

arnaud.nauwynck@gmail.com

Maven

Introduction to Concepts:
POM, Dependencies, Plugins, Phases

This document:

<http://arnaud-nauwynck.github.io/docs/Maven-Intro-Concepts.pdf>

What is Maven ?

The image is a composite of two distinct parts. On the left, a screenshot of a Google search results page is displayed. The search term 'maven' has been entered into the search bar. The results show the first few entries: 'Maven – Welcome to Apache Maven' (https://maven.apache.org/), 'Maven – Introduction' (https://maven.apache.org/what-is-maven.html), 'Maven – Download Apache Maven' (https://maven.apache.org/download.cgi), and 'Apache Maven - Wikipedia' (https://en.wikipedia.org/wiki/Apache_Maven). A large blue oval on the left side contains the handwritten text '31 M!!'. On the right, there is a cartoon illustration of Bart Simpson from 'The Simpsons'. He is standing in front of a green chalkboard, holding a piece of chalk and looking towards it. The chalkboard has the same handwritten text 'I will use Google before asking dumb questions.' repeated multiple times in a white, cursive font.

<https://maven.apache.org/>



Apache / Maven / Welcome to Apache Maven

Last Published: 2016-11-04

MAIN

[Welcome](#)

[License](#)

[Download](#)

[Install](#)

[Configure](#)

[Run](#)

[IDE Integration](#)

[ABOUT MAVEN](#)

[What is Maven?](#)

[Features](#)

[FAQ](#)

[Support and Training](#)

[DOCUMENTATION](#)

[Maven Plugins](#)

[Index \(category\)](#)

[Running Maven](#)

[User Centre](#) >

[Pluqin Developer Centre](#) >

Welcome to Apache Maven

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

If you think that Maven could help your project, you can find out more information about in the "About Maven" section of the navigation. This includes an in-depth description of [what Maven is](#), a list of some of its main features, and a set of [frequently asked questions about what Maven is](#).

This site is separated into the following sections, depending on how you'd like to use Maven:

Use	Download, Install, Run Maven	Configure, Use Maven and Maven Plugins
	Information for those needing to build a project that uses Maven	Information for those wanting to use Maven to build their project, including a "10 minute test" that gives a practical overview of Maven's main features in just 10 minutes and plugin list for more information on each plugin
Extend	Write Maven Plugins	Improve the Maven Repository
	Information for those who may or may not be using Maven, but want to provide a plugin for shared functionality or to accompany their own product or toolset	Information for those who may or may not use, but are interested in getting project metadata into the repository
Contribute	Help Maven	Develop Maven
	Information if you'd like to get involved: Maven is an open source community and welcomes contributions.	Information for those who are currently developers, or who are interested in contributing to the Maven project itself

A software project management and comprehension tool.

Based on a project object model (POM),
.. project's build, reporting and documentation..

<https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>



Apache / Maven / Maven in 5 Minutes Last Published: 2016-11-04

MAIN

- Welcome
- License
- Download
- Install
- Configure
- Run
- IDE Integration

ABOUT MAVEN

- What is Maven?
- Features
- FAQ
- Support and Training

DOCUMENTATION

- Maven Plugins
- Index (category)
- Running Maven

User Centre

- Maven in 5 Minutes
- Getting Started Guide

Maven in 5 Minutes

Prerequisites

You must have an understanding of how to install software on your computer. If you do not know how to do this, please ask someone at your office, school, etc or pay someone to explain this to you. The Maven mailing lists are not the best place to ask for this advice.

Installation

Maven is a Java tool, so you must have Java installed in order to proceed.

First, [download Maven](#) and follow the [installation instructions](#). After that, type the following in a terminal or in a command prompt:

```
1. mvn --version
```

It should print out your installed version of Maven, for example:

```
1. Apache Maven 3.0.5 (r01de14724cdef164cd33c7c8c2fe155faf9602da; 2013-02-19 14:51:28+0100)
2. Maven home: D:\apache-maven-3.0.5\bin\..
3. Java version: 1.6.0_25, vendor: Sun Microsystems Inc.
4. Java home: C:\Program Files\Java\jdk1.6.0_25\jre
5. Default locale: nl_NL, platform encoding: Cp1252
6. OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
```

Depending upon your network setup, you may require extra configuration. Check out the [Guide to Configuring Maven](#) if necessary.

If you are using Windows, you should look at [Windows Prerequisites](#) to ensure that you are prepared to use Maven on Windows.

Creating a Project

← → ⌂ <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html> ⌂

M G perso dev Misc

Security

COMMUNITY

Community Overview

How to Contribute

Maven Repository

Getting Help

Issue Tracking

Source Repository

The Maven Team

PROJECT DOCUMENTATION

Project Information >

MAVEN PROJECTS

Archetype

Doxia

JXR

Maven

Parent POMs

Plugins

Plugin Testing

Plugin Tools

Resource Bundles

SCM

Shared Components

Skins

Surefire

Wagon

Under this directory you will notice the following **standard project structure**.

```
1. my-app
2. | -- pom.xml
3. '-- src
4.   | -- main
5.   |   '-- java
6.   |       '-- com
7.   |           '-- mycompany
8.   |               '-- app
9.   |                   '-- App.java
10.  '-- test
11.    '-- java
12.     '-- com
13.       '-- mycompany
14.         '-- app
15.             '-- AppTest.java
```

The `src/main/java` directory contains the project source code, the `src/test/java` directory contains the test source, and the `pom.xml` file is the project's Project Object Model, or POM.

The POM

The `pom.xml` file is the core of a project's configuration in Maven. It is a single configuration file that contains the majority of information required to build a project in just the way you want. The POM is huge and can be daunting in its complexity, but it is not necessary to understand all of the intricacies just yet to use it effectively. This project's POM is:

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.   <modelVersion>4.0.0</modelVersion>
4.
5.   <groupId>com.mycompany.app</groupId>
6.   <artifactId>my-app</artifactId>
7.   <version>1.0-SNAPSHOT</version>
8.   <packaging>jar</packaging>
9.
```

Wikipedia

Not logged in Talk Contributions Create account Log in



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
العربية
Čeština
Deutsch
Español

Article Talk

Read Edit View history

Search Wikipedia



Apache Maven

From Wikipedia, the free encyclopedia



This article has multiple issues. Please help improve it or [hide]
discuss these issues on the talk page. (Learn how and when to remove these template messages)

- This article may be too technical for most readers to understand. (October 2015)
- This article relies too much on references to primary sources. (October 2015)
- This article needs additional citations for verification. (March 2012)

Maven is a build automation tool used primarily for Java projects.

The word *maven* means "accumulator of knowledge" in Yiddish.^[3]

Maven addresses two aspects of building software: first, it describes how software is built, and second, it describes its dependencies.

Contrary to preceding tools like Apache Ant, it uses conventions for the build procedure, and only exceptions need to be written down. An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins. It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging. Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven 2 Central Repository, and stores them in a local cache.^[4] This local cache of downloaded artifacts can also be updated with artifacts created by local projects. Public repositories can also be updated.

Maven can also be used to build and manage projects written in C#, Ruby, Scala, and other languages. The Maven project is hosted by the Apache Software Foundation, where it was formerly part of the

Apache Maven

The Apache Maven logo, consisting of the word "maven" in a stylized font where the 'm' is blue, 'a' is orange, 'v' is red, and 'e' is black.

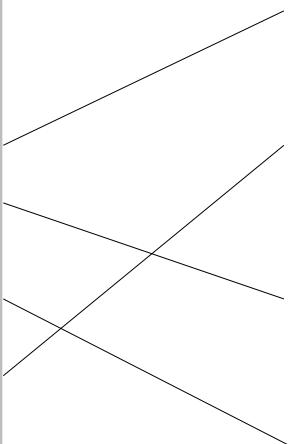
Developer(s)	Apache Software Foundation
Initial release	13 July 2004; 12 years ago
Stable release	3.3.9 ^[1] / 22 November 2015; 11 months ago ^[2]
Repository	git-wip-us.apache.org/repos/asf/maven.git ^[3]
Development status	Active
Written in	Java
Operating system	Cross-platform
Type	Build tool
License	Apache License 2.0
Website	maven.apache.org ^[4]



Maven is a Build Automation Tool

Contents [hide]

- 1 Example
- 2 Concepts
 - 2.1 Project Object Model
 - 2.2 Plugins
 - 2.3 Build lifecycles
 - 2.4 Dependencies
- 3 Maven compared with Ant
- 4 IDE integration



The diagram illustrates the structure of a Maven configuration file, `pom.xml`. It shows the root element `pom.xml` branching down into three main sections: `<dependencies>`, `<plugins>`, and `<packaging>` (which further branches into `<phase>`). Each section is preceded by a line that originates from the corresponding section header in the sidebar.

`pom.xml`

`<dependencies>`

`<plugins>`

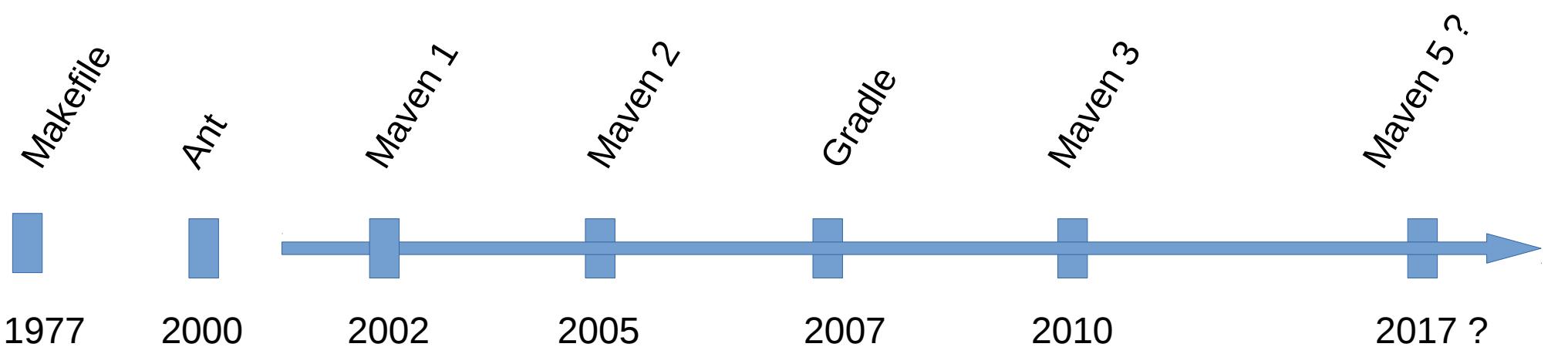
`<packaging>` `<phase>`

**1st, it describes
how software is built,**

**2nd, it describes
its dependencies.**

... use conventions for build

Maven History



Ant
= portable Make
in xml
for Javac, Jar, ..

Maven 1
= Ant + Jelly
= xml build tasks
+ scripts
(<if>, <var> ..)
+ rules / lifecycle...

Maven 2
= Java Mojo
pom.xml

Maven 3
= better & compatible
optional .mvnw
=> pom in
yaml, groovy, ..
(not widespread)

?? Split
pom vs build
Yaml / groovy

Authors, Company, Community

Jason Van Zyl



Worked on Turbine
Author of
- Velocity
- Maven

founder of Sonatype

Sonatype Company
(core Contributor + Product = Nexus)

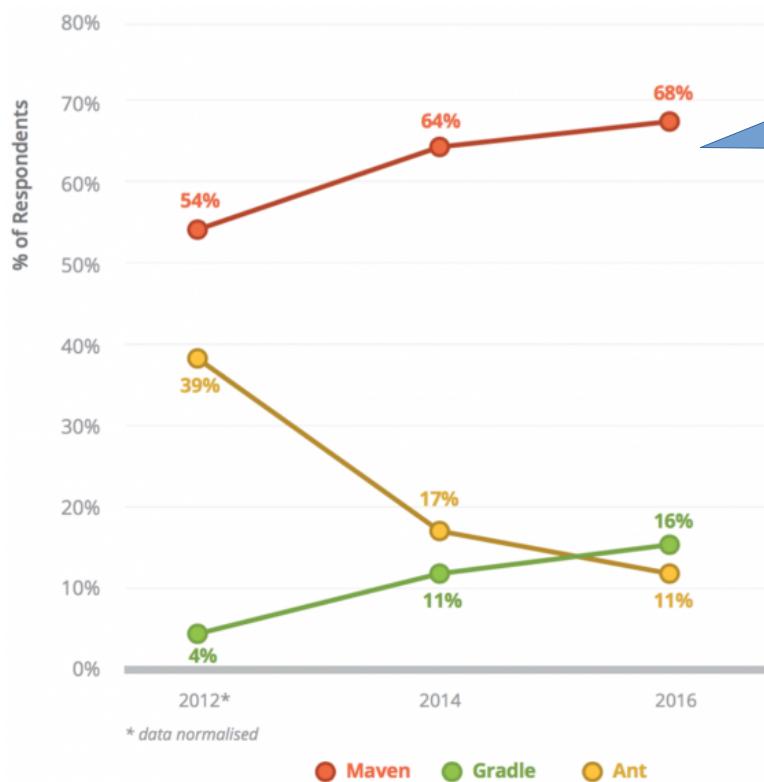
A screenshot of the Sonatype website. The header includes the company logo, navigation links for About, Blog, Partners, Community, Support, and social media icons. The main section features the heading "About Sonatype" and the subtext "We created Nexus to accelerate software innovation.", with a "WATCH THE VIDEO" button below it.

Apache Foundation
= Open Source Community

A screenshot of the Apache Software Foundation website. The header features the Apache logo and navigation links for Home, About, Projects, People, Get Involved, Download, and Support Apache. The main content area highlights the foundation's values: OPEN., INNOVATION., and COMMUNITY., each with a brief description. A sidebar on the right contains links for Google Custom Search, The Apache Way, Contribute, and ASF Sponsors.

PMC Chair / Member / Contributor / Plugin-Developers / Users

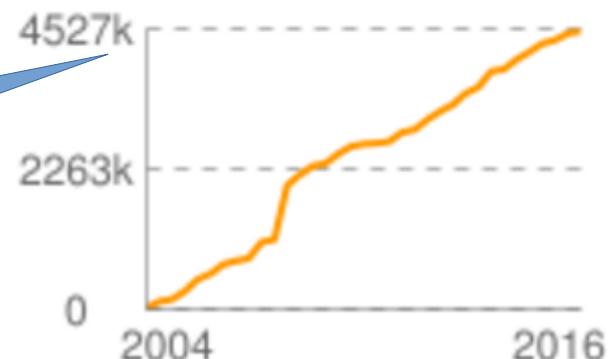
Maven Adoption



68% uses maven
It JUST Work !!



Indexed Artifacts (4.53M)



4.5 Millions published
artifacts (=jars)

Yet Another Build Tool

Makefile, Imake, Cmake,
Ant, Maven, Gradle,
MsBuild, Sbt, Gulp, Grunt,
...

Declarative vs Imperative

Describe WHAT ... not HOW

Imperative

Do **1/** This,
Then **2/** That,
Then **3/** Also That

...

And **N/** You have finished
you want to **restart** ?

Implicit Convention over Configuration

Given standards

When You follow them

Then it just works

Implicit Convention over Configuration

a Java Program is src/*.java files
compiled in classes/*.class
using javac + classpath

assembled in jar file

tested with Junit test
bla bla bla

Implicit + Declarative = Surprising Magic ...

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  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>fr.an.tests</groupId>
  <artifactId>test-mvn1</artifactId>
  <version>1.0-SNAPSHOT</version>

</project>
```

```
$ mvn install -o
[INFO] Scanning for projects...
[INFO] -----
[INFO] Reactor Build Order:
[INFO]
```

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.219 s
[INFO] Finished at: 2016-11-05T
[INFO] Final Memory: 19M/348M
[INFO] -----
```





WIKIPEDIA
The Free Encyclopedia

Maven Core Concepts 1 : (declarative) Project Object Model

Contents [hide]

- 1 Example
- 2 Concepts
 - 2.1 Project Object Model
 - 2.2 Plugins
 - 2.3 Build lifecycles
 - 2.4 Dependencies
- 3 Maven compared with Ant
- 4 IDE integration



Mandatory GAV Declaration

Unique ID : GAV = Group / Artifact / Version

```
<groupId>fr.an.tests</groupId>
<artifactId>test-mvn</artifactId>
<version>1.0-SNAPSHOT</version>
```

Group = like dns domain name, reverse
com.<<company>>, org.<<...>>, <<country>>. <<...>>. ...

Artifact = name of the final jar

naming convention: some-library-name

Version = major.minor.fix (-SNAPSHOT)?

Optional Project Information

```
<!-- More Project Information -->
<name>...</name>
<description>...</description>
<url>...</url>
<inceptionYear>...</inceptionYear>
<licenses>...</licenses>
<organization>...</organization>
<developers>...</developers>
<contributors>...</contributors>

<!-- Environment Settings -->
<issueManagement>...</issueManagement>
<ciManagement>...</ciManagement>
<mailingLists>...</mailingLists>
<scm>...</scm>
<prerequisites>...</prerequisites>
<repositories>...</repositories>
<pluginRepositories>...</pluginRepositories>
<distributionManagement>...</distributionManagement>
<profiles>...</profiles>
```

Example Project Information

```
<artifactId>maven-jar-plugin</artifactId>
<version>3.0.2</version>
<packaging>maven-plugin</packaging>

<name>Apache Maven JAR Plugin</name>
<description>Builds a Java Archive (JAR) file from the compiled pro

<contributors>
  <contributor>
    <name>Jerome Lacoste</name>
    <email>jerome@coffeebreaks.org</email>
    <organization>CoffeeBreaks</organization>
    <organizationUrl>http://www.coffeebreaks.org</organizationUrl>
    <timezone>+1</timezone>
    <roles>
      <role>Java Developer</role>
    </roles>
  </contributor>
</contributors>

<prerequisites>
  <maven>${mavenVersion}</maven>
</prerequisites>

<scm>
  <connection>scm:svn:http://svn.apache.org/repos/asf/maven/plugins</connection>
  <developerConnection>scm:svn:http://svn.apache.org/repos/asf/maven/plugins</developerConnection>
  <url>http://svn.apache.org/viewvc/maven/plugins</url>
</scm>
```

Typical Maven Generated Site

Then you recognise pom infos

PROJECT DOCUMENTATION

- Project Information
- About
- Summary
- Dependency Information
- Team
- Source Code Management
- Issue Management
- Mailing Lists
- Dependency Management
- Dependencies

OVERVIEW

- Introduction
- Goals
- Usage
- FAQ
- License
- Download

EXAMPLES

- Creating an Executable JAR File
- Manifest Customization
- Using Your Own Manifest File
- Additional attached JAR
- Create Test JAR
- Include/Exclude content

Project Summary

Project Information

Field	Value
Name	Apache Maven JAR Plugin
Description	Builds a Java Archive (JAR) file from the compiled project classes and resources.
Homepage	https://maven.apache.org/plugins/maven-jar-plugin/

Project Organization

Field	Value
Name	The Apache Software Foundation
URL	https://www.apache.org/

"Powered by" Logo

You can add your own "Powered by" logo to your site. To do this, you add a <poweredBy> element in your `site.xml` like this:

```
1. <project>
2. ...
3. <poweredBy>
4.   <logo name="Maven" href="http://maven.apache.org/" 
5.     img="http://maven.apache.org/images/logos/maven-feather.png"/>
6. </poweredBy>
7. ...
8. </project>
```

Menu when you see this logo

Built by 

Logo comes from maven (maven-site-plugin)

Basic Declarations

```
<!-- The Basics -->
<parent>...</parent>
<packaging>jar</packaging> <!-- implicit: jar -->

<dependencies>
    <dependency>...</dependency>
</dependencies>
<dependencyManagement>...</dependencyManagement>

<modules>
    <module>...</module>
</modules>
<properties>...</properties>

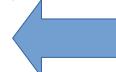
<!-- Build Settings -->
<build>
    <plugins>
        <plugin>...</plugin>
    </plugins>
</build>
<reporting>...</reporting>
```



Maven Core Concepts 2 : Dependencies

Contents [hide]

- 1 Example
- 2 Concepts
 - ✓ 2.1 Project Object Model
 - 2.2 Plugins
 - 2.3 Build lifecycles
 - 2.4 Dependencies**
- 3 Maven compared with Ant
- 4 IDE integration



Dependency Declaration

Describe WHAT

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

Not HOW

```
$ mvn install
[INFO] Scanning for projects...
[INFO]
```

First execution:
Download

Next execution:
reuse local repository file

```
[INFO] -----
Downloading: https://oss.sonatype.org/content/repositories/releases/junit/junit/4.12/junit-4.12.jar
Downloaded: https://oss.sonatype.org/content/repositories/releases/junit/junit/4.12/junit-4.12.jar (308 KB at 150.9 KB/sec)
[INFO]
```

```
/home/arnaud/.m2/repository
$ ls -l junit/junit/4.12/
total 348
-rw-r--r-- 1 arnaud arnaud 314932 Nov  6 12:35 junit-4.12.jar
-rw-r--r-- 1 arnaud arnaud      40 Nov  6 12:35 junit-4.12.jar.sha1
-rw-r--r-- 1 arnaud arnaud  23678 Feb 18 2016 junit-4.12.pom
-rw-r--r-- 1 arnaud arnaud      40 Feb 18 2016 junit-4.12.pom.sha1
-rw-r--r-- 1 arnaud arnaud     648 Nov  2 00:15 m2e-lastUpdated.properties
-rw-r--r-- 1 arnaud arnaud    348 Nov  6 12:35 _remote.repositories
```

WHERE ?

<http://maven.search.org>

The Central Repository

SEARCH | ADVANCED SEARCH | BROWSE | QUICK STATS

logback-classic

SEARCH

[About Central](#)

[Advanced Search](#) | [API Guide](#) | [Help](#)

All Day DevOps 2016

15 time zones 15 hours 54 sessions 100% free

[All Day DevOps - Register Now](#)

Browse Central For [ch.qos.logback : logback-classic : 1.1.7](#)

Click on a link above to browse the repository.

Project Information

GroupId: ch.qos.logback

ArtifactId: logback-classic

Version: 1.1.7

Project Object Model (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/pom-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-parent</artifactId>
    <version>1.1.7</version>
  </parent>
  <artifactId>logback-classic</artifactId>
  <packaging>jar</packaging>
  <name>Logback Classic Module</name>
  <description>logback-classic module</description>
  <dependencies>
    <dependency>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-classic</artifactId>
      <version>1.1.7</version>
    </dependency>
    <dependency>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-core</artifactId>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <scope>compile</scope>
    </dependency>
  </dependencies>

```

Dependency Information

Apache Maven

```
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.1.7</version>
</dependency>
```

Apache Buildr

Apache Ivy

Groovy Grape

Gradle/Grails

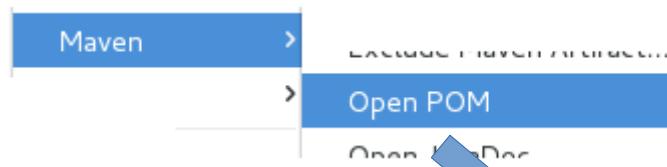
Scala SBT

Leiningen

Transitive Dependencies

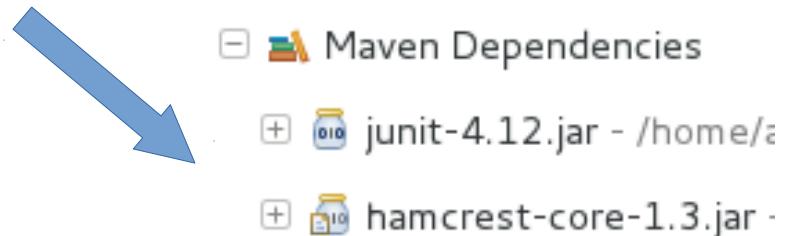
```
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.12</version>
<scope>test</scope>
</dependency>
```

Transitive Relation Definition:
A → B and B → C .. then A → C



A screenshot of a Maven interface showing the contents of the 'junit:junit:4.12.pom' file. The code is:

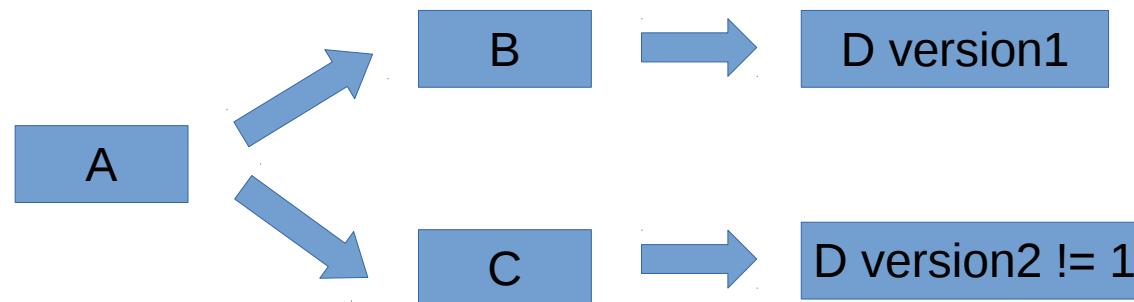
```
<dependencies>
  <dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-core</artifactId>
    <version>1.3</version>
  </dependency>
</dependencies>
```



Transitive Dependencies Conflicts

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version> <!-- ==> dependency hamcrest-core:1.3 -->
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-library</artifactId>
    <version>1.2</version> <!-- ==> dependency hamcrest-core:1.2 -->
</dependency>
```



Dependencies Omitted for Conflict

**Given Java ClassLoader load once each class
When you have conflict
Then 1 jar or 2 would be useless,
Maven omit oldest jar version**

The screenshot shows the Maven Dependency Hierarchy and Resolved Dependencies interface. On the left, the 'Dependency Hierarchy' panel lists dependencies: junit: 4.12 [test], hamcrest-core: 1.3 [compile], hamcrest-library: 1.2 [compile], and hamcrest-core: 1.2 (omitted for conflict with 1.3) [compile]. The last item is highlighted with a blue background. On the right, the 'Resolved Dependencies' panel shows the actual dependencies included in the build: hamcrest-core: 1.3 [compile], hamcrest-library: 1.2 [compile], and junit: 4.12 [test]. Below the interface, a terminal window displays the command '\$ mvn dependency:tree' and its output, which includes the omitted dependency.

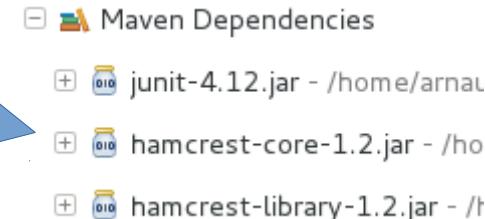
```
$ mvn dependency:tree
```

```
[INFO] --- maven-dependency-plugin:2.8:tree (default-cl)
[INFO] fr.an.tests:test-mvn-archetype1:jar:1.0-SNAPSHOT
[INFO] +- junit:junit:jar:4.12:test
[INFO] | \- org.hamcrest:hamcrest-core:jar:1.3:compile
[INFO] \- org.hamcrest:hamcrest-library:jar:1.2:compile
```

Can Override default configuration

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.hamcrest</groupId>
      <artifactId>hamcrest-core</artifactId>
      <version>1.2</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Override to use specific version

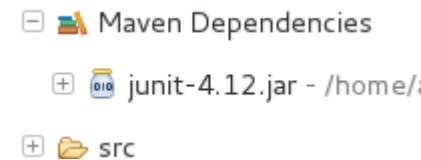


Maven Dependencies

- + junit-4.12.jar - /home/arnau...
- + hamcrest-core-1.2.jar - /ho...
- + hamcrest-library-1.2.jar - /t...

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
    <exclusions>
      <exclusion>
        <groupId>org.hamcrest</groupId>
        <artifactId>hamcrest-core</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

Override to exclude dependency



Maven Dependencies

- + junit-4.12.jar - /home/...
- + src

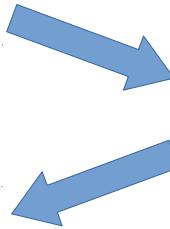
DependencyManagement

avoid duplicate version, use parent

```
<dependencyManagement>
  <dependencies>
    <dependency>
```



```
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <scope>test</scope>
    </dependency>
```



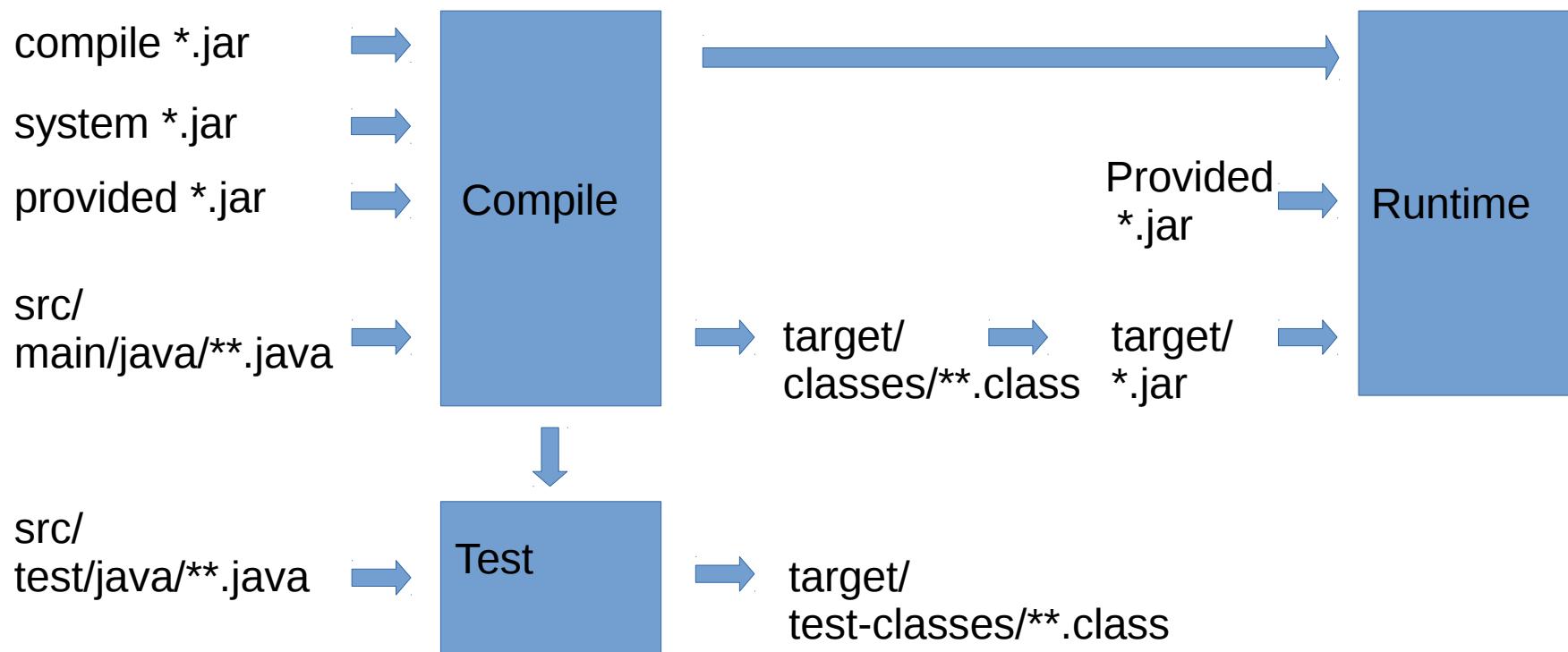
```
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <scope>test</scope>
      <version>4.12</version>
```

A screenshot of an IDE showing a warning message. The message says: "Duplicating managed version 4.12 for junit". It also indicates "2 quick fixes available:" and lists "Remove version declaration" and "Ignore this warning". A "Press 'F2' for focus" placeholder is visible at the bottom right of the message box.

```
</dependency>
  <!-- Duplicating managed version 4.12 for junit
  2 quick fixes available:
  Remove version declaration
  Ignore this warning -->
```

Dependency Scopes

```
<scope>compile</scope>  <!-- default -->
<scope>runtime</scope>   <!-- at RT, not compile-time (ex: ojdbc16.jar) -->
<scope>provided</scope>  <!-- should be on server (ex: servlet.jar) -->
<scope>system</scope>    <!-- should be in system (ex: <<jdk>>/tools.jar) -->
<scope>test</scope>      <!-- only for src/test/java, not src/main -->
<scope>import</scope>    <!-- for dependencyManagement -->
```



Dependency Summary

Would be more concise than xml:
“junit:junit:4.12:test”

Use dependencyManagement
versions in parent only

don't be too verbose
transitive dependencies => implicit

don't be too implicit
use exact versions, not *

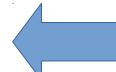


Example

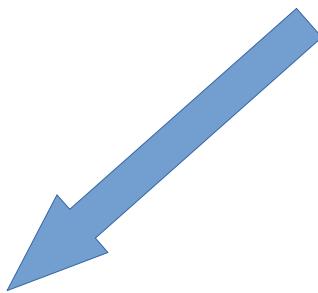
Getting Started with Mvn & Eclipse

Contents [hide]

- 1 Example
- 2 Concepts
 - ✓ 2.1 Project Object Model
 - 2.2 Plugins
 - 2.3 Build lifecycles
 - ✓ 2.4 Dependencies
- 3 Maven compared with Ant
- 4 IDE integration

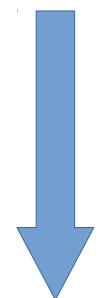


Maven Tools Usages

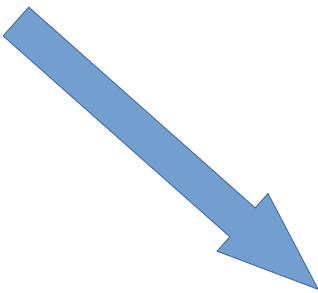


In Command Line

```
$ mvn [ ]
```



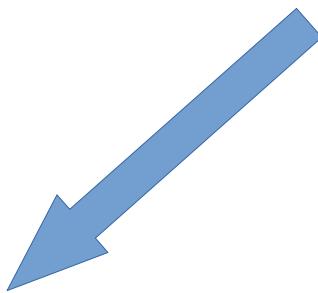
In IDE Eclipse
Buit-in support



In CI Server Jenkins..
Built-in support

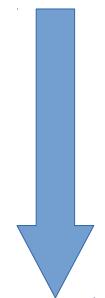


Maven Tools Usages

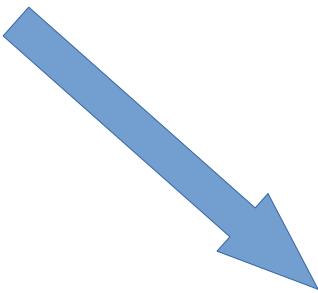


In Command Line

```
$ mvn [ ]
```



In IDE Eclipse
Buit-in support



In CI Server Jenkins..
Built-in support



Installation= Unzip + export PATH



Apache / Maven / Installing Apache Maven Last Published: 2016-11-04

MAIN
Welcome
License
Download
Install
Configure
Run
IDE Integration

ABOUT MAVEN
What is Maven?
Features

Installing Apache Maven

The installation of Apache Maven is a simple process of extracting the archive and adding the `bin` folder with the `mvn` command to the `PATH`.

Detailed steps are:

- Ensure `JAVA_HOME` environment variable is set and points to your JDK installation
- Extract distribution archive in any directory

```
1. unzip apache-maven-3.3.9-bin.zip
```

1: Unzip

or

```
1. tar xzvf apache-maven-3.3.9-bin.tar.gz
```

Unix-based Operating System (Linux, Mac OS X)

- Check environment variable value

```
1. echo $JAVA_HOME  
2. /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home
```

- Adding to `PATH`

```
1. export PATH=/opt/apache-maven-3.3.9/bin:$PATH
```

2: export JAVA_HOME & PATH

Mvn command line

```
$ mvn --version
Apache Maven 3.3.9
Maven home: /opt/ds

$ mvn --help

usage: mvn [options] [<goal(s)>] [<phase(s)>]

Options:
-----[INFO] Scanning for projects...
[INFO] -----
[INFO] Reactor Build Order:
[INFO]
```

mvn install

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.219 s
[INFO] Finished at: 2016-11-05T18:18:01+01:00
[INFO] Final Memory: 19M/348M
[INFO] -----
```

Typical Commands

Simple phases

```
mvn clean install
```

With options

```
mvn install -f ./pom.xml  
-o -DskipTests -Pprofile...
```

Plugin Goals

```
mvn springboot:run
```

Start Edit a Pom.xml manually ?

```
<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>fr.an.tests</groupId>
  <artifactId>test-mvn1</artifactId>
  <version>1.0-SNAPSHOT</version>

</project>
```

XML = Langage for Computers
... not for Humans

You only need this identity card :
GAV = group / artifact / version

New Maven Project

mvn install

Copy & Paste



mvn archetype:generate
mvn install



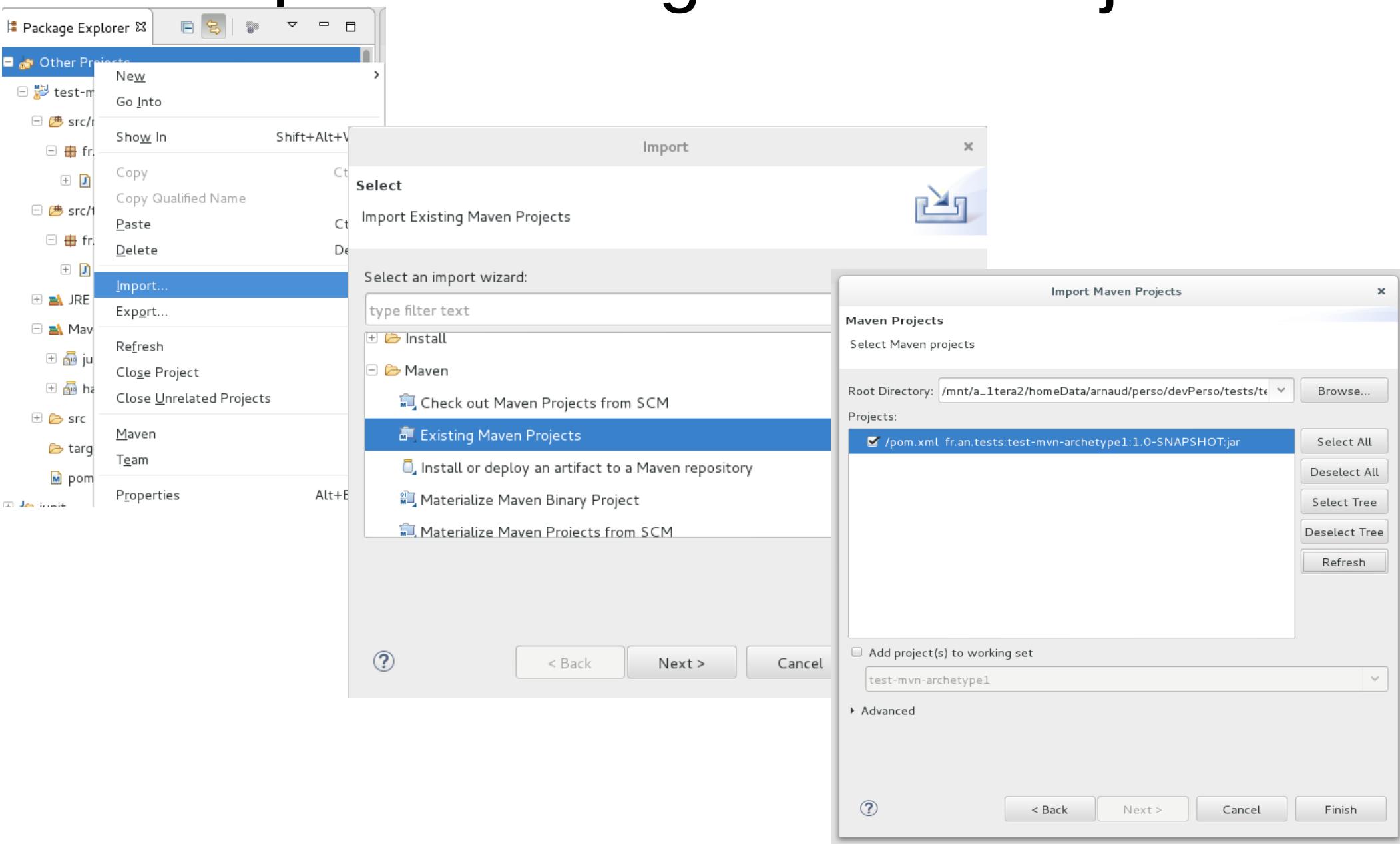
New Project... >
Type: Maven Project
Choose archetype...

Import existing project ... >
Type : maven

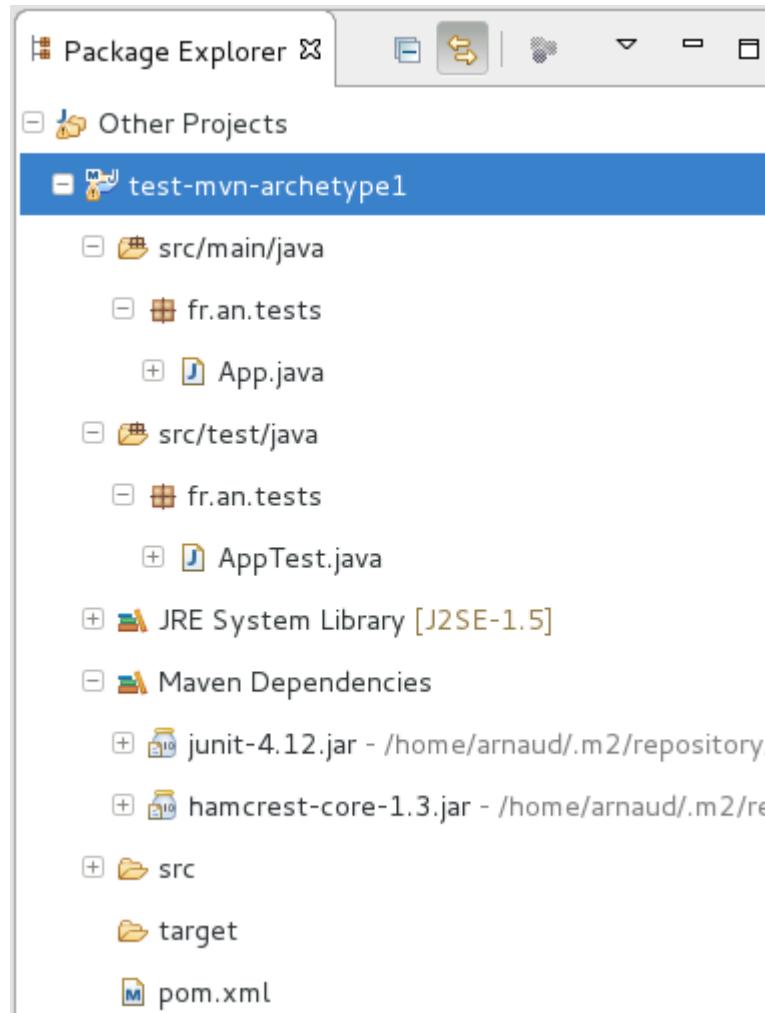


mvn install

Eclipse Import Existing Maven Project



First Maven Project in Eclipse



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/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>fr.an.tests</groupId>
  <artifactId>test-mvn1</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

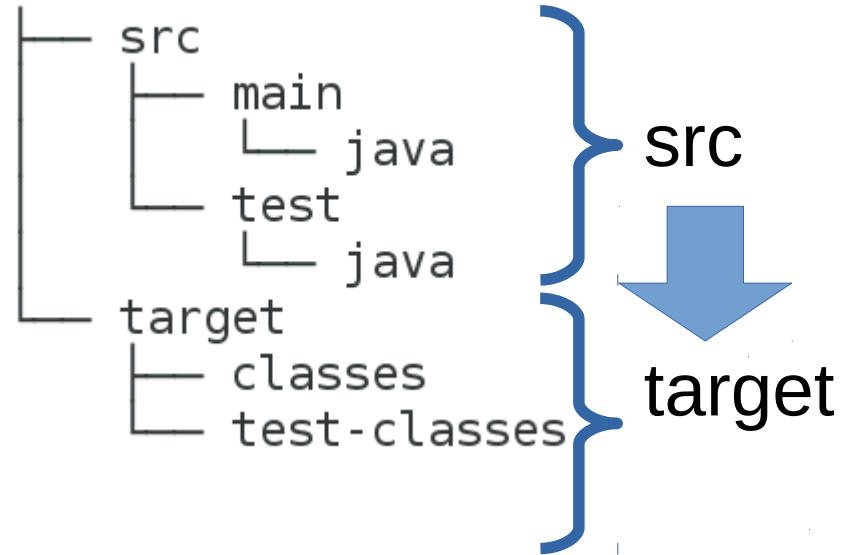
Standard Source Project Layout

```
$ tree
.
└── pom.xml
└── src
    ├── main
    │   └── java
    │       └── fr
    │           └── an
    │               └── tests
    │                   └── App.java
    └── test
        └── java
            └── fr
                └── an
                    └── tests
                        └── AppTest.java
11 directories, 3 files
$
```

Mvn compile (or install)

```
$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building test-mvn1 1.0-SNAPSHOT
[INFO] -----
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ test-mvn1 ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /mnt/a_1tera2/homeData/arnaud/perso/devPerso/tests/test-mvn-archetype1/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ test-mvn1 ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file to /mnt/a_1tera2/homeData/arnaud/perso/devPerso/tests/test-mvn-archetype1/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ test-mvn1 ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /mnt/a_1tera2/homeData/arnaud/perso/devPerso/tests/test-mvn-archetype1/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ test-mvn1 ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file to /mnt/a_1tera2/homeData/arnaud/perso/devPerso/tests/test-mvn-archetype1/target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ test-mvn1 ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ test-mvn1 ---
[INFO] Building jar: /mnt/a_1tera2/homeData/arnaud/perso/devPerso/tests/test-mvn-archetype1/target/test-mvn1-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ test-mvn1 ---
[INFO] Installing /mnt/a_1tera2/homeData/arnaud/perso/devPerso/tests/test-mvn-archetype1/target/test-mvn1-1.0-SNAPSHOT.jar to /home/arnaud/.m2/repository/fr/an/tests/test-mvn1/1.0-SNAPSHOT/test-mvn1-1.0-SNAPSHOT.jar
[INFO] Installing /mnt/a_1tera2/homeData/arnaud/perso/devPerso/tests/test-mvn-archetype1/pom.xml to /home/arnaud/.m2/repository/fr/an/tests/test-mvn1/1.0-SNAPSHOT/test-mvn1-1.0-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.089 s
```

src → target directories



compile

`src/main/java` → `target/classes`

compile-test

`src/test/java` → `target/test-classes`

jar

`target/classes` → `target/*.jar`

install

`target/*.jar` → `~/.m2/repository/...`
`groupId/artifactId/version/*.jar`

deploy

`.m2/repository/... .jar` → `http://nexus-repo/...`
`groupId/artifactId/version/*.jar`

Mvn clean

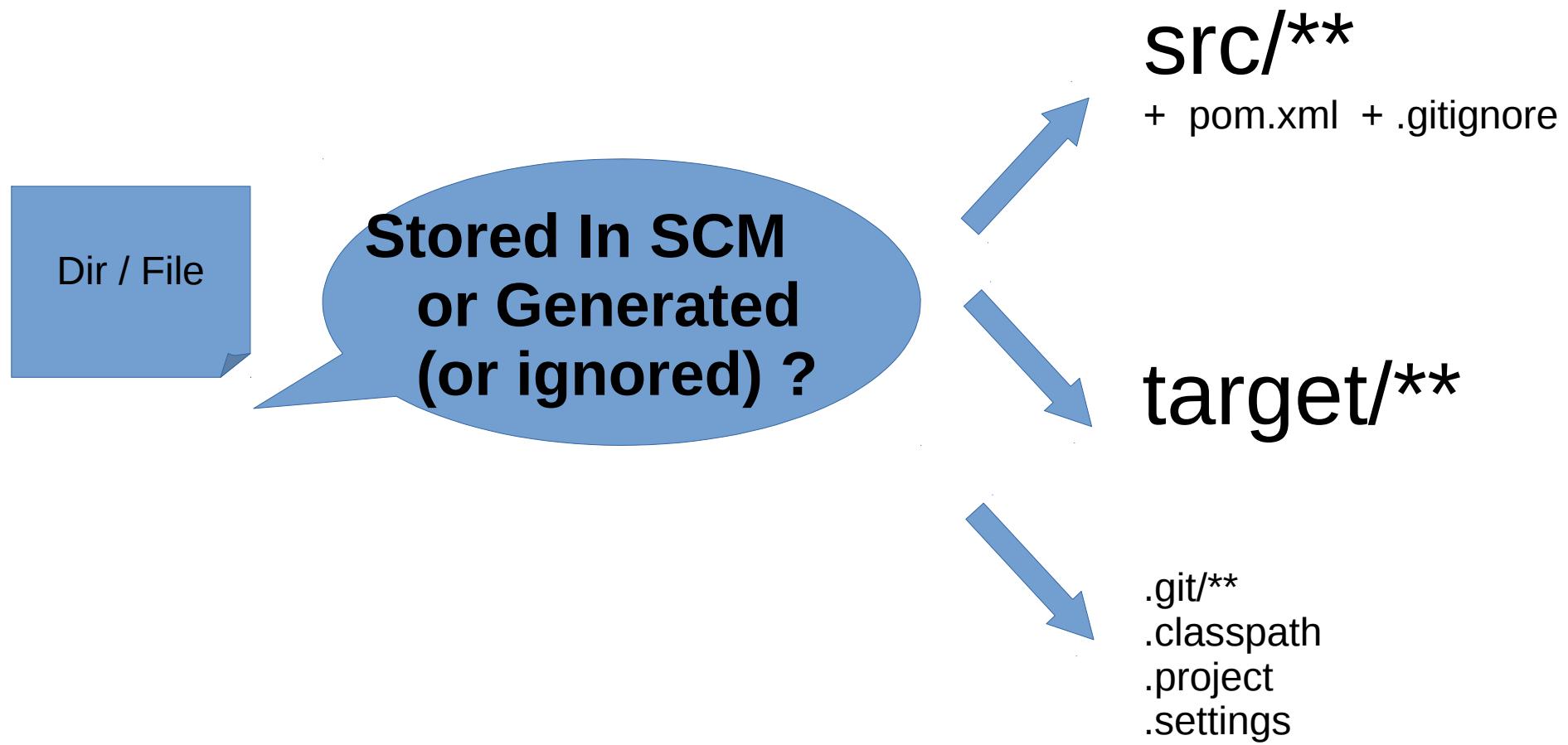
src vs target + .gitignore

```
$ mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building test-mvn1 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ test-mvn1 ---
[INFO] Deleting /mnt/a_1tera2/homeData/arnaud/perso/devPerso/tests/test-mvn-archetype1/target
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.223 s
[INFO] Finished at: 2016-11-06T18:58:30+01:00
[INFO] Final Memory: 6M/283M
[INFO] -----
```

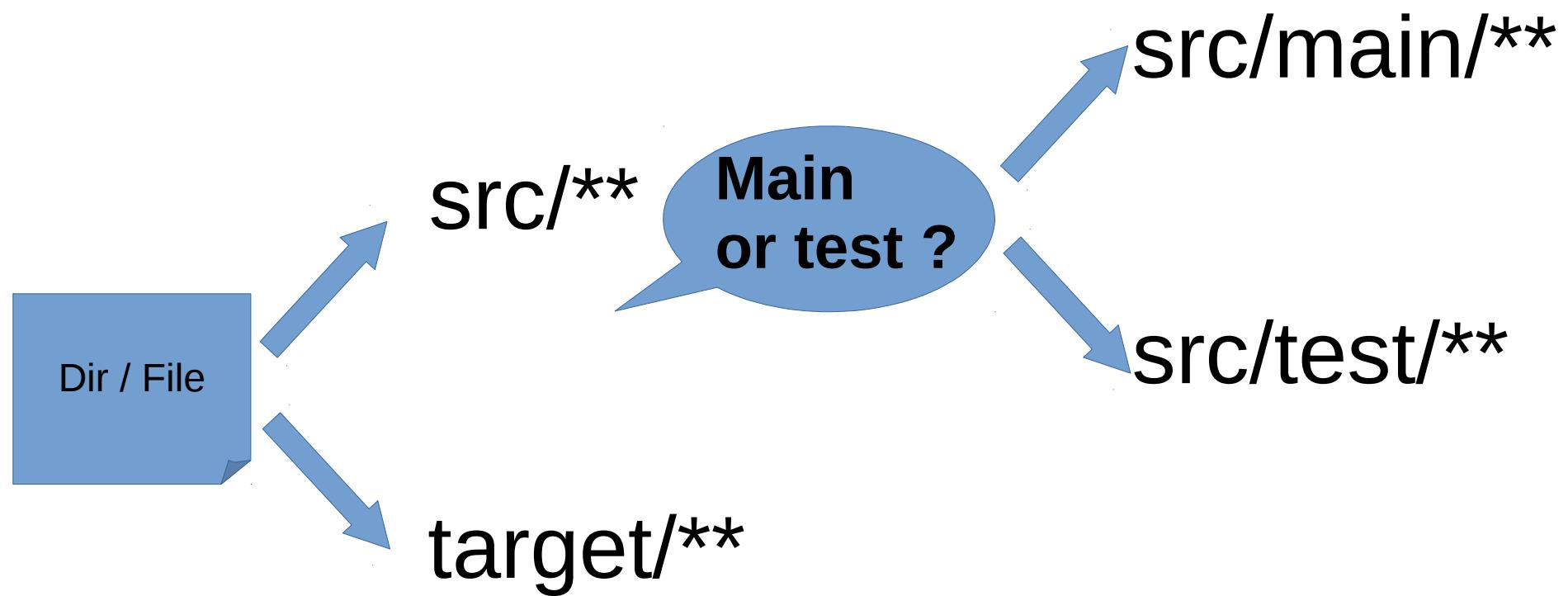
Typical .gitignore for target/, .project, .classpath, ...

```
$ ls
pom.xml  src  target
$ 
$ ls -a
.  ..  .classpath  .git  .gitignore  pom.xml  .project  .settings  src  target
$ 
$ cat .gitignore
target
.project
.classpath
.settings
```

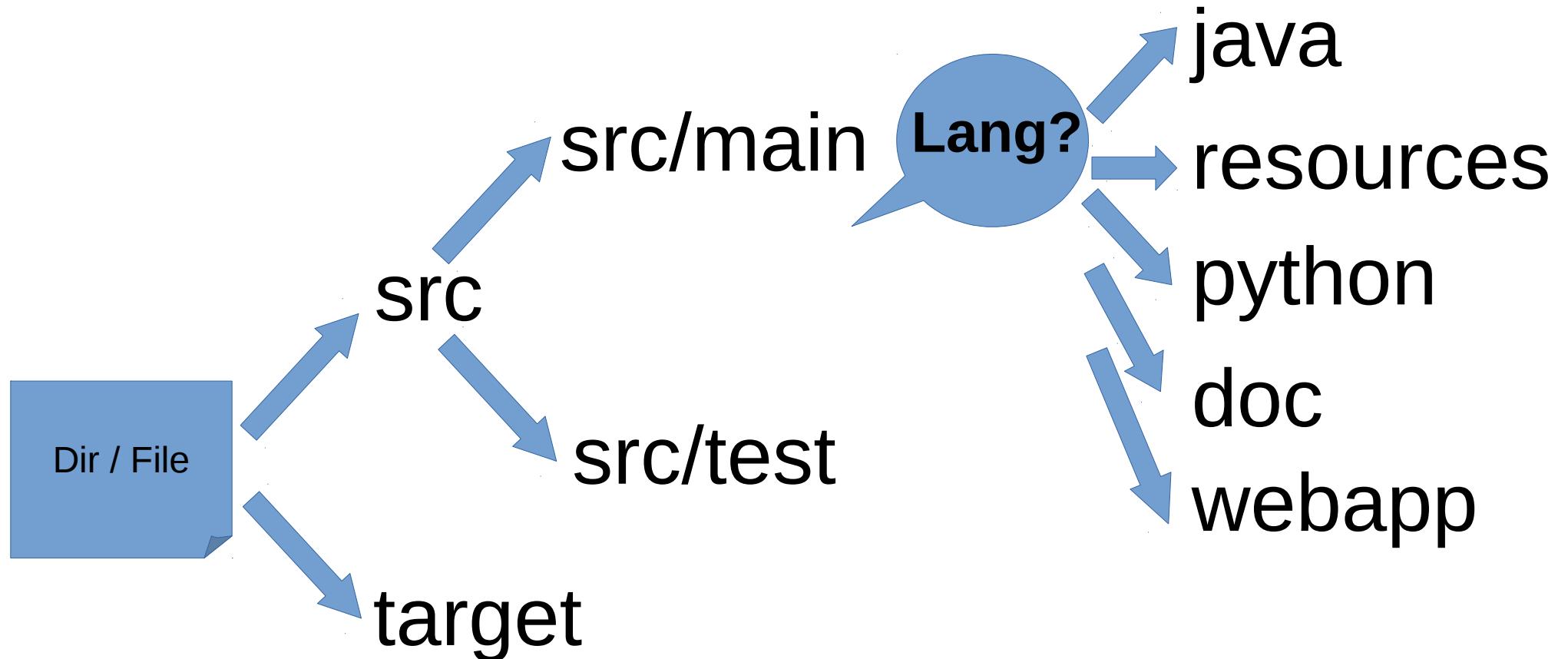
Project Layout Explained by Dichotomy Questions



Project Layout Explained (bis)



Project Layout Explained (ter)





Maven Core Concepts 3 : Plugins – Goals

Contents [hide]

- [Example](#)
- [2 Concepts](#)
 - [2.1 Project Object Model](#)
 - [2.2 Plugins](#)
 - [2.3 Build lifecycles](#)
 - [2.4 Dependencies](#)
- [3 Maven compared with Ant](#)
- [4 IDE integration](#)



<https://maven.apache.org/plugins/>

MAIN

Welcome

License

Download

Install

Configure

Run

IDE Integration

ABOUT MAVEN

What is Maven?

Features

FAQ

Support and Training

DOCUMENTATION

Maven Plugins

Index (category)

Running Maven

User Centre

Available Plugins

Scroll for 100 more

Maven is - at its heart - a plugin execution framework; all work is done by plugins. Looking for a specific goal to execute? This page lists the core plugins and others. There are the build and the reporting plugins:

- **Build plugins** will be executed during the build and they should be configured in the `<build/>` element from the POM.
- **Reporting plugins** will be executed during the site generation and they should be configured in the `<reporting/>` element from the POM. Because the result of a Reporting plugin is part of the generated site, Reporting plugins should be both internationalized and localized. You can read more about the [localization of our plugins](#) and how you can help.

Supported By The Maven Project

To see the most up-to-date list browse the Maven repository, specifically the `org/apache/maven/plugins` subfolder. (*Plugins are organized according to a directory structure that resembles the standard Java package naming convention*)

Plugin	Type*	Version	Release Date	Description	Source Repository	Issue Tracking
Core plugins				Plugins corresponding to default core phases (ie. clean, compile). They may have multiple goals as well.		
<code>clean</code>	B	3.0.0	2015-10-22	Clean up after the build.	SVN	JIRA
<code>compiler</code>	B	3.6.0	2016-10-29	Compiles Java sources.	SVN	JIRA
<code>deploy</code>	B	2.8.2	2014-08-27	Deploy the built artifact to the remote repository.	SVN	JIRA
<code>failsafe</code>	B	2.19.1	2016-01-03	Run the JUnit integration tests in an isolated classloader.	GIT	JIRA
<code>install</code>	B	2.5.2	2014-08-27	Install the built artifact into the local repository.	SVN	JIRA
<code>resources</code>	B	3.0.1	2016-06-03	Copy the resources to the output directory for including in the JAR.	SVN	JIRA
<code>site</code>	B	3.5.1	2016-04-15	Generate a site for the current project.	SVN	JIRA
<code>surefire</code>	B	2.19.1	2016-	Run the JUnit unit tests in an isolated classloader.	GIT	JIRA

.m2/repository/org/apache/maven

First launch mvn ... will download ~150Mo ...

```
$ ls
apache-maven          maven-artifact-manager    maven-plugin-parameter-documenter   plugins
archetype              maven-builder-support    maven-plugin-registry             plugin-testing
archetypes             maven-compat            maven-plugin-tools             plugin-tools
doxia                  maven-core               maven-plugin-tools-api        release
enforcer               maven-dependency-plugin  maven-profile                   reporting
indexer                maven-embedder          maven-project                  scm
its                     maven-error-diagnostics  maven-project-builder       shared
jxr                     maven-model             maven-repository-metadata     skins
maven                  maven-model-builder      maven-script                  surefire
maven-aether-provider  maven-monitor          maven-script-ant             wagon
maven-ant-tasks         maven-parent           maven-settings
maven-archiver          maven-plugin-api        maven-settings-builder
maven-artifact          maven-plugin-descriptor  maven-toolchain
```

Plugins ...

```
$ pwd
/home/arnaud/.m2/repository/org/apache/maven
$ du -sh
166M .
```

Maven .m2/repository/ .../plugins standard plugins in local repository

```
$ cd plugins/
$ ls
maven-acr-plugin          maven-idea-plugin
maven-ant-plugin           maven-install-plugin
maven-antrun-plugin        maven-invoker-plugin
maven-archetype-plugin     maven-jar-plugin
maven-assembly-plugin      maven-jarsigner-plugin
maven-build-helper         maven-javadoc-plugin
maven-changelog-plugin     maven-jxr-plugin
maven-changes-plugin       maven-linkcheck-plugin
maven-checkstyle-plugin    maven-metadata-central.xml
maven-clean-plugin          maven-metadata-central.xml.sha1
maven-compiler-plugin      maven-metadata-jboss-public-repository-group.xml
maven-dependency-plugin    maven-metadata-jboss-public-repository-group.xml.sha1
maven-deploy-plugin        maven-metadata-local.xml
maven-doap-plugin          maven-metadata-maven-central.xml
maven-docck-plugin         maven-metadata-maven-central.xml.sha1
maven-ear-plugin           maven-metadata-repo.jenkins-ci.org.xml
maven-eclipse-plugin       maven-metadata-repo.jenkins-ci.org.xml.sha1
maven-ejb-plugin           maven-patch-plugin
maven-enforcer-plugin      maven-pdf-plugin
maven-failsafe-plugin      maven-plugin-parent
maven-gpg-plugin            maven-plugin-plugin
maven-help-plugin          maven-plugins
maven-plugins-aggregator   maven-pmd-plugin
maven-project-info-reports-plugin
maven-rar-plugin           maven-reactor-plugin
maven-release-plugin        maven-remote-resources-plugin
maven-repository-plugin    maven-resources-plugin
maven-scm-publish-plugin   maven-shade-plugin
maven-site-plugin          maven-site-plugin
maven-source-plugin         maven-source-plugin
maven-stage-plugin          maven-stage-plugin
maven-surefire-plugin       maven-surefire-plugin
maven-surefire-report-plugin maven-surefire-report-plugin
maven-toolchains-plugin    maven-war-plugin
maven-verifier-plugin      resolver-status.properties
```

Use build/plugins

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.2</version>
    </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>build-helper-maven-plugin</artifactId>
      <version>1.12</version>
    </plugin>
  
```

Use plugin {

Use Another {

Declare plugin Dependencies ... with GAV
(as “build dependencies” but in section plugins)
=> Plugin will register itself in build lifecycle
see next for configuring..

Plugin <configuration>

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.2</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
```

Override
plugin
configuration

<source> = “-source” for javac compiler ...
list of options => cf next

Eclipse Auto-Completion for Plugin Configuration

The screenshot shows the Eclipse IDE interface with a code editor displaying XML configuration for a Maven plugin. A tooltip is open over the XML element `<source>`, providing detailed information about the attribute:

`required: false`
`type: String`
`expression: ${maven.compiler.source}`
`default: 1.5`

The tooltip also includes a descriptive text: "The -source argument for the Java compiler." At the bottom of the tooltip, there is a message: "Press 'Ctrl+Space' to show XML Template Proposals".

```
<version>3.2</version>
<configuration>
    <source>1.8</source>
    <t <!--> showDeprecation
    <t <!--> showWarnings
    <t <!--> skip
    <t <!--> skipMain
    <t <!--> skipMultiThreadWarning
    <t <!--> source
    <t <!--> staleMillis
    <e
<goals>
```

Maven Plugin Doc Site

<https://maven.apache.org/plugins>

Apache  / Maven / Plugins / Apache Maven Compiler Plugin / compiler:compile

OVERVIEW

Introduction

Goals

Usage

FAQ

License

Download

EXAMPLES

Compile Using A
Different JDK

Compile Using -source
and -target javac

compiler:compile

Full name:

org.apache.maven.plugins:maven-compiler-plugin:3.6.0:compile

Description:

Compiles Java source code.

Attributes:

- Requires a Maven project to be executed.
- Requires dependency resolution of artifacts in scope: `compile`.
- Since version: `2.0`.
- Binds by default to the lifecycle phase `compile`.

Optional Parameters

`source`

String

2.0

The `-source` argument for the Java compiler.

Default value is: `1.5`.

User property is: `maven.compiler.source`.

`staleMillis`

int

2.0

Sets the granularity in milliseconds of the last modification date for testing whether a source needs recompilation.

Default value is: `0`.

User property is: `lastModGranularityMs`.

`target`

String

2.0

The `-target` argument for the Java compiler.

Default value is: `1.5`.

User property is: `maven.compiler.target`.

`useIncrementalCompilation`

boolean

2.0

to enable/disable incremental compilation feature

See Plugin Goals & Usage

In 2016
default value to change
for jdk8 !!

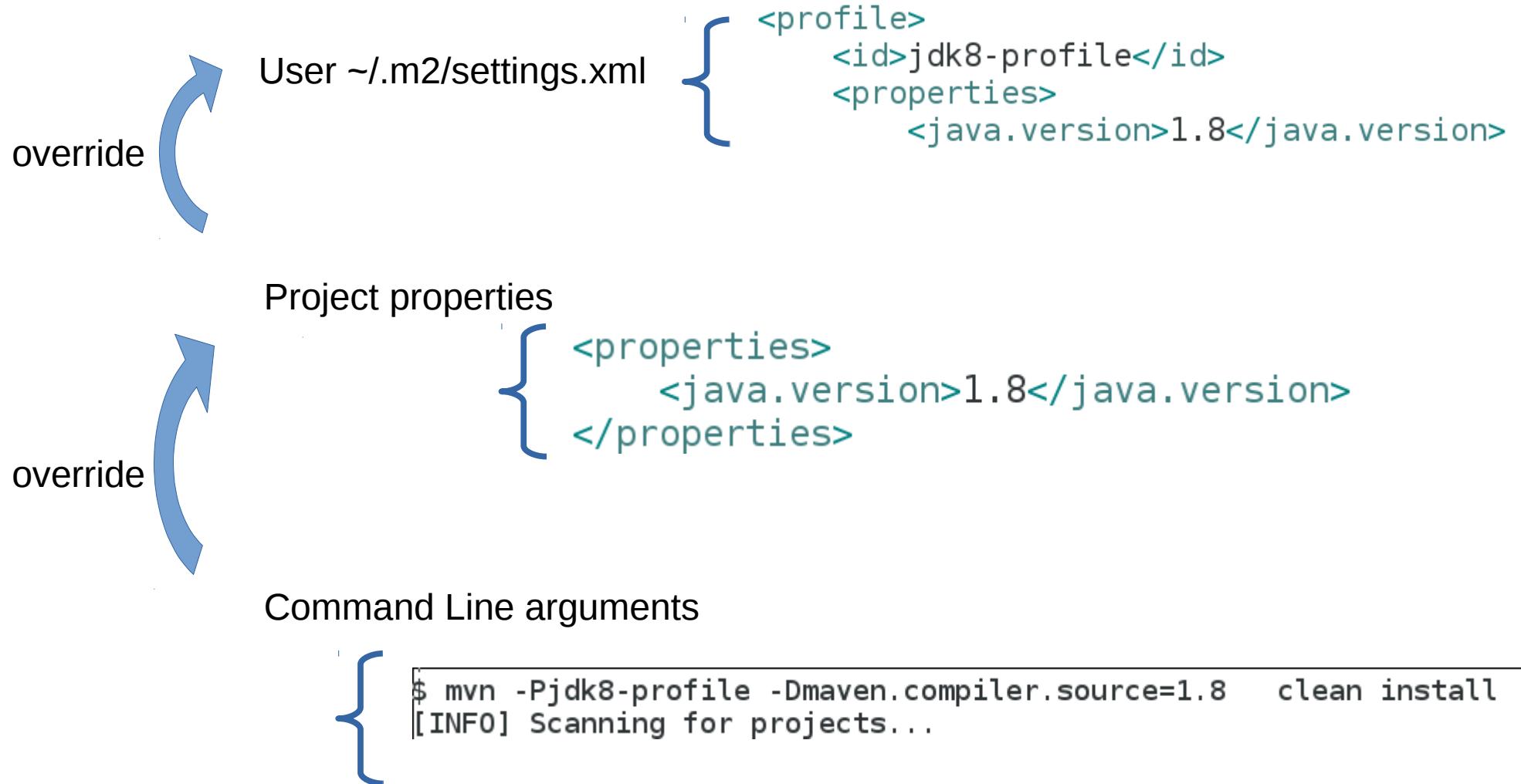
Configuration Override Properties



```
Project properties {<properties>
    <java.version>1.8</java.version>
</properties>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
        <source>1.8</source>
        <!-- default to <source>${maven.compiler.source}</source> -->
```

Properties Override Hierarchy (bis)



plugins/executions

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>build-helper-maven-plugin</artifactId>
  <version>1.12</version>
  <executions>
    <execution>
      <id>add-source</id>
      <phase>generate-sources</phase> ← When processing lifecycle phase
      <goals>
        <goal>add-source</goal> ← Then call plugin goal
      </goals>
      <configuration>
        <sources>
          <source>target/generated-source</source> ← With this parameter
        </sources>
      </configuration>
    </execution>
  </executions>
</plugin>
```

1 execution

Can add others

...

When processing lifecycle phase

Then call plugin goal

With this parameter

Plugin Execution Override

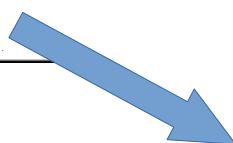


Command Line Explicit Plugin Goal Execution

```
$ mvn clean install | grep '@'  
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ test-mvn ---  
[INFO] --- build-helper-maven-plugin:1.12:add-source (add-source) @ test-mvn ---  
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ test-mvn ---  
[INFO] --- maven-compiler-plugin:3.2:compile (default-compile) @ test-mvn ---  
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ test-mvn ---  
[INFO] --- maven-compiler-plugin:3.2:testCompile (default-testCompile) @ test-mvn ---  
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ test-mvn ---  
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ test-mvn ---  
[INFO] --- maven-install-plugin:2.4:install (default-install) @ test-mvn ---
```

```
$  
$ mvn [ ]
```

Call build lifecycle 1..* phase(s)
=> sequence of plugin goals



Call explicit 1..* plugin goal(s)

```
$ mvn compiler:compile | grep '@'  
[INFO] --- maven-compiler-plugin:3.2:compile (default-cli) @ test-mvn ---
```

Example of Plugin Goals

```
mvn help:effective-pom
```

```
mvn dependency:tree
```

```
mvn springboot:run
```

```
mvn sonar:sonar
```

```
mvn compiler:compile
```

```
# using explicit group:artifact:goal
```

```
mvn org.apache.maven.plugins:maven-compiler-plugin:compile
```

```
# explicit version  group:artifact:version:goal
```

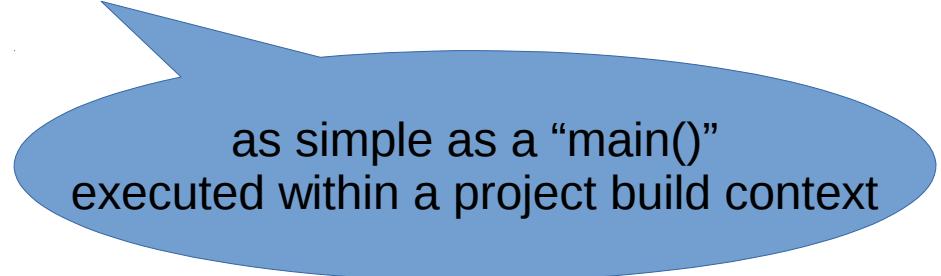
```
mvn org.apache.maven.plugins:maven-compiler-plugin:3.1:compile
```

Plugin Internal “Mojo” Classes

```
import org.apache.maven.plugin.AbstractMojo;
import org.apache.maven.plugins.annotations.Mojo;

@Mojo(name = "hello-world")
public class MyHelloWorldMojo extends AbstractMojo {

    public void execute() {
        getLog().info("Hello Mojo World!");
    }
}
```



as simple as a “main()”
executed within a project build context

Run Mojo Hello World ...

```
$ mvn fr.an.tests:test-mvn-mojo:1.0-SNAPSHOT:hello-world
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building test-mvn-mojo Maven Mojo 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- test-mvn-mojo:1.0-SNAPSHOT:hello-world (default-cli) @ test-mvn-mojo
[INFO] Hello Mojo World!
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.144 s
[INFO] Finished at: 2016-11-06T23:30:52+01:00
[INFO] Final Memory: 5M/283M
[INFO] -----
```

Mojo Context Injection @Parameter

```
@Mojo(name = "add-source", defaultPhase = LifecyclePhase.GENERATE_SOURCES,  
public class AddSourceMojo extends AbstractMojo {  
  
    @Parameter(required = true) Inject from <configuration><sources>..  
    private File[] sources; default properties for values  
  
    @Parameter(readonly = true, defaultValue = "${project}")  
    private MavenProject project; Inject the FAMOUS pom  
  
    public void execute() {  
        for (File source : this.sources) {  
            this.project.addCompileSourceRoot(source.getAbsolutePath());  
            if (!(getLog().isInfoEnabled()))  
                continue;  
            getLog().info("Source directory: " + source + " added.");  
        }  
    }  
}
```



Maven Core Concepts 4 : Build Lifecycle - Phases

Contents [hide]

- ✓ Example
- 2 Concepts
 - ✓ 2.1 Project Object Model
 - ✓ 2.2 Plugins
 - 2.3 Build lifecycles**
 - ✓ 2.4 Dependencies
- 3 Maven compared with Ant
- 4 IDE integration





Maven Phases

Build lifecycles [edit]

Build lifecycle is a list of named *phases* that can be used to give order to goal execution. One of Maven's standard lifecycles is the *default lifecycle*, which includes the following phases, in this order:^[12]

```
1 validate
2 generate-sources
3 process-sources
4 generate-resources
5 process-resources
6 compile
7 process-test-sources
8 process-test-resources
9 test-compile
10 test
11 package
12 install
13 deploy
```

Goals provided by plugins can be associated with different phases of the lifecycle. For example, by default, the goal "compiler:compile" is associated with the "compile" phase, while the goal "surefire:test" is associated with the "test" phase. Consider the following command:

Register Goals Execution in Phases

```
<execution>
  <id>add-source</id>
  <phase>generate-sources</phase>
  <goals>
    <goal>add-source</goal>
```

← Explicit
Goal Execution
per <phase>

```
<plugin>
  <groupId>fr.an.tests</groupId>
  <artifactId>test-mvn-mojo</artifactId>
  <version>1.0-SNAPSHOT</version>
  <extensions>true</extensions>
</plugin>
```

← Implicit
Goal Execution
per Phase

```
@Mojo(name = "compile-hello-world", defaultPhase=LifecyclePhase.COMPILE)
@Execute(goal = "compile-hello-world",
  phase = LifecyclePhase.COMPILE, lifecycle = "default")
public class MyCompileHelloWorldMojo extends AbstractMojo {
```

Project Type → LifeCycle → Phases → Plugins Mojo

```
<groupId>fr.an.tests</groupId>
<artifactId>test-mvn-archetype1</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
```

repository/org/apache/maven/plugins/
Maven-jar-plugin-3.0.2.jar

META-INF/plexus/components.xml

```
<!--
 | Defining the phases with their appropriate plugins
 | and versions which will be executed during the 'default'
 | life cycle.
-->
<component>
<role>org.apache.maven.lifecycle.mapping.LifecycleMapping</role>
<role-hint>jar</role-hint>
<implementation>org.apache.maven.lifecycle.mapping.DefaultLifecycleMapping</implementation>
<configuration>
<lifecycles>
<lifecycle>
<id>default</id>
<!-- START SNIPPET: jar-lifecycle -->
<phases>
<process-resources>
  org.apache.maven.plugins:maven-resources-plugin:2.7:resources
</process-resources>
<compile>
  org.apache.maven.plugins:maven-compiler-plugin:3.5.1:compile
</compile>
<process-test-resources>
  org.apache.maven.plugins:maven-resources-plugin:2.7:testResources
</process-test-resources>
<test-compile>
  org.apache.maven.plugins:maven-compiler-plugin:3.5.1:testCompile
</test-compile>
<test>
  org.apache.maven.plugins:maven-surefire-plugin:2.19.1:test
</test>
<package>
  org.apache.maven.plugins:maven-jar-plugin:3.0.2:jar
</package>
<install>
  org.apache.maven.plugins:maven-install-plugin:2.5.2:install
</install>
<deploy>
  org.apache.maven.plugins:maven-deploy-plugin:2.8.2:deploy
</deploy>
</phases>
</lifecycle>
</lifecycles>
</component>
```

Conclusion

Questions ?

arnaud.nauwynck@gmail.com

Only a “Short”
Introduction to Concepts...
other docs:

<http://arnaud-nauwynck.github.io/>
<http://arnaud.nauwynck.free.fr/>

This document:

<http://arnaud-nauwynck.github.io/docs/Maven-Intro-Concepts.pdf>