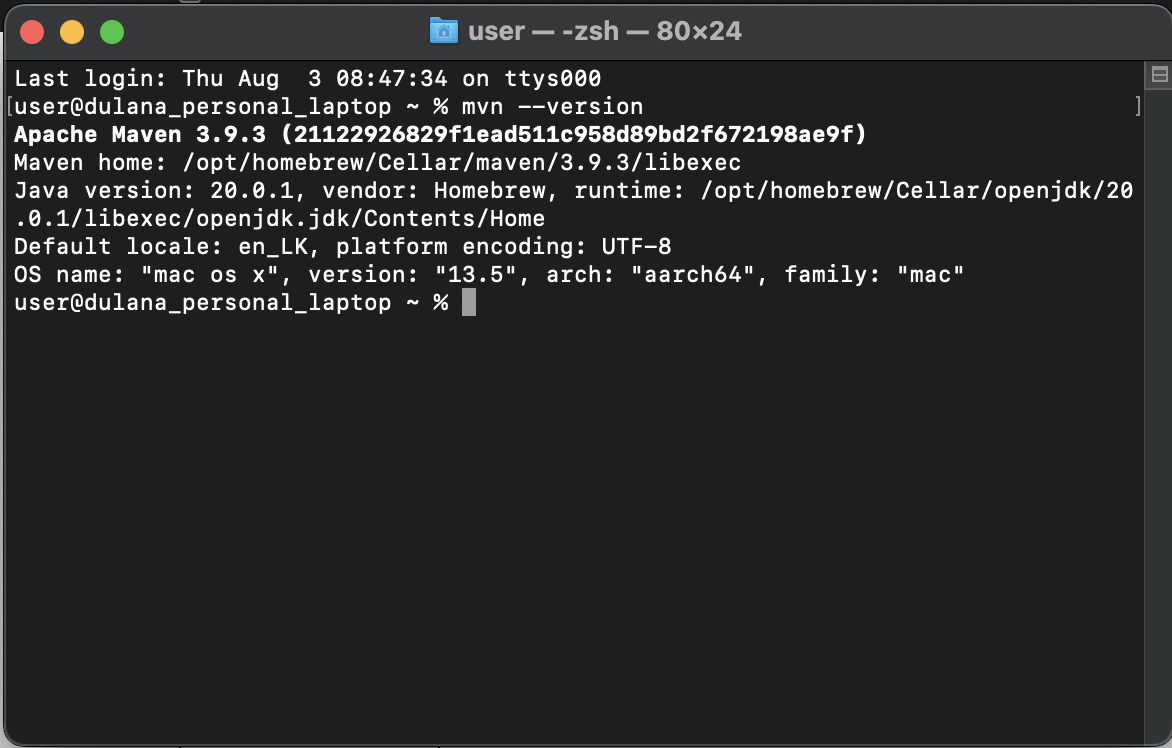
**3. Maven**

1. Why do we need a build tool like maven?

Maven is chiefly used for Java-based projects, helping to download dependencies, which refers to the libraries or JAR files. The tool helps get the right JAR files for each project as there may be different versions of separate packages.

1. Install maven
2. Display output of maven version



1. What is the pom.xml file?

The pom.xml file is a fundamental part of Apache Maven, which is a popular build automation and project management tool for Java and other programming languages. The term "POM" stands for "Project Object Model." The pom.xml file is an XML (Extensible Markup Language) file that serves as the configuration and information hub for a Maven project.

In the context of Maven, a project refers to a software project with its source code, dependencies, and build settings. The pom.xml file contains various elements and settings that define how the project should be built, packaged, and managed. Some of the essential elements found in the pom.xml file include:

* Project Information
* Project Dependencies
* Build Configuration
* Project Modules
* Plugins
* Repositories

When you run a Maven build for a project, Maven reads the pom.xml file, interprets its configuration, resolves dependencies, and executes the build process accordingly.

Overall, the pom.xml file plays a crucial role in managing Maven projects, and it helps maintain consistency and standardization across different projects by providing a centralized and declarative way to define project settings and dependencies.

1. Explain these tags found in pom.xml files?

|  |  |
| --- | --- |
| groupId | This tag represents the group or organization that owns the project. It's used to uniquely identify the project in a global context. Conventionally, it is based on a reversed domain name (e.g., com.example) to ensure uniqueness across projects. The groupId is a crucial part of the artifact coordinates, along with artifactId and version, that Maven uses to identify and resolve dependencies. |
| artifactId | This tag defines the unique identifier for the project's main artifact (usually a JAR or WAR file). The artifactId combined with the groupId and version forms a unique identifier for the project's artifact, which is used when deploying or resolving dependencies. |
| version | This tag specifies the version number of the project. Versions allow you to differentiate between different releases or versions of your project's artifact. Maven uses this information to manage dependencies and determine which version of a dependency to use. |
| packaging | This tag indicates the type of artifact that the project produces. The default packaging is usually "jar" for Java projects, but it can also be set to "war" for web application projects, "pom" for a parent POM, and so on. |
| dependencies | This tag contains a list of project dependencies. It is used to declare external libraries and other projects that your project relies on to build and run correctly. Dependencies specified within this tag will be automatically downloaded and included in your project during the build process. |
| dependency | This tag is used within the dependencies tag to define a single project dependency. It includes the groupId, artifactId, and version elements to specify the exact library or project required. |
| properties | This tag allows you to define project-level properties that can be used to configure various aspects of the build process, plugins, or other parts of the pom.xml file. Using properties enhances maintainability and allows you to centralize configuration values. |

1. Create a method which accept an integer as parameter and returns the square of it
2. Add junit (v5) dependency
3. Add a unit test to test the method
4. Run unit tests with maven. What is the command you used?
5. Create a Student class with following attributes and add getters/setters

* id: int
* name: String
* age: int
* subjects: List<String>

1. In your main method create and student object and set these values

id: 1

name: “john”

age: 20

subjects: [“Maths”, “English”, “History”]

1. Then print these student values using getters (e.g: s.getName())
2. Add lombok dependency and remove getter/setter methods from Student class
3. Explain the usage of these commands

|  |  |
| --- | --- |
| mvn clean | This command is used to clean the project's build directory. It removes all the files and directories generated by previous builds. This is helpful to start a fresh build from scratch and ensure that no artifacts from previous builds interfere with the current build process. |
| mvn install | The install command is used to compile the project, package it into a JAR (or other packaging formats), and then install it in the local Maven repository. The local repository is a local cache on your system where Maven stores the project's artifacts (e.g., JARs, POMs) for reuse by other projects. Other projects on your machine can then refer to these artifacts as dependencies in their pom.xml files. |
| mvn package | The package command is similar to mvn install, but it does not install the project's artifacts in the local Maven repository. Instead, it creates the packaged output (e.g., JAR, WAR, or other packaging formats) in the target directory of your project. This command compiles the code, runs tests (if any), and packages the compiled code along with resources into an artifact. |

1. Explain 3 types of maven repositories

* **Local Repository:**

The local repository is a local cache on your development machine where Maven stores all the project dependencies and artifacts downloaded from remote repositories. When you build a project for the first time or encounter a new dependency, Maven downloads the necessary artifacts from remote repositories and stores them in the local repository. Subsequent builds can then reuse these artifacts, reducing the need to download them again from the remote repositories.

* **Remote Repository:**

Remote repositories are central repositories that are hosted remotely, usually on the internet. These repositories contain a vast collection of open-source and third-party libraries, plugins, and other dependencies that can be downloaded and used in Maven projects. Examples of popular remote repositories include Maven Central Repository, JCenter, and others.

When Maven encounters a new dependency that it needs to download, it searches the local repository first. If the dependency is not found locally, it then looks for the artifact in the configured remote repositories. If the artifact is found, it is downloaded and cached in the local repository for future use.

* **Remote Repository Proxy (Mirror Repository):**

A remote repository proxy, also known as a mirror repository, acts as an intermediary between your local Maven build and the original remote repository. The purpose of a mirror repository is to improve build performance and provide control over the dependencies used in your project.

Instead of directly accessing the remote repository, Maven first checks the mirror repository. If the artifact is present in the mirror, it is retrieved from there. If not, the mirror repository will retrieve it from the original remote repository and cache it. This helps in reducing the latency and network load, especially for geographically distributed teams or in environments with slow internet connections.

1. Add your codes and answer sheet to a directory named “maven-basic-training” and push it to your training github repository.