Apache Maven: A Comprehensive Guide

1. Setting Up Maven

Prerequisites:

- **Java Development Kit (JDK):** Install Java (JDK 8 or later).
- **Maven Installation:**
  - Download Maven from Apache Maven Website.
  - Extract and set the `MAVEN_HOME` environment variable.
  - Add the Maven `bin` directory to the `PATH`.
  - Verify installation:
  - `mvn -version`

2. Project Structure

A typical Maven project follows a standard directory structure:

```
/my-maven-project
  ├── src
  │    ├── main
  │    │    ├── java         # Application source code
  │    │    ├── resources    # Configuration files
  │    ├── test
  │    │    ├── java         # Test cases
  │    │    ├── resources    # Test configuration
  ├── target                 # Compiled bytecode and packaged files
  ├── pom.xml                # Maven configuration file
  ├── .mvn                   # Internal Maven configuration
```

3. POM File (Project Object Model)

The `pom.xml` file is the heart of a Maven project. It contains project configuration, dependencies, and build instructions.

Basic Example of `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>com.example</groupId>
    <artifactId>my-maven-app</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
```

```
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Key Elements in POM:

- `<groupId>`: Unique identifier for a project (e.g., company name).
- `<artifactId>`: Name of the project artifact (e.g., JAR file).
- `<version>`: Version of the application.
- `<dependencies>`: List of external libraries used in the project.

---

4. Building, Testing, and Packaging a Project

Maven provides **lifecycle commands** to build and test the project.

Basic Commands:

```
mvn compile      # Compiles the source code
mvn test         # Runs unit tests
mvn package      # Packages the project (JAR/WAR)
mvn clean        # Cleans the project (deletes `target` folder)
mvn install      # Installs package into local repository
mvn deploy       # Deploys the package to a repository
```

---

5. Dependency Management and Repository

Maven automatically **downloads and manages dependencies** from repositories.

Maven Repositories:

1. **Local Repository:** Stored in `~/.m2/repository` (caches dependencies).
2. **Central Repository:** Hosted by Maven (https://repo.maven.apache.org).
3. **Remote Repositories:** Third-party repositories (https://mvnrepository.com/).

Adding Dependencies in `pom.xml`:

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>5.3.10</version>
    </dependency>
</dependencies>
```

Dependency Scopes:

Scope	Description
`compile`	Default scope, available during all build phases.
`provided`	Required for compilation but provided at runtime (e.g., Servlet API).
`runtime`	Not needed for compilation but required during execution.
`test`	Available only during testing.
`system`	Uses local dependencies outside the Maven repository.

---

6. Maven Lifecycles and Phases

Maven follows a predefined **build lifecycle**, divided into **phases**.

Three Main Lifecycles:

1. **Default Lifecycle** – Builds and deploys projects.
2. **Clean Lifecycle** – Cleans project files.
3. **Site Lifecycle** – Generates documentation.

Key Phases in Default Lifecycle:

Phase	Description
`validate`	Validates `pom.xml`.
`compile`	Compiles Java source code.
`test`	Runs unit tests.
`package`	Packages compiled code into JAR/WAR.
`verify`	Verifies integration tests.
`install`	Installs package to local repository.
`deploy`	Deploys package to a remote repository.

Example:

```
mvn clean package
```

This runs the **clean** phase, then **compiles, tests, and packages** the project.

---

7. Maven Archetypes

Maven **archetypes** are templates used to quickly generate a new project.

Create a New Maven Project Using Archetype:

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

Common Archetypes:

Archetype	Description
`maven-archetype-quickstart`	Creates a simple Java project.
`maven-archetype-webapp`	Generates a web application (WAR).
`maven-archetype-j2ee-simple`	Creates a J2EE project.
`maven-archetype-plugin`	Generates a custom Maven plugin.

Conclusion

Maven automates project builds and manages dependencies efficiently.
Standard directory structure ensures consistency across projects.
POM file (`pom.xml`) defines project dependencies and build configurations. Maven lifecycles and phases allow structured execution of tasks.
Archetypes help in quickly setting up different types of projects.
