
* Once we done with official release is called proper release
* Developers release all these partial releases to the snapshot repo and official release to the release repo
* In pom.xml, we need to set snapshot and release repositories, if the version name as snapshot, it will go to snapshot repository and if its mentioned only version number, then it will go to release repo