Introduction à Subversion

Nous allons dans ce TP apprendre à travailler avec Subversion en simulant 2 utilisateurs sur un dépôt distant. Nous disposons d'un répertoire de travail init contenant le fichier point.py suivant

point.py

```
class point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return 'point: (%e, %e)'%(self.x, self.y)

if __name__ == '__main__':
    print point(2, 3)
```

Nous voulons créer un dépôt Subversion avec uniquement ce fichier. Pour cela, nous allons nous servir de la commande svnadmin.
Dans un premier temps, créez un répertoire server_depot qui fera office de dépôt distant.

```
terminal$ mkdir /chemin/vers/server_depot
terminal$ svnadmin create /chemin/vers/server\_depot
```

Maintenant que notre serveur local est prêt, nous allons initialiser notre projet via la commande svn import

```
terminal$ svn import init file:///chemin/vers/server_depot/trunk  \
          -m "initialisation du projet"
```

Nous allons maintenant créer la copie du dépôt pour nos deux utilsateurs.

1. Créez un répertoire depot_de_user1 qui servira de répertoire de copie du dépôt à l'utilisateur 1.

2. Créez un répertoire depot_de_user2 qui servira de répertoire de copie du dépôt à l'utilisateur 2.

3. Allez dans le répertoire depot_de_user1 et importer le trunk du server_depot via la commande svn checkout ou svn co

```
terminal$ svn co file:///chemin/vers/server_depot/trunk .
```

4. faire de même pour l'utilisateur 2 dans le répertoire depot_de_user2.

   Les 2 utilisateurs ont maintenant leur copie de travail et ils sont prêts à bien avancer sur ce projet. L'utilisateur 1 décide de modifier un peu le fichier point.py

point.py

```
class point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return 'point: (%e, %e)'%(self.x, self.y)

    def __add__(self, p):
        return point(self.x + p.x, self.y + p.y)

if __name__ == '__main__':
    print point(2, 3)
```

   Il met à jour le dépôt distant via la commande svn commit ou svn ci

```
terminal$ svn ci -m "ajout de la méthode addition dans point"
```

   De son coté, l'utilisateur 2 a lui aussi modifié le fichier point.py de la manière suivante

point.py

```
class point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return 'point: (%e, %e)'%(self.x, self.y)

    def __sub__(self, p):
        return point(self.x - p.x, self.y - p.y)

if __name__ == '__main__':
    print point(2, 3)
```

Il cherche lui aussi à mettre à jour le dépôt. Il tape donc la même commande que l'utilisateur 1

```
terminal$ svn ci -m "ajout de la méthode soustraction dans point"
```

Expliquez ce qui se passe. Pour régler le problème, il est nécessaire de mettre à jour la copie via la commande svn update ou svn up

```
terminal$ svn up
```

Nous sommes face à un conflit car les mêmes lignes du fichier point.py ont été modifiées. Plusieurs solutions s'offrent à vous

- p: marque ce conflit pour résolution ultérieure
- e: résout manuellement le conflit avec un éditeur
- mc: accepte ma version pour tous les conflits
- tc: accepte l'autre version pour tous les conflits

Nous supposons que nous n'avons pas l'éditeur adéquat. Nous allons donc utiliser la commande p afin de dire que nous allons gérer le problème plus tard.

Ouvrez à présent le fichier point.py et résolvez le conflit en laissant que les lignes nécessaires. Uitlisez la commande svn resolve pour dire que le conflit est résolu. Vous pouvez à présent resoumettre les modifications de l'utilisateur 2

```
terminal$ svn resolved point.py
terminal$ svn ci -m "ajout de la méthode soustraction dans point"
```

Les conflits peuvent néanmoins se régler automatiquement si le source n'est pas modifié au même endroit. Par exemple, l'utilisateur 1 met à jour sa copie pui ajoute, après la méthode `__sub__`, la méthode suivante

point.py

```
class point:
    ...

    def __mul__(self, p):
        return point(self.x*p.x, self.y+p.y)

if __name__ == '__main__':
    print point(2, 3)
```

et l'utilisateur 2 change au niveau du main

point.py

```
if __name__ == '__main__':
    print point(2, 3)
    print point(2, 3) + point(3, 4)
    print point(2, 3) - point(3, 4)
```

Suivre la même procédure que précédemment (chaque utilisateur essaye de mettre à jour le dépôt). Que constatez-vous ?
L'utilisateur 1 souhaiterait maintenant développer ses idées sans géner les autres développeurs. Pour cela, il va créer une branche au projet initial.

```
terminal$ svn mkdir file:///chemin/vers/server_depot/branches
terminal$ svn ci -m "ajout du repertoire branches"
terminal$ svn copy file:///chemin/vers/server_depot/trunk \
          file:///chemin/vers/server_depot/branches/user1
```

Il utilise ensuite la commande svn switch pour se retrouver dans sa branche

```
terminal$ svn switch file:///chemin/vers/server_depot/branches/user1
terminal$ svn info
```

Maintenant qu'il est bien dans son espace de travail et qu'il ne gène personne, il décide d'ajouter un fichier rectangle.py

rectangle.py

```python
from point import point

class rectangle:
    def __init__(self, p1, p2):
        self.p1 = p1
        self.p2 = p2

    def __str__(self):
        s = 'rectangle avec les points\n'
        s += '\t' + self.p1.__str__()
        s += '\n\t' + self.p2.__str__()
        return s

if __name__ == "__main__":
    r = rectangle(point(0, 0), point(1, 1))
    print r
```

En utilisant la commande svn status, il constate que rectangle.py a un ! devant signifiant que subversion ne sait pas quoi faire de ce fichier. Il utilise alors la commande svn add pour ajouter ce fichier

```
terminal$ svn add rectangle.py
terminal$ svn ci -m "ajout du fichier rectangle.py"
```

L'utilisateur 1 continue à faire ses développements dans son coin.
Continuer à modifier les fichiers, à en ajouter, ... et à faire un ensemble de svn commit.

De son coté, l'utilisateur 2 continue à travailler avec la branche initiale et il se rend compte qu'il y a un gros bug dans le fichier point.py au niveau de la méthode **__mul__**.
Il fait les modifications suivantes:

point.py

```
class point:
    ...

    def __mul__(self, p):
        return point(self.x*p.x, self.y*p.y)

...
```

```
terminal$ svn commit -m "correction d'un bug dans point.py"
```

Il fait remonter l'information à l'ensemble des développeurs. L'utilisateur 1 est donc obligé de faire ces changements sur sa branche pour corriger lui aussi ce bug. Il va pour cela utiliser la commande svn merge

```
terminal$ svn merge file:///chemin/vers/server_depot/trunk
terminal$ svn commit -m "fusion avec le tronc"
```

L'utilisateur 1 décide finalement de revenir à une version antérieure de son travail. Il utilise la commande svn log pour avoir des informations sur les différents svn commit.

```
terminal$ svn log -v
```

Il repère la révision qu'il souhaite (on pourra prendre par exemple la révision avant

l'ajout du fichier rectangle.py), puis il utilise la commande svn update pour revenir à cette révision. On supposera dans cet exemple que l'on souhaite revenir à la révision 5.

```
terminal$ svn up -r 5
terminal$ svn commit -m "retour à la révision 5"
```

Et les développements continuèrent ...