Dependency Management with Maven

PU Machine learning mit Java, Weka und UIMA

Nils Reiter,
nils.reiter@uni-koeln.de

November 10, 2020
Winter term 2020/21

Version Control

- Git: Software to do version control
  - GitHub: Platform with services around git
- Commit: Set of changes, potentially in multiple files
  - Each commit knows its predecessor
- Branch: Pointer to a commit
  - ...but it feels/behaves like an actual branch

Version Control

- Git: Software to do version control
  - GitHub: Platform with services around git
- Commit: Set of changes, potentially in multiple files
  - Each commit knows its predecessor
- Branch: Pointer to a commit
  - ...but it feels/behaves like an actual branch
- Merging: Combining changes from multiple branches
  - Most of the time, merging happens automatically
    - In each case, take the most recent version
  - Sometimes, we have to merge manually
- GUIs help a lot

Exercise 1

`https://github.com/idh-cologne-machine-learning-mit-java/exercise-01`

Exercise 1

`https://github.com/idh-cologne-machine-learning-mit-java/exercise-01`

- GitHub compare view:

  `https://github.com/idh-cologne-machine-learning-mit-java/exercise-01/compare/b7f3c517bc836b77cf0140e3696f15bd89ceb5b6...main`
- Commit ids can be abbreviated – as long as they are unique
  - Seven characters is enough in most cases:
    b7f3c517bc836b77cf0140e3696f15bd89ceb5b6 → b7f3c517
- Protection of main branch

Section 2

Maven

Introduction

- Maven: A build tool
  - Build process: Convert source code to binary code
  - More than pure compilation: Libraries (in correct versions), ressources, packages, documentation, testing, ...
- Build process often automated (continuous integration, CI)
  - Maven can also run on GitHub
- Documentation: `https://maven.apache.org/guides/`

How to use Maven

Two core ingredients

`pom.xml`

- "Project Object Model"
- XML-file
- Contains all meta data about the build process
- `groupId`/`artifactId`/`version` identifies a Maven artifact

How to use Maven

Two core ingredients

`pom.xml`

- "Project Object Model"
- XML-file
- Contains all meta data about the build process
- `groupId`/`artifactId`/`version` identifies a Maven artifact

Standard Directory layout

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

Repositories

Local

- Maven maintains a local repository in your home directory
  - Unix: `~/.m2`
  - Windows: `C:\Users\<USERNAME>\.m2`
- Dependencies are installed in this directory
- Classpath is maintained to include the necessary directories
- Local repository can become quite large (my laptop: 7GB), cleanup from time to time

Repositories

Local

- Maven maintains a local repository in your home directory
  - Unix: `~/.m2`
  - Windows: `C:\Users\<USERNAME>\.m2`
- Dependencies are installed in this directory
- Classpath is maintained to include the necessary directories
- Local repository can become quite large (my laptop: 7GB), cleanup from time to time

Remote repositories

- Central repository: Filled by the maven community (maintained by Sonatype)
  - `https://search.maven.org`
- Large organisations may maintain their own internal repositories

Build Lifecycle

Lifecycle	Description
`validate`	All necessary information present
`compile`	Compile source code
`test`	Run unit tests on the compiled code
`package`	Package into a jar file
`verify`	Run tests on the jar file
`install`	Install into the local maven repository
`deploy`	Copy package to a remote maven repository

Table: Most important Maven default lifecycle phases

Build Lifecycle

Lifecycle	Description
`validate`	All necessary information present
`compile`	Compile source code
`test`	Run unit tests on the compiled code
`package`	Package into a jar file
`verify`	Run tests on the jar file
`install`	Install into the local maven repository
`deploy`	Copy package to a remote maven repository

Table: Most important Maven default lifecycle phases

- Can be used on command line:
  - `$ mvn package` to create a jar file
  - `$ mvn clean compile` to remove old compilations and compile again
- Technically, each phase is handled by a different plugin
- Other lifecycles: `clean` and `site`

pom.xml Sections

Properties

```
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>
  ...
</project>
```

- Properties are specified in top-level `properties`-element
- Some properties are project-global (`project.build.sourcEncoding`)
- Some properties specific to some plugins (`maven.compiler.target`)

pom.xml Sections

Build

```
<project ...>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>GROUPID OF PLUGIN</groupId>
        <artifactId>ARTIFACT ID OF PLUGIN</artifactId>
        <version>VERSION OF PLUGIN</version>
        <configuration>PLUGIN CONFIGURATION PARAMETERS</configuration>
        <executions>
          <execution>
            <id>generate</id>
            <goals>
              <goal>generate</goal>
            </goals>
          </execution>
        </executions></plugin><plugin>...</plugin></plugins></build>
  ...
</project>
```

pom.xml Sections

Build

- Allows to change each lifecycle
- Example use cases
  - Generate source code from XML files (before compiling)
  - Generate test resources
  - Exclude specific resources from packaging
  - Compile other languages
  - `https://github.com/nilsreiter/CorefAnnotator/blob/master/pom.xml`

pom.xml Sections

Dependencies

```
<project ...>
  ...
  <dependencies>
    <dependency>
      <groupId>org.glassfish.jaxb</groupId>
      <artifactId>jaxb-runtime</artifactId>
      <version>2.3.2</version>
    </dependency>
    <dependency>...</dependency>
  </dependencies>
  ...
</project>
```

- Top-level element
- Specify each dependency as triple `groupId`, `artifactId`, and `version`

pom.xml Sections

Dependencies

```
<project ...>
  ...
  <dependencies>
    <dependency>
      <groupId>org.glassfish.jaxb</groupId>
      <artifactId>jaxb-runtime</artifactId>
      <version>2.3.2</version>
    </dependency>
    <dependency>...</dependency>
  </dependencies>
  ...
</project>
```

- Top-level element
- Specify each dependency as triple `groupId`, `artifactId`, and `version`
- All dependencies in maven universe!

Additional elements

- `scope`: compile/provided/runtime/test/system
- `classifier`: Arbitrary string to distinguish variants
- `type`: jar/war/ejb-client/...

pom.xml Sections

Dependencies

Specification	Description
`1.0`	1.0 if no other version appears earlier in dependency tree
`[1.0]`	1.0 and only 1.0.
`(,1.0]`	Any version $<=$ 1.0
`[1.2,1.3]`	Any version between 1.2 and 1.3, inclusive
`[1.0,2.0)`	Any version between 1.0 inclusive and 2.0 exclusive
`[1.5,)`	Any version greater than or equal to 1.5
`(,1.0],[1.2,)`	Any version $<=$ 1.0 or $>=$ 1.2, but not 1.1
`(,1.1),(1.1,)`	Any version except 1.1

Table: Specifying versions for dependencies in Maven. Works best with "semantic versioning" `https://semver.org/spec/v1.0.0.html`

pom.xml

Properties

```
<project>
  <properties>
    <version.javafx>14.0.2.1</version.javafx>
  </properties>
  ...
  <dependencies><dependency>
      <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>${version.javafx}</version>
      </dependency><dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-swing</artifactId>
        <version>${version.javafx}</version>
      </dependency></dependencies>
</project>
```

- Properties can be used throughout the POM
- Can be defined in other ways as well: Profiles, inheritance, default, system OS, ...
  - `http://maven.apache.org/ref/3.6.3/maven-model-builder/#Model_Interpolation`

Inheritance

- Each POM inherits from its parent POM
- Root-POM:
  `https://maven.apache.org/ref/3.6.3/maven-model-builder/super-pom.html`

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <version>1</version>
  </parent>
  <artifactId>my-module</artifactId>
</project>
```

- This POM inherits `groupId` and `version` from its parent

Inheritance

What is inherited by a child POM?

- dependencies
- developers and contributors
- plugin lists (including reports)
- plugin executions with matching ids
- plugin configuration
- resources

Summary

- Maven to organise your dependency
- Works across computers
  - because dependencies are downloaded automatically from Maven central
- Formal definition of the build process
  - Replication
  - Documentation
- Maven will be used throughout the semester for dependency handling

Section 3

Exercise 02

Exercise 02

```
42
Incorrect value

100
0,0,1,0,0,0,0

140
0,0,1,0,2,0,0

5
0,0,0,0,0,0,1

200
0,1,0,0,0,0,0

1000
2,0,0,0,0,0,0

1025
2,0,0,0,1,0,1
```

- Given an amount, ATM tells us how many of which banknote it would give us
- Assumption: Euro banknotes system (500, 200, 100, 50, 20, 10, 5)
- Exercise 02
  - Change output such that we use a library via maven
  - Add a pre-calculation check that works only in Java 8 and higher
- `https://github.com/idh-cologne-machine-learning-mit-java/exercise-02`