Common Maven Plugins

Maven is a framework to work with plugins.

Overview Maven Lifecycle Plugins:

Maven clean plugin->

Actually problems happen if we do not clean and re package then all old files remains intact which can result in unpredictable results.

Plugins defined in your build section plugins tag will be executed during the build of your project.

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-clean-plugin</artifactId>
      <version>3.2.0</version>
      <executions>
        <execution>
          <id>auto-clean</id>
          <phase>initialize</phase>
          <goals>
            <goal>
              clean
            </goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Adding this if we click on maven package we first delete the target folder and then again compile it.
So actually we are performing mvn clean package.
We are hooking into the default phase of initialising the build. We are telling every time we are building delete everything and start from fresh.

Maven compiler plugin-> uses Javax.tools out of the jvm as compiler. This is a default class compiler available in the jvm.

If we click on plugins section of sidebar and hit on any of the goal then that specific plugin goal will run but if we click on one of the phase in lifecycle then all the stages up to that will be executed.

Maven Resources Plugin -> resources:copy-resources copy resources from one file to another.
It actually helps in copying resources from main and test into target folder. For main resources they get copy into classes folder of target folder.
We can also reset resources folder location.

Maven Surefire plugin -> by default it looks for files ending with Test, beginning with test or their plurals, or end with test case, in test directory.
It also does pojo test.
It runs every flavour of Junit, Cucumber, TestNG,Spock,etc.
Every Junit test have void as return type. Every method should start with test…
We can also see surefire report in target folder.

Maven jar plugin ->

Maven deploy plugin -> where the jar file is put on central repo is decided by pom file.

We can add plugin in pom.xml to make the jar file as executable one. It has addClassPath property which makes that available on classpath on running.
Java -jar <jar file name> -> be on the parent directory for running.
While running the executable file it wants all the dependencies to be in same class path. We can do by copying the jar of it in same target folder. This makes it available during runtime.

Sometime if we delete target folder then if we are on same directory on terminal it will not work. We have to again go to the same directory by going back.

Maven deploy plugin-> helps to publish our artifact to remote repo. It is given by <distributionManagement> tag in pom.xml. This works in conjunction with settings.xml in .m2 directory. We have to specify username and passport for our repo.

Maven site plugin -> generate the website for our project.
It can generate xml, apt (almost plain text), fml, XDoc.
On running site cmd we get its output as site in target folder.
You can check your website by pasting that index.html in site folder in chrome.
There will also show info from pom.xml about dependency tree and all.
If site phase does not run try to explicitly write the plugin in the build.

Clicking on package in lifecycles run all default phases up to package.

Not include .idea, .iml or target folder in GitHub.

Maven clean:clean -> run the clean plugin and clean goal of plugin.

Maven install -> installs the artifact into .m2 repository.