Maven

Maven Index:
==========

Build Tools/Build Automation/Management/Process

==========

--> Build Management: is a process that we compile and assemble all the source code(written by developers) into object files.

ex: 100 app.java files
100 object files(.class files)

>>Grunt
>>Gulp
>>Ant--> Java(Apache Foundation)
>>Gradle--> Alternative for Maven
>>Maven--> latest and updated one

Ant vs Maven:

>> actions are defined in ant(so much of scripting) >> in maven say what to do not how to do
>> sequences are defined in ant >> how to build is defined in maven (life cycle)
>> no default directory layout >> it fallows standard directory structure
>> ant fallows you >> you need to fallow maven
>> librarys are part of source code >> librarys are not part of source code
(difficult to maintain)

--> diff with other tools
>> open source
>> it is not only build tool and also project management tool
>> it has set of standards and object modules,so no need to instruct
>> default project lifecycle
>> dependency management

>> Compiling Source Code
>> Packing Biniries/artifacts

```
>> running Automated tests
>> Deploying to production system
>> Creating Documentation

Variables:
---------
Environment Variables
      # user
      # system ($PATH)
echo $HOME
echo $SHELL
env
VARIABLENAME=vinodh
unset VARIABLENAME

-->if we want to use variables globals(in all shel windows)then
export name=vinodh (in bashrc file)

Maven Installation in Windows:
------------------------------
--> install java
--> Download java JDK & JRE (or)
http://www.oracle.com/technetwork/java/javase/downloads/index.html
--> Go to-->mycomputer-->properties-->Advanced system settings--
>environment variables-->system variables
--> path    ;C:\Program Files\Java\jdk1.8.0_131\bin;C:\Program
Files\Java\jre1.8.0_131\bin  ---> to system variables PATH by seperater ;
            JAVA_HOME should point to JDK(without bin)

--> install Maven
Go to this website to downloab(Zip)--> maven.apache.org/download.cgi
      D:\Apache_Maven ---> MAVEN_HOME in system variables
      path--> ;D:\Apache_Maven\bin

Maven Installation in Linux::
=============================
  Java Environmental Variables
  ----------------------------
      vi /etc/profile.d/java.sh
      export JAVA_HOME=/opt/java/jdk1.8.0_211
      export PATH=${JAVA_HOME}/bin:${PATH}
      source /etc/profile.d/java.sh (to execute script file)

  maven
  -----
      vi /etc/profile.d/maven.sh
      export MAVEN_HOME=/opt/maven/apache-maven-3.6.1
      export PATH=${MAVEN_HOME}/bin:${PATH}
      source /etc/profile.d/maven.sh
        mvn --version
           o/p:- Apache Maven 3.5.4
(1edded0938998edf8bf061f1ceb3cfdeccf443fe; 2018-06-17T19:33:14+01:00)
            Maven home: /usr/local/src/apache-maven
```

```
           Java version: 9.0.4, vendor: Oracle Corporation, runtime:
/opt/java/jdk-9.0.4
           Default locale: en_US, platform encoding: UTF-8
           OS name: "linux", version: "4.17.6-1.el7.elrepo.x86_64",
arch: "amd64", family: "unix"

verify whether java/maven is installed or not in CMD prompt by typing
below commands
Javac --> compiler
java -->keyword
java -version --> runtime environment
mvn --version

How Maven works:(Architecture)
==============================
                            Build System
                          --------------------------
                            |                   |
                          |        POM.XML    (conf file)  |      HTTP

       local repository <<------|   goals               |---------------
>> Remote Repository
                          |                   |
      (maintained by Maven opensource Community)
                          |        Maven      |
                          ----------------------------
--> it works as a GOALS, internally goals as plugins/jar files which has
the future of when and what it has to do
      eg:- maven do testing; --->>  then it call plugin to do testing

--> remote maven repository located in - http://repo1.maven.org/maven2
--> local repo located in c:/user/vinodh --> .M2 --> Repository


Default lifecycle:  mvn compile package
==================
      1. generate-source (.java files)
      2. compile -->all .java files into .class files(object files)
      3. test  ---> Unit test (a peace of code)
      4. package --> deliveriable or executable or Artifacts(which
contains all)
      5. integration-test(pre and post)
      6. install        mvn compile package
      --> clean :- it deletes all runtime files
      --> site : - documentation(99% we will not use, very rare cases
like audits...)

Example Maven Goles:
--------------------
To invoke a Maven build you set a lifecycle “goal”
mvn install
note:- mvn -f pom.xml <goal>
Invokes generate and compile, test, package, integration-test, install
```

mvn clean
Invokes just clean

mvn clean compile
Clean old builds and execute generate, compile

mvn compile install
Invokes generate, compile, test, integration-test, package, install

mvn test clean
Invokes generate, compile, test then clean

Note:
diff source and binary code
1. source code which we can customize
2. binary code is a product which we can buy/use directly

Standard Directory Layout:

>> if you want to work with maven project, then we need to follow the maven standard directory structure through which maven will work.
main-->actual source code, lib files,additinal info, property files....etc
test --> unit testing files

once you start compile, maven will go to src/main folder to compile (what are the files you gave over there)

GAV:
====
--> how maven identify which plugin or project to select when we instruct a goal. (G.A.V)
G(groupid) -- string rep company name / group name / business org on which u doing project.
A(artifactid) -- string rep product or deliverable(final output of your product)
V(versionid) -- Major.Minar.Patch/Maintanance(add SNAPSHOT to identify in development)
packaging -- build type identified using the packaging element
eg : - pom ,jar(default),war,ear
note: - by keeping pom in packaging it acts as a parent pom of all modules
mvn archetype:generate

--> jar - java archive(default package maven uses which contains group of .class files, so we group this to get a particular behaviour)
--> war - web archive - contain group of jar + config + xml (for web based projects)
--> ear - enterprice application

Note:- How maven knows,where java files,what it has to do,where to keep files and fetch files....etc this all done by below two files
to run maven default life cycle
1)dir structure

2) pom.xml file in dir

POM:(conf file)

Project object model is fundamental unit of work in maven,POM is an xml file that contains information about project and configuration details used by maven to build project. pom conf file contains below list.
--> atlest one pom.xml file should be there in product/project
@ Describe a project
@ name and Version, Artifact type,source code location, Dependencies
@ Plugins
@ Profiles(Alternate build configuration)
@ it uses XML by default

Plugins:-
---
if we want instruct anything to maven through goals we will do, goals internally have plugins/jar files.
1. Build Plugins : we will use this for entire life cycle
2. Reporting Plugins : create documentation of product (for site phase only)

```
<build>
     <plugins>
              <plugin>
        1. GAV - how maven identifies plugins
        2. when you have to run the plugin
        3. how to use plugin(like conn DB, insall, disconnect...etc)
        4. what exactly to do
              </plugin>
        -- plugin 2 infomation
    </plugins>
</build>
```

```
<project>
  <build>
        <plugins>
          <plugin>
             <groupId>org.apache.maven.plugins</groupId>
             <artifactId>maven-antrun-plugin</artifactId>
             <version>1.1</version>
             <executions>
               <execution>
                   <id>id.clean</id>
                   <phase>clean</phase>
                   <goals>
                   <goal>run</goal>
                   </goals>
                   <configuration>
```

```
                    <tasks>
                       <echo>hallo world=============</echo>
                    </tasks>
                    </configuration>
                </execution>
             </executions>
          </plugin>
       </plugins>
    </build>

<plugin>
          <groupId>org.codehaus.mojo</groupId>
            <artifactId>exec-maven-plugin</artifactId>
            <version>1.6.0</version>
            <configuration>
               <executable>mvn</executable>
                    <arguments>--version</arguments>
            </configuration>
</plugin>

</plugins>
</build>
</project>
```

Note:- what plugin we selecting, what syntax(GAV) of plugin, And how to call....
--> maven ant plugin, maven exec plugin....

how to call a individual plugin:
---
mvn <GOAL>
mvn <PLUGIN>:<GOAL_NAME>
mvn exec:exec
mvn exec:java
--> mvn <plugin>:<goal> ---> we can call plugin directly without phase/goal

SNAPSHOT:
=========
1. it is under development build (or) dev copy which is not yet finalized(only we will change before releasing to client)
2. other projects are depends on this, if i rebuild the jar name other proj looking for this

Maven Profile:
==========
def:- buid profile is a set of configurationns values which can be used to set or override dafault values of maven build.
using a build profile, you can customize buid for different environments such as production v/s developmennt.

--> some times you want to execute only default plugins not all mentioned in build, at that time we can use.
mvn clean (default)

```
      mvn -Pdemo  specify goal(all plugins)

<profiles>
       <profile>
      <id>demo</id>
               <build>
                </build>
       </profile>
</profiles>

--> profile can activate many types like env, os, settings.xml in
repo...etc

<profile>
  <id>test</id>
    <activation>
       <property>
      <name>env</name>
       <value>test</value>
      </property>
    </activation>
</profile>
-----------------------------------------------
Multy-Module Projects:
======================

--> if you have 1000 files in app.java project it is diffcult to
maintain, so make modules/components like add, sub, dev of calculater
project and copy src,pom file in each.
 note: - by keeping pom in packaging it acts as a parent pom of all
modules (parent and child relationship)     demo (parent) > add, sub
(child)
<modules>
      <module>add</module>
      <module>sub</module>
</modules>

Maven has 1st class multi-module support
Each maven project creates 1 primary artifact
A parent pom is used to group modules

issues -1:
 >> executing all modules every time

overcome:
parent and child relationship, by keeping 'pom' file in "packaging"
Ex:-

  <groupId>EBU</groupId>
  <artifactId>Parent-module</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
    <modules>
        <module>Child-jar</module>
```

```
        <module>child-war</module>
    </modules>

issue-2:
>> Dependencies
-->adding add.jar to subtract for dependency..

<dependencies>
    <dependency>
      <groupId>training</groupId>
      <artifactId>subtract</artifactId>
      <version>1.0 SNAPSHORT</version>
    </dependency>
  </dependencies>

note:-
<dependencies>                    |
    <dependency>             |
      <groupId>junit</groupId>          |
      <artifactId>junit</artifactId>    |------------>> junit plugin is
default plugin for performing test phase
      <version>3.8.1</version>          |
      <scope>test</scope>         |
    </dependency>            |
  </dependencies>            |

by using "install" phase in add module, then add.jar will move to local
repo

mvn install--> copying jar file form local project folder to local
repository

giving parent gav in child ==>>complete parent and child rel

Dependencys how maven know:
------------------
--> if sub is depend on add file then we need to keep add file GAV into
sub file dependency.
--> error :- not able to find add file, then install add file from local
project folder to local repository
              mvn install
```