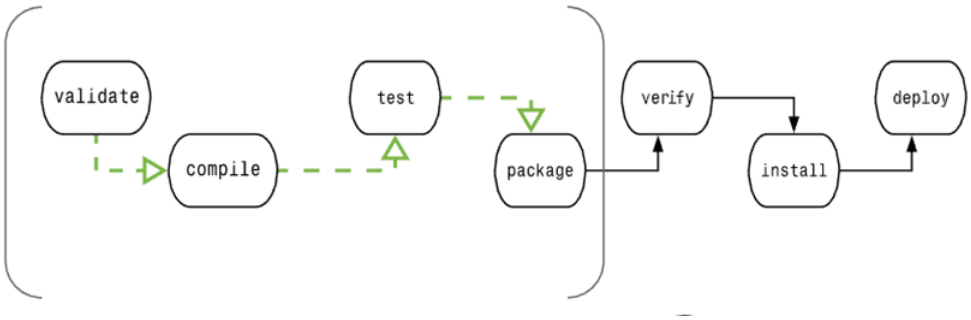
[CS304] Tutorial 6 - Maven & Build Systems

Part 1. Maven Intro & Setup

Maven is a software project management and comprehension tool, can be used for building and managing any Java-based project. 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.

```
1.Making the build process easy
2.Providing a uniform build system
3.Providing quality project information
4.Encouraging better development practices
```

A Build Lifecycle is Made Up of Phases. Each of these build lifecycles is defined by a list of build phases.



When the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar, war), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository. For example, the default lifecycle comprises of the following phases:

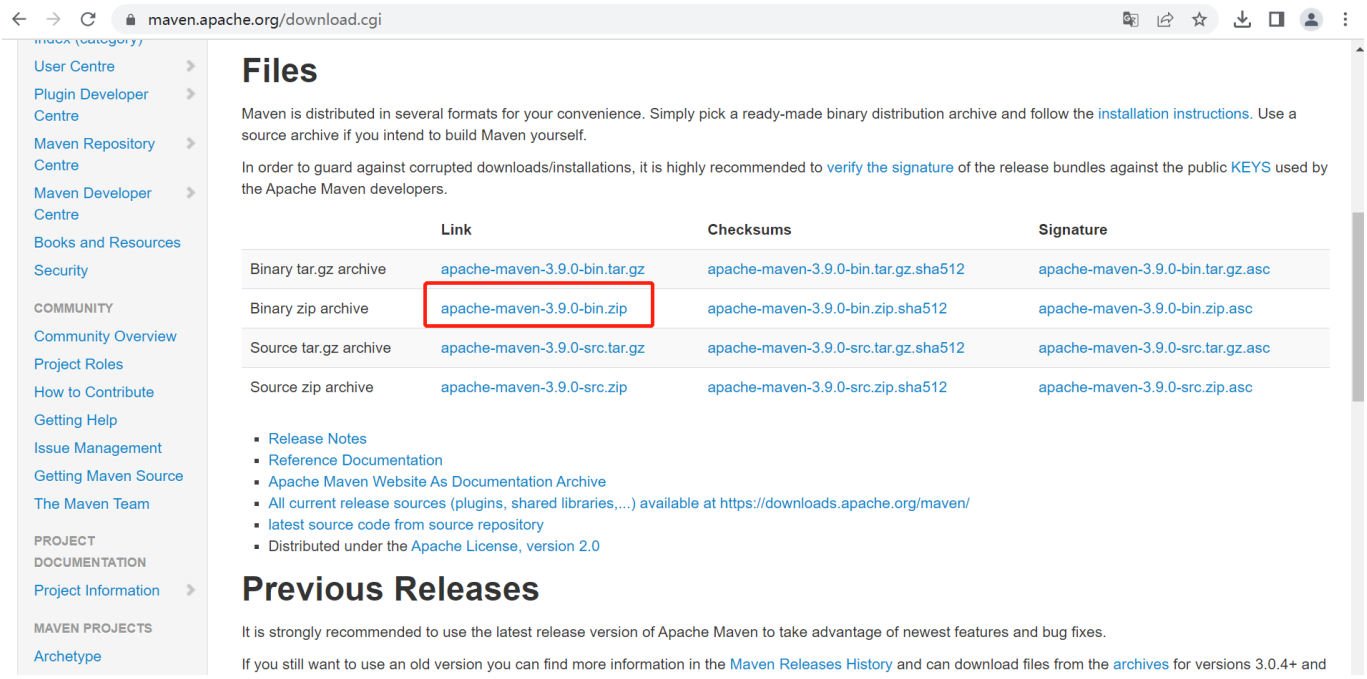
```
validate - validate the project is correct and all necessary information is
available
compile  - compile the source code of the project
test     - test the compiled source code using a suitable unit testing framework.
These tests should not require the code be packaged or deployed
package  - take the compiled code and package it in its distributable format, such
as a JAR.
verify   - run any checks on results of integration tests to ensure quality
criteria are met
install  - install the package into the local repository, for use as a dependency
in other projects locally
deploy   - done in the build environment, copies the final package to the remote
repository for sharing with other developers and projects.
```

These lifecycle phases (other lifecycle phases not shown here) are executed sequentially to complete the default lifecycle.

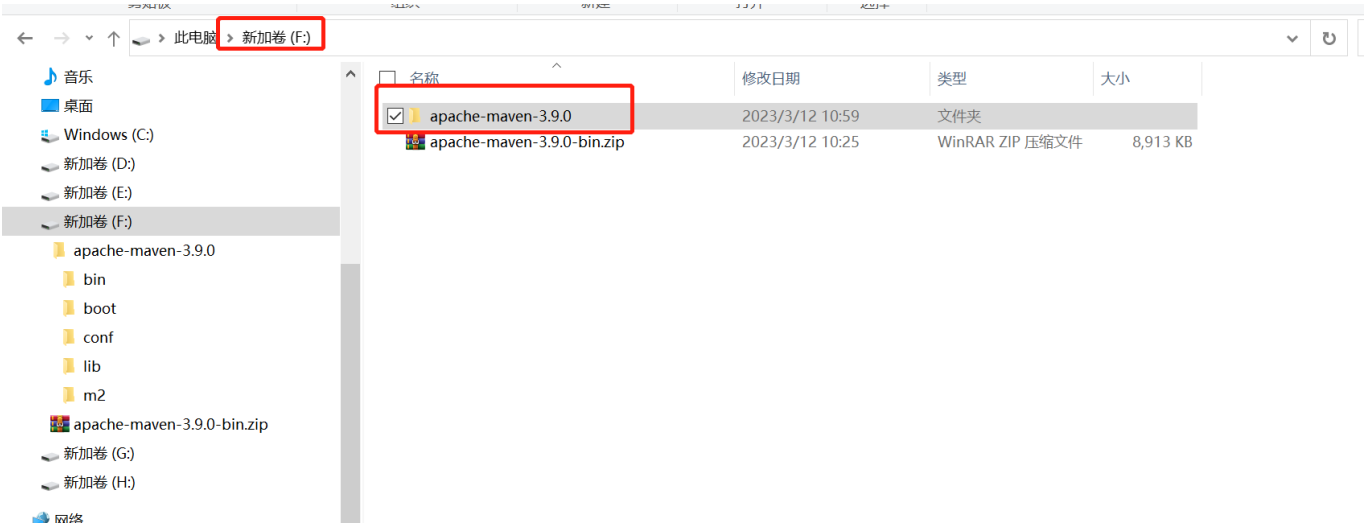
Maven official website:

https://maven.apache.org/index.html

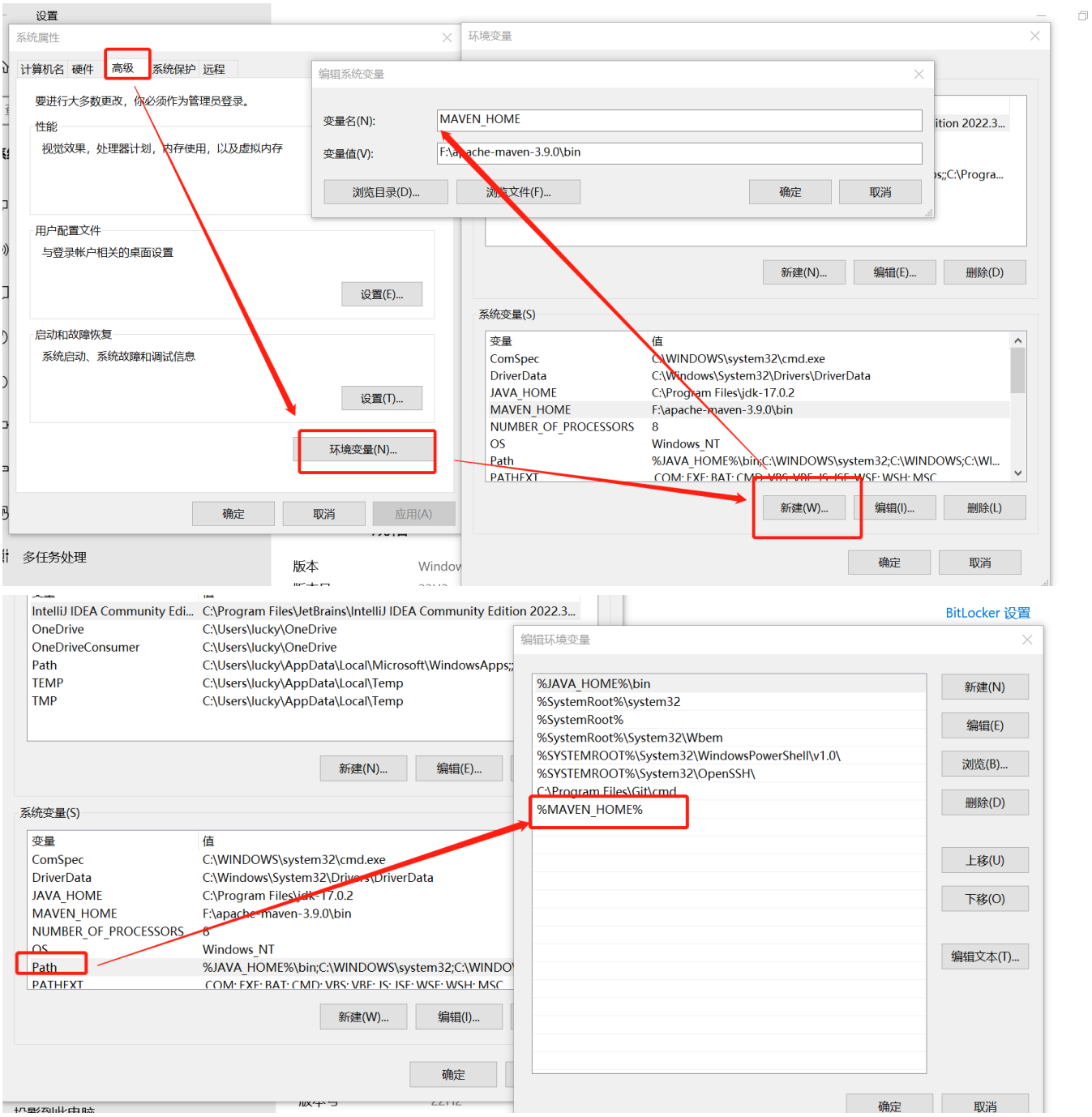
Download and install here:

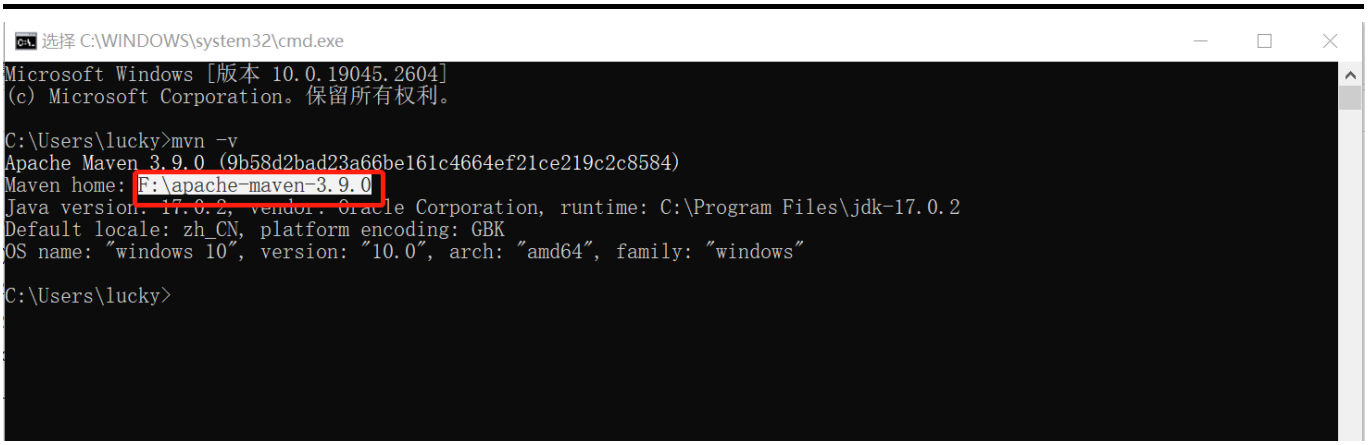


Extract distribution archive in any directory:



Config environment variables: MAVEN_HOME





Part 2. Hello Maven & Basic Commands

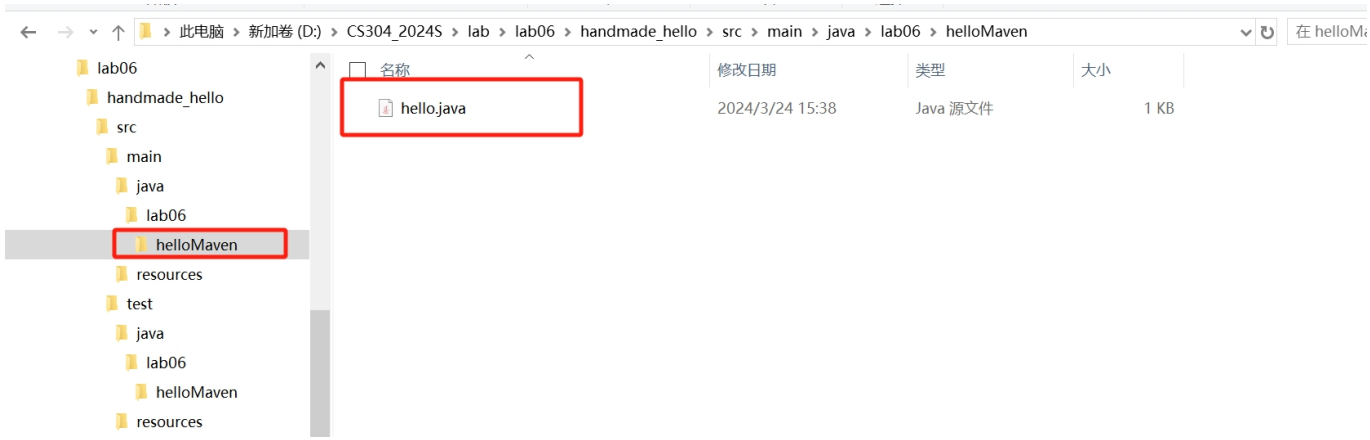
1. Create a simple Java project

1. Create files and `pom.xml` below:

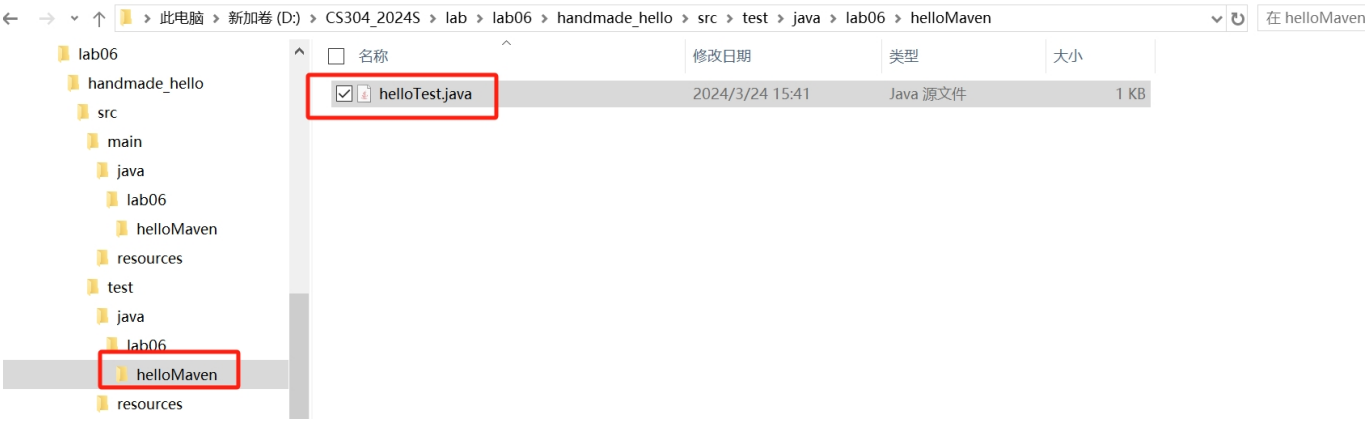
The `src/main/java` directory contains the project source code, the `src/test/java` directory contains the test source, and the `pom.xml` file is the project's Project Object Model, or POM.

2. Create `hello.java` (in `\src\main\java\lab06\helloMaven`) and `helloTest.java`(in `\src\test\java\lab06\helloMaven`):

```
package lab06.helloMaven;

public class hello {
    public static void main(String[] args) {
        hello hi=new hello();
        hi.echo("CS304");
    }

    public String echo(String name){
         String echo="hello,"+name;
         System.out.println(echo);
         return echo;
    }
}
```

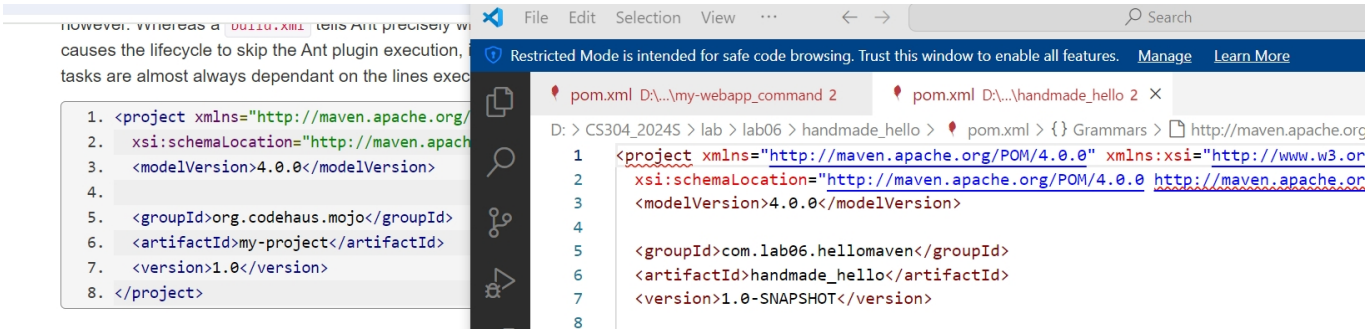


```
package lab06.helloMaven;

import org.junit.Test;
import org.junit.Assert;

public class helloTest {

    @Test
    public void testEcho(){
        hello hi_test=new hello();
        String echoTest=hi_test.echo("CS304");
        Assert.assertEquals("hello,CS304",echoTest);

    }

}
```



3. Maven Coordinates

1. modelVersion: pom model version.
2. groupId: This is generally unique amongst an organization or a project.
3. artifactId: The artifactId is generally the name that the project is known by.



Update our maven coordinates:

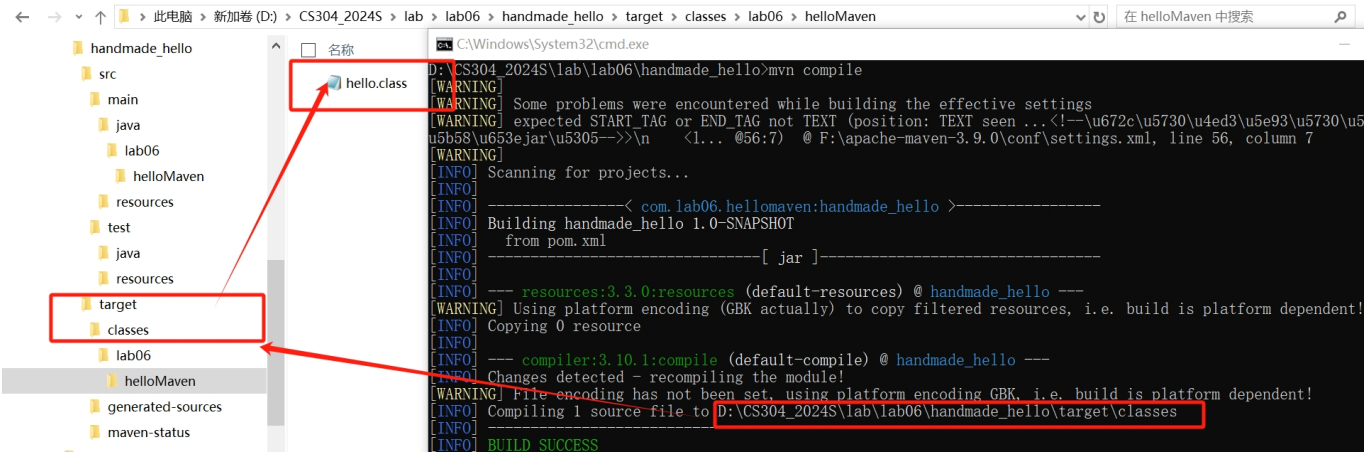
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```

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

  <groupId>com.lab06.hellomaven</groupId>
  <artifactId>handmade_hello</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

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

</project>
```

4. Run basic maven commands:

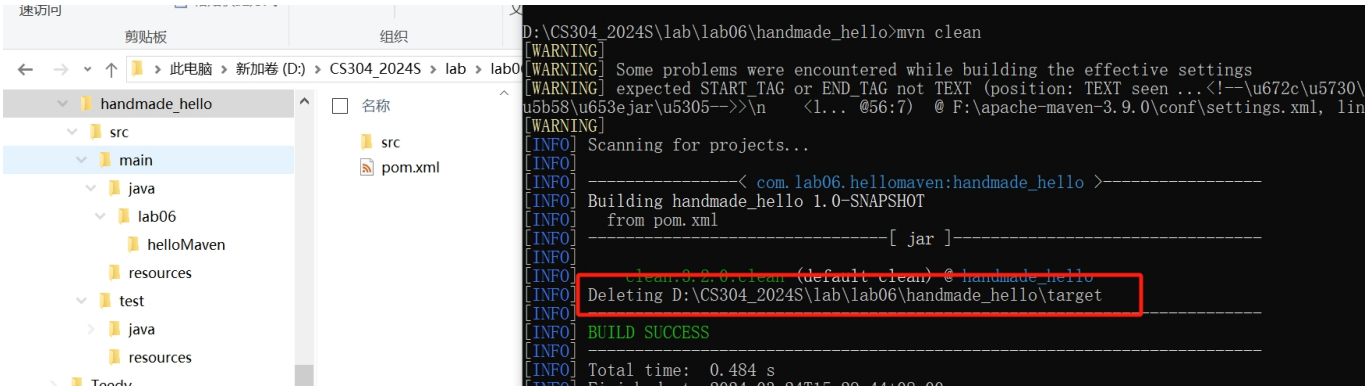
1.

```
mvn compile
```



2.

```
mvn clean
```

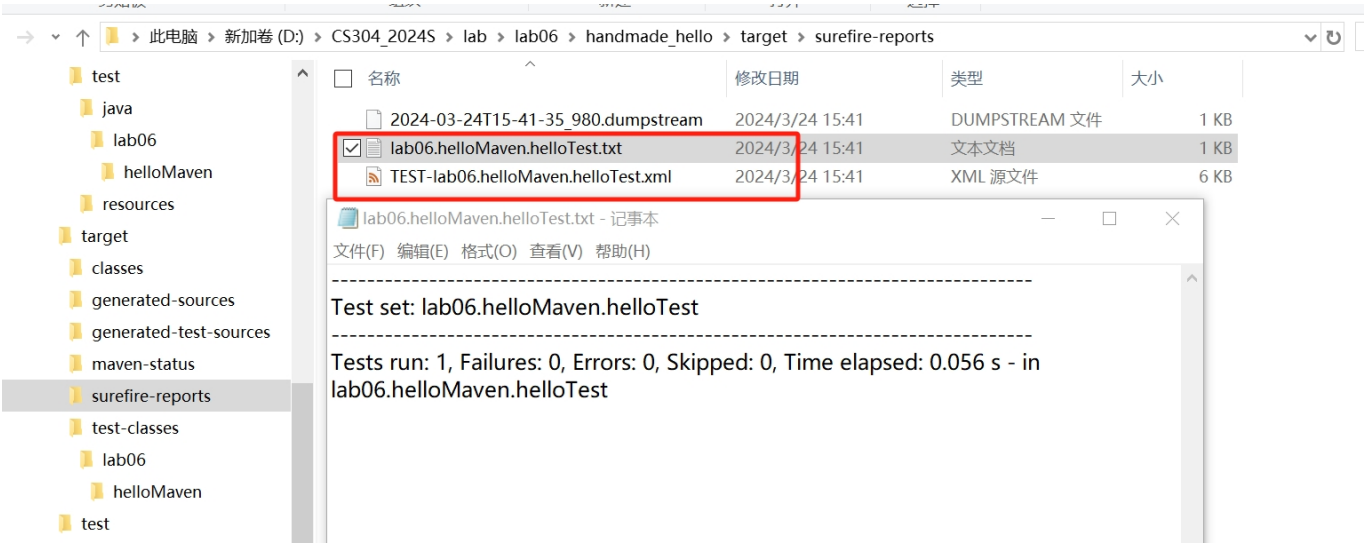


3.

```
mvn test
```

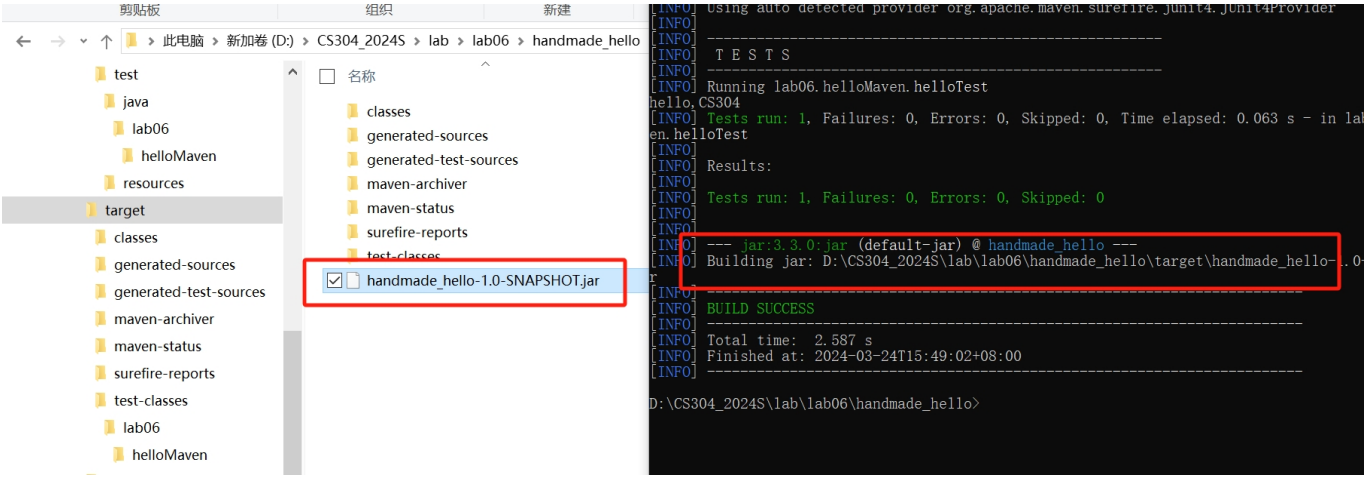
```
[INFO]
[INFO] ----------------< com.lab06.hellomaven:handmade_hello >----------------
[INFO] Building handmade_hello 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.3.0:resources (default-resources) @ handmade_hello ---
[WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- compiler:3.10.1:compile (default-compile) @ handmade_hello ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- resources:3.3.0:testResources (default-testResources) @ handmade_hello ---
[WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- compiler:3.10.1:testCompile (default-testCompile) @ handmade_hello ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding GBK, i.e. build is platform dependent!
[INFO] Compiling 1 source file to D:\CS304_2024S\lab\lab06\handmade_hello\target\test-classes
[INFO]
[INFO] --- surefire:3.0.0-M8:test (default-test) @ handmade_hello ---
[INFO] Using auto detected provider org.apache.maven.surefire.junit4.JUnit4Provider
Downloading from aliyunmaven: https://maven.aliyun.com/repository/public/org/apache/maven/surefire/surefire-junit4/3.0.0
-M8/surefire-junit4-3.0.0-M8.jar
Downloaded from aliyunmaven: https://maven.aliyun.com/repository/public/org/apache/maven/surefire/surefire-junit4/3.0.0-
M8/surefire-junit4-3.0.0-M8.jar (18 kB at 12 kB/s)
Downloading from aliyunmaven: https://maven.aliyun.com/repository/public/org/apache/maven/surefire/surefire-junit4/3.0.0
-M8/surefire-junit4-3.0.0-M8.pom
Downloaded from aliyunmaven: https://maven.aliyun.com/repository/public/org/apache/maven/surefire/surefire-junit4/3.0.0-
M8/surefire-junit4-3.0.0-M8.pom (2.9 kB at 7.2 kB/s)
Downloading from aliyunmaven: https://maven.aliyun.com/repository/public/org/apache/maven/surefire/common-junit4/3.0.0-M
8/common-junit4-3.0.0-M8.pom
Downloaded from aliyunmaven: https://maven.aliyun.com/repository/public/org/apache/maven/surefire/common-junit4/3.0.0-M8
/common-junit4-3.0.0-M8.pom (3.0 kB at 6.5 kB/s)
Downloading from aliyunmaven: https://maven.aliyun.com/repository/public/org/apache/maven/surefire/common-junit4/3.0.0-M
8/common-junit4-3.0.0-M8.jar
Downloaded from aliyunmaven: https://maven.aliyun.com/repository/public/org/apache/maven/surefire/common-junit4/3.0.0-M8
/common-junit4-3.0.0-M8.jar (26 kB at 49 kB/s)
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running lab06.helloMaven.helloTest
hello,CS304
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.056 s - in lab06.helloMaven.helloTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

Check the test result:



4.

```
mvn package
```

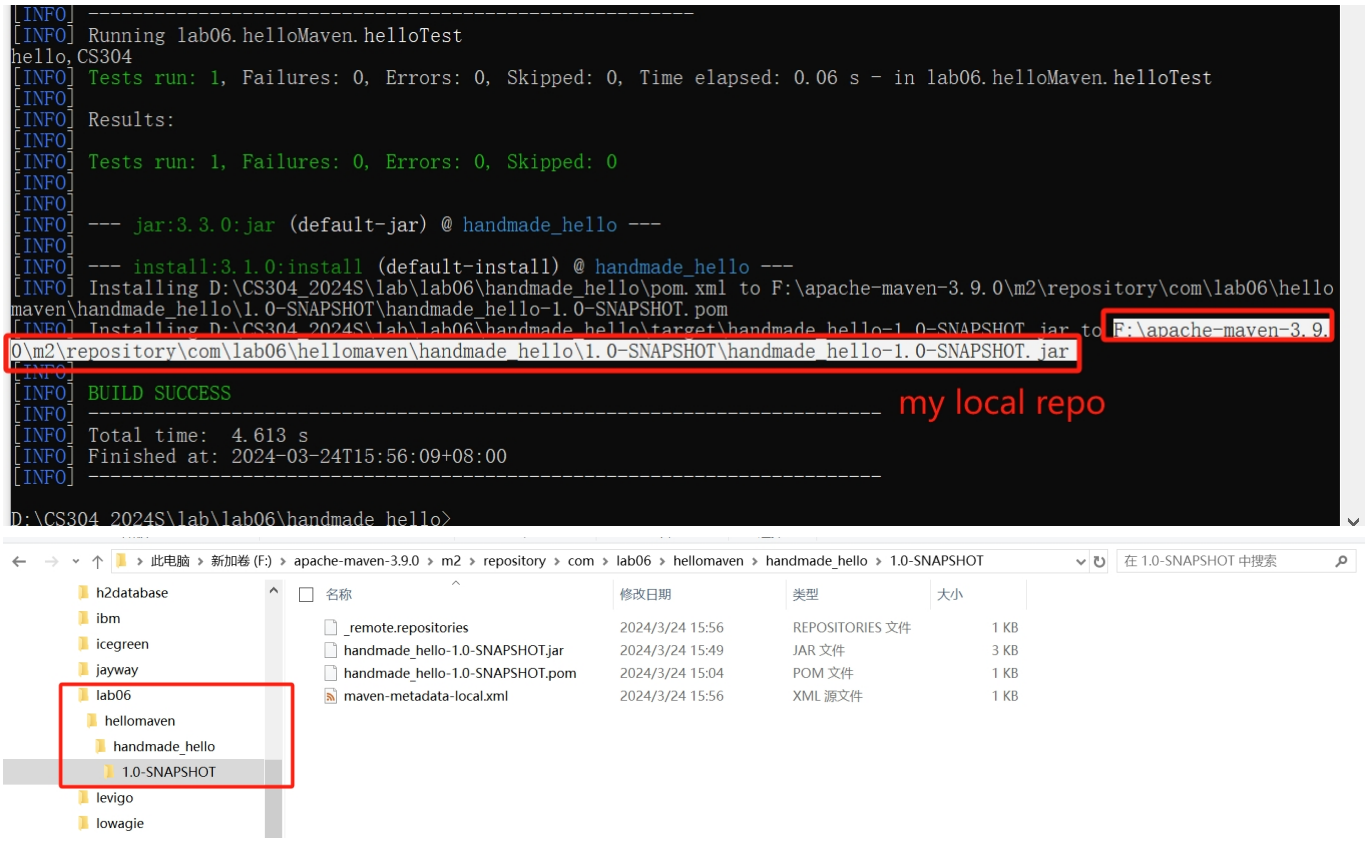



```
java -cp target/handmade_hello-1.0-SNAPSHOT.jar lab06.helloMaven.hello
```

```
D:\CS304_2024S\lab\lab06\handmade_hello>java -cp target/handmade_hello-1.0-SNAPSHOT.jar lab06.helloMaven.hello
hello,CS304
```

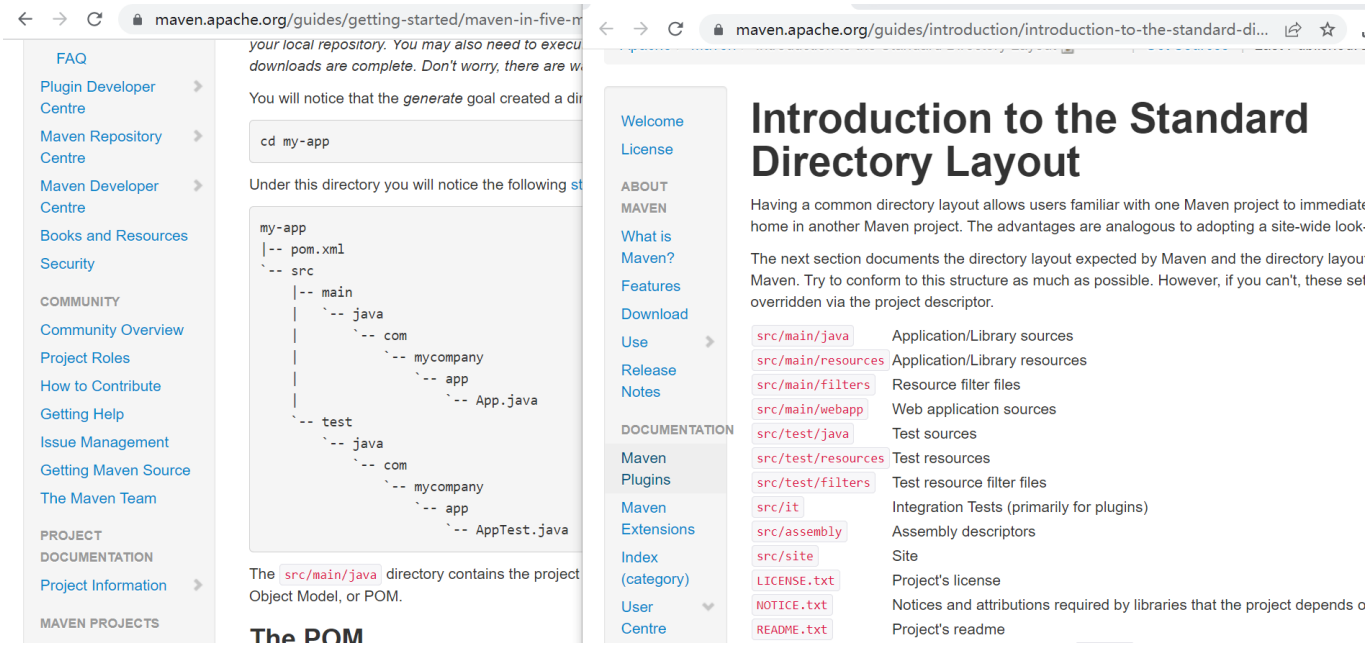
5.

```
mvn install
```



2. Standard Directory Layout

Maven has its own standard directory layout: https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

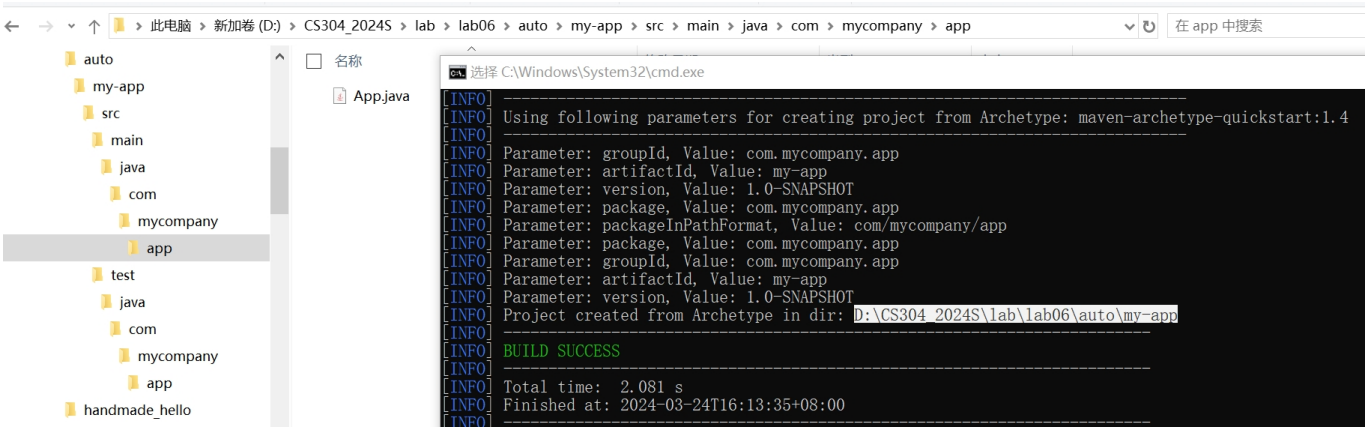



```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -
DinteractiveMode=false
```

We use this command to create a directory and start it. This `archetype:generate` goal created a simple project based upon a maven-archetype-quickstart archetype.

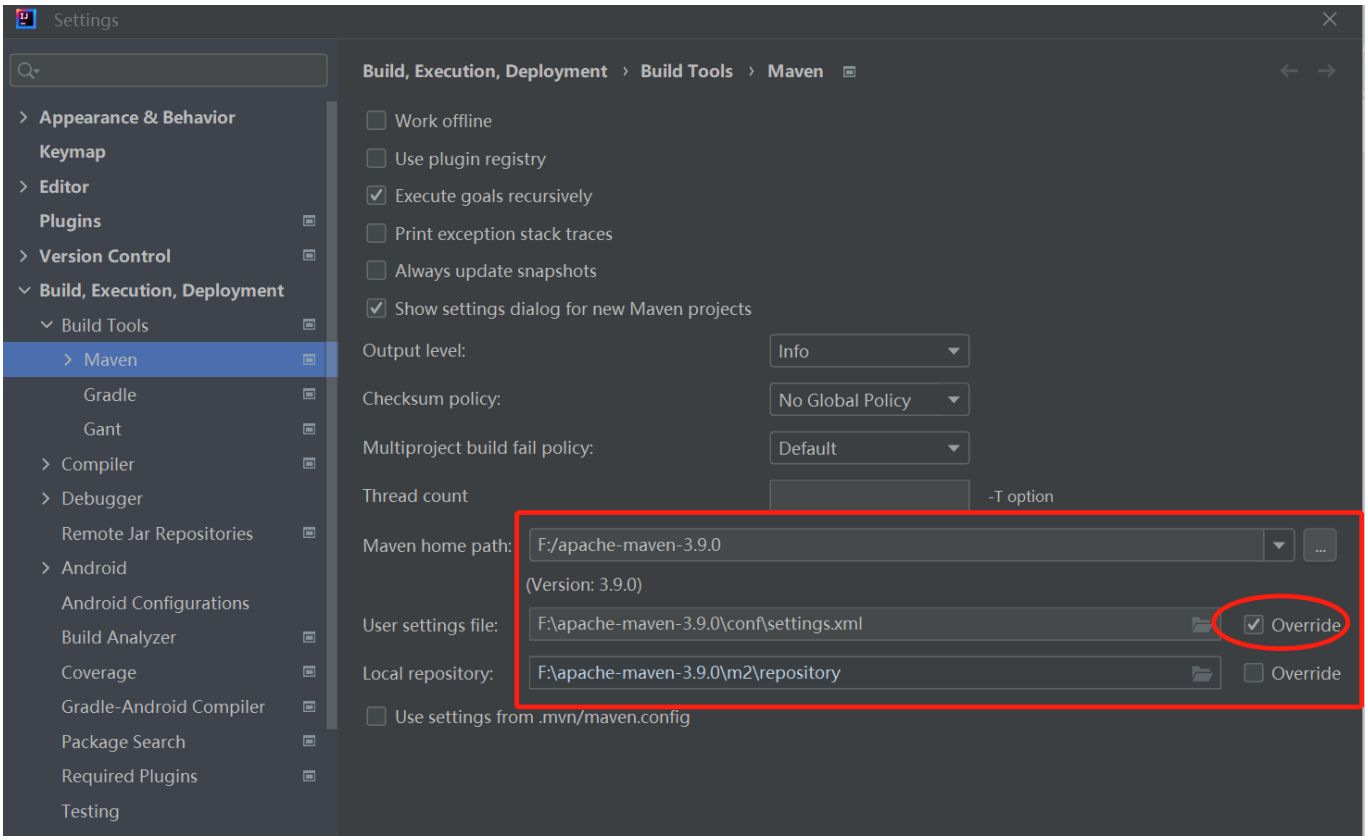
You can rename as you like:

```
-DgroupId=com.mycompany.app
-DartifactId=my-app
```

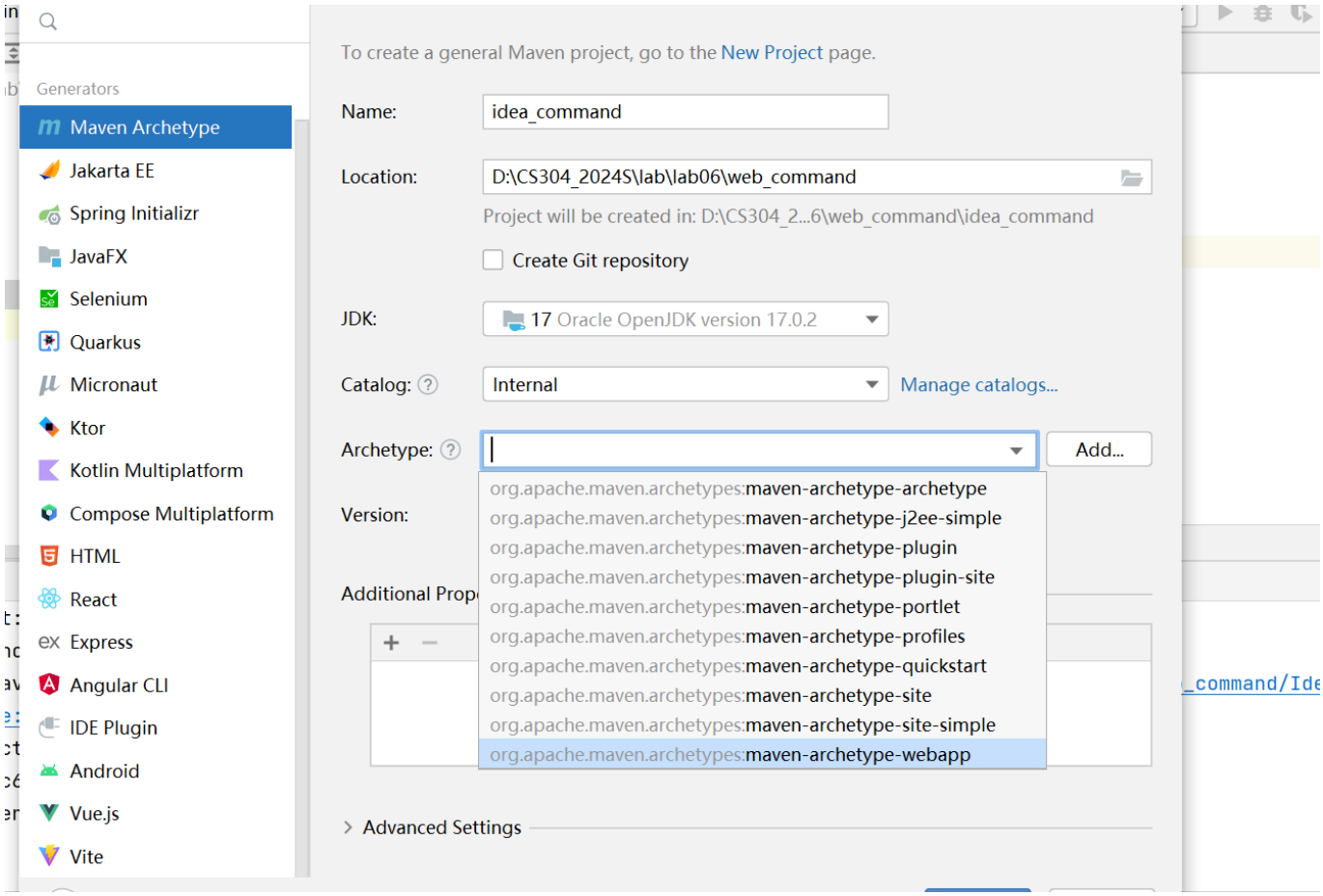


Part 3. Use Maven in IDEA & Run a web app with Jetty

Maven is bundled in IDEA. You can also modify the Maven configuration in IDEA: File -> Settings:



1. Create a Maven project



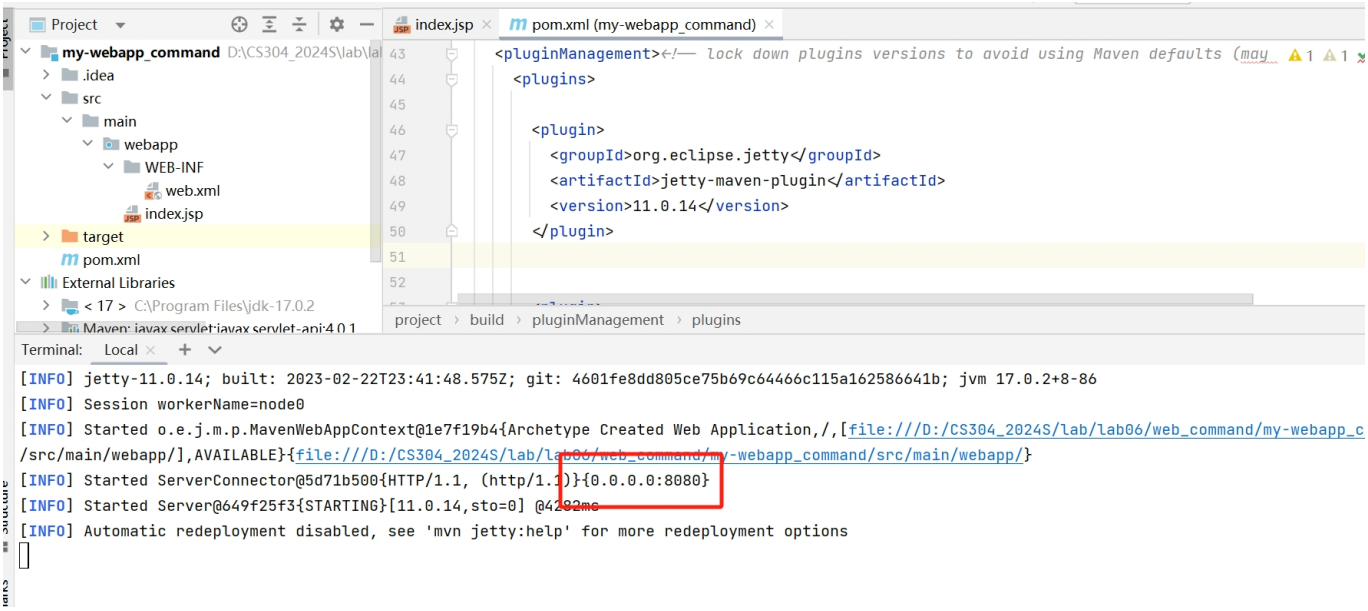
2. Add the java servlet dependency

3. Add jetty plugin:

```
<plugin>
  <groupId>org.eclipse.jetty</groupId>
  <artifactId>jetty-maven-plugin</artifactId>
  <version>11.0.14</version>
</plugin>
```

4. Run jetty server:

```
mvn install
mvn jetty:run
```



5. Open your browser:

Hello World!

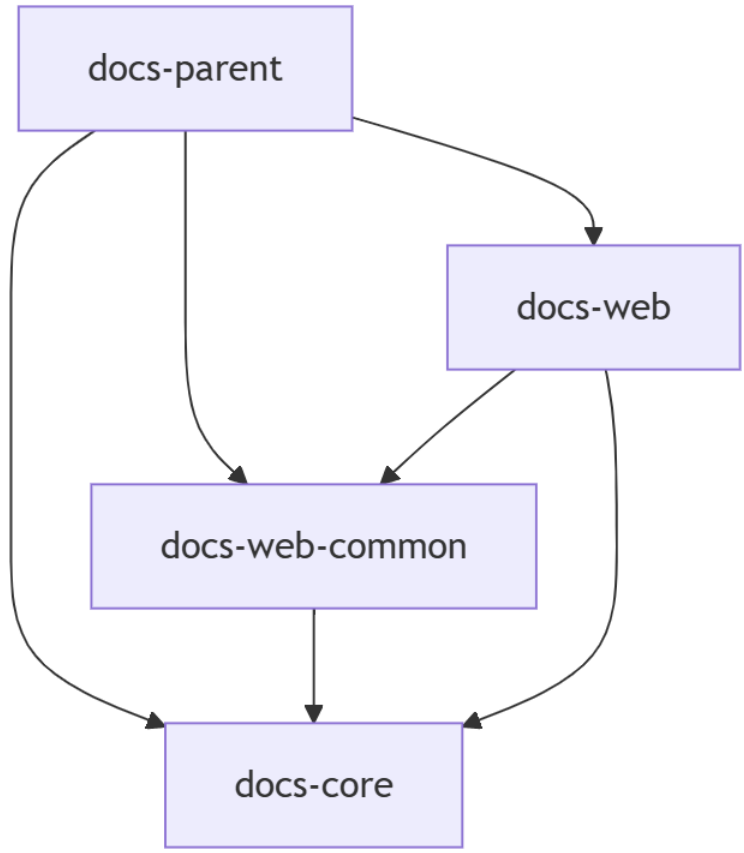
Part 4. Explore Teedy's dependencies

You may view project dependencies using maven command:

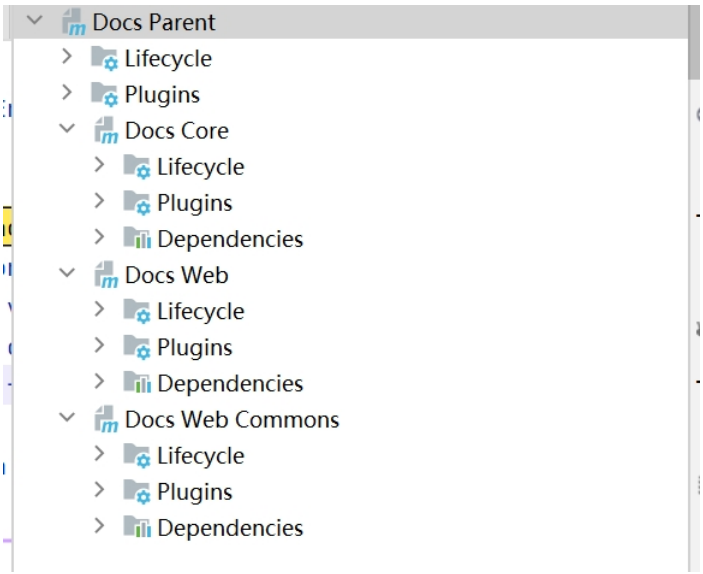
```
mvn dependency:tree
```

or in IDE, right click and select "Show Diagram".

Here is Teedy's module dependency:



Each module has its own `pom.xml`:



1. Root POM

The root `pom.xml` serves as the parent POM for the entire project

1. Module declaration

```
    <!--manage shared configurations and dependencies for child modules."-->
    <packaging>pom</packaging>

    <!--Module list-->
    <modules>
```

```
    <module>docs-core</module>
    <module>docs-web-common</module>
    <module>docs-web</module>
  </modules>
```

2. Dependency Management

```
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>com.sismics.docs</groupId>
        <artifactId>docs-core</artifactId>
        <version>${project.version}</version>
      </dependency>
    <dependencies>
    ......
  <dependencyManagement>
```

2. Docs-core

This is a core module of a document management system project. Here are the main dependency categories:

- Persistence Layer:
  - Hibernate Core - For ORM (Object-Relational Mapping)
  - PostgreSQL - Database driver for production
  - H2 Database - For testing purposes
- Utility Libraries:
  - Joda-Time - For date/time handling
  - Google Guava - General utility library
  - Apache Commons (Compress, Lang3, Email) - Various utility functions
  - FreeMarker - Template engine
  - BCrypt - For password hashing
- Search & Text Processing:
  - Apache Lucene (multiple modules) - For search functionality:
- Document Processing:
  - PDFBox - PDF manipulation
  - etc.
- Network/Communication:
  - OkHttp - HTTP client
  - Apache Directory API - For LDAP integration

    - Commons Email - For email functionality

- Logging:
    - Log4j
    - SLF4J (multiple modules)

- Testing:
    - JUnit - For unit testing

```
    <dependency>

      <groupId>com.h2database</groupId>

      <artifactId>h2</artifactId>

      <scope>test</scope>

    </dependency>
```

Teedy defines 2 profiles: dev and prod, which correspond to different configurations.

```
  <profiles>
    <!-- Development profile (active by default) -->
    <profile>
      <id>dev</id>
      <activation>
        <activeByDefault>true</activeByDefault>
        <property>
          <name>env</name>
          <value>dev</value>
        </property>
      </activation>

....

    <!-- Production profile -->
    <profile>
      <id>prod</id>
    </profile>
  </profiles>
```

They could be used as:

```
mvn clean install -P prod
mvn clean install -Denv=dev
```

3. Docs-web-common

This is a common web module that provides shared web functionality, REST API capabilities, and utilities that can be used across other web modules in the project. It sits between the core module and web-specific implementations.

4. Docs-web

The docs-web module is packaged as a WAR file and serves as the main web application interface for Teedy. The module integrates with docs-core for business logic and docs-web-common for shared functionality.