Environnements de développement (intégrés)

Développement collaboratif (CVS), automatisation de la compilation (Ant), documentation (Javadoc)

Patrick Labatut
labatut@di.ens.fr
http://www.di.ens.fr/~labatut/

Département d'informatique
École normale supérieure

Centre d'enseignement et de recherche en technologies de l'information et systèmes
École des ponts

Transparents inspirés de ceux du cours de Mihaela Sighireanu en 2006-2007.

Plan

Introduction 1/2

Le travail en équipe impose une coordination :

- à la conception,
- à la réalisation (codage),
- au planning du travail.

Un système de contrôle de version (du code source) permet :

- un contrôle d'accès au code,
- de maintenir un historique des modifications,
- de coordonner le travail simultané sur le code,
- de coordonner le travail sur plusieurs versions (« branches ») du code.

CVS[1] est un tel système (libre).

[1] *Concurrent Versions System*

Introduction 2/2

Concurrent :

- plusieurs utilisateurs peuvent accéder au code en même temps,
- plusieurs branches de développement peuvent exister en même temps.

Il existent d'autres systèmes (libres[2]) de contrôle de version : Subversion (SVN), GIT, ...

CVS est complètement intégré à Eclipse[3].

[2] Et plus modernes...

[3] Il existe cependant un greffon Eclipse pour Subversion, Subclipse : `http://subclipse.tigris.org/`.

Vocabulaire 1/2

Dépôt[4] : emplacement de stockage des fichiers sous contrôle de version.

Création du dépôt (en local) : `cvs -d chemin_absolu_du_depot init`

Le client CVS permet d'accéder à un dépôt local ou distant :

- en local, par ex. :

  `:local :/home/cvs`
- en distant, sur un serveur CVS dédié, par ex. :

  `:pserver :login@cvs.dev.java.net :/cvs`
- en distant, via une méthode d'authentification (SSH), par ex. :

  `:ext :login@cvs.savannah.gnu.org :/sources/classpath`

Contenu d'un dépôt : modules ($\sim$ projets).

Contenu d'un module : fichiers (sources) partagés ayant (chacun) un numéro de version ($\sim$ ressources).

[4] *Repository*, en anglais.

Vocabulaire 2/2

Opérations :

- importation de code source existant (n'étant pas encore sous contrôle de version) dans le dépôt (*import*),
- récupération d'une copie de travail d'un module (*checkout*), c'est sur cette copie que l'on travaille,
- mise-à-jour (*update*) de la copie de travail,
- ajouter un fichier/répertoire au dépôt (*add*),
- supprimer un fichier/répertoire au dépôt (*remove*),
- validation des modifications (*checkin*/*commit*),
- consulter l'historique/les différences (*log*/*history*/*diff*/*status*),
- attribuer un nom (*tag*).

Cycle de travail

1. récupérer une copie de travail d'un module (*checkout*),
2. modifier cette copie/ajouter/enlever des fichiers/répertoires (*add*/*remove*),
3. mettre à jour la copie de travail (*update*),
4. valider les modifications (*commit*),
5. (éventuellement) attribuer un nom à la version actuelle (*tag*),
6. aller à l'étape 2.

En pratique, des conflits dus à des modifications concurrentes (signalés à l'étape 3) doivent être résolus...

CVS sous Eclipse

Perspective dédiée à l'exploration de dépôt CVS : *CVS Repository Exploring*.
Rajouter un dépôt : *[Mouse-R / New / Repository Location...]*.
Récupérer une copie de travail d'un module comme nouveau projet :

- perspective CVS : *[Mouse-R / Checkout]*,
- partout : *[Mouse-R / Import. . . / CVS / Projects From CVS]*.

Opérations (tout est dans le menu *[Mouse-R / Team]*) :

- valider : *[Mouse-R / Team / Commit. . .]*,
- mettre à jour : *[Mouse-R / Team / Update]*,
- synchroniser (valider/metter à jour/résoudre les conflits) : *[Mouse-R / Team / Synchronize with Repository. . .]* (ouvre la perspective *Team Synchronizing*)
- importer : *[Mouse-R / Team / Share Project. . .]*,
- nommer : *[Mouse-R / Team / Tag as Version. . .]*,
- consulter l'historique : *[Mouse-R / Team / Show History]*.

Introduction

L'outil `make` d'automatisation de la compilation en C/C++ est un standard de fait.
Problèmes[5] :

- plusieurs version différentes incompatibles (GNU make, BSD make, MS nmake),
- suppose l'existence d'un shell (plutôt orienté Unix),
- extensible uniquement via des commandes shell,
- syntaxe rigide/archaïque/difficile à parser (*Tab*, ...).

Apache Ant[6] offre une alternative adaptée à Java :

- syntaxe XML (facile à générer et surtout à interpréter et à étendre) (`build.xml` $\sim$ `Makefile`),
- notion de tâche $\sim$ cible des `Makefile`,
- facilement extensible.

+/- considéré comme un outil standard d'automatisation de la compilation en Java.

[5] + le fait que `javac` compile automatiquement les dépendances des sources...
[6] `http://ant.apache.org/`

Ant sous Eclipse

Création d'un fichier `build.xml` pour Ant : *[Mouse-R / Export... / Ant Buildfiles]*

Attention : par défaut, le fichier `build.xml` généré utilise le compilateur Java intégré à Eclipse et n'est donc pas « portable » (de même, les bibliothèques de tests unitaires utilisées sont celles fournies avec Eclipse).

Création d'un projet Eclipse à partir de sources utilisant un fichier `build.xml` : *[File / New. . . / Project / Java Project From Existing Ant Buildfile]*

Lancement de tâches définies dans un fichier `build.xml` :

- *[Mouse-R / Run As / Ant Build]* (tâche par défaut),
- *[Mouse-R / Run As / Ant Build...]* (sélection des tâches à lancer).

Introduction

Système d'annotation du code (classe/interface/méthode/champ/...) : rend possible la génération automatique de documentation (sous forme de pages HTML le plus souvent[7]).

Développé par Sun, standard pour Java[8]

Pourquoi/pour qui écrire de la documentation ?

- pour spécifier (précisément) le comportement d'un code source,
- pour les autres développeurs utilisant un code source,
- pour vous-même, dans le futur...

[7] Il est possible de générer d'autre type de documentation (LaTeX, ...) ou d'information (changement d'API) à partir des annotations Javadoc...

[8] Il existe des équivalents (standards ou non) pour d'autres langages : Doxygen pour le C/C++ (+/- compatible avec Javadoc), Pod pour le Perl, ...

Écriture de documentation 1/2

Indiquer un commentaire Javadoc : `/** Commentaire Javadoc */`.

Au sein d'un commentaire Javadoc, on peut utiliser des marqueurs[9] spéciaux :

- `@param` : paramètre d'une méthode,
- `@return` : valeur de retour par une méthode,
- `@exception`/`@throws` : exception (éventuellement) levée par une méthode,
- `@author` : auteur(s),
- `@version` : version,
- `@see` : référence,
- `@since` : existe depuis,
- `@literal` : texte à ne pas interpréter (par ex. : `{@literal ArrayList<String>}`)
- ...

[9] *tag*, en anglais.

Écriture de documentation 2/2

En plus de ces *tags*, il est possible d'utiliser du HTML (basique, $\leq$ 3.2) : `<code>`, `<ul>`, `<b>`, `<i>`, `&lt ;`...

Il existe des conventions officielles pour l'écriture de documentation Javadoc :

- usage de la troisième personne,
- commencer par un verbe,
- utiliser `this` plutot que « l'objet »
- ...

`http://java.sun.com/j2se/javadoc/writingdoccomments/`

Création avec l'outil `javadoc` :

```
javadoc -d apidocs @liste_de_fichiers
```

Crée/met à jour un répertoire `apidocs` avec la documentation au format HTML.

Exemple

```
1  /**
2   * The <code>ImageLoader</code> simplifies loading images from the
3   * filesystem.
4   * @author Author1
5   * @author Author2
6   */
7  public class ImageLoader {
8      /** The unique instance of this class. */
9      private static ImageLoader imageLoader = new ImageLoader();
10     /* ... */
11
12     /** Initializes the unique instance of this class. */
13     private ImageLoader() {
14         /* ... */
15     }
16
17     /**
18      * Loads an image from the given file.
19      * @param   fileName  the file to load the image from.
20      * @return  the newly created image, or <code>null</code> if the image
21      *          could not be loaded from the given file.
22      */
23     public Image loadImage(String fileName) {
24         /* ... */
25     }
26
27     /**
28      * Returns the unique instance of the <code>ImageLoader</code> class.
29      * @return an <code>ImageLoader</code> object.
30      */
31     public static ImageLoader getDefaultImageLoader() {
32         return imageLoader;
33     }
34 }
```

Création de documentation dans Eclipse

Remarque : [Ctrl-Espace] fonctionne dans les commentaires (pour `@param`).

Création de documentation :

- *[Mouse-R / Export... / Java / Javadoc]*,
- Sélection des packages et des classes,
- Utilisation de la visibilité pour générer :
  - la documentation interne,
  - la documentation externe (dans le cas d'une bibliothèque),
- Choix du répertoire de destination pour la documentation,
- Options de génération : index/barre de nav./hiérarchie/...