Introdução à Engenharia de Software

José Mendes 107188

2023/2024

1 Maven

1.1 O que é o Maven?

É uma **ferramenta de gestão de projetos**, que inclui:

- Um **project object model** (POM) que descreve o projeto;
- Um conjunto de **standards**;
- Um **lifecycle** do projeto;
- Um sistema de gestão de **dependências**;
- Lógica para **executar plugins** em **fases** específicas do ciclo de vida.

Convenção sobre configuração (layout do projeto é padronizado).

1.2 Layout de Diretórios Padronizado

POM - Contém uma descrição completa do projeto de como construir o projeto.

src - Diretório que contém todo o código fonte para construir o projeto, o seu site, ...

target - Diretório que contém os resultados da construção, tipicamente um JAR ou WAR, juntamente com os ficheiros intermedios.

1.3 POM

Maven é baseado no conceito de um **Project Object Model** (POM). Este é um ficheiro XML, que está sempre localizado no diretório base do projeto como **pom.xml** (os users definiram POMs que estendem o Super POM).

O POM contém informação sobre o projeto e vários detalhes de configuração usados pelo Maven para construir o projeto.

O POM é declarativo, não necessita de detalhes de procedimento.

1.3.1 Estrutura do POM

O POM contém 4 categorias de descrição e configuração:

- Informação geral do projeto, isto é, informação human-readable;
- Configuração do build, que pode incluir, adicionar plugins, afixar plugins objetivo ao ciclo de vida;
- Ambiente de construção, que descreve o ambiente "familiar" em que o Maven está;
- Relações POM, isto é, coordenadas, herança, agregação, dependências.

1.4 Coordenadas Maven

As coordenadas definem o lugar único do projeto no universo Maven. São compostas por 3 partes: <**groupId**>, <**artifactId**> e <**version**> (The Maven trinity!).

As versões de um projeto são usadas para agrupar e ordenar lançamentos:

$$< major\_version > . < minor\_version > . < incremental\_version > - < qualifier >$$

Exemplo: 1.0.0-SNAPSHOT ou 1.2.3-alpha-2

Se o qualifier contiver a palavra chave SNAPSHOT, então o Maven vai expandir este token para uma data e hora convertida para o formato UTC.

- **groupId** - Nome da empresa, organização, equipa, . . . , normalmente usando a convenção de nomes de domínio invertidos (reverse URL naming, ex: org.apache.maven);
- **artifactId** - Nome único do projeto dentro do groupId;
- **version** - Versão do projeto;
- **packaging** - Tipo de empacotamento do projeto (jar (default), war, . . .);
- **classifier** - Classificador opcional para distinguir artefactos

Nota: As coordenadas Maven identificam unicamente um projeto.

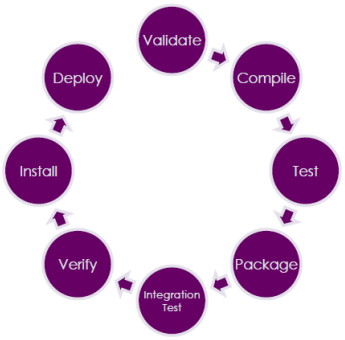
1.5 Ciclo de Vida Maven

Um ciclo de vida é uma sequência organizada de fases, que dão ordem a uma sequência de objetivos. Estes objetivos são empacotados em plugins que estão ligados as fases.

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	

Chamar uma fase específica num ciclo de construção, vai executar todas as fases anteriores a essa fase.



1. **Validate** - Valida que a estrutura do projeto está correta. (ex: verifica se todas as dependências foram transferidas e estão disponíveis no repositório local);
2. **Compile** - Compila o código fonte, converte os ficheiros **.java** em **.class**, e armazenando-os no diretório **target/classes**;
3. **Test** - Corre testes unitários para o projeto;
4. **Package** - Empacota o código compilado num formato distribuível como **JAR** ou **WAR**;
5. **Integration Test** - Corre testes de integração para o projeto;
6. **Verify** - Corre verificações para verificar que o projeto é válido e que cumpre os critérios de qualidade;
7. **Install** - Instala o código empacotado no repositório Maven local, para uso como dependência noutros projetos locais;
8. **Deploy** - Copia o pacote final de código para o repositório remoto para partilha com outros developers e projetos.

1.6 Ciclo de Vida de Construção

O processo para contruir e distribuir um projeto. Consiste em vários passos designados por **fases**.

Algumas fases default são:

- **validate**
- **compile**
- **test**
- **package**
- **deploy**

1.7 Goals e Plugins

Os Goals são operações fornecidas pelas ferramentas Maven.

Cada fase é uma sequência de Goals, em que cada Goal é responsável por uma tarefa específica. Quando corremos uma fase, todos os Goals ligados a essa fase são executados, na ordem em que estão definidos.

Algumas Maven Plugins:

- resources
- compiler
- surefire
- jar, war

1.8 Arquétipos (Archetypes)

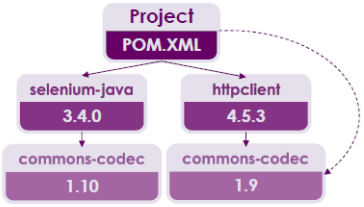
Um Archetype é um template para um projeto Maven, que pode ser usado para criar um novo projeto rapidamente.

Exemplo: *maven-archetype-quickstart* ou *maven-archetype-webapp*

Users podem criar os seus próprios Archetypes e publicá-los através de catálogos.

1.9 Gestor de Dependências

Uma **dependência** de um projeto é uma biblioteca da qual o projeto depende. Adicionar uma dependência ao projeto é simples, basta adicionar a dependência ao POM. O Maven vai automaticamente procurar a dependência no repositório local, e se não encontrar, vai procurar no repositório remoto e transferi-la.



2 Git e GitHub

2.1 Sistemas de Controlo de Versões

Um sistema de controlo de versões (também conhecido como sistema de controlo de código fonte) faz o seguinte:

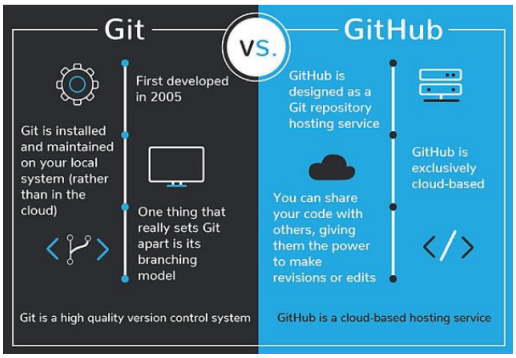
- Mantém várias (antigas e novas) versões de tudo (não só código fonte);
- Pede por comentários quando se fazem alterações;
- Permite "check-in" e "check-out" de ficheiros para saber em que ficheiros outras pessoas estão a trabalhar;
- Mostra as diferenças entre versões;

2.1.1 Vantagens

Ao trabalhar sozinho: Fornece uma "máquina do tempo" para voltar atrás para uma versão anterior, e fornece um bom suporte de diferentes versões do mesmo projeto.

Ao trabalhar em equipa: Simplifica muito trabalhar em concurrencia, dando "merge" de alterações feitas por diferentes pessoas.

2.2 o que é Git e GitHub



Quando fazemos "git init" num diretório de um projeto, ou quando fazemos "git clone" de um projeto existente, o Git cria um repositório (.git).

Em qualquer momento, podemos fazer um "snapshot" de tudo no dirétorio do projeto e guardar este no repositório. Este "snapshot" é chamado de **commit object**.

Um **commit** ocorre quando fazemos alterações que estão prontas para serem guardadas no repositório.

Quando realizamos um commit, o Git guarda um **commit object**:

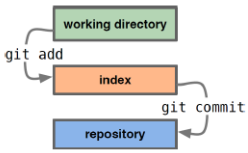
- Um estado completo do projeto, incluindo todos os ficheiros;
- O primeio não possui pai;
- Normalmente, pegamos num commit object, fazemos alterações, e criamos um novo commit object, pelo que a maior parte dos commit objects têm apenas um pai;
- Quando fazemos **merge** de dois commit objects, forma um commit object com dois pais.

Pelo que, os commit objects formam uma **DAG** (Directed Acyclic Graph). O Git é tudo sobre usar e manipular este grafo.

2.2.1 Mensagem de Commit

Os commits são "baratos" pelo que os devemos fazer com frequência, e com mensagens descritivas sobre o que foi alterado. Devem ter apenas uma linha.

Como não devemos dizer muito numa linha, devemos fazer vários commits.



2.3 Manter simples

- If you:
  - Make sure you are current with the central repository
  - Make some improvements to your code
  - Update the central repository before anyone else does
- Then you don't have to worry about resolving conflicts or working with multiple branches
  - All the complexity in git comes from dealing with these
- Therefore:
  - Make sure you are up-to-date before starting to work
  - Commit and update the central repository frequently