

# Maven - Forge - Intégration continue

Emmanuel Coquery

`emmanuel.coquery@univ-lyon1.fr`

`http://liris.cnrs.fr/~ecoquery`

→ Enseignement



# Autour du développement

Au delà du code :

- Tests (unitaires, intégration, fonctionnels)
- Documentation
- Partage des sources
- Suivi de bugs / évolutions
- Qualité du code
- Distribution

→ cycles de développement lourds à gérer

# Outils

- Frameworks de tests
- Générateurs de documentation
- Gestionnaires de version
- Gestionnaires de tickets
- Outils d'audit de code
- Scripts, *builders*

# Maven

## Automatisation du traitement des phases du cycle de vie

- Peut être vu comme un « super Makefile »
  - Java comme langage de script
- Lance l'exécution d'outils :
  - Compilation
  - Test automatisés
  - Archives, Déploiement
  - Génération de documentation
  - ...

Alternatives : CMake, Premake, etc

# Architecture

- Basée sur un système de plugins
  - Un plugin ↔ un script Java
    - i.e. une classe avec une méthode particulière
    - paramétrable via un bout de XML
  - Une exécution de maven ↔ suite d'exécution de plugins
  
- Nombreux plugins disponibles
  - Pas tous installés au départ
  - Système de téléchargement de plugins à la demande

# Phases et cycles de vie

- Une phase regroupe un ensemble de tâches (goals)
  - 1 tâche → 1 plugin
- Un cycle de vie est une suite de phases
  - Le déclenchement d'une phase déclenche les phases précédentes du cycle de vie
- Le cycle de vie dépend du *packaging* (jar,war, ...)
  - *packaging* = type de projet
  - Format d'archive
  - Ordre des phases
  - Affectation tâches → phases
  - Préconfiguration des tâches
  - peut être reconfiguré selon les besoins du projet



## Exemple : phases du *packaging* jar

| Phase                  | Tâche(s)                             |
|------------------------|--------------------------------------|
| process-resources      | <code>resources:resources</code>     |
| compile                | <code>compiler:compile</code>        |
| process-test-resources | <code>resources:testResources</code> |
| test-compile           | <code>compiler:testCompile</code>    |
| test                   | <code>surefire:test</code>           |
| package                | <code>jar:jar</code>                 |
| install                | <code>install:install</code>         |
| deploy                 | <code>deploy:deploy</code>           |

# Projet maven : organisation des fichiers

- pom.xml ← config. du projet
- src/ ← sources
  - main/ ← à distribuer
    - java/ ← code Java
    - resources/ ← fichiers à distribuer (config appli, images, etc)
    - webapp/ ← ressources web (pour les war : html, jsp, js, images)
    - javacc/ ← grammaire pour générer les *parsers*
    - ...
  - test/ ← uniquement pour les tests
    - java/, resources/, javacc/, etc
- target/ ← tout ce qui est généré, il est supprimé par clean



# Repository Maven

- Dépôt contenant :
  - Des plugins
  - Des bibliothèques (en général Java)
- Sur le web
  - Téléchargement automatique à la demande
  - Défaut : `http://repo1.maven.org`
  - Miroirs (Nexus, Archiva, etc)
- Local : `/.m2/repository`
  - contient les archives des projets locaux
    - phase `install`
  - cache pour les *repository* web

# Classpath et dépendances

## Utilisation de libs externes

- Téléchargement
- Gestion des versions
- Transitivité
- Configuration du CLASSPATH

Egalement utilisé pour les plugins

# Dépendances : *scope*

### Scope vs Classpath

|          | compilation | test | exécution | déploiement |
|----------|-------------|------|-----------|-------------|
| compile  | x           | x    | x         | x           |
| provided | x           | x    | x         |             |
| runtime  |             | x    | x         | x           |
| test     |             | x    |           |             |

(+ system, import)

# Archetypes

- Complexité inhérente aux projets maven
  - Difficultés de mise en œuvre
- Archetype = mini-projet de départ
  - D'un type particulier
  - Préconfiguré
- Exemples
  - maven-archetype-quickstart
  - spring-mvc-jpa-archetype
- En ligne de commande : `mvn archetype:generate`

# Intégration dans les EDI

## Projets

- Netbeans : par défaut
- Eclipse :
  - Plugin m2e + connecteurs
  - `mvn eclipse:eclipse`
    - configure un projet Eclipse
    - qui correspond au projet maven
- IntelliJ : par défaut

## Exécution :

- Intégrée dans tous les EDI (via plugin dans Eclipse)

# Forges logicielles

Outil de travail collaboratif pour le développement :

- Espace collaboratif
  - Partage de documents
  - Wiki
  - Dépôt (SVN, Mercurial, Git, etc)
- Gestion des tâches
  - Calendrier, Gantt
  - Bug tracking
  - Support, tâches diverses
- Système d'annonces

# forge.univ-lyon1.fr

- Forge Redmine
- Dépôts Mercurial
- Intégration SI Lyon 1 (LDAP + CAS)
  - Disponible aux étudiants et personnels
  - Utilisable pour les TPs
  - Obligatoire pour le projet MultiMIF

Merci à : J. Delmas, V. Hurtevent, F. Langlade-Bellone,  
Y. Mazzer, J. Nassimian, A. Plagnol



# Mercurial

- Gestionnaire de versions distribué
  - À la git / darcs / bazaar /etc
- Utilisable
  - En ligne de commande (hg)
  - Via une interface dédiée
    - Tortoise hg, Murky, MacHg, HgkExplorer, etc
  - Dans un EDI
    - Intégration Eclipse, Netbeans, IntelliJ, Emacs, etc



# Commandes de base

- Création
  - init, clone
- Fichiers
  - add, remove, move, revert, status
- Versions
  - commit, update, tag
- Branches
  - branch, branches, merge, heads
- Synchronisation de dépôts
  - pull, push

# Scénario simple

## ① Début du travail

- ① Clone d'un dépôt distant
- ② Modification d'un fichier
- ③ Commit
  - Pour l'instant, pas de modification du dépôt distant
- ④ Push vers le dépôt distant

## ② Plus tard ...

- ① Pull du dépôt distant
- ② Update
- ③ GOTO 1.2

# Suivi des tâches (gestionnaire de tickets)

- Les tâches ont
  - Une description
  - Un statut (nouveau, assigné, fermé, etc)
- Les tâches peuvent avoir :
  - Une échéance
  - Un pourcentage de réalisation
  - Un temps estimé/passé
- Découpage en sous-tâches
- Tâches liées
- Liens commit (revision 32)/tâche (numéro 1234) :
  - ref #1234 dans le message de commit
  - ref r32 dans les commentaires de la tâche

# Intégration Continue - contexte

## Sur des projets

- de grande taille
- impliquant de nombreuses personnes
- avec des itérations courtes



# Intégration Continue - principes

- Automatisation des phases du cycle de vie
  - Compilation, test, mise à disposition de binaires
- Institutions de bonnes pratiques
  - Commit réguliers
  - La branche par défaut compile
  - ...
- Surveillance
  - Tableaux de bord, etc

# Serveurs d'IC

Permet d'exécuter régulièrement :

- Checkout
- Compilation
- Test
- Audit de code

Fournit les différents tableaux de bord

ex : Hudson/Jenkins

# Sonar

## Outil d'audit de code

- Analyse exécutée lors du cycle de vie
  - Via e.g. un goal maven
- Fourni des tableaux de bord :
  - Qualité du code
  - Couverture des tests unitaires

# Références

- <http://maven.apache.org/>
- <http://mercurial.selenic.com/>