**All build systems are essentially the same**

1. Compile source code

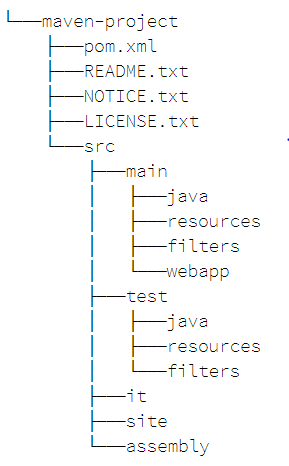
2. Copy resource

3. Compile and run tests

4. Package project

5. Deploy project

6. Cleanup



🡨 **This is the folder structure for the maven project (JAR)**

**Project name** (**G:A:V**)

**G:** Group ID (no spaces)

Business Unit / Company Name

**A:** Artifact ID (no spaces)

Name of Project

**V:** Version

Format: {major}.{minor}.{maintenance}

**MAVEN Download and Installation**

1. go to the maven download website <https://maven.apache.org/download.cgi>

2. copy link address of Binary tar.gz archive (for Linux).

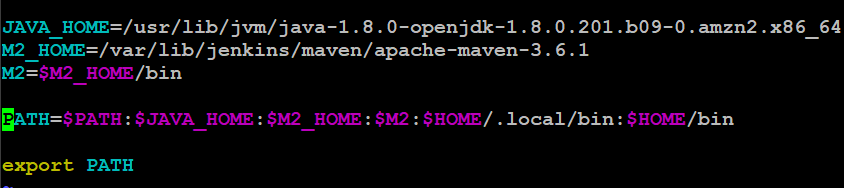
3. create a directory in **jenkins folder** (cd /var/lib/jenkins) where we install maven.

4. in that created directory **# wget link address.**

5. **# tar -xvzf downloaded file** (to extract maven folder).

6. **#cd extracted folder** (we will find **bin** folder in this extracted maven folder).

7. **#pwd** you will get the path, now update that path in bash profile.



8. now in jenkins install the plugin **Maven Invoker, Unleash Maven.**

9. in the global tool configuration under maven section, put the path and save (no error).

10. **# mvn --version :** it should show the version of maven and java.



It is going to show you maven version as well as java version.

If there is no error, then maven is installed successfully.

**Maven build lifecycle**

**Goal** is the single unit of task which does some real work. For example, the compile goal (runs as mvn compiler:compile) which compiles the Java source. All goals are provided by plugins, either by default plugins or by user defined plugins (configured in pom file).

**Phase** is a group of ordered goals or in other words: zero or more plugin goals are bound to a phase (either by default or by user). For example, the compile phase (runs as "mvn compile") consists only compiler:compile goal (plugin). Running a phase basically runs the plugins bound with it.

* **Default life cycle : main lifecycle responsible for project deployment**
* Generate -sources / Generate -resources
* Compile
* Test
* Package
* Install
* Deploy
* **Clean life cycle : clean project and remove all files generated by previous build**
* **Site life cycle : to create project site documentation**

**# mvn archetype:generate** (it will create directories with sample standard directory format)

**# mvn clean package** (it will first perform clean operation and then invoke package operation)

**# mvn archetype:generate -DgroupID=demo -DartifactID=samplewar -DarchetypeArtifactID =maven-archetype-webapp -DinteractiveMode=false**

**Default life cycle:**

**Generate sources/Generate resources \*1 -> Compile \*2-> Test \*3-> Package \*4**

**# mvn compile , # mvn test , #mvn package** (when we do \*4, \*3\*2\*1 are done automatically)

**# mvn package** (it will default generate the .jar file)

**Clean life cycle: # mvn clean**

Its going to delete entire thing which is generated during runtime. (target folder)

\*\*To perform any operation, you should be in the home directory where **pom.xml** file is present

\*\*Interactive mode : its going to ask you for group ID, artifact ID, version

\*\***JAR/WAR** generated after #mvn package is called **Build Artifact** or **Deliverable**

**POM:** Project Object Model (Configuration file)

POM is an XML file that contains information about project and configuration details used by maven to build the project.

* Describes a project
* Name and version, artifact type, source code locations, dependencies
* Plugins
* Profiles
* Uses XML by default
* Not the way Ant uses the XML

**MAVEN plugin management:**

* **Build type plugin:** they execute during the build and configured in </build> pom.xml
* **Reporting type plugin:** they execute during site generation, configured in </reporting> pom.xml

Maven available plugin -> select plugin -> usage (tells how exactly we have to use it).