

# *Construction et gestion de développement avec Maven 3.0*

Bruno Mermet  
novembre 2017



# *Maven : Kesako ?*

- Un outil de construction d'application
    - Génère une application « déployable » à partir d'un code source
    - Compile
    - Exécute des tests
  - Un outil de gestion de développement
    - Gère aussi
      - Documentation & Rapports
      - Site web
      - ...
- 
-

# *Maven : choix sous-jacents*

- Privilégier la standardisation à la liberté (*Convention over Configuration*)
    - Structure standard des répertoires d'une application
    - Cycle de développement standard d'une application
    - Maven se débrouille souvent tout seul !
  - Factoriser les efforts
    - Un dépôt global regroupant les ressources/bibliothèques communes
    - Des dépôts locaux
    - Un dépôt personnel (~/.m2)
  - Multiplier les possibilités
    - Une application légère
    - De nombreux plugins, chargés automatiquement au besoin
- 
-

# Maven : vocabulaire


- **Plugin**  
Extension de l'application de base, proposant un ensemble de buts
  - **But (*Goal*)**  
Tâche proposée par un plugin permettant de lancer un certain nombre d'action lorsqu'il est invoqué par `mvn plugin:but`.  
Paramétré par `-Dparam=valeur`
  - **Phase (*Maven Lifecycle Phase*)**  
Phase du cycle de développement d'un logiciel, généralement associée à des buts et exécutée par `mvn phase`
  - **Artefact (*Artifact*)**  
Application dont le développement est géré via Maven
  - **POM (Project Object Model)**  
Fichier xml décrivant les spécificités du projets (par rapport à `Build.xml`, décrit non pas tout, mais juste le « non standard »)
- 
-

# *Plugins et buts*

## *Exemples*

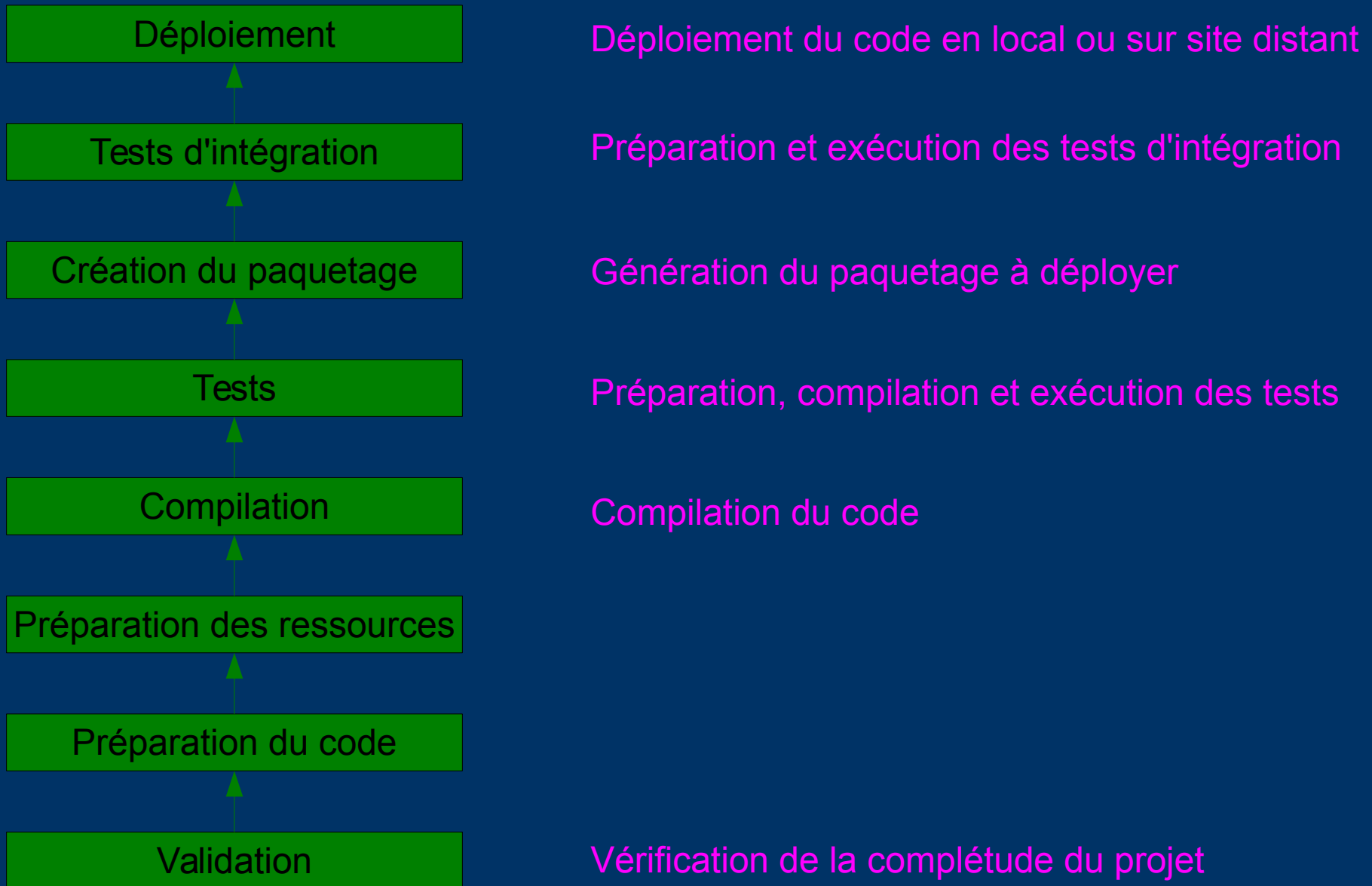
- Plugin archetype
    - Pour créer de nouveaux projets standards
    - Buts : generate, create
  - Plugin compiler
    - Pour la compilation de code java
    - But : compile, test-compile
  - Plugin surefire
    - Pour l'exécution des tests
    - But : test
- 
-

# *Buts et phases*

- Exécution d'un but
  - Exécution du but seulement
- Exécution d'une phase 
  - Exécution de la phase précédente
  - Exécution du but associé



# Étapes du cycle de vie Maven par défaut



# Phases du cycle de vie Maven par défaut

## 1. De la Validation à la compilation

Compilation

process-classes

Post-traite les classes (optimisation du byte-code par exemple)

compile

Compile (!) les sources

Préparation des ressources

process-resources

Copie et traite les ressources pour inclusion dans le paquetage

generate-resources

Génère les ressources à inclure dans le paquetage

Préparation Du code

process-sources

Traite le code source, par exemple pour filtrer certaines valeurs

generate-sources

Génère tout code source nécessaire à la compilation

Validation

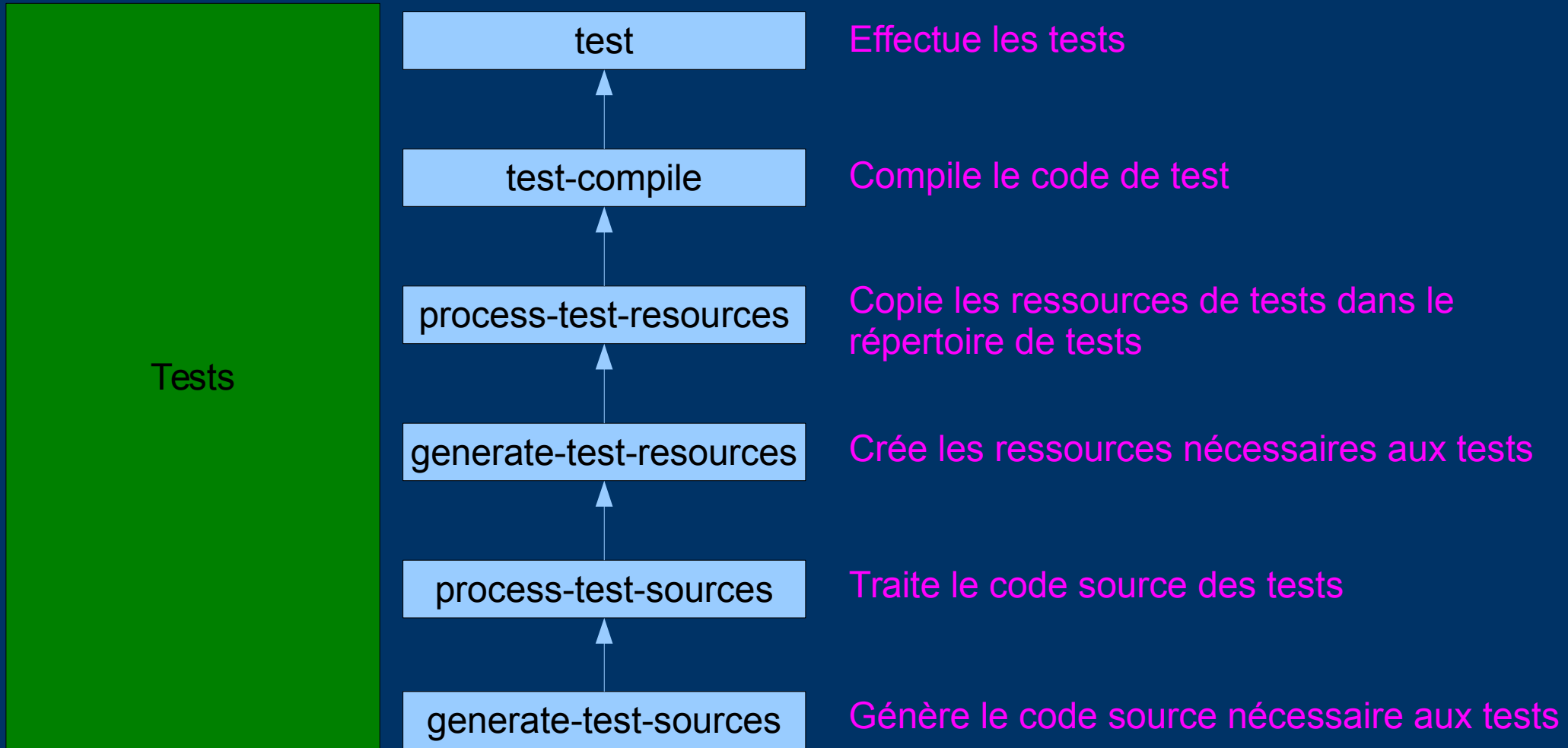
validate

Vérifie que le projet est complet et que toutes les informations requises sont disponibles



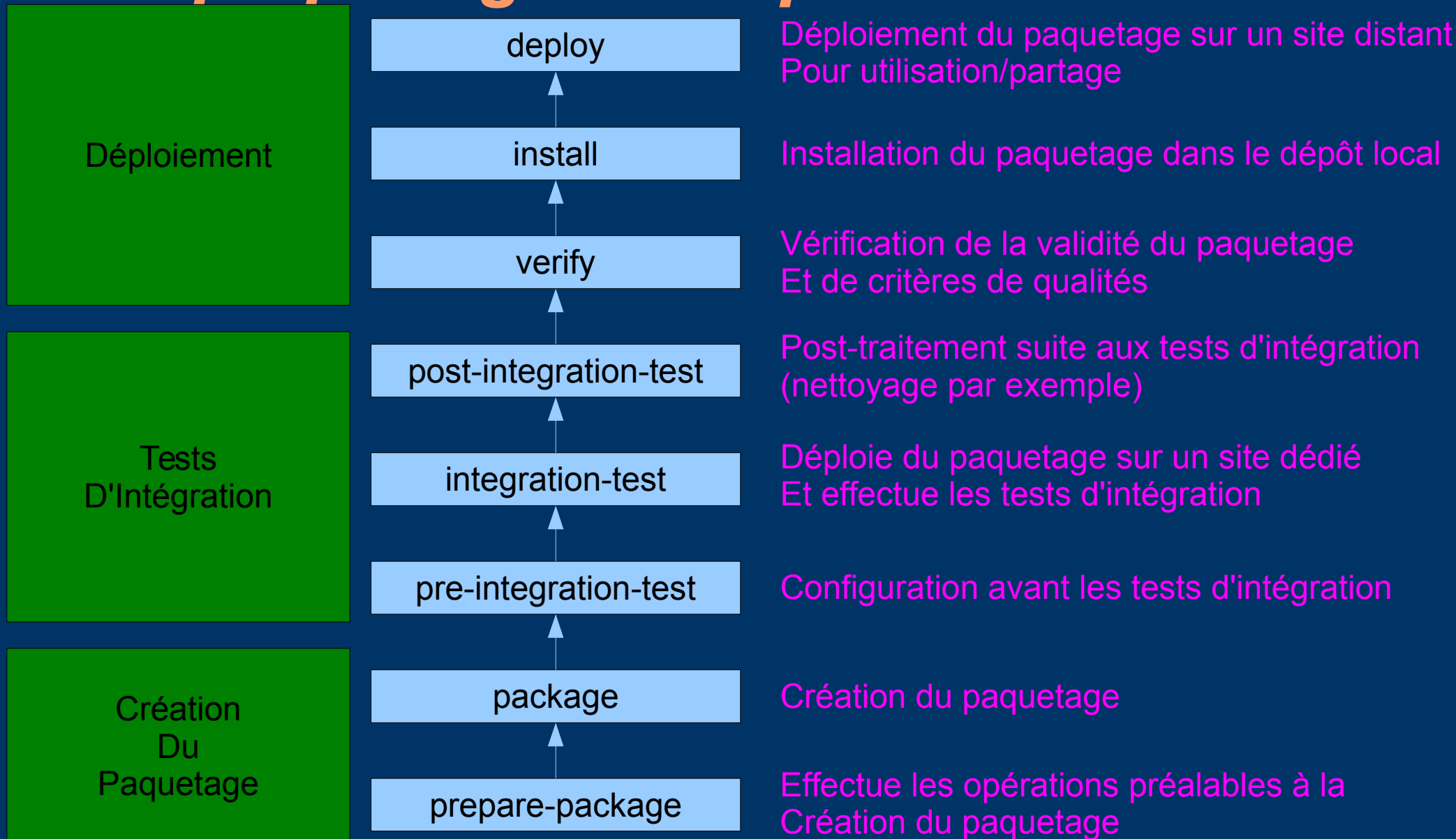
# Phases du cycle de vie Maven par défaut

## 2. Tests



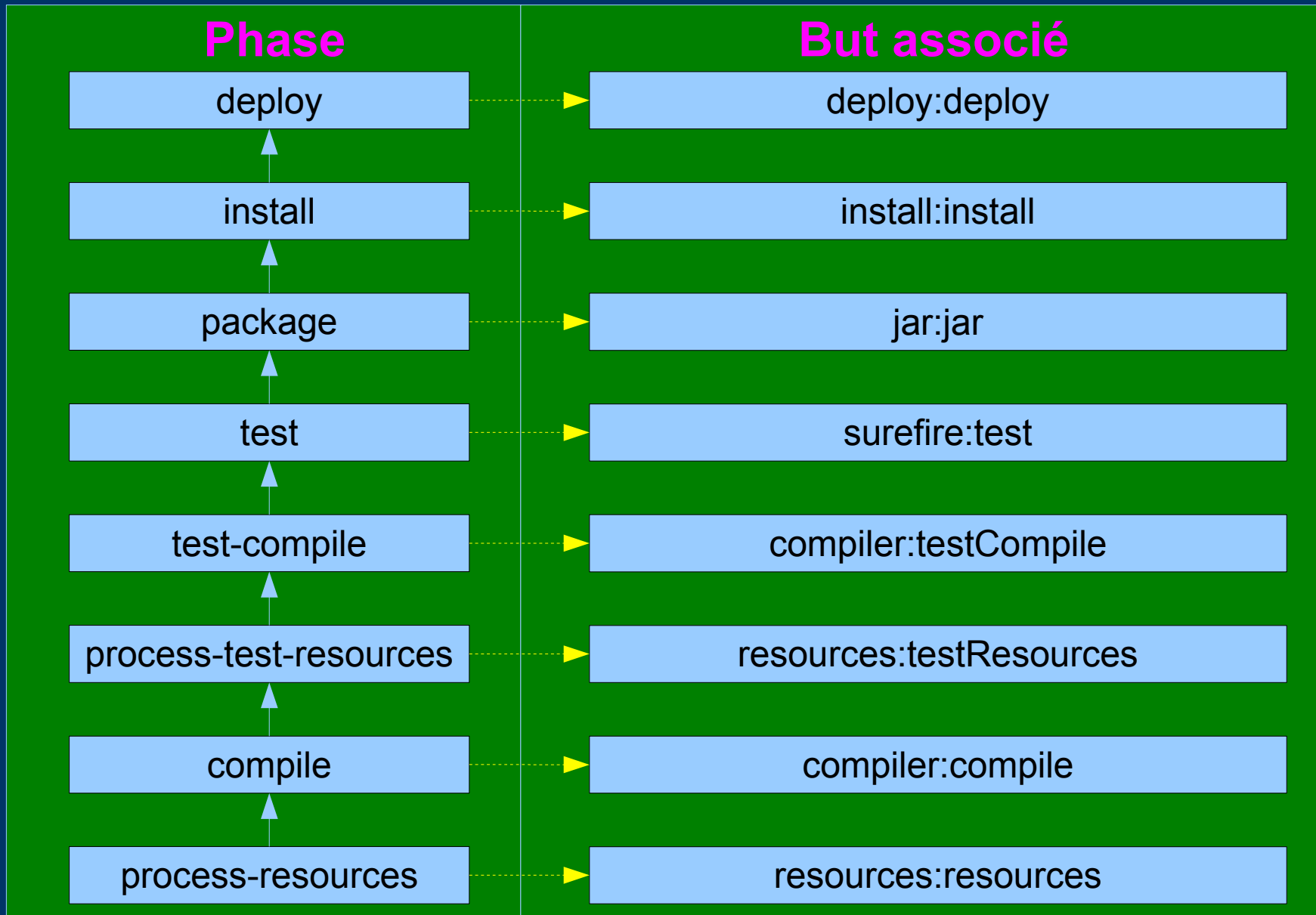
# Phases du cycle de vie Maven par défaut

## 3. Du packaging au déploiement

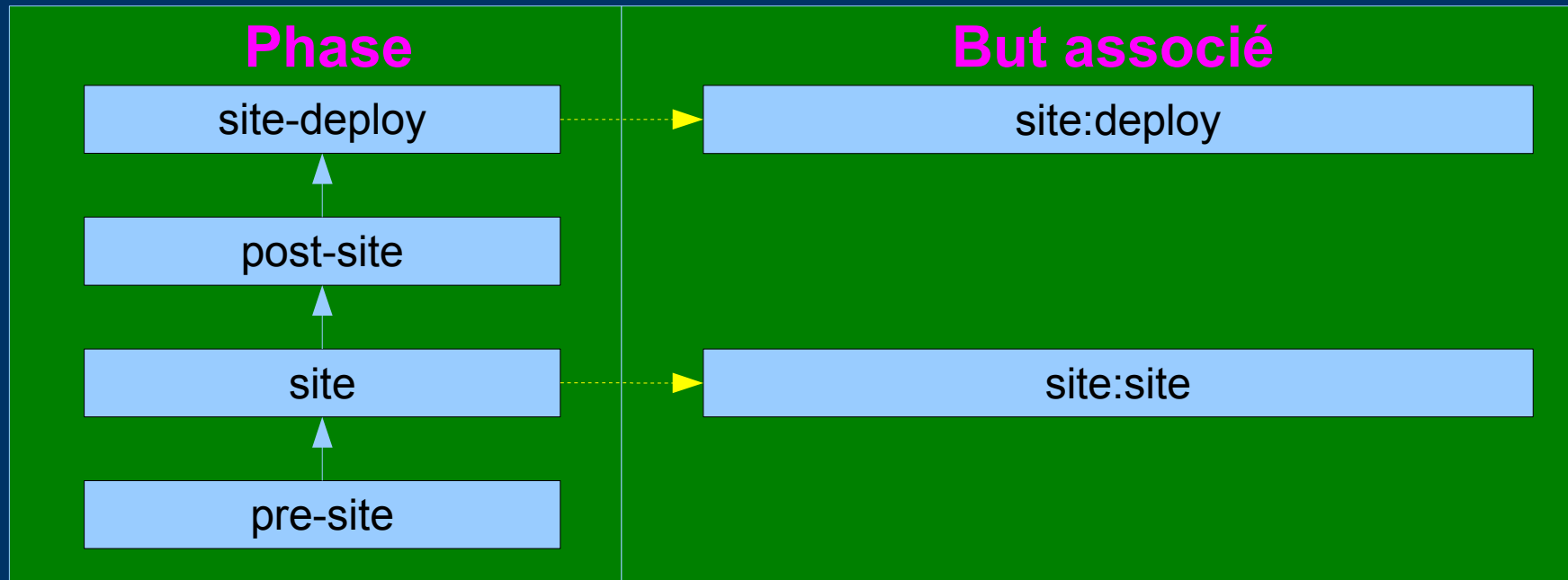


# Cas particulier

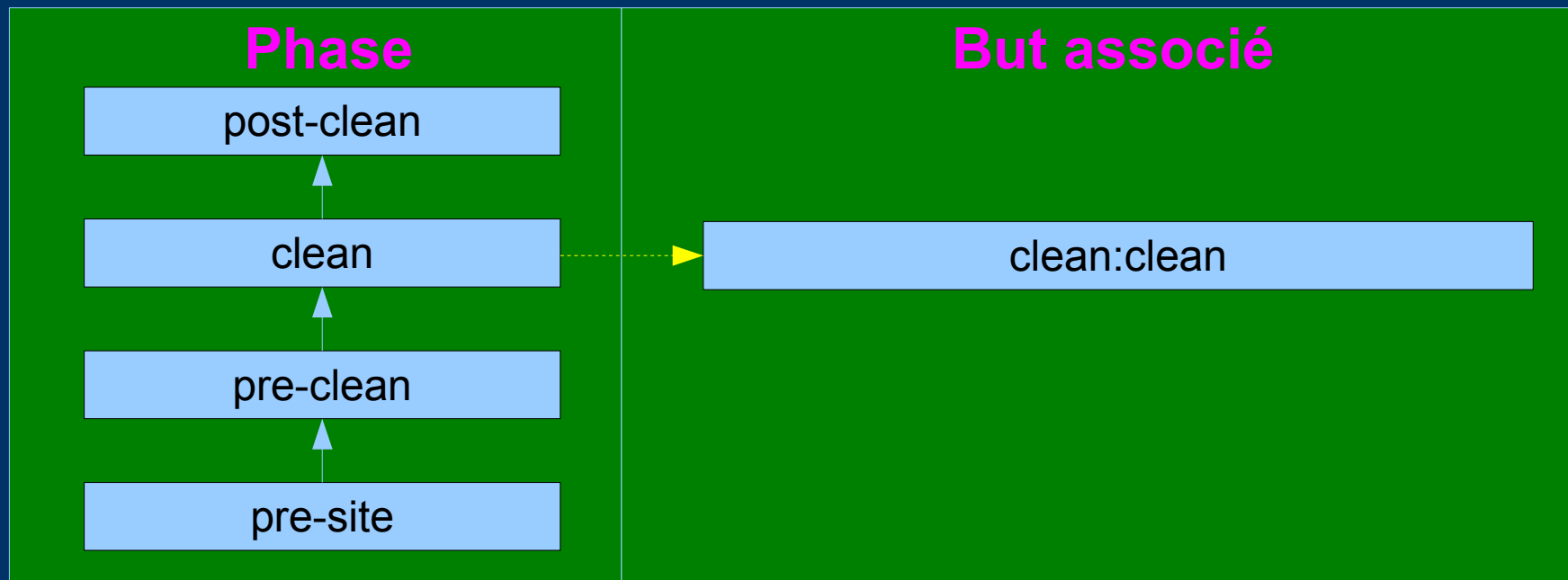
## Cycle de vie pour un fichier jar



# Phases du cycle de vie Maven pour le site

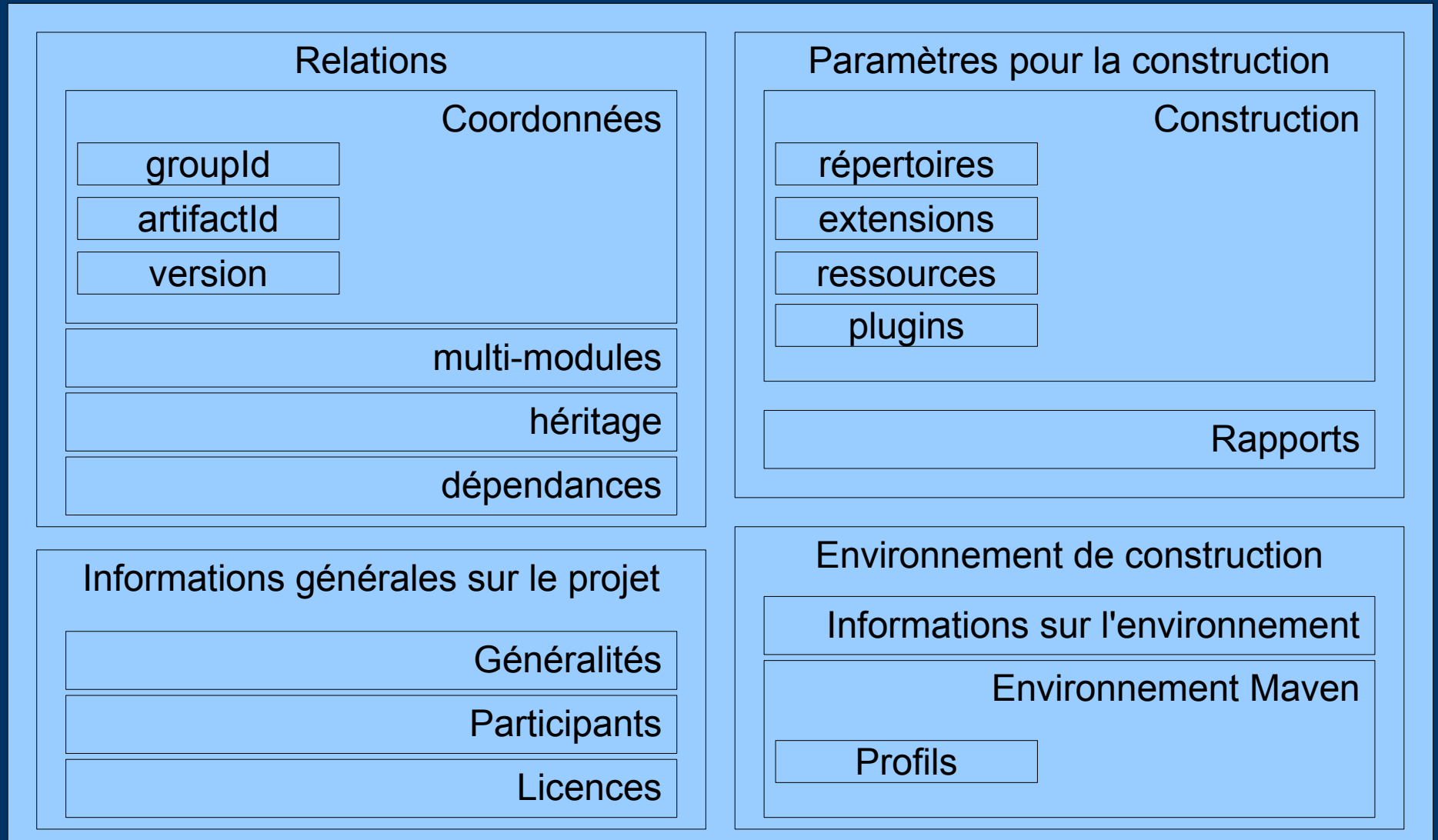


# Phases du cycle de vie Maven pour le nettoyage



# Project Object Model

## Structure générale



# *Project Object Model*

## *Un premier exemple*

```
<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>edu.mermet</groupId>
<artifactId>EssaiProjetMaven</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<name>EssaiProjetMaven</name>
<url>http://maven.apache.org</url>
```

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
</project>
```

# Coordonnées maven

- Identifiant
    - GroupId
      - En général, le nom du domaine à l'envers
    - ArtifactId
      - Ce que l'on veut
    - Version
      - `majorVersion.minorVersion.incrementalVersion-qualifier`
      - Permet de spécifier une version minimale lors d'une dépendance avec `order <num,num,num,alpha>`
        - Si contient SNAPSHOT, remplacé par `dateHeureUTC` lors de la génération du package
        - Non importé par défaut
  - Packaging
    - Type de paquetage à produire (jar, war, ear, etc.)
- 
-



# Dépendances

- Spécifiées par
    - Un identifiant (group/artifact/version)
    - Une portée (*scope*) : *compile* par défaut
  - Récupération du projet
    - Soit localement
    - Soit sur un dépôt distant
  - Version
    - 3.8.1 : si possible 3.8.1
    - [3.8,3.9] : de 3.8 à 3.9 inclus
    - (3.8,4.0) : à partir de 3.8, mais avant 4.0
    - (,4.0) : antérieur à 4.0
    - [3.8,] : à partir de 3.8
    - [3.8.1] : 3.8.1 absolument
  - Portée
    - Compile : nécessaire à la compilation, et inclus dans le paquetage
    - Provided : nécessaire à la compilation, non packagé (ex : Servlets)
    - Runtime : nécessaire pour exécution et test, mais pas compilation (ex : driver jdbc)
    - Test : nécessaire uniquement pour les tests
    - System : comme *provided* + chemin à préciser, mais à éviter
- 
-

# Structure des répertoires

## Répertoire Du Projet

- pom.xml
- src
  - main
    - java
    - resources
  - test
    - java
    - resources
- target
  - classes
  - test-classes

# *Installation de maven*

## *Variables d'environnement à définir*

- M2\_HOME

Doit contenir le répertoire d'installation de maven

- PATH

Rajouter \$M2\_HOME/bin

- JAVA\_HOME

Doit contenir le répertoire d'installation de java



# Création d'un projet Maven Plugin archetype

- Suggestion
    - `mvn archetype:generate -DgroupId=monGroupe -DartifactId=monAppli -Dversion=1.0-SNAPSHOT`
    - Choisir le type de projet « maven-archetype-quickstart »
    - Confirmer avec « Y »
  - Remarques
    - Les arguments non renseignés seront demandés à l'exécution
- 
-

# Créer un projet Java depuis les dernières versions

- Maintenant, une version supérieure à java 1.5 est recommandée. Plusieurs solutions :
  - Rajouter les lignes suivantes dans le POM généré avec l'archetype maven-archetype-quickstart :

```
<maven.compiler.source>1.8</maven.compiler.source>  
<maven.compiler.target>1.8</maven.compiler.target>
```
  - Utiliser un autre archetype comme :

```
com.github.ngeor:archetype-quickstart-jdk8
```

# Structure d'un projet nouvellement créé

- Commande

```
Mvn archetype:generate -DgroupId=edu.mermet  
-DartifactId=exemple -Dversion=1.0-SNAPSHOT
```

- Structure

- exemple

- pom.xml

- src

- main

- java

- edu

- mermet

- App.java

- test

- java

- edu

- mermet

- AppTest.java



# Fichiers de base générés : pom.xml

```
<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/P
OM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>edu.mermet</groupId>
  <artifactId>exemple</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>exemple</name>
  <url>http://maven.apache.org</url>
```

```
<properties>
  <project.build.sourceEncoding>UTF-8
</project.build.sourceEncoding>
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
</project>
```

# *Fichiers de base générés : App.java*

```
package edu.mermet;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
    }
}
```



# Fichiers de base générés : AppTest.java

```
package edu.mermet;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

/**
 * Unit test for simple App.
 */

public class AppTest extends TestCase
{
    /**
     * Create the test case
     *
     * @param testName name of the test case
     */
}
```

```
public AppTest( String testName )
{
    super( testName );
}

/**
 * @return the suite of tests being tested
 */
public static Test suite()
{
    return new TestSuite( AppTest.class );
}

/**
 * Rigorous Test :-)
 */
public void testApp()
{
    assertTrue( true );
}
}
```

# Utilisation de base du projet

## Compilation

- Prépambule

Exécuter toutes les commandes depuis le répertoire contenant le fichier pom.xml

- Compilation

- Commande

- mvn compile

- Bilan

- exemple/target/classes/edu/mermet/App.class

- Exécution (à des fins de test)

`mvn exec:java -Dexec.mainClass=edu.mermet.App`

---

---

# Utilisation de base du projet

## Packaging et Installation

- Packaging
  - Commande
    - `mvn package`
  - Bilan
    - Ne recompile pas le code, mais compile la classe de test
    - Exécution avec succès du seul test unitaire
    - Création de
      - `exemple/target/test-classes`
      - `exemple/target/surefire-reports`
      - `exemple/target/exemple-1.0-SNAPSHOT.jar`
- Installation
  - Commande
    - `mvn install`
  - Bilan
    - Ré-exécution des tests\*
    - Création de `$HOME/.m2/repository/edu/mermet/exemple/1.0-SNAPSHOT/exemple-1.0-SNAPSHOT.jar, ...`

\*pour éviter cela, `mvn install -Dmaven.test.skip=true`

---

---

# Utilisation de base du projet

## Génération d'un site web

- Commande
  - mvn site
- Bilan
  - exemple/target/site
    - CSS
      - maven-base.css, maven-theme.css
      - print.css, site.css
    - images
      - collapsed.gif, expanded.gif
      - external.png
      - icon\_error\_sml.gif, icon\_info\_sml.gif
      - icon\_success\_sml.gif, icon\_warning\_sml.gif
      - logos
        - build-by-maven-black.png
        - build-by-maven-white.png
        - maven-feather.png
      - newwindow.png

# Maven et les tests

- Exécution des tests à chaque invocation de la phase `test`

## Évitement

- En ligne de commande : `-Dmaven.test.skip=true`
- Via le POM

```
<plugins><plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<configuration>
<skip>true</skip>
</configuration>
</plugin></plugins>
```

- Echec d'un test au moins => blocage de la phase `package`

## Évitement

- En ligne de commande : `-Dmaven.test.failure.ignore=true`
- Via le POM

```
<plugins><plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<configuration>
<testFailureIgnore>true</testFailureIgnore>
</configuration>
</plugin></plugins>
```

---

---

# Utilisation d'un jar externe

## Exemple avec log4J

- pom.xml

```
...  
  
<dependencies>  
  <dependency>  
    <groupId>log4j</groupId>  
    <artifactId>log4j</artifactId>  
    <version>1.2.14</version>  
  </dependency>  
  <dependency>  
    <groupId>junit</groupId>  
    <artifactId>junit</artifactId>  
    <version>3.8.1</version>  
    <scope>test</scope>  
  </dependency>  
</dependencies>  
</project>
```

- App.java

```
package edu.mermet;  
  
import org.apache.log4j.Logger;  
import org.apache.log4j.ConsoleAppender;  
import org.apache.log4j.SimpleLayout;  
  
/**  
 * Hello world!  
 */  
public class App  
{  
    private static Logger log = Logger.getLogger(App.class);  
  
    public static void main( String[] args )  
    {  
        ConsoleAppender ca = new ConsoleAppender(new  
SimpleLayout ());  
        ca.setTarget (ConsoleAppender.SYSTEM_ERR);  
        ca.activateOptions (); log.addAppender (ca);  
        log.info ("avant");  
        System.out.println ( "Hello World!" );  
        log.info ("Après");  
    }  
}
```

- Compilation/exécution

```
mvn compile  
mvn exec:java -Dexec.MainClass=edu.mermet.App
```

# Utilisation de fichiers de ressources

- `exemple/src/main/resources/message.txt`

`coucou`

- `App.java`

```
package edu.mermet;
```

```
import org.apache.log4j.*;
import java.io.*;
```

```
/**Hello world! */
```

```
public class App {
```

```
    private static Logger log = Logger.getLogger(App.class);
```

```
    public static void main( String[] args ) {
```

```
        ConsoleAppender ca = new ConsoleAppender(new SimpleLayout());
```

```
        ca.setTarget(ConsoleAppender.SYSTEM_ERR);
```

```
        ca.activateOptions();
```

```
        log.addAppender(ca);
```

```
        String message = "Hello world";
```

```
        try {
```

```
            InputStream is = App.class.getClassLoader().getResourceAsStream("message.txt");
```

```
            BufferedReader br = new BufferedReader(new InputStreamReader(is));
```

```
            message = br.readLine();
```

```
            br.close();
```

```
        } catch (IOException ioe) {}
```

```
        log.info("avant");
```

```
        System.out.println(message);
```

```
        log.info("Après");
```

```
    }
```

```
}
```

## Compilation

```
mvn compile
```

## Conséquences

```
Création de exemple/target/classes/messages.txt
```

# Profils

- Principe

Ensemble nommé de configurations dans le pom.xml

- Utilisation

- Permettre de tester des configurations différentes (sur sélection manuelle)
- Permettre une configuration automatique selon la plateforme d'exécution de Maven





# *Profil nommé avec sélection manuelle*

```
<profiles>
  <profile>
    <id>production</id>
    <build>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <configuration>
            <debug>>false</debug>
            <optimize>>true</optimize>
          </configuration>
        </plugin>
      </plugins>
    </build>
  </profile>
</profiles>
```



# *Profil activé automatiquement*

```
<profiles>
  <profile>
    <id>dev</id>
    <activation>
      <jdk>1.5</jdk>
      <os>
        <name>Windows XP</name>
        <family>Windows</family>
        <arch>x86</arch>
        <version>5.1.2600</version>
      </os>
      <property>
        <name>customProperty</name>
        <value>BLUE</value>
      </property>
      <file>
        <exists>file2.properties</exists>
        <missing>file1.properties</missing>
      </file>
    </activation>
    ...
  </profile>
</profiles>
```

# *Profil activé par défaut*

```
<profiles>
  <profile>
    <id>dev</id>
    <activation>
      <activeByDefault>>true</activeByDefault>
    </activation>
    ...
  </profile>
</profiles>
```



# Héritage entre POMs

- Principe

- Tout « pom » hérite d'un autre
- POM racine : « super-POM »
- Définition d'un héritage

```
<parent>
```

```
    <groupId>groupe</groupId>
```

```
    <artifactId>projet</artifactId>
```

```
    <version>version</version>
```

```
</parent>
```

- Propriété

- Transitivité
  - On hérite de toutes les propriétés du super-POM, mais on peut les redéfinir
- 
-

# Maven et subversion

## Plugin scm (Source Code Management)

- Configuration dans le pom

```
<scm>
  <connection>scm:svn:http://somerepository.com/svn
_repo/trunk</connection>
  <developerConnection>scm:svn:https://somereposito
ry.com/svn_repo/trunk</developerConnection>
  <url>http://somerepository.com/view.cvs</url>
</scm>
```

- Utilisation

- `mvn scm:checkout`
- `mvn scm:update`
- `mvn scm:checkin`
- ...

Connection : pour la lecture seule  
DeveloperConnection : pour l'écriture

# Projets multi-modules

- Un répertoire parent avec un pom référençant les sous-modules

```
<packaging>pom</packaging>
```

```
...
```

```
<modules>
```

```
<module>m1</module>
```

```
<module>m2</module>
```

```
</modules>
```

- Un sous-répertoire par module, avec un pom par module
- Utilisation

- Depuis le répertoire parent

les commandes mvn s'appliquent à tous les modules, en tenant compte des éventuelles dépendances

- Depuis un sous-répertoire

les commandes ne s'appliquent qu'au sous-module courant

---

---

# Quelques configurations pratiques

## 1. Junit dernière version

```
<dependency>
```

```
  <groupId>junit</groupId>
```

```
  <artifactId>junit</artifactId>
```

```
  <version>4.12</version>
```

```
  <scope>test</scope>
```

```
</dependency>
```

---

---

# Quelques configurations pratiques

## 2. Java 8

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

---

---



# Quelques configurations pratiques

## 3. Jar exécutable

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>2.4</version>
      <configuration>
        <archive>
          <manifest>
            <mainClass>edu.mermet.App</mainClass>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
```

---

---

# Quelques configurations pratiques

## 4. OneJar

```
<build>
  <plugins>
    <plugin>
      <groupId>com.jolira</groupId>
      <artifactId>onejar-maven-plugin</artifactId>
      <version>1.4.4</version>
      <executions>
        <execution>
          <goals>
            <goal>one-jar</goal>
          </goals>
          <configuration>
            <mainClass>edu.mermet.App</mainClass>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

---

---

# Quelques configurations pratiques

## 5. PostGres (version 8.4)

```
<dependency>  
  <groupId>postgresql</groupId>  
  <artifactId>postgresql</artifactId>  
  <version>8.4-702.jdbc4</version>  
</dependency>
```



# *Tests unitaires et Tests d'intégration*

- Principe général
  - Les tests unitaires sont lancés par surefire lors de la phase « test »
  - Les tests d'intégration sont lancés par failsafe lors de la phase « integration-test », préalable à la phase « verify »

# *Distinguer les tests d'intégration*

- De base
  - Les tests unitaires sont dans des fichiers :
    - `**/Test*.java`
    - `**/*Test.java`
  - les tests d'intégration sont dans des fichiers :
    - `**/IT*.java`
    - `**/*IT.java`
- Sinon
  - Configurable (voir doc.)



# Configuration du plugin failsafe

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-failsafe-plugin</artifactId>
      <version>2.19.1</version>
      <executions>
        <execution>
          <goals>
            <goal>integration-test</goal>
            <goal>verify</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```



# Intégration d'un rapport au site web

- Configuration

```
<reporting>
```

```
<plugins>
```

```
<plugin>
```

```
<groupId>org.apache.maven.plugins</groupId>
```

```
<artifactId>maven-surfing-report-plugin</artifactId>
```

```
<version>2.19.1</version>
```

```
</plugin>
```

```
</plugins>
```

```
</reporting>
```

- Exécution

```
mvn site
```

---

---