

# Overall Table of Contents



- ▶ **Build Profiles**
- ▶ **Repositories**
- ▶ **Plugins**

## Build Profiles



# Table of Contents



- ▶ Introduction to Build Profiles
- ▶ Explicit Profile Activation
- ▶ Profile Activation via Maven Settings
- ▶ Profile Activation via System Variables
- ▶ Profile Activation via Operating System
- ▶ Profile Activation via Present/Missing File



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 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**

```
<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><input type="checkbox"/></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><input type="checkbox"/></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**



2

## Explicit Profile Activation



## ► Explicit Profile Activation

- Explicit activation uses “-P” option in a CLI command
- “-P” option requires a comma-delimited list of profile-ids
- **Example :**
  - **mvn groupId:artifactId:goal -P profile-1,profile-2**



## ► Explicit Profile Activation

- In the example :
  - Profile id is “**test**”
  - **Example :** “**mvn package -P test**”  
will execute the profile
- Profile uses **maven-antrun-plugin**
- It copies the “**env.test.properties**” file to  
“**env.properties**” file
- So the app will use the test properties

```
<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>
  <profiles>
    <profile>
      <id>test</id>
      <build>
        <plugins>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-antrun-plugin</artifactId>
            <version>1.1</version>
            <executions>
              <execution>
                <phase>test</phase>
                <goals>
                  <goal>run</goal>
                </goals>
                <configuration>
                  <tasks>
                    <task>
                      <echo>Using env.test.properties</echo>
                      <copy file="src/main/resources/env.test
                        .properties"
                        tofile="${project.build.outputDirectory}
                          /env.properties"/>
                    </task>
                  </tasks>
                </configuration>
              </execution>
            </executions>
          </plugin>
        </plugins>
      </build>
    </profile>
  </profiles>
</project>
```



3

## Profile Activation via Maven Settings



### Profile Activation via Maven Settings

- ▶ When you install Maven, a directory named **“.m2”** is created **under your Home Directory**
- ▶ **“settings.xml”** (user located in that directory
- ▶ If it's not there, you can create one
- ▶ To activate a profile **profile id** should be declared under **<activeProfile>** tag
- ▶ **No need to trigger** the profile

```
1 <settings xmlns = "http://maven.apache.org/POM/4.0.0"
2 xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
3 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4 http://maven.apache.org/xsd/settings-1.0.0.xsd">
5 <mirrors>
6 <mirror>
7 <id>maven.dev.snaponglobal.com</id>
8 <name>Internal Artifactory Maven repository</name>
9 <url>http://repo1.maven.org/maven2/</url>
10 <mirrorOf>*</mirrorOf>
11 </mirror>
12 </mirrors>
13 .
14 .
15 .
16 <activeProfiles>
17 <activeProfile>test</activeProfile>
18 </activeProfiles>
19 </settings>
20
```



# 4

## Profile Activation via System Variables



## Profile Activation via System Variables

- Activation occurs when **system property** is specified with **-D** option

```
1 <profiles>
2   <profile>
3     <activation>
4       <property>
5         <name>debug</name>
6       </property>
7     </activation>
8   .
9   .
10  .
11 </profile>
12 </profiles>
13
```

mvn groupId:artifactId:goal -Ddebug

```
<profiles>
  <profile>
    <activation>
      <property>
        <name>!debug</name>
      </property>
    </activation>
    .
    .
    .
  </profile>
</profiles>
```

mvn groupId:artifactId:goal





## Profile Activation via System Variables

- Activation occurs when **system property** is specified with **-D** option

```
<profiles>
<profile>
  <activation>
    <property>
      <name>debug</name>
      <value>!true</value>
    </property>
  </activation>
  .
  .
</profile>
</profiles>
```

mvn groupId:artifactId:goal -Ddebug=false

mvn groupId:artifactId:goal -Denvironment=test

```
<profiles>
<profile>
  <activation>
    <property>
      <name>environment</name>
      <value>test</value>
    </property>
  </activation>
  .
  .
</profile>
</profiles>
```



5

## Profile Activation via Operating System



## Profile Activation via Operating System

- ▶ It is defined under **<os>** tag
- ▶ In the example **Windows XP** will **trigger** the profile

```
1- <profile>
2-   <id>test</id>
3-   <activation>
4-     <os>
5-       <name>Windows XP</name>
6-       <family>Windows</family>
7-       <arch>x86</arch>
8-       <version>5.1.2600</version>
9-     </os>
10-   </activation>
11- </profile>
12-
```



6

## Profile Activation via Present/Missing File



## Profile Activation via Present/Missing File

- In the example, profile is triggered when **the file** `target/generated-sources/axistools/wsd12java/org/apache/maven` is **missing**

```
1 <profiles>
2   <profile>
3     <activation>
4       <file>
5         <missing>target/generated-sources/axistools/wsd12java/org/apache
          /maven</missing>
6       </file>
7     </activation>
8     ...
9   </profile>
10 </profiles>
11
```



## Repositories



# Table of Contents



- ▶ Introduction to Repository
- ▶ Local Repository
- ▶ Central Repository
- ▶ Third-Party Repository



1

## Introduction to Repositories



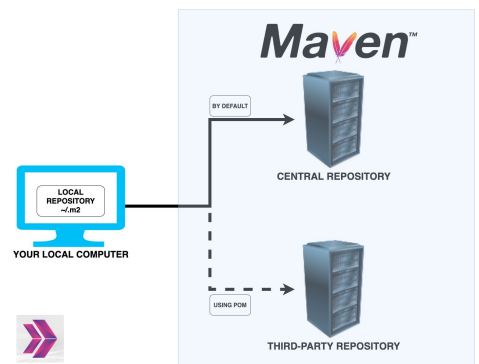
# Introduction to Repositories

- ▶ 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

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7     <groupId>com.companyname.projectgroup</groupId>
8     <artifactId>project</artifactId>
9     <version>1.0</version>
10    <dependencies>
11        <dependency>
12            <groupId>com.companyname.common-lib</groupId>
13            <artifactId>common-lib</artifactId>
14            <version>1.0.0</version>
15        </dependency>
16    </dependencies>
17    <repositories>
18        <repository>
19            <id>companyname.lib1</id>
20            <url>http://download.companyname.org/maven2/lib1</url>
21        </repository>
22        <repository>
23            <id>companyname.lib2</id>
24            <url>http://download.companyname.org/maven2/lib2</url>
25        </repository>
26    </repositories>
27 </project>
```



# Plugins



# Table of Contents



- ▶ What is a Plugin?
- ▶ Types of Plugins



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?

