Creating the repositories

Creating the repositories

This guide will recommend installing both the Maven internal repositories and the Maven proxy on a single machine and to make this guide a bit more readable this machine will be called NUCLEUS.

Prerequisites

A web server already running and configured on NUCLEUS.

A web server is needed so that the internal repositories can be accessed from the local desktops.

scp access to NUCLEUS

It is out of scope for this guide to walk through setting scp up on the server.

The local desktop will need a SSH client which are available for Unix at `http://www.openssh.com/` and Windows at `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`

Shell access to NUCLEUS

Shell access is needed to execute commands and to configure the initial repository layout. This may be done in conjunction with a sysadmin if you do not have the required privileges.

Write access to a web server visible directory.

In this guide the Maven repositories will be installed into a location that has been configured for access by your web server.

This guide will use the location WEB_ROOT/maven2_repositories.

Create the repositories

The following artifacts are ones stored in a corporate environment:

- inhouse released - artifacts released in-house
- inhouse snapshot - snapshots made in-house
- external non free - lke Oracle JDBC, commercial (licensed or bought) stuff
- external free - free but with annoying license restrictions so they can not be housed on ibiblio. Or any other projects not yet on ibiblio (in which case file a request to get it loaded into ibiblio see http://maven.apache.org/guides/mini/guide-ibiblio-upload.html)

Note: these categories are paraphrased from the Proximity web site http://proximity.abstracthorizon.org/ which comes preconfigured for similar scenario (local repositories with proxied remote repositories).

The corporate environment should only contain artifacts as categorised above, any other artifacts should be obtained from the proper repository and not be duplicated in the corporate repository.
However these remote non-corporate environment repositories may be proxied. See #Creating a Maven proxy. Remember a proxy and a repository are different concepts.

In this guide the logically different artifacts are stored in physically separate repositories. There is no physical reason why you could not store them within the one repository but to make it obvious where artifacts have come from they are separated.

It is best practice to separate releases from snapshots. In general other people are only interested in the released versions, they are not interested in snapshots unless they want to work on the bleeding edge. A side effect of including snapshots in your release repository is that a snapshot is always considered newer than a release and unless the pom specifies a specific version people will be using (potentially unstable) snapshot versions.

In the WEB_ROOT/maven2_repositories directory (which your sysadmin has already created) create these directories:

- inhouse
- inhouse_snapshot
- external_free
- external_non_free

If you are on a Unix machine ensure that the WEB_ROOT/maven2_repositories directory has g+ws permission (the s bit ensures that files and directories created will have the same group access control). Recommend that a unix group called "maven" be created and that these directories be owned by that group. Anyone who requires deploy access to these repositories should be included in the "maven" group.

Configure settings.xml

Now that the repositories have been created, each **deployer**'s settings.xml file will need to specify these repositories in the server section. **Developer**'s should **NOT** have these definitions in their settings.xml file.

Once you have Continuous Integration up and running, then developer's should no longer have permissions to deploy to the inhouse_snapshot repository as this should be happening every time continuous integration runs.

```
<servers>
    ...
        <server>
            <id>inhouse</id>
            <username>DEVELOPERS_USERNAME</username>
            <privateKey>PATH/TO/SSH.key</privateKey>
            <filePermissions>664</filePermissions>
            <directoryPermissions>775</directoryPermissions>
        </server>

        <server>
            <id>inhouse_snapshot</id>
            <username>DEVELOPERS_USERNAME</username>
            <privateKey>PATH/TO/SSH.key</privateKey>
            <filePermissions>664</filePermissions>
            <directoryPermissions>775</directoryPermissions>
        </server>

        <server>
            <id>external_free</id>
            <username>DEVELOPERS_USERNAME</username>
            <privateKey>PATH/TO/SSH.key</privateKey>
            <filePermissions>664</filePermissions>
            <directoryPermissions>775</directoryPermissions>
        </server>

        <server>
            <id>external_non_free</id>
            <username>DEVELOPERS_USERNAME</username>
            <privateKey>PATH/TO/SSH.key</privateKey>
            <filePermissions>664</filePermissions>
            <directoryPermissions>775</directoryPermissions>
        </server>
    ....
    </servers>
```

Deploy artifacts into a repository

Deploying inhouse artifacts

Inhouse artifacts should have the correct sections of the pom completed so that the deploy goal will work without having to specify any extra arguments.

```
...
    <distributionManagement>
        <repository>
            <id>inhouse</id>
            <name>Inhouse Internal Release Repository</name>
            <url>

scp://NUCLEUS/PATH/TO/WEB_ROOT/maven2_repositories/inhouse</url>
        </repository>
        <snapshotRepository>
            <id>inhouse_snapshot</id>
            <name>Inhouse Internal Snapshot Repository</name>
            <url>

scp://NUCLEUS/PATH/TO/WEB_ROOT/maven2_repositories/inhouse_snapshot</url>
            <uniqueVersion>true</uniqueVersion>
        </snapshotRepository>
    </distributionManagement>
...
```

The "uniqueVersion" setting defaults to true and will mean that the uploaded snapshot will look like:

```
<artifactId>-<version>-<yyyyMMdd.HHmmss>-<buildNumber>
```

instead of just

```
<artifactId>-<version>
```

Deployment should be done via Continuous Integration, but until you get that setup you can do it manually.

To deploy the artifact run: (Note: use -N if you don't want maven to run in recursive mode)

```
mvn deploy
```

Deploying external free or external non free artifacts

External free (ones not available from ibiblio because of annoying license restrictions) and external non free artifacts can be deployed to their respective repository.

Since these are normally well versioned jar files there is no need for any snapshots in these repositories.

Additional documentation on the deploy goal is available at `http://maven.apache.org/plugins/maven-deploy-plugin/deploy-file-mojo.html`.

Consider the example of deploying the WebLogic jars. The WebLogic jar is an external non free jar and therefore needs to be deployed into the external_non_free repository.
Assume that the version of WebLogic is 8.1sp5 and the weblogic.jar is in the current directory then use the following command:

Note: The trailing slashes are used to represent the command continues onto the next line. This works fine on Unix but not on Windows. You will need to ensure that this command is all on one line with the trailing slashes removed.

```
mvn deploy:deploy-file -DgroupId=weblogic -DartifactId=weblogic
-Dversion=8.1.5 \
  -DgeneratePom=true \
  -Dpackaging=jar \
  -Dfile=weblogic.jar \
  -DrepositoryId=external_non_free \

-Durl=scp://NUCLEUS/PATH/TO/WEB_ROOT/maven2_repositories/external_non_free
```

Defining repositories locations

Here are the options available to you for defining your repository locations.

- Define all repositories in settings.xml (maven contacts all repostories defined in settings.xml)
- Define all repositories in pom.xml (inheritance boostrap problem)
- Define a bootstrap repository definition in settings.xml profile pointing to your maven proxy (preferred)

By defining these repositories in the settings.xml they will automatically get checked for artifacts. An alternative is to define these directly in your project's pom.xml file.

Using the pom.xml is a better choice as the location of repositories is knowledge the project should have and therefore should be located in pom.xml and not settings.xml. However once you start using project inheritance and the repository definitions are located in the parent pom.xml then you will run into the problem that maven can no longer find your internal artifacts. See http://www.nabble.com/RE%3A-maven-proxy-and-snapshots-problem-t1111404.html#a2943362 for a coherent discussion of this problem.

In the discussion that follows it is assumed that these repositories do not contain any Maven plugins. If plugins are needed then the pluginRepositories section will need to be completed with appropriate details.
Inhouse Corporate plugins are likely to be the only ones needing a repository definition.

Define all repositories in settings.xml (maven contacts all repostories defined in settings.xml)

In this configuration each developer needs to maintain the list of repositories in their settings.xml file.
What you will notice as you run maven commands is that each repository is contacted to see if an artifact is available. While this spam information is not harmful it gets distracting and does slow down the discovery of artifacts.

In the following setup the profile "repositoryDefinitions" is made active, but "inhouseSnapshot" is de-active by default. If you want your project to be built against snapshot versions then activate the profile on the mvn command line via "-PinhouseSnapshot", or altenatively include inhouseSnapshot in repositoryDefinitions.

```
...
    <profiles>
        <profile>
            <id>repositoryDefinitions</id>
            <repositories>
                <repository>
                    <id>inhouse</id>
                    <name>Inhouse Repository</name>
                    <url>http://NUCLEUS/maven2_repositories/inhouse</url>
                    <releases>
                        <enabled>true</enabled>
                    </releases>
                    <snapshots>
                        <enabled>false</enabled>
                    </snapshots>
                </repository>
                <repository>
```

```
                    <id>external_free</id>
                    <name>External Free Repository</name>

<url>http://NUCLEUS/maven2_repositories/external_free</url>
                    <releases>
                        <enabled>true</enabled>
                    </releases>
                    <snapshots>
                        <enabled>false</enabled>
                    </snapshots>
                </repository>
                <repository>
                    <id>external_nonfree</id>
                    <name>External Non-Free Repository</name>

<url>http://NUCLEUS/maven2_repositories/external_non_free</url>
                    <releases>
                        <enabled>true</enabled>
                    </releases>
                    <snapshots>
                        <enabled>false</enabled>
                    </snapshots>
                </repository>
            </repositories>
        </profile>
        <profile>
            <id>inhouseSnapshot</id>
            <repositories>
                <repository>
                    <id>inhouse_snaphot</id>
                    <name>Inhouse Snapshot Repository</name>

<url>http://NUCLEUS/maven2_repositories/inhouse_snapshot</url>
                    <releases>
                        <enabled>false</enabled>
                    </releases>
                    <snapshots>
                        <enabled>true</enabled>
                        <updatePolicy>always</updatePolicy>
                    </snapshots>
                </repository>
            </repositories>
        </profile>
        ...
    </profiles>
  ...
  <activeProfiles>
```

```
    <activeProfile>repositoryDefinitions</activeProfile>
  </activeProfiles>
```

Define all repositories in pom.xml (inheritance boostrap problem)

In this configuration all pom files need to include the repository definitions.

In your project's pom.xml include the following:

```
<repositories>
        <repository>
            <id>inhouse</id>
            <name>Inhouse Release Repository</name>
            <url>http://NUCLEUS/maven2_repositories/inhouse</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>external_free</id>
            <name>External Free Repository</name>
            <url> http://NUCLEUS/maven2_repositories/external_free</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>external_nonfree</id>
            <name>External Non-Free Repository</name>
            <url>
                http://NUCLEUS/maven2_repositories/external_non_free</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>inhouse_snaphot</id>
            <name>Inhouse Snapshot Repository</name>
            <url> http://NUCLEUS/maven2_repositories/inhouse_snapshot</url>
            <releases>
                <enabled>false</enabled>
            </releases>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </repository>
    </repositories>
```

This situation described below will occur when you checkout the module separately and the parent directory does not contain the parent pom (as maven will by default also check the parent directory for the parent pom). In this situation your maven build will fail because it will not be able to located the parent pom.
Therefore **ALL** pom.xml files will need this, including module pom.xmls.

That is, if your module includes a reference to a parent and the parent defines the repositories, then the module pom.xml must **ALSO** define the repositories.
e.g:

```
<parent>
        <groupId>GROUP_ID</groupId>
        <artifactId>PARENT_POM_ID</artifactId>
        <version>1.0</version>
    </parent>
    ...
    <!-- Repository definitions need to be included as well -->
```

Define a bootstrap repository definition in settings.xml profile pointing to your maven proxy (preferred)

In this configuration you must configure settings.xml to include the repository for use as a bootstrap repository, create and deploy a boostrap pom.xml, and use the bootsrap pom as the parent for all root projects.

The bootstrap definition is included as a profile and is only needed when your environment is fresh and clean (since you have only checked out a module and the parent pom and the bootstrap pom are not also checked out and also not in your local repository). Once the bootstrap profile has been used to obtain the parent and boostrap poms the profile does not need to be active anymore.

Modify settings.xml

```
<profiles>
        <!-- profile
        | Specifies a set of introductions to the build process, to be
activated using one or more of the
        | mechanisms described above. For inheritance purposes, and to
activate profiles via <activatedProfiles/>
        | or the command line, profiles have to have an ID that is unique.
        |
        | An encouraged best practice for profile identification is to use
a consistent naming convention
        | for profiles, such as 'env-dev', 'env-test', 'env-production',
'user-jdcasey', 'user-brett', etc.
        | This will make it more intuitive to understand what the set of
introduced profiles is attempting
        | to accomplish, particularly when you only have a list of profile
id's for debug.
        -->
        <profile>
            <id>bootstrap<id>
            <repositories>
                <repository>
                    <id>inhouse</id>
                    <name>Inhouse Repository</name>
                    <url>http://NUCLEUS/maven2_repositories/inhouse</url>
                    <releases>
                        <enabled>true</enabled>
                    </releases>
                    <snapshots>
                        <enabled>false</enabled>
```

```
                    </snapshots>
                </repository>
                <repository>
                    <id>inhouse_snapshot</id>
                    <name>Inhouse Snapshot Repository</name>

<url>http://NUCLEUS/maven2_repositories/inhouse_snapshot</url>
                    <releases>
                        <enabled>false<enabled>
                    </releases>
                    <snapshots>
                        <enabled>true</enabled>
                    </snapshots>
                </repository>
            </repositories>
        </profile>
    </profiles>

    <!-- activeProfiles
    | List of profiles that are active for all builds.
    |
    -->
```

```
    <activeProfiles>
    </activeProfiles>
```

Create bootstrap pom.xml

Note that each time you make a change to the bootstrap pom.xml instead of using a version of the form major.minor.patch just use a sequential number. The normal version format assumes backward compatibility or breaking changes and these assumption do not make sense.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>bootstrap</groupId>
    <artifactId>bootstrap-pom</artifactId>
    <packaging>pom</packaging>
    <name>Bootstrap POM</name>
    <version>1</version>
    <description>A bootstrap pom.xml is needed for maven to resolve the
following problem.
        On a new machine there are no artifacts in the local repository and
everything must be
        obtained from the remote repository. A module pom does not contain
any repository
        definitions as these are contained within the parent pom. When
maven attempts to
        retrieve the parent pom, since there are no additional repository
definitions, it
        assumes the artifact is located on the "central" repository and
retrieval fails.
        Repository definitions can be located in either settings.xml or the
pom.xml.
        Placing the repository definitions in the settings.xml file
increases the
        maintenance burden as every developer must ensure this file is kept
up to date. To
        minimize this maintenance all the repository definitions needed are
contained
        within a bootstrap pom which all project's main pom will depend
upon. Then only the
        repository location for the bootstrap artifact is needed in each
developer's
        settings.xml file and can be turned on as needed via
profiles.</description>
    <repositories>
        <repository>
            <id>inhouse</id>
            <name>Inhouse Release Repository</name>
            <url>http://NUCLEUS/maven2_repositories/inhouse</url>
            <releases>
                <enabled>true</enabled>
            </releases>
```

```
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>external_free</id>
            <name>External Free Repository</name>
            <url> http://NUCLEUS/maven2_repositories/external_free</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>external_nonfree</id>
            <name>External Non-Free Repository</name>
            <url>
                http://NUCLEUS/maven2_repositories/external_non_free</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>inhouse_snaphot</id>
            <name>Inhouse Snapshot Repository</name>
            <url> http://NUCLEUS/maven2_repositories/inhouse_snapshot</url>
            <releases>
                <enabled>false</enabled>
            </releases>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </repository>
    </repositories>
    <distributionManagement>
        <repository>
            <id>inhouse</id>
            <name>Inhouse Internal Release Repository</name>
            <url>

scp://NUCLEUS/PATH/TO/WEB_ROOT/maven2_repositories/inhouse</url>
        </repository>
```

```
    </distributionManagement>
</project>
```

Deploy the bootstrap pom

Before the bootstrap pom can be used it needs to be deployed to your internal repositories.
Use:

```
mvn deploy
```

Include boostrap pom.xml as parent into your project's pom.xml

In your parent pom.xml, not your module's pom.xml, include a reference to the boostrap pom as the parent.

```
<parent>
        <groupId>bootstrap</groupId>
        <artifactId>bootstrap-pom</artifactId>
        <version>1</version>
    </parent>
```

Activating the bootstrap profile

If you have a clean local repository and only the module checked out (so that the parent pom is not available in the parent directory), then you must activate the bootstrap profile so that maven knows the repositories it can contact for obtaining the deployed pom files (your parent pom and bootstrap:bootstrap-pom).

Turning on the bootstrap profile is done via the -P command followed by a list of the profiles that should be enabled, e.g.:

```
mvn -Pbootstrap install
```

Once you have the parent pom and bootstrap pom in your local repository there is no need to activate the bootstrap profile again.

Advanced Repositories (Using Internally Released versions of Snapshot Plugins)

At some stage you may find a bug in Maven or may wish to use unreleased features of a plugin that are available in a snapshot version and you need to control which plugin snapshots you want to use instead of using the snapshot version of all plugins.

To do this you will need to create a plugin repository and deploy into it the plugins that you want control over. Be aware that when the released versions of these plugins become available that they will obsolete the snapshot version and if the released version does not contain the bug fixes you require then you will need to redeploy the snapshot again (after patching from the latest code base)

Update your parent pom with:

```
<pluginRepositories>
        <pluginRepository>
            <id>internal_plugins</id>
            <name>Internal Plugin Repository</name>
            <url>http://NUCLEUS/maven2_repositories/internal_plugins</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
    </pluginRepositories>
```

And follow the instructions at Patching Maven Plugins for the steps on how to internally release the snapshot plugins

Conclusion

After following this guide you should have four internal repositories configured for your corporate environment:

- inhouse
- inhouse_snapshot
- external_free
- external_non_free

You can deploy your snapshot artifacts to the inhouse_snapshot repository, your released artifacts to the inhouse repository and populate the external_free and external_non_free repositories with jar files as needed.