Repositories

Repositories

Maven introduces the concept of **repositories**. A repository can be seen as an **artifact version control system**. Ultimately, all projects products are going to end up in a repository. In fact, Maven uses repositories to store about anything : plugins, dependencies, extensions, ... Whenever it needs a artifact, Maven will look into its different repositories to find it. There are two different types of repositories : **local** and **remote**.

There is always one local repository for a given Maven installation accessible from the file system. The local repository serves two purposes, it holds a cache of all the different artifacts retrieved remotly and contains your installed projects. An installed project is a project which has been built and that you want to be able to use in some other local projects but you aren't ready to deploy it yet and makes it available to other developers.

Remote repositories allow you to makes your project artifacts available to other developers. A remote repository can be accessed throughout a network protocol. Maven has built-in support for a couple of protocols (file, http, scp, ...) and extensions are available for several others (ftp for instance) - The network protocols topic is addressed more deeply in the Configuration trails **[todo:link]**. Those repositories can be some **external repositories** set up by a third organization to provide their artifacts (for instance, Maven's official repository on Ibiblio) or an **internal repository** u sed by your corporation to store and share in-house artifacts. As long as it is accessed remotely, Maven doesn't care and doesn't see any difference.

> **Why not using a SCM system instead?**
> Some people might wonders at this point why do we need repositories. Isn't that the purpose of a SCM system ? Well, not really. A SCM system's role is to handle the versionning of the files constituting one and only one project. An artifact is independent of the project, it has a versionning of his own and therefore it doesn't serve any purpose to save it along the project in the SCM system.

The repository structure

Do you remember all this stuff we learned in the Naming Convention lesson? Well, it's time to put them to good use. Like projects, repositories are structured in a standard way so Maven can find quickly the needed artifacts. A developer doesn't have to deal directly with this stuff but it is pretty important to have a good comprehension of it. Otherwise, you will have a hard time finding dependencies informations and declaring them in your project.

A Maven repository is organized in a similar fashion as Java packages. A project groupd id and artifact id are mirrored on the file system with directory names. For instance, let's take the Doxia core project, which group id is *org.apache.maven.doxia* and artifact id is *doxia-core*. Hence the project artifact is stored under *repository_root_dir/org/apache/maven/doxia/doxia-core.*

But how is Maven able to differentiate versions ? Well, the artifact is always stored under a subdirectory corresponding to the artifact version number, So doxia-core 1.0 would appears under the subdirectory *repository_root_dir/org/apache/maven/doxia/doxia-core/1.0/*.This approach works well for official releases but what about snapshot releases? Knowing I can make several releases of my project in one day, does Maven create a new subdirectory for every one of them?. Of course not! Otherwise the project directory would become quickly a big mess. Maven will create only one subdirectory for instance 1.0-SNAPSHOT where every snapshot releases of the version 1.0 will be kept.

Maven can stores four kind of files associated to a project : the project artifact archive, some project related content (javadoc archive, source archive), the project pom file and key signature files so downloaded files integrity can be verified. All the project filenames respect the following format : *artifactId-version.extension*. This is a good practice Maven enforces so that the version can be determined at a glance without having to check a possibly non-existent manifest, or compare file sizes with the official releases (something I am sure you have done in the past and didn't feel comfortable about). This convention is also used to deal with snapshot releases in a way that avoid the creation of many subdirectories. Maven offers two ways of dealing with snapshot releases : overwriting the previous snapshot release or assigning a unique version number comprised of the timestamp and build number (how to choose the strategy for a given repository is covered in more details in the Configuration trail **[todo link]**).

[todo :screenshot]

> **Metadatas files**
> As you can see, the repository also contains some metadata files. Those files are used for the moment by Maven to speed up its research in the repository but they may have other use in the future once more information are added to them. For the moment, think of them as index files.

Dealing with repositories

Basically, to use repositories you have absolutely nothing to do. Maven handles all that boring stuff for you 😊. Well this is not totally true, you still have to configure the different repositories you use but fortunately this is a one-time operation. Repositories configuration is covered in the Configuration trails [todo:lnk].