What is Maven

CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents

CLARUSWAY©
WAY TO REINVENT YOURSELF

1 Introduction to Maven

CLARUSWAY©
WAY TO REINVENT YOURSELF

Introduction to Maven

- First, it was used at **Apache's Jakarta Alexandria Project** in 2001
- What Maven did was to **simplify the build processes**

Introduction to Maven

- As a project management tool, Maven :
  - builds **multiple projects** easily,
  - **publishes documentation** for the projects,
  - accomplishes an **easy deployment**,
  - **helps in collaboration** with development teams.

Introduction to Maven

- Maven can :
  - **manage the versions** of consecutive builds,
  - **compile** source code into binary,
  - **download dependencies**,
  - **run tests**,
  - **package** compiled code
  - **deploy** artifacts

2 Features of Maven

CLARUSWAY©
WAY TO REINVENT YOURSELF

Features of Maven

- **Easy to start** with Maven
- Variety of **options**
- **Same structure** across different projects
- **Easy to integrate** into a developing team
- It has a **powerful dependency management tool**
- **Large repository** of libraries

CLARUSWAY©
WAY TO REINVENT YOURSELF

Features of Maven

- **Extra features** with plugins
- **Different outputs** like a **jar**, **ear** or **war**
- Maven can **generate a website**
- Maven can **support the older versions**

3 Directory Structure

Directory Structure

- Project structure **should conform** to
- The most important file is the **pom file**
  - defines project's **config details**

```
└──maven-project
    ├──pom.xml
    ├──README.txt
    ├──NOTICE.txt
    ├──LICENSE.txt
    └──src
        ├──main
        │   ├──java
        │   ├──resources
        │   ├──filters
        │   └──webapp
        ├──test
        │   ├──java
        │   ├──resources
        │   └──filters
        ├──it
        ├──site
        └──assembly
```


POM File

Table of Contents

1 Introduction to POM File

CLARUSWAY©
WAY TO REINVENT YOURSELF

Introduction to POM File

- It is an XML file
- **P**roject **O**bject **M**odel is the **starting point** for a Maven project
- It contains **configurations** about the project
- When a task or goal is executed, **Maven searches for** the **POM file**

Introduction to POM File

- POM defines
  - Project **dependencies**
  - Plugins and **goals** to be executed
  - **Build profiles**
  - Other information like the **project version**, **description**, **developers**, **mailing lists**, and more...

Introduction to POM File

- There **must** be a POM file in every Maven project
- **All POMs need** at least
  - Project tag
  - modelVersion tag
  - **g**roupId tag
  - **a**rtifactId tag
  - **v**ersion (Last three called as **gav** in short)

```
<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/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.companyname.project-group</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>
</project>
```


Introduction to POM File

- **Project tag** is the **root** of the file
- It should reference a basic schema settings such as **apache schema** and **w3.org** specification

```
<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/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.companyname.project-group</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>
</project>
```

- **Model version** describes the **version of Maven**
- **Group Id** is the id of the project's group (Simply it shows the **company** or the **organization** or the **owner of the project**)

Introduction to POM File

- **Group Id** should be long enough to give **uniqueness** to the project
- **Artifact id** is the id for specifying the project under the group

```
<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/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.companyname.project-group</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>
</project>
```

- It shows the **name of the project** like pet-clinic-server
- **Version** defines the version number of the project

2 Super POM

Super POM

- Super POM is Maven's **default POM**
- **All POMs extend** the Super POM **unless explicitly set**
- Super POM and project POM creates the **Effective POM**
- Which is the **overall configuration** file
- Effective POM can be examined by running

  "**mvn help:effective-pom**"

- Effective POM -

```
<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/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.companyname.project-group</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>
   <build>
      <sourceDirectory>C:\MVN\project\src\main\java</sourceDirectory>
      <scriptSourceDirectory>src/main/scripts</scriptSourceDirectory>

      <testSourceDirectory>C:\MVN\project\src\test\java
         </testSourceDirectory>
      <outputDirectory>C:\MVN\project\target\classes</outputDirectory>
      <testOutputDirectory>C:\MVN\project\target\test-classes
         </testOutputDirectory>
      <resources>
         <resource>
            <mergeId>resource-0</mergeId>
            <directory>C:\MVN\project\src\main\resources</directory>
         </resource>
      </resources>
      <testResources>
         <testResource>
            <mergeId>resource-1</mergeId>
            <directory>C:\MVN\project\src\test\resources</directory>
         </testResource>
      </testResources>
      <directory>C:\MVN\project\target</directory>
      <finalName>project-1.0</finalName>
```

1

```
<pluginManagement>
   <plugins>
      <plugin>
         <artifactId>maven-antrun-plugin</artifactId>
         <version>1.3</version>
      </plugin>
      <plugin>
         <artifactId>maven-assembly-plugin<  /artifactId>
         <version>2.2-beta-2</version>
      </plugin>
      <plugin>
         <artifactId>maven-clean-plugin<  /artifactId>
         <version>2.2</version>
      </plugin>

      <plugin>
         <artifactId>maven-compiler-plugin</artifactId>
         <version>2.0.2</version>
      </plugin>
      <plugin>
         <artifactId>maven-dependency-plugin</artifactId>
         <version>2.0</version>
      </plugin>
      <plugin>
         <artifactId>maven-deploy-plugin</artifactId>
         <version>2.4</version>
      </plugin>

      <plugin>
         <artifactId>maven-ear-plugin</artifactId>
         <version>2.3.1</version>
      </plugin>
      <plugin>
         <artifactId>maven-ejb-plugin</artifactId>
         <version>2.1</version>
      </plugin>
```

2

```
   <plugin>
      <artifactId>maven-install-plugin</artifactId>
      <version>2.2</version>
   </plugin>

   <plugin>
      <artifactId>maven-jar-plugin</artifactId>
      <version>2.2</version>
   </plugin>
   <plugin>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>2.5</version>
   </plugin>
   <plugin>
      <artifactId>maven-plugin-plugin</artifactId>
      <version>2.4.3</version>
   </plugin>

   <plugin>
      <artifactId>maven-rar-plugin</artifactId>
      <version>2.2</version>
   </plugin>
   <plugin>
      <artifactId>maven-release-plugin</artifactId>
      <version>2.0-beta-8</version>
   </plugin>
   <plugin>
      <artifactId>maven-resources-plugin</artifactId>
      <version>2.3</version>
   </plugin>

   <plugin>
      <artifactId>maven-site-plugin</artifactId>
      <version>2.0-beta-7</version>
   </plugin>
```

3

- Effective POM -

```
        <plugin>
          <artifactId>maven-source-plugin</artifactId>
          <version>2.0.4</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.4.3</version>
        </plugin>
        <plugin>
          <artifactId>maven-war-plugin</artifactId>
          <version>2.1-alpha-2</version>
        </plugin>
      </plugins>
    </pluginManagement>

    <plugins>
      <plugin>
        <artifactId>maven-help-plugin</artifactId>
        <version>2.1.1</version>
      </plugin>
    </plugins>
  </build>

  <repositories>
    <repository>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
      <id>central</id>
      <name>Maven Repository Switchboard</name>
      <url>http://repo1.maven.org/maven2</url>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <releases>
        <updatePolicy>never</updatePolicy>
      </releases>
```

4

```
      <plugins>
        <plugin>
          <artifactId>maven-help-plugin</artifactId>
          <version>2.1.1</version>
        </plugin>
      </plugins>
    </build>

    <repositories>
      <repository>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
        <id>central</id>
        <name>Maven Repository Switchboard</name>
        <url>http://repo1.maven.org/maven2</url>
      </repository>
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <releases>
          <updatePolicy>never</updatePolicy>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
        <id>central</id>
        <name>Maven Plugin Repository</name>
        <url>http://repo1.maven.org/maven2</url>
      </pluginRepository>
    </pluginRepositories>
    <reporting>
      <outputDirectory>C:\MVN\project\target/site</outputDirectory>
    </reporting>
</project>
```

5

3 Project Inheritance

CLARUSWAY©
WAY TO REINVENT YOURSELF

Project Inheritance

- As in the object-oriented programming, POM files **can also be inherited** by other POM files
- Child POM can either **inherit** or **override**
- Parent POM is a **general template**
- **Not every item** in the parent is inherited
- Some elements should be declared specifically
- Like **artifactId**, **name**, and **prerequisites**

Project Inheritance

- Parent POM's **packaging** tag should have the value "**pom**"

Parent

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.clarusway.mojo</groupId>
  <artifactId>my-parent</artifactId>
  <version>2.0</version>
  <packaging>pom</packaging>      <!-- NOTICE THE PACKAGING TYPE -->
</project>
```

Project Inheritance

- Child is related to parent **by specifying the parent element**
- If you want to inherit an element you should remove it

Parent

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.clarusway.mojo</groupId>
  <artifactId>my-parent</artifactId>
  <version>2.0</version>
  <packaging>pom</packaging>      <!-- NOTICE
</project>
```

Inherited

Child

```
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>com.clarusway.mojo</groupId>
    <artifactId>my-parent</artifactId>
    <version>2.0</version>
    <relativePath>../my-parent/pom.xml</relativePath>
  </parent>

  <artifactId>my-project</artifactId>
</project>
```


Build Lifecycles

Table of Contents

1 Introduction to Build Lifecycles

CLARUSWAY©
WAY TO REINVENT YOURSELF

Introduction to Build Lifecycles

- A Build Lifecycle is **a track** that is comprised of **different number of phases**
- A phase is a **job unit** or a **specific stage** in a lifecycle

Introduction to Build Lifecycles

- There are **three built-in lifecycles :**
  - **default**, **clean**, and **site**
  - Default is the **main lifecycle**
  - Clean is used for **cleaning the project**
  - Site lifecycle is used for building the **project's website**

Introduction to Build Lifecycles

- Each life cycle has a different number of phases
  - Default build lifecycle has **23**
  - Clean lifecycle has **3**
  - Site lifecycle has **4 phases**

Introduction to Build Lifecycles

- **Using Command-Line :**
  - Maven CLI commands generates your outputs
  - For example,
    - "**mvn package**" gives you a "**jar, war or ear ...**"
    - "**mvn test**" gives your test code's results
    - "**mvn clean**" cleans the artifacts of a previous command

2 Clean Lifecycle

Clean Lifecycle

- Clean Lifecycle has **three phases**
  - **pre-clean**, **clean**, and **post-clean**
- These phases are **in sequence**
- When a phase is called (for example "mvn post-clean"), **phases prior to that phase** are also **run**

Clean Lifecycle

- It cleans the project's target directory
- **Pre-clean** phase is used for **any task prior to** the cleanup
- **Post-clean** phase is used for **any task following** the cleanup

3 Default Lifecycle

Default Lifecycle

- Default lifecycle is used **for application build**
- There are **23 phases** in Default Lifecycle
- The most important phases are :
  - **validate:** validates if the project has necessary information
  - **compile:** compiles the source code
  - **test-compile:** compiles the test source code

Default Lifecycle

- The most important phases are :
  - **test:** runs unit tests
  - **package:** packages compiled source code
    - **packaging tag** in POM.xml changes the output

Default Lifecycle

- The most important phases are :
  - **integration-test:** processes and deploys the package if needed to run integration test
  - **install:** installs the package to local repository
  - **deploy:** copies the package to a remote repository

4 Site Lifecycle

Site Lifecycle

- Site lifecycle has **four phases**
  - **pre-site**, **site**, **post-site**, **site-deploy**
- For Site Lifecycle, the **Site Plugin is used**
- The plugin's **main duty** is to **generate a website**

Build Profiles

1 Introduction to Build Profiles

Introduction to Build Profiles

- A Build profile is a kind of **mechanism for triggering** a set of **build configurations**
- Configurations determine **different build environments** like **production**, **stage**, **test**, or **development** environment

Introduction to Build Profiles

We have two profiles in here!

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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/xsd/maven-4.0.0.xsd">
    <parent>
        <groupId>com.clarusway.maven</groupId>
        <artifactId>profiles</artifactId>
        <version>1.0</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>profile-1</artifactId>
    <profiles>
        <profile></profile>
        <profile></profile>
    </profiles>
    <build>
        <!-- you can map a variable with the  ${} syntax -->
        <resources>
            <resource>
                <directory>src/main/resources</directory>
                <filtering>true</filtering>
            </resource>
        </resources>
    </build>
</project>
```


Introduction to Build Profiles

- First profile is **activated by default unless** another profile in the same POM is activated
- Its **activation property** "**env**" has the value "**dev**"
- Other properties are listed under the **properties tag**

```xml
<profiles>
    <profile>
        <id>dev</id>
        <activation>
            <!-- this profile is active by default -->
            <activeByDefault>true</activeByDefault>
            <!-- activate if system properties 'env=dev' -->
            <property>
                <name>env</name>
                <value>dev</value>
            </property>
        </activation>
        <properties>
            <db.driverClassName>com.mysql.jdbc.Driver</db
                .driverClassName>
            <db.url>jdbc:mysql://localhost:3306/dev</db.url>
            <db.username>clarus</db.username>
            <db.password>123456789</db.password>
        </properties>
    </profile>
    <profile></profile>
</profiles>
```

Introduction to Build Profiles

- Second profile is **not activated by default**
- Its **activation property** "**env**" has the value "**prod**"
- Database url is changed to production db

```
<profiles>
    <profile></profile>
    <profile>
        <id>prod</id>
        <activation>
            <!-- activate if system properties 'env=prod' -->
            <property>
                <name>env</name>
                <value>prod</value>
            </property>
        </activation>
        <properties>
            <db.driverClassName>com.mysql.jdbc.Driver</db
                .driverClassName>
            <db.url>jdbc:mysql://database-1.cdbs4t6jyjpw.us-east-1.rds
                .amazonaws.com:3306/prod</db.url>
            <db.username>clarus</db.username>
            <db.password>123456789</db.password>
        </properties>
    </profile>
</profiles>
```

run to activate => **mvn ..... -Denv=prod**

Introduction to Build Profiles

- All profiles should have a way of activation
- Maven Build Profiles can be activated in **five different ways**
  - Using **explicit** profile **activation**
  - **Maven settings**
  - **System variables**
  - **Operating System** Settings
  - **Present/Missing files**

Repositories

CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents

CLARUSWAY©
WAY TO REINVENT YOURSELF

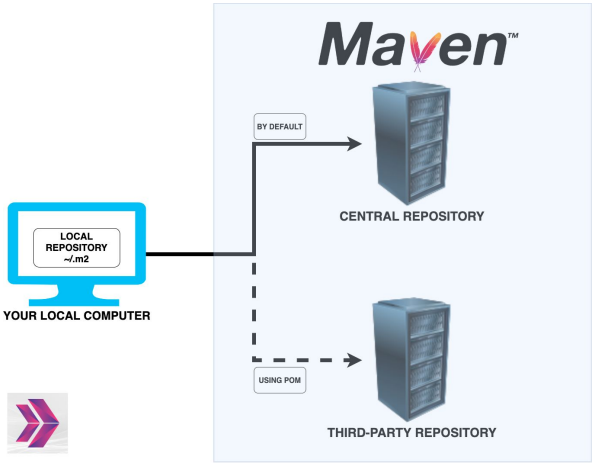
1 Introduction to Repositories

Introduction to Repositories

- Repository is a source where all library **jars**, **plugins**, **dependencies**, or any other project-specific **artifacts** are stored
- While your project runs, **these resources** are **used silently**
- There are **two types** of repositories
  - **Local** and **remote**
- Local repository is **your own computer**

Introduction to Repositories

- **Remote** repository can be **separated into two**
  - **Central** repository and **Third-Party** repository
- **By default central** repository **is used** as the remote repository
- You can also configure to use a third-party repository



2 Local Repository

Local Repository

- As mentioned, local repository is **in your local computer**
- **Maven creates** this directory
- It **continuously develops** it whenever you use a resource from a remote repository

Local Repository

- After adding a resource into your POM file, **Maven automatically downloads** all the dependency jars into your local repository
- It **doesn't reach out** to remote repository **if the resource exists** in local
- By default, local repository is under your Home Directory

Local Repository

> **Tip:** In general, you should not need to do anything with the local repository on a regular basis, except clean it out (~/.m2 directory) if you are short on disk space (or erase it completely if you are willing to download everything again).

3 Central Repository

Central Repository

- Maven central repository is the **default remote repository**
- When Maven **cannot find a dependency** in the local repository, it tries to find it in the central repository
- Central repo is **located in** this url **https://repo.maven.apache.org/maven2/**
- **No configuration** is needed to use the central repo

4 Third-Party Repository

Third-Party Repository

- Central repository is not the only choice
- **Any organization** or **any individual** can host a remote repository
- **You need to configure** it in the POM file

Third-Party Repository

- In the example, third-party repositories are specified under **<repositories>** and **<repository>** fields

```
<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/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.companyname.projectgroup</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>
   <dependencies>
      <dependency>
         <groupId>com.companyname.common-lib</groupId>
         <artifactId>common-lib</artifactId>
         <version>1.0.0</version>
      </dependency>
   </dependencies>
   <repositories>
      <repository>
         <id>companyname.lib1</id>
         <url>http://download.companyname.org/maven2/lib1</url>
      </repository>
      <repository>
         <id>companyname.lib2</id>
         <url>http://download.companyname.org/maven2/lib2</url>
      </repository>
   </repositories>
</project>
```

Plugins

CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents

1 What is a Plugin?

What is a Plugin?

- Plugin is the **heart of Maven** framework
- A **unit work** in Maven or a **single output** is produced by a specific Maven Plugin
- **Some** of the plugins are **bound to** some of the **phases** of Maven Build Lifecycles
- But **some** are **independent**

What is a Plugin?

- Plugins do the works like **creating jar** files, **war** files, **compiling code**, **compiling unit test** code, creating **project documentation** or **JavaDoc** (Java Documentation), and so on
- One of the **simplest plugins** in Maven is the **clean plugin**

What is a Plugin?

- Maven Clean Plugin is responsible for **removing the target directory** of a Maven project
- When you run **mvn clean**, Maven executes the **clean goal** as defined **in** the **clean plug-in**
- Goals in Maven can be executed via the command-line interface within the format specified below :
  - **mvn [plugin-name]:[goal-name]**

What is a Plugin?

- If you want to run **both the clean phase** and compiler plugin's **compile goal**, you should run the command
  - **mvn clean compiler:compile**
- All plugins should have the **minimum requirement** of having the **groupId**, **artifactId**, and **version** elements

2 Types of Plugins

Types of Plugins

- There are **two types** of plugins :
  - **Build** Plugins and **Reporting** Plugins
- Build plugins are configured under **<build>** tag
- They **run** during the **build time**

Types of Plugins

- Reporting Plugins are configured under **<reporting>** tag
- They run while you are **generating the site** for the project
- Maven plugins are configured by specifying a **<configuration>** element

THANKS!

Any questions?

phase-goal-plugin

Maven Build Lifecycle	Phases	Goals	Plugin
	pre-clean		
clean	clean	clean	Clean
	post-clean		

phase-goal-plugin

Phase	plugin:goal
`process-resources`	`resources:resources`
`compile`	`compiler:compile`
`process-test-resources`	`resources:testResources`
`test-compile`	`compiler:testCompile`
`test`	`surefire:test`
`package`	`jar:jar`
`install`	`install:install`
`deploy`	`deploy:deploy`