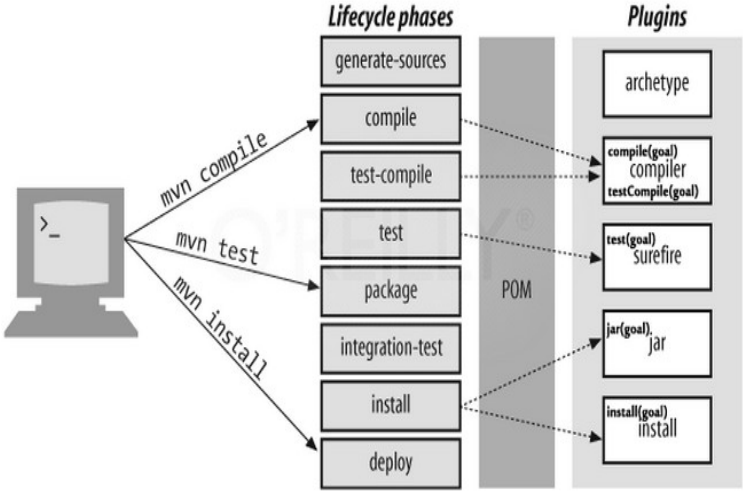
Maven Life Cycle

"Maven is based around the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined."

" For the person building a project, this means that it is only necessary to learn a small set of commands to build any Maven project, and the POM will ensure they get the results they desired."



Apache Maven Project - Introduction

There are three built-in build lifecycles:

default lifecycle handles project deployment.
clean lifecycle handles project cleaning.
site lifecycle handles the creation of project's site documentation.

Default lifecycle
The default lifecycle has the following build phases:

validate: validate the project is correct and all necessary information is available.

compile: compile the source code of the project.

test: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed.

package: take the compiled code and package it in its distributable format, such as a JAR.

integration-test: process and deploy the package if necessary into an environment where integration tests can be run.

verify: run any checks to verify the package is valid and meets quality criteria.

install: install the package into the local repository, for use as a dependency in other projects locally.

deploy: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.
These lifecycle phases (plus the other lifecycle phases not shown here) are executed sequentially to complete the default lifecycle.

To do all those, we only need to call the last build phase to be executed, in this case, deploy:

$ mvn deploy

Clean Lifecyle
pre-clean
clean
post-clean

Default Lifecyle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	**package**
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	**install**
generate-test-resources	**deploy**
processs-test-resources	

Site Lifecyle
pre-site
site
post-site
site-deploy

That is because if we call a build phase, it will execute not only that build phase, but also every build phase prior to the called build phase.

mvn clean install
The following command tells Maven to do the clean action in each module before running the install action for each module.

$ mvn clean install
In other words, mvn clean install clears any compiled files we have, making sure that we're really compiling each module from scratch.

Note that clean is in a separate lifecycle, so it's not called by default.

A target folder holds Maven-generated temporary files and artifacts. There are times when the target folder becomes huge or when certain files that have been cached need to be cleaned out of the folder. The clean goal accomplishes exactly that, as it attempts to delete the target folder and all its contents.

Maven goals vs phases
"Build processes generating artifacts typically require several steps and tasks to be completed successfully. Examples of such tasks include compiling source code, running a unit test, and packaging of artifacts. Maven uses the concept of goals to represent such granular tasks." - ref #2

Goals in Maven are packaged in plug-ins, which are essentially a collection of one or more goals. For example, the compiler is the plug-in that provides the goal compile.

Some phases have goals bound to them by default. And for the default lifecycle, these bindings depend on the packaging value.

Goals are executed in phases which help determine the order goals get executed in.

This simplifies project dependency management greatly.

The default Maven lifecycle bindings show which goals get run in which phases by default. The compile phase goals will always be executed before the test phase goals which will always be executed before the package phase goals and so on.

A goal not bound to any build phase could be executed outside of the build lifecycle by direct invocation.

The order of execution depends on the order in which the goal(s) and the build phase(s) are invoked.

For example, consider the command below. The clean and package arguments are build phases, while the dependency:copy-dependencies is a goal (of a plugin).

$ mvn clean dependency:copy-dependencies package
If this were to be executed, the clean phase will be executed first (meaning it will run all preceeding phases of the clean lifecycle, plus the clean phase itself), and then the dependency:copy-dependencies goal, before finally executing the package phase (and all its preceeding build phases of the default lifecycle).