Maven Basics

Pom can inherit from parent pom….

When maven project calls for dependency, maven looks in
Local repo first.
Local repo -> <user home>/.m2/
.m2 -> hidden folder.
Then it's gonna search for central or other depending on configuration.

SNAPSHOT-> development version, maven will look for newer versions (stable) to modify it. Otherwise it will stay forever.
Even if newer version are found in cached local repo then it will copy it.

To check for .m2 file go to -> ~/.m2
You can check for repository folder where you can find your path to your artifact. We can check their versions installed.
Inside that jar and pom will be stored. Without pom maven does not know the dependencies for that artefact.

To prevent malicious content being pulled from central repo, maven keeps a hashed file for checking.

Https://repo1.maven.org - > directory structure maven will look to resolve artifacts.
There is more friendly web site -> https://mvnrepository.com/

On this site, maven will use coordinates to locate the artifact.

——————————————————————

Maven Wagon -> unified api used to publish maven artefact on central maven repo or get them also. It helps in maintaining communication with diff providers while talking to maven repositories. Also helps to setup proxy for the corporate.

———————————————————————
xsd -> xml schema document

Spring boot also inherits pom of spring.

To look out at effective pom ->
mvn help:effective-pom effective-pom
To see in intellij go to pom and right click to find options there find maven and then effective pom.

——————————————————————

Class paths for maven -> compiled, runtime and test.
Runtime scope -> needed for oracle, etc. database to connect.
Import -> standardise versions across projects.

Mvn dependency:tree -> to check for dep. tree plugin.
You can see maven coordinates and scope there for artefact.

Site -> assembly descriptors site, these are values for site.
Maven can create a website for us.

————————————————

Maven life cycles:
Default life cycle has no binding plugin -> it is defined by the type of jar, war, etc.

Maven offers many default packaging types that include a jar, war, ear, pom, rar, ejb, and maven-plugin. Each packaging type follows a build lifecycle that consists of phases. Usually, every phase is a sequence of goals and performs a specific task.

————————————————

Maven wrapper -> script wrapper around Maven that can be distributed with your project. Helps to distribute maven with project. It makes our build portable so that we can run from shell script. Helps to work with maven without having maven on specific machine.

mvn -N io.takari:maven:wrapper

Command to run wrapper.

./mvnw —version -> to know wrapper version

There is also a hidden file .mvn which contains maven-wrapper.properties. It will show the version is is requiring.

mvn -N io.takari:maven:wrapper -Dmaven=3.6.0
Helps in upgrading version of wrapper so that we have consistent version of maven and maven wrapper.

The Maven Wrapper is an excellent choice for projects that need a specific version of Maven (or for users that don't want to install Maven at all). Instead of installing many versions of it in the operating system, we can just use the project-specific wrapper script.

The option *-N* means *–non-recursive* so that the wrapper will only be applied to the main project of the current directory, not in any submodules.
After executing the goal, we'll have more files and directories in the project:

- *mvnw*: it's an executable Unix shell script used in place of a fully installed Maven
- *mvnw.cmd*: it's the Batch version of the above script
- *mvn*: the hidden folder that holds the Maven Wrapper Java library and its properties file

After that, we can run our goals like this for the Unix system:
./mvnw clean install

It is used when when we have no control over continuous integration server in our corporate and download specific maven version.