Introduction à Subversion (1)

L'objectif de ce TP est de vous initier au Système de contrôle de versions (Subversion), que nous conseillons fortement d'utiliser lors de la conception de votre projet.
Ce TP se fait en binôme. Cette première partie consiste à comprendre les principes d'utilisation de svn. Pour cela, le binôme travaillera sur la même machine.
La seconde partie vise à tester l'aspect multi-utilisateur de svn. Dans cette partie, le binôme travaillera sur des machines différentes.

1 Qu'est ce que Subversion ?

Subversion gère de façon centralisée, sur un serveur, les versions successives d'un ensemble de fichiers et de la hiérarchie de répertoires qui les contient. Il prend en compte les modifications **simultanées** faites par plusieurs développeurs sur un même fichier, et propose des moyens pour résoudre les conflits pouvant survenir.

2 Principes généraux de Subversion

Subversion maintient l'ensemble de fichiers dans un répertoire qu'on appelle **dépôt** (**repository** en anglais). Ce dépôt peut être sur le disque local ou sur celui d'une machine distante. Dans le cas de ce TP, le dépôt se situe sur une machine distante (voir section 4.1)[1].
Toutes les versions successives de l'ensemble des fichiers et des répertoires sont stockées dans le dépôt (rien ne se perd) et chacune de ces versions est étiquetée par un entier. Cet entier commence à 1 pour la version d'origine, et est augmenté à chaque fois qu'une nouvelle version d'un ou plusieurs fichiers est publiée dans le dépôt avec la sous-commande **commit** de la commande svn (voir section 4.6).
Un membre de l'équipe peut obtenir auprès de Subversion une **copie de travail** (working copy) de ces fichiers et répertoires dans son **espace de travail** (working space ou encore sandbox) par la commande **checkout** (voir section 4.3). Par défaut la copie de travail contient les versions les plus récentes des fichiers et chaque répertoire de cette copie contient un répertoire .svn mémorisant des informations de gestion utilisées par Subversion, par exemple les versions des fichiers depuis le checkout ou le dernier commit ou update).
Une fois la copie de travail effectuée, il est alors possible de modifier (par exemple, avec un éditeur de texte) cette copie des fichiers puis mettre à jour le dépôt (on dira publier) avec ses modifications (commande commit). Lors du commit, chaque fichier réellement modifié fait l'objet d'un nouveau numéro de version qui est le nouveau numéro de version du projet.
Il peut aussi intégrer à sa copie de travail les modifications publiées entre-temps par les autres membres grâce à la commande **update** (voir la seconde partie de ce TP d'initiation à SVN).
Le checkout est en principe fait une seule fois pour chaque nouveau membre.
Les commandes suivantes n'ont qu'un effet local (sur la copie de travail), elles doivent être suivies d'une publication (commande **commit**) :

- **add** permet d'ajouter des fichiers ou des répertoires et leurs contenus sous le contrôle de Subversion.
- **move** permet de changer le nom d'un fichier ou d'un répertoire contrôlé par Subversion.
- **delete** permet de supprimer des fichiers ou des répertoires et leurs contenus du contrôle de Subversion.
- **resolved** permet de signifier qu'un conflit de modification d'un fichier a été résolu. Un tel conflit ne peut se produit que lors d'un update quand il se trouve qu'une modification provenant du dépôt et qu'une modification faite dans la copie de travail se recouvrent.

[1] Cette fonctionnalité n'est pas illustrée dans cette première partie

Ces quatre commandes ne prendront effet que lors du prochain commit qui demandera au dépôt de se créer une nouvelle version correspondant à la nouvelle structure. Bien entendu le membre ayant exécuté ce commit verra la même évolution dans sa copie de travail.

3 Références et aide en ligne

Le site de Subversion :
`http://subversion.tigris.org`
Les documents de Philippe Durif, dont le TP s'inspire :
`http://www2.lifl.fr/~durif/svn/index.php`
Le livre en ligne sur Subversion :
`http://svnbook.red-bean.com`
L'aide fournie par le client svn :

- La commande **svn help** pour avoir une aide générale à propos de svn;
- La commande **svn help commit** pour avoir de l'aide à propos de la sous-commande commit. Bien entendu, cela fonctionne avec toutes les sous-commandes de svn.

4 Prise en main de Subversion

Dans cette section, vous allez utiliser l'interface client svn pour gérer différentes versions d'un projet. Il s'agit de confier à Subversion la version initiale du TP d'Initiation à ant, puis d'en faire deux nouvelles versions qui incorporent de nouvelles fonctionnalités. Ensuite il sera possible de retrouver chacune des trois versions stockées dans le dépôt de Subversion.

Les deux membres du binôme doivent travailler sur la même machine pour cette partie du TP.

4.1 Dépôt utilisé

Si vous avez précisé votre binôme à votre enseignant, vous disposez d'un dépôt **temporaire**, se situant à l'adresse[2] :

- `https://svn.fil.univ-lille1.fr/svn/`*login*`_PL_TP_SVN`

Par exemple, si les logins du binome sont *"dupontm"* et *"jacksonj"*, alors le dépôt est :

- `https://svn.fil.univ-lille1.fr/svn/dupontm_PL_TP_SVN`

Si vous ne l'avez pas fait, il ne vous est pas possible de faire ce TP.

4.2 Placer une version initiale du projet dans le dépôt

Subversion permet soit de commencer le projet à zéro, soit d'importer une version initiale du projet. Dans la section qui suit, nous illustrons comment importer la version initiale d'un projet.

4.2.1 Préparation des fichiers nécessaires

Dans un premier temps, vous allez récupérer les fichiers qui feront office de version initiale.

Manipulation 1

1. *Créez un répertoire nommé TP_PL dans l'espace de travail de la machine que vous utilisez.*
2. *Téléchargez l'archive* `tp_svn.tar.gz` *sur le portail.*
3. *Décompressez-la dans le répertoire TP_PL.*

[2] dans cette adresse, *login* correspond au login du membre du binome étant le premier dans l'ordre alphabétique.

4. *Supprimez l'archive.*

Le répertoire *TP_PL* ressemble maintenant à :

```
TP_PL/
'-- TP_Test_SVN
    |-- build.xml
    |-- rapport
    |   '-- rapport.txt
    '-- src
        '-- ant
            '-- example
                '-- HelloWorld.java
```

Le répertoire *TP_Test_SVN* constitue la version initiale du projet.

4.2.2 Ajouter la version initiale du projet dans le dépôt

Il s'agit ici de mettre, une fois pour toutes, la version initiale de ce nouveau projet dans le dépôt (sous-commande **import**).

La commande import ne peut être utilisée qu'une seule fois par dépôt.

Manipulation 2

1. *Placez vous dans le répertoire TP_PL*
2. *Entrez la commande :*
   `svn import -m "Version initiale du TP" TP_Test_SVN`
   `https://svn.fil.univ-lille1.fr/svn/login_PL_TP_SVN`
   L'option ***-m*** *permet de spécifier un message qui sera associé à la version initiale du projet.*
3. *Spécifiez le login de l'un des deux binômes, et le mot de passe qu'il utilise dans les salles du M5*
4. *Supprimez le répertoire TP_PL*

Remarquez que la commande import affiche tous les fichiers et répertoires mis sous le contrôle de Subversion ainsi que le numéro initial de version (numéro 1). Cet affichage produit (dans le cadre de ce TP) :

```
Ajout          TP_Test_SVN/src
Ajout          TP_Test_SVN/src/ant
Ajout          TP_Test_SVN/src/ant/example
Ajout          TP_Test_SVN/src/ant/example/HelloWorld.java
Ajout          TP_Test_SVN/rapport
Ajout          TP_Test_SVN/rapport/rapport.txt
Ajout          TP_Test_SVN/build.xml

Révision 1 propagée.
```

4.3 Prendre une copie de travail du Projet

On ne travaille jamais directement dans le dépôt : pour tester ou modifier le logiciel, il faut d'abord en prendre une copie de travail avec la commande **checkout**. En principe un checkout n'est fait qu'une seule fois par un développeur qui commence à travailler sur le projet.

Manipulation 3

1. *Créez un répertoire intitulé* ***workingSpace***
2. *Placez-vous dans ce répertoire. Il sera utilisé comme espace de travail (voir section 2).*

3. *Effectuez une copie de travail du dépôt, en entrant la commande :*
   `svn checkout https://svn.fil.univ-lille1.fr/svn/login_PL_TP_SVN`

Remarquez que checkout affiche tous les répertoires et fichiers obtenus auprès du dépôt pour constituer votre copie de travail, ainsi que le numéro de version :

```
A    login_PL_TP_SVN/rapport
A    login_PL_TP_SVN/rapport/rapport.txt
A    login_PL_TP_SVN/src
A    login_PL_TP_SVN/src/ant
A    login_PL_TP_SVN/src/ant/example
A    login_PL_TP_SVN/src/ant/example/HelloWorld.java
A    login_PL_TP_SVN/build.xml
Révision 1 extraite.
```

Dans chaque répertoire créé, un répertoire *.svn* a été ajouté. Il sert à gérer la copie de travail (liste des fichiers/répertoires gérés, numéros de version des fichiers/répertoires depuis la dernière commande checkout, commit ou update).
A partir de maintenant, toutes les opérations s'effectueront à partir du répertoire login_PL_TP_SVN

4.4 Consulter l'état de votre copie de travail

Il est possible à tout moment de connaitre l'état de votre copie de travail (fichiers modifiés ou ajoutés, numéro de version, ...) à l'aide de la sous-commande **status**[3], en écrivant : `svn status -vu`
Attention dans notre cas, beaucoup de colonnes n'affichent rien car rien n'a été fait depuis le checkout !

4.5 Modifications du fichier build.xml

Il s'agit de faire évoluer le projet de son état initial à une nouvelle version : nous souhaitons ajouter la fonctionnalité **clean** au fichier **build.xml**.

Exercice 1 : La modification du fichier **build.xml** ne nécessite aucun moyen sortant de l'ordinaire : vous pouvez utiliser l'éditeur de votre choix.

1. En utilisant la commande ant **delete**, écrivez une cible **clean** qui supprime le répertoire **classes** (question 1.4 du TP d'initiation à ant).
2. Modifiez le fichier **build.xml** de façon à ce que la cible *clean* soit exécutée à chaque appel de la cible **compile** (question 1.5 du TP d'initiation à ant).

Observez le nouvel état de votre espace de travail, à l'aide de la sous-commande **status -uv**, et interprétez le à l'aide du tableau 1 situé en annexe.

Question 1 *Y a-t-il des différences par rapport à la première fois que vous avez appelé **status** ? Lesquelles ?*

4.6 Publication dans le dépôt de cette modification

La commande **commit** sert à mettre à jour le dépôt avec les dernières modifications effectuées dans la copie de travail. Si au moins un des fichiers de la copie de travail est modifié par rapport à sa version la plus récente dans le dépôt, la commande commit augmente de 1 la version globale du projet et les nouvelles versions des fichiers modifiés sont publiées dans le dépôt avec ce même numéro de version ainsi que dans la copie de travail.

Manipulation 4

1. *A l'aide de **commit**, mettez à jour le dépôt avec les dernières modifications apportées dans l'espace de travail :*
   `svn commit -m "clean intégré à build.xml"`
   *L'option **-m** permet de spécifier un message qui sera associé à la nouvelle version du projet.*

[3] Voir le tableau 1 en annexe pour interpréter les valeurs affichées

2. *Affichez le statut de l'espace de travail à l'aide de la commande **status -uv**.*

Remarquez que seuls les fichiers modifiés sont publiés et que la version 2 leur est donnée ainsi qu'au projet. On peut le revoir avec la commande status.

Manipulation 5

1. *Effectuez un **commit** à nouveau.*

Question 2 *Qu'observez-vous ?*

Ainsi, si la copie de travail ne comporte aucune modification par rapport à la version la plus récente du dépôt, le commit n'a aucun effet.

4.7 Observer quelques historiques

Il est possible de vérifier les différentes versions des fichiers du dépôt à l'aide de la sous-commande **log**.

Manipulation 6

1. *Observez les différentes versions des fichiers **HelloWorld.java** et **build.xml** à l'aide de la commande : `svn log nomDuFichier`*

Question 3 *Qu'observez-vous ?*

De même, il est possible de récupérer la liste des différentes versions du projet sur le dépôt, à l'aide de la commande :
`svn log https://svn.fil.univ-lille1.fr/svn/login_PL_TP_SVN`

4.8 Seconde modification du projet

Cette manipulation va vous amener à mettre de nouveaux fichiers sous le contrôle de SVN avec la commande locale **add**.

Exercice 2 :

1. Créez une nouvelle classe intitulée **Text** dans le package **ant.example**. Cette classe dispose d'un attribut **text** qui est une chaîne de caractères, et de deux méthodes :
   - **getText** qui retourne la valeur de l'attribut **text**
   - **setText** qui permet de redéfinir la valeur de l'attribut **text**.
2. Modifiez la class **HelloWorld**, de sorte que la méthode **main** :
   (a) créée une instance de la classe **Text**, dont l'attribut **text** a pour valeur "(Text) Hello World !".
   (b) affiche la valeur cet attribut.

4.9 Ajouter les nouveaux fichiers au dépôt

Une fois les modifications terminées, on souhaite les enregistrer sur le dépôt.

Manipulation 7

1. *A l'aide de **commit**, mettez à jour le dépôt avec les dernières modifications apportées dans l'espace de travail. N'oubliez pas de préciser le message associé au commit à l'aide de l'option **-m**.*
2. *Affichez la liste des fichiers disponibles dans le dépôt à l'aide de la commande : `svn list -R https://svn.fil.univ-lille1.fr/svn/login_PL_TP_SVN`*
3. *Affichez le statut de l'espace de travail à l'aide de la commande **status -uv**.*

Question 4 *Qu'observez vous ?*

Question 5 *Selon vous, comment expliquer l'absence du fichier **Text.java** dans le dépôt, et le point d'interrogation dans le statut de l'espace de travail ?*

Contrairement au fichier **HelloWorld.java**, le fichier **Text.java** n'existe pas encore dans le dépôt. Ainsi, il n'est pas pris en compte directement lors de l'appel à **commit** : il faut l'ajouter à l'aide de la sous-commande **add**.

Manipulation 8

1. *A l'aide de **add**, ajoutez le fichier **Text.java** dans l'espace de travail :*
   `svn add src/ant/example/Text.java`
2. *Affichez la liste des fichiers disponibles dans le dépôt.*
3. *Affichez le statut de l'espace de travail à l'aide de la commande **status -uv**.*

Question 6 *Qu'observez vous ?*

Add est une commande à effet purement local, un **commit** reste nécessaire pour publier les nouveaux fichiers dans le dépôt.

Manipulation 9

1. *A l'aide de **commit**, mettez à jour le dépôt avec les dernières modifications apportées dans l'espace de travail. N'oubliez pas de préciser le message associé au commit à l'aide de l'option **-m**.*
2. *Affichez la liste des fichiers disponibles dans le dépôt.*

Dès qu'une application devient importante, il devient nécessaire d'organiser les différents fichiers qui la composent dans une hiérarchie de répertoires. L'application évoluant, de nouveaux répertoires seront introduits dont il faudra demander la prise en compte par Subversion (à nouveau avec la commande **add**).

Exercice 3 : On souhaite ajouter au projet le nouveau paquetage **ant.tools** contenant les deux types **One** et **Two**.

1. Créez un répertoire intitulé **src/ant/tools**, par exemple à l'aide de la commande shell **mkdir**.
2. Créez les deux classes **One** et **Two** dans le package **ant.tools**. Ces deux classes ne disposent d'aucune méthode ni d'attributs.
3. Ajoutez le répertoire **src/ant/tools** à la copie de travail à l'aide de la sous-commande svn **add**.
4. A l'aide de **commit**, mettez à jour le dépôt avec les dernières modifications apportées dans l'espace de travail. N'oubliez pas de préciser le message associé au commit à l'aide de l'option **-m**.

Un appel à `svn status -uv` doit produire un affichage similaire à :

```
5       1 login       rapport/rapport.txt
5       1 login       rapport
5       5 login       src/ant
5       5 login       src/ant/tools
5       5 login       src/ant/tools/One.java
5       5 login       src/ant/tools/Two.java
5       4 login       src/ant/example
5       4 login       src/ant/example/Text.java
5       3 login       src/ant/example/HelloWorld.java
5       4 login       src
5       2 login       build.xml
5       5 login       .
```

4.10 Changer le nom d'un répertoire ou d'un fichier

La gestion d'un projet de grande ampleur vous amènera parfois à modifier le nom d'un fichier ou d'un répertoire. Pour illustrer ceci, on décide de changer le nom du paquetage **tools** en **misc**.
Tout comme l'ajout de nouveaux fichiers, il est nécessaire de spécifier ces changements à l'espace de travail. Cela s'effectue à l'aide de la sous-commande svn **move**.

Manipulation 10

1. *Renommez le répertoire **tools** en **misc** dans l'espace de travail à l'aide de la commande :* `svn move src/ant/tools src/ant/misc`.
2. *Affichez le statut de l'espace de travail à l'aide de la sous-commande svn **status -uv**.*
3. *A l'aide de **commit**, mettez à jour le dépôt avec les dernières modifications apportées dans l'espace de travail. N'oubliez pas de préciser le message associé au commit à l'aide de l'option **-m**.*
4. *Affichez le statut de l'espace de travail à l'aide de la sous-commande svn **status -uv**.*

Question 7 *Qu'observez-vous ?*

La sous-commande svn **move** crée, dans la copie de travail, le répertoire misc puis effectue effectivement le mouvement du contenu du répertoire tools vers misc.

4.11 Arrêter la gestion d'un répertoire ou d'un fichier

La gestion d'un projet vous amènera aussi à supprimer des fichiers. Par exemple, les classes *One* et *Two* sont des classes vides, que l'on ne souhaite plus avoir dans le projet.
Tout comme l'ajout ou la modification du nom de fichiers, il est nécessaire de spécifier ces changements à l'espace de travail.

Manipulation 11

1. *Affichez le statut de l'espace de travail à l'aide de la sous-commande svn **status -uv**.*
2. *Supprimez le package **ant.misc** à l'aide de la commande svn **delete** :* `svn delete src/ant/misc`
3. *Affichez le statut de l'espace de travail à l'aide de la sous-commande svn **status -uv**.*
4. *Affichez le contenu du répertoire **src/ant/misc** dans votre espace de travail, par exemple à l'aide de la commande shell **ls**.*

Question 8 *En observant le résultat des commandes **status -uv** et **ls**, pouvez-vous spécifier ce que fait la sous-commande svn **delete** ?*

La sous-commande svn **delete** détruit, dans la copie de travail, le fichier spécifié. S'il s'agit d'un répertoire, le contenu du répertoire est détruit. Le répertoire lui-même ne sera détruit que lors du prochain appel à **commit**.

Manipulation 12

1. *A l'aide de **commit**, mettez à jour le dépôt avec les dernières modifications apportées dans l'espace de travail. N'oubliez pas de préciser le message associé au commit à l'aide de l'option **-m**.*
2. *Affichez le statut de l'espace de travail à l'aide de la sous-commande svn **status -uv**.*

4.12 Supprimer une copie de travail

Nous avons fini de faire les modifications liées à cette première partie du TP. Nous n'avons donc plus besoin de la copie de travail : toutes les versions dont nous avons besoin se trouvent sur le serveur Subversion.
La suppression de la copie de travail revient simplement à supprimer le répertoire où elle se situe. Dans notre cas, il s'agit du répertoire **workingSpace**.

Manipulation 13

1. *Supprimez le répertoire **workingSpace**, par exemple à l'aide de la commande shell **rm**.*

Lors de la conception de votre projet, certains problèmes peuvent survenir et corrompre votre espace de travail (crash de disque dur, virus, etc.). Dans ce cas, vous aboutissez à une situation similaire à celle qui suit la manipulation de la section précédente.
Pour récupérer une version valide de travail, il vous suffit d'effectuer à nouveau les opérations de la section 4.3.

4.13 Récupérer la version précédente du projet dans l'espace de travail

Lors de la conception d'un projet, vous allez sûrement être confrontés à une situation où les modifications faites sur la version courante de votre espace de travail ne sont pas satisfaisantes. Dans ce cas, il est possible de retourner à une ancienne version du projet à l'aide de la sous-commande svn **update**.

Manipulation 14

1. *Retournez à la version initiale (version 1) de votre projet à l'aide de la commande :*
   `svn update -r1`

Exercice 4 : A l'aide des sous-commandes svn **log** et **update**, faites en sorte de retourner au projet tel qu'il était à la fin de la section 4.7.

Vérifiez que le contenu du fichier build.xml correspond bien à celui de la version souhaitée : il doit contenir la cible clean.

4.14 Récupérer le livrable de chaque version

La commande **export** est l'inverse de la commande **import** : elle permet de récupérer les sources du projet mais sans aucune des informations de gestion obtenues par un **checkout** (pas de répertoire .svn). Autrement dit export permet d'obtenir les sources livrables au client. Grâce aux numéro de révision globaux et à la commande log, on va pouvoir récupérer facilement différentes versions du livrable.
Toutes les manipulations qui suivent se vont se faire dans un répertoire nouvellement créé, que nous appellerons **livrables**.

Manipulation 15

1. *Créez un répertoire **livrables** sur votre session, par exemple à l'aide de la commande shell **mkdir**.*

4.14.1 Récupérer le livrable de la version initiale

La sous-commande **export** récupère une version livrable du projet, et la place dans le répertoire courant. Il est possible de préciser le numéro de la version que l'on souhaite récupérer. Nous allons récupérer une telle version dans un répertoire intitulé **livrables/version1**.

Manipulation 16

1. *Créez un répertoire **livrables/version1***
2. *Placez-vous dans ce répertoire*
3. *A l'aide de la sous-commande svn **export**, récupérez la version initiale de votre projet sur le serveur Subversion en entrant la commande :*

```
svn export --revision 1 https://svn.fil.univ-lille1.fr/svn/login_PL_TP_SVN
```

4.14.2 Récupérer le livrable de la version prenant en compte uniquement ”clean”

En combinant l’utilisation de **export** avec l’utilisation de la sous-commande **log**, il est possible de créer facilement un livrable avec n’importe quelle version du projet. En effet, **log** vous permet d’afficher les différents numéros de version, ainsi que le commentaire qui a été fourni lors de la création de la version (avec l’option **-m** lors du commit).

Exercice 5 : Cet exercice vise à créer un livrable du projet, dont la version ne prend en compte que ”clean”.

1. Créez un répertoire **livrables/version_clean**
2. A l’aide de la commande **log**, retrouvez la version du projet pour laquelle la commande clean venait d’être ajoutée.
3. A l’aide de la commande **export**, créez dans le répertoire **livrables/version_clean** un livrable du projet pour cette version.

4.14.3 Récupérer le livrable de la version la plus récente

Un livrable de la version la plus récente du projet peut être obtenue en ne spécifiant pas l’option **–revision** dans la sous-commande **export** :

```
svn export https://svn.fil.univ-lille1.fr/svn/login_PL_TP_SVN
```

5 Exercice de synthèse

Quel que soit le cas, vous ne DEVEZ PAS utiliser l’option force de la commande svn.

5.1 Préparation

Pour simplifier le travail effectué dans l’exercice qui suit, nous allons dans un premier temps supprimer le contenu du dépôt.

Manipulation 17

1. *Supprimez **build.xml** à l’aide de la sous-commande svn **delete***
2. *Supprimez **rapport/** à l’aide de la sous-commande svn **delete***
3. *Supprimez **src/** à l’aide de la sous-commande svn **delete***
4. *Effectuez un **commit**, en spécifiant avec l’option **-m** un message explicite marquant le début de l’exercice.*

5.2 L’Exercice

Exercice 6 : Dans cet exercice, vous allez devoir utiliser la plupart des commandes decrites dans la section précédente.

1. Dans un premier temps, faites de sorte de créer l’arborescence qui suit **dans votre espace de travail** Subversion (et pas dans le dépôt !!!!) :

```
‘-- PL_Projet
    |-- build.xml
    ‘-- src
        ‘-- pl
            ‘-- ClassePrincipale.java
```

La classe **ClassePrincipale** est une classe sans attributs, et dont l'unique méthode est la méthode statique **main(String[] args)**, qui ne fera pour l'instant qu'afficher le message "Hello World !" à l'écran.

Le fichier **build.xml** contiendra une cible par défaut **exec_clean** qui :

(a) Crée un répertoire **classes**

(b) Compile dans **classes** les fichiers java

(c) Exécute la méthode **main** de la classe **ClassePrincipale**

(d) Supprime le répertoire **classes** et son contenu

2. Après vous être assurés que le programme fonctionne, et que le répertoire **classes** est bien supprimé après un appel à la commande **ant**, effectuez une mise à jour de votre dépôt.

La sous-commande import a déjà été utilisée sur ce dépôt. Il vous faudra donc utiliser les sous-commandes add et commit.

3. Vous allez faire évoluer ces fichiers, pour atteindre l'arborescence qui suit **dans votre espace de travail** :

```
'-- PL_Projet
    |-- build.xml
    '-- src
        '-- pl
            '-- project
                |-- Game.java
                |-- GameCharacter.java
                '-- Main.java
```

La classe **ClassePrincipale** y est renommée en **Main**, et le package des classes devient **pl.project** à la place de **pl**.

De plus, deux nouvelles classes sans attributs ni méthodes sont ajoutées dans ce package : **Game** et **GameCharacter**.

4. Après vous être assurés que le programme fonctionne après ces modifications (la commande **ant** doit s'exécuter sans produire d'erreurs), effectuez la mise à jour de votre dépôt.

A Annexes

Table 1: Quelques informations pour comprendre ce qui est imprimé par **status**.

colonne 1	
’ ’	fichier non modifié depuis son **checkout**
’A’	fichier ajouté avec **add** mais le **commit** n’a pas été fait
’C’	fichier avec un conflit de modification non résolu suite à un **update**
’M’	fichier modifié localement depuis le **checkout**, le dernier **update** ou le dernier **commit**
’D’	fichier supprimé localement depuis le **checkout**, le dernier **update** ou le dernier **commit**
’?’	fichier ou répertoire non géré par Subversion
colonne 4	
’ ’	fichier ajouté à l’aide de la commande **add**.
’+’	fichier ajouté provenant de la commande **move**.
colonne 8	
’ ’	copie de travail issue de la plus récente version du serveur
’*’	une version plus récente se trouve sur le serveur
colonne 9	
numéro de version local de la copie de travail	
colonne 10	
numéro de version d’origine : version du dépôt ayant produit ou mis à jour la copie de travail	
colonne 11	
auteur de la version d’origine	
colonne 12	
nom du répertoire ou fichier	