SCALA WITH MAVEN

By Adrian Null

Introduction

Maven is a build/project management tool. It favours "convention over configuration"; it can greatly simplify builds for "standard" projects and a Maven user can usually understand the structure of another Maven project just by looking at its `pom.xml` (Project Object Model). Maven is a plugin-based architecture, making it easy to add new libraries and modules to existing projects. For example, adding a new dependency usually involves only 5 extra lines in the `pom.xml`. These "artifacts" are downloaded from repositories such as The Central Repository.

You can also check out the official example project which uses the same Scala plugin we will show here.

Jumping Ahead

If you're familiar with Maven, you can go ahead with the Scala Maven Plugin.

The Scala Maven Plugin

We'll be using the Scala Maven Plugin (GitHub repo, website) (formerly known as the maven-scala-plugin; renamed to honour the new naming policy where only Maven core

plugins are prefixed with "maven"), by far the dominant plugin for Scala projects. *Note: the plugin includes Scala from the Central Repository so there's no need to install it yourself if you're compiling with Maven.*

Getting Maven

Linux (Debian)

On Debian and Debian-derivatives, Maven is usually available via `apt-get`. Just do `(sudo) apt-get install maven` and you're good to go.

OSX

OSX prior to 10.9 (Mavericks) comes with Maven 3 built in. If you don't have it, you can get it with the package managers MacPorts, Homebrew, or Fink. The Scala Maven Plugin requires Maven 3.0+

Manually (Red Hat Linux, OSX, Windows)

You can download Maven from its Apache homepage. After extracting it (`tar -zxvf apache-maven-X.X.X-bin.tar.gz` , or use something like 7-zip) to your directory of choice (on Linux and OSX, Unix-like systems, I like to put them in `/opt/` . On Windows I would probably put this in `C:/`), you need to add Maven to your environment Path variable:

- Linux/OSX (option 1): Create a symlink to `/usr/bin` , which is already on your Path
  - `ln -s /usr/bin/mvn /opt/apache-maven-X.X.X/bin/mvn`
- Linux/OSX (option 2): Add the Maven `bin` folder directly to your path, using your shell configuration file (e.g. `~/.bash_profile`)
  - Add `export PATH=$PATH:/opt/apache-maven-X.X.X/bin` to `.bash_profile` (or whatever profile for the shell you use)
  - Example: `echo "export PATH=$PATH:/opt/apache-maven-X.X.X/bin" >> ~/.bash_profile`
- Linux/OSX (option 3): Make a `mvn` shell script in an existing path location
  - Example: you have `$HOME/bin` in your path
  - Put the folder you extracted in `$HOME/bin` (`mv apache-maven-X.X.X "$HOME/bin/"`)
  - Create a file `mvn` in `$HOME/bin`
  - Add `"$HOME/bin/apache-maven-X.X.X/bin/mvn" $@` to it, and `chmod u+x mvn` to make it executable
  - This is probably the least intrusive way; `$HOME/bin` is usually added to the user's path by default, and if not, it's a useful thing to do/have anyways. The shell script simply invokes the Maven location (which is at some other location) and passes on the arguments

- Windows
  - Hit Start. Right click on “My Computer” and go to “Properties”
  - This should bring you to “Control Panel -> System and Security -> System”, giving an overview of your computer
  - On the left sidebar there should be four options; click on “Advanced system settings” (fourth one)
  - Under the “Advanced” tab, hit “Environment Variables…” in the bottom right
  - Note: I recommend creating/editing your User variables (top box). You can do the same with System variables though (bottom box)
  - Create a new variable called “MAVEN3_HOME”. Point this to your Maven folder (e.g. `C:\apache-maven-X.X.X`). Use backslashes to be safe, and do not include a trailing slash
  - Create a new variable called “MAVEN3_BIN” with this value: `%MAVEN3_HOME%\bin`
  - Edit your Path variable: being careful not to change anything else, append `;%MAVEN3_BIN%` to it
  - You’ll need to restart cmd to see these changes

Creating Your First Project

The easiest way to create new projects is using an “archetype”. An archetype is a general skeleton structure, or template for a project. Think back to “convention over configuration”; in our case, the Scala Maven Plugin provides an archetype for scala projects.

You run the archetype plugin like this:

```
mvn archetype:generate -DarchetypeGroupId=net.alchim31.maven -Darchetype
```

If this is your first time, you’ll notice that Maven is downloading many jar files. Maven resolves dependencies and downloads them as needed (and only once). Right now, Maven is downloading its core plugins.

Next, Maven will ask you for a groupId, artifactId, and package. You can read the guide to naming conventions, but in short:

- groupId: inverted domain name (e.g. com.my-organization)
- artifactId: project name (e.g. playground-project)

- version: anything you want, but I recommend you read and follow the guidelines for Semantic Versioning (e.g. 0.0.1)
- package: the default is the groupId, but you can change this (e.g. com.my-organization)

The groupId and artifactId together should serve as a globally unique identifier for your project

When it's done, you should see a new folder named with the artifactId. `cd` into it and run:

```
mvn package
```

You'll see Maven downloading dependencies including the Scala library (as mentioned above), JUnit, ScalaTest, and Specs2 (the latter three are test frameworks; the archetype includes an example "Hello world" program, and tests with each of the frameworks).

Explaining this Archetype

In your project root, you'll see a `pom.xml`, `src` folder, and `target` folder (target folder only appears after building). *Note: this archetype also includes a* `.gitignore`

Inside the `src` folder you'll see `main` and `test`; `main` includes your application code, and `test` includes your test suites. Inside each of those you'll find a `scala` folder, followed by your package structure (actually, `test/scala` includes a sample package, but you should replace this with your own package and tests). If you want to mix Scala and Java source code, simply add a `java` folder inside `main` or `test`.

`target` includes generated/built files, such as `.class` and `.jar` files. You can read about `pom.xml` at the Maven page.

Example structure:

- pom.xml
- src
  - main
    - scala
      - com/my-package/... *.scala
    - java

        - com/my-package/... *.java
    - test
        - scala
            - com/my-package/... *.scala
        - java
            - com/my-package/... *.java
- target ...

Again, you can read more about the Scala Maven Plugin at its website.

Creating a Jar

By default the jar created by the Scala Maven Plugin doesn't include a `Main-Class` attribute in the manifest. I had to add the Maven Assembly Plugin to my `pom.xml` in order to specify custom attributes in the manifest. You can check the latest version of this plugin at the project summary or at The Central Repository

```
<project ...>
    <modelVersion>X.X.X</modelVersion>
    ...
    <licenses>
        ...
    </licenses>

    <properties>
        ...
    </properties>

    <dependencies>
        ...
    </dependencies>

    <build>
        ...
        <plugins>
            ...
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>2.4</version>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptor
                    </descriptorRefs>
                    <archive>
```

```
                        <manifest>
                            <mainClass>com.your-package.MainClass</mainC
                        </manifest>
                    </archive>
                </configuration>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

After adding this, `mvn package` will also create `[artifactId]-[version]-jar-with-dependencies.jar` under `target`. *Note: this will also copy the Scala library into your Jar. This is normal. Be careful that your dependencies use the same version of Scala, or you will quickly end up with a massive Jar.*

Useful commands

- `mvn dependency:copy-dependencies` : copy all libraries and dependencies to the `target/dependency` folder
- `mvn clean`
- `mvn package` : compile, run tests, and create jar

Integration with Eclipse (Scala IDE)

There are instructions at the Scala Maven Plugin FAQs, but I thought I'd expand a bit. The maven-eclipse-plugin is a core plugin (all plugins prefixed with "maven" are, and are available by default) to generate Eclipse configuration files. However, this plugin does not know about our new Scala source files. We'll be using the build-helper-maven-plugin to add new source folders:

```
...
<plugin>

    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <executions>
        <execution>
```

```
            <id>add-source</id>
            <phase>generate-sources</phase>
            <goals>
                <goal>add-source</goal>
            </goals>
            <configuration>
                <sources>
                    <source>src/main/scala</source>
                </sources>
            </configuration>
        </execution>
        <execution>
            <id>add-test-source</id>
            <phase>generate-sources</phase>
            <goals>
                <goal>add-test-source</goal>
            </goals>
            <configuration>
                <sources>
                    <source>src/test/scala</source>
                </sources>
            </configuration>
        </execution>
    </executions>
</plugin>
...
```

After adding that to your `pom.xml`:

1. Run `mvn eclipse:eclipse` - this generates the Eclipse project files (which are already ignored by our archetype's `.gitignore`)
2. Run `mvn -Declipse.workspace="path/to/your/eclipse/workspace" eclipse:configure-workspace` - this adds an `M2_REPO` classpath variable to Eclipse
3. Run `mvn package` to ensure you have all the dependencies in your local Maven repo

Unfortunately, the integration isn't perfect. Firstly, open up the generated `.classpath` file (it will be hidden by default as it's a dotfile, but it should be in your project root directory; where you ran `mvn eclipse:eclipse`). You should see something like this near the top.

```
<classpathentry kind="src" path="src/test/scala" output="target/test-cla
<classpathentry kind="src" path="src/main/scala" including="**/*.java"/>
```

Change the `*.java` to `*.scala` (or duplicate the lines and change them to `*.scala` if you also have Java sources).

Secondly, open the `.project` eclipse file (again, in the same folder). Change `<buildSpec>` and `<natures>` to look like this. Now Eclipse knows to use the Scala editor and it won't think that everything is a syntax error.

```
<buildSpec>
  <buildCommand>
    <name>org.scala-ide.sdt.core.scalabuilder</name>
  </buildCommand>
</buildSpec>
<natures>
  <nature>org.scala-ide.sdt.core.scalanature</nature>
  <nature>org.eclipse.jdt.core.javanature</nature>
</natures>
```

Finally, in Eclipse, under "File" choose "Import..." and find your project.

Using m2eclipse-scala for Eclipse integration

m2eclipse-scala is a work in progress, and their website/repository may have updated information. It aims to ease integration between m2eclipse and Scala IDE for Eclipse.

Under "Help -> Install New Software", enter "https://alchim31.free.fr/m2e-scala/update-site" and hit enter. You should see "Maven Integration for Eclipse -> Maven Integration for Scala IDE".

After it installs, go to "New -> Project -> Other" and select "Maven Project". Search fo "scala-archetype" choose the one with the group "net.alchim31.maven". The wizard will more or less guide you through what was done with `mvn archetype:generate` above, and you should end up with a new Scala project!

To import an existing project, simply go to "File -> Import -> Existing Maven Project" and find the directory containing your project.

Adding Dependencies

The first thing I do is look for "Maven" in the project page. For example, Google's [Guava] page includes Maven Central links. As you can see in the previous link, The Central

Repository conveniently includes the snippet you have to add to your `pom.xml` on the left sidebar.

If you can't find Maven information at the project page, try a Google search for "[project

If you can't find Maven information at the project page, try a Google search for "[project name] maven". Sometimes, you still won't find anything. For scopt (Scala command line option parser), I couldn't find the latest version from Google. However, manually searching The Central Repository did

Afterwards, running

```
mvn package
```

Will download any new dependencies before packaging

Other Useful Reading

I'm not going to explain all of Maven in this tutorial (though I hope to add more in the future, because I do feel that the resources are a bit scattered), so here are some useful articles:

- Maven Lifecycle (explanation of goals like clean, package, install)

Contributors to this page:

 oswaldo  SethTisue  komainu8  ashawley  Philippus

 OlivierBlanvillain  heathermiller