# Goals and Phases

## Contents

## Overview

Maven uses goals and phases to outline the life cycle that a build will follow to deploy and distribute a project.
There are three built in life cycles:

- **Default** is the main life cycle and is responsible for the deployment of a project.
- **Clean** is used to remove all of the files generated from the last build.
- **Site** is used to create the project's site documentation.

Each of these life cycles consist of a sequence of phases, the default life cycle has 23, clean has 3, and site has 4.

## Tutorial

### Maven Phase

A Maven phase is a stage in the Maven build lifecycle, each having a specific task to complete.
Each phase is executed in a specific order, so if we run a specific phase it will execute everything up to and including that phase.
For example if we run the `deploy` phase in the default life cycle, it will run the entire default life cycle since `deploy` is the last phase.
We can specify a phase to build our project using the following command.

```
mvn deploy
```

Switching `deploy` out for whichever phase we want to run.
Below you can find a list of every phase for each of the life cycles and the task that they perform, as well as which order they will be run in if you run the entirety of a life cycle.

#### Default Life Cycle

1. `validate` : validates that the project is correct and all necessary information is available.
2. `initialize` : initialises the build state, for example setting properties or creating directories.
3. `generate-sources` : generates any source code for inclusion in the compilation.
4. `process-sources` : processes the source code, checking to see if any filtering needs to be applied.
5. `generate-resources` : generates the resources needed for inclusion in the package.

6. `process-resources` : copies and processes the resources into the destination directory so that it is ready to be packaged.
7. `compile` : compiles the source code of the project.
8. `process-classes` : post-processes the generated files from compilation by performing bytecode enhancement on Java classes.
9. `generate-test-sources` : generates any test source code needed for inclusion in the compilation.
10. `process-test-sources` : processes the test source code, checking to see if any filtering needs to be applied.
11. `generate-test-resources` : creates the resources for testing.
12. `process-test-resources` : copies and processes the resources into the test destination directory.
13. `test-compile` : compiles the test source code into the test destination directory.
14. `process-test-classes` : post-processes the generated files from test compilation by performing bytecode enhancement on Java classes.
15. `test` : runs the tests using a suitable testing framework, these tests should not require the code to be packaged or deployed.
16. `prepare-package` : performs any operations needed to prepare a package before the packaging.
17. `package` : takes the compiled code and packaged it in a distributable format, such as a JAR or WAR file.
18. `pre-integration-test` : performs any actions required before the integration tests are executed, for example setting up the required environment.
19. `integration-test` : processes and deploys the package (if needed) into the environment where the integration tests are run.
20. `post-integration-test` : performs actions required after integration tests have been run, for example cleaning up the environment.
21. `verify` : runs any checks to verify the package is valid and meets the quality criteria.
22. `install` : installs the package into the local repository, this allows it to be used as a dependency in other local projects.
23. `deploy` : this is done in an integration or release environment and copies the final package to the remote repository so that it can be used as a dependency on any project that has access to that repository.

### Clean Life Cycle

1. `pre-clean` : executes processes needed prior to the actual cleaning of the project.
2. `clean` : removes all files generated by the previous build of the project.
3. `post-clean` : executes processes needed to finalise the cleaning of the project.

### Site Life Cycle

1. `pre-site` : executes processes needed prior to the project site generation.
2. `site` : generate the project's site documentation.
3. `post-site` : executes processes needed to finalise the site generation and prepares for site deployment.
4. `site-deploy` : deploys the generated site documentation to the specified web server.

## Maven Goal

Maven phases are made up of plugin goals; even though the phases are responsible for a specific part of the life cylce, their implementation can vary, this can be defined with the plugin goals bound to those phases.
A plugin goal is responsible for a specific task within the phase and contributes to the building and managing of a project, each goal can be bound to zero or more phases.
If a goal is not bound to any of the phases then it can be executed outside of the life cycle by direct invocation.

If a goal is bound to multiple build phases, it will be called in all of those phases.
Each phase can have zero or more goals bound to it, if the phase has zero goals bound to it, then it will not execute; however if it has one or more goals bound to it then it will run all of those goals.

If we had a plugin goal named `dependency:copy-dependencies` then it can be invoked like this:

```
mvn dependency:copy-dependencies
```

## Example

The order in which phases and goals are executed depends upon the order in which they are invoked, for example if we want to invoke both the `clean` and `install` phases by running this:

```
mvn clean install
```

then Maven will first run all phases up to and including `clean` within the clean lifecycle, then it will run all phases up to and including `install` in the default life cycle.
Each phase that is ran will run each of its goals in order, including any user defined plugin goals.
If we wanted to also invoke a user defined plugin goal within the same command, we can add the goal invocation into the command at the point we want it to be executed.
So if we want to invoke the `dependency:copy-dependencies` goal in between `clean` and `install` then we will have a Maven command like the following.

```
mvn clean dependency:copy-dependencies install
```

## Exercises

There are no exercises for this module.