Prise de contact avec subversion.

Le tp se déroule sous Linux et en console.

1-Exploration de dépôts

Voici quelques URLs de projets célèbres:

Apache: http://svn.apache.org/repos/asf/httpd

GCC: http://gcc.gnu.org/svn/gcc/

Sans récupérer de copie locale, répondre aux questions suivantes:

1. Quel est le numéro de la dernière révision des deux référentiels ?
2. Lequel des deux projets à fait le plus de modifications ?
3. Quel est le nombre et le nom des différentes branches du projet httpd de la Fondation Apache ? Comparer avec les tags...
4. Comment afficher le fichier README de la dernière révision de la branche 2.2.x ?
5. Que produit la commande « svn blame » appliquée au README de la branche 2.2.x ?
6. Quelle est la révision publiée par « jorton » ?

2-Création d'un référentiel.

Créer un dépôt subversion dans /tmp/tpsvn/repos avec la commande « svnadmin », puis exécuter la commande:

```
ls -al /tmp/tpsvn/repos
```

Comment savoir que c'est un dépôt ?

Quel est le type de système de fichier utilisé par le dépôt ?

Quel est l'UUID du dépôt ?

Comment peut-on connaître cet UUID en utilisant svn ?

Créer un répertoire « trunk » dans le dépôt.

Observe-t-on une différence avec précédemment lorsque l'on exécute la commande:

```
ls -al /tmp/tpsvn/repos
```

Comment faire pour vérifier que le répertoire « trunk » à bien été créé ?

3-Importer un premier projet dans le référentiel.

On veut créer un petit programme tout simple qui affiche "Bonjour" suivi du premier paramètre passé en argument. Ce qui donnera par exemple:

```
./hello Chabal
Bonjour Chabal
```

Créer un répertoire /tmp/tpsvn/import.

Créer ensuite dans le répertoire « import » un répertoire « doc » puis un fichier « doc/index.html » contenant une aide sommaire pour le programme « hello ».

Créer à l'intérieur du répertoire « import » le fichier source « hello.c » du programme.

Le compiler et l'exécuter.

Quand il fonctionne correctement, supprimer l'exécutable et les fichiers temporaires (s'il y en a), puis importer le projet dans le dépôt subversion à l'aide de la commande « import ».

Comment faire pour vérifier que l'opération à bien fonctionné ?

Quelle est le N° de révision du dépôt maintenant ?

Que faut-il faire avec le fichier hello.c ?

4-Récupération du projet avec un checkout.

Récupérer une copie locale du dépôt dans /tmp/tpsvn/sandbox avec la commande « checkout »

Qu'affiche la commande:

```
ls -al /tmp/tpsvn/sandbox
```

Aller dans le répertoire « trunk »

Qu'affiche la commande:

```
svn status -v
```

Modifier le fichier le fichier hello.c pour que le programme affiche "Bonjour subversion" si aucun paramètre n'est passé en argument.

Qu'affiche la commande:

```
svn status -v
```

Renommer le fichier « hello.c » en « bye.c »:

```
mv hello.c bye.c
```

Qu'affiche la commande:

```
svn status -v
```

Et la commande:

```
ls -l
```

Qu'observe-t-on après avoir exécuté la commande:

```
svn revert hello.c
```

Que constate-t-on cependant ? Pourquoi ?

5-Validation des modifications

Renommer le fichier « bye.c » en « hello.c ».

Contrôler qu'il fonctionne correctement et « commiter » la modification sur le serveur.

Pourquoi n'a-t-on pas besoin de donner l'URL du dépôt ?

Pourquoi n'a-t-on pas besoin de faire le « ménage » avant de commiter les changements ?

Taper ensuite les commandes:

```
svn log
svn log hello.c
```

Que constate-t-on ?

Quelle différence avec la commande:

```
svn blame hello.c
```

Supprimer le fichier hello.c, puis exécuter les commandes:

```
svn status -v
ls -l
```

Si on exécute à nouveau la commande « svn revert » que ce passe-t-il ? Pourquoi ?

Exécuter la commande:

```
rm -fr /tmp/tpsvn/sandbox/.svn
```

Exécuter la commande:

```
svn status -v
```

Que se passe-t-il ?

6-Ajout d'un fichier au projet.

Supprimer le contenu de la sandbox et récupérer une copie du dépôt avec:

```
rm -fr /tmp/tpsvn/sandbox/*
svn co file:///tmp/tpsvn/repos /tmp/tpsvn/sandbox/
```

Aller dans le répertoire « trunk » de la sandbox, puis créer un fichier Makefile pour le projet, qui permette de compiler l'application en release et en debug avec la première cible « first » qui compile en release et la cible « all » qui compile les deux versions.

(On appellera la version debug « hello_d » et la version release « hello ».)

Taper "make", puis « make all » pour s'assurer que ça marche...

Ajouter le Makefile à subversion.

Comment vérifier que le fichier à bien été ajouté ?

7-Renommer un fichier

Renommer avec subversion le fichier hello.c en bonjour.c. (Penser à mettre à jour le Makefile...)

Une fois la modification terminée (y compris le commit), faire un « export » vers le répertoire /tmp/tpsvn/livrable.

Quelle est la différence avec un « checkout » ?

Renommer directement sur le dépôt le répertoire « doc »:

file:///tmp/tpsvn/repos/trunk/doc en file:///tmp/tpsvn/repos/trunk/documentation

Comparer les résultats des deux commandes:

```
svn status -v
svn status -u
```

Que constate-t-on après avoir exécuté la commande:

```
svn update
```

Afficher les différences entre la première et la dernière version du fichier hello.c

8-Gestion des conflits

Sortir de la sandbox et créer un répertoire « conflit ».

Faire dans ce répertoire deux checkout du projet pour deux utilisateurs (fictifs) différents:

```
svn co file:///tmp/tpsvn/repos/trunk/ harry
svn co file:///tmp/tpsvn/repos/trunk/ sally
```

Aller dans le répertoire de Harry et modifier le fichier bonjour.c pour que le programme affiche:

```
(c) Harry
Bonjour Chabal
```

(Ajouter une instruction sur une ligne avant l'instruction qui existait...)

Commiter la modification.

Aller dans le répertoire de Sally et modifier le fichier bonjour.c pour que le programme affiche:

```
Bonjour Chabal
(c) Sally
```

Commiter la modification.

Que se passe-t-il ? Pourquoi ?

Finalement corriger le programme pour qu'il affiche pour Harry:

```
Bonjour Chabal
(c) Sally & Harry
```

et pour Sally:

```
Bonjour Chabal
(c) Harry & Sally
```

Commiter la modification.

Que se passe-t-il ? Pourquoi ?

Résoudre le problème et commiter la correction.

9-Gestion des branches et des labels

Maintenant que cette version est prête, il est temps de faire un tag pour pouvoir l'identifier facilement.

1-Création d'un tag:

D'un point de vue technique, un tag est une copie, il suffit donc de faire une copie du « trunk » dans « tags/rel-1.0 ».

On peut faire le « copy » aussi bien dans la sandbox que directement sur le serveur, soit

```
svn copy file:///tmp/tpsvn/repos/trunk/
file:///tmp/tpsvn/repos/tags/rel-1.0 -m"Creation du label rel-1.0"
```

ou encore (en supposant qu'on est dans le répertoire /tmp/tpsvn/repos):

```
svn copy /trunk tags/rel-1.0

svn ci -m "Creation du label rel-1.0"
```

Quelle que soit la méthode utilisée, contrôler que la copie a bien fonctionné avec un « svn ls file:///tmp/tpsvn/repos/tags/ ».

Pour améliorer la qualité de votre logiciel on vous demande d'ajouter aux option de compilation les options -Wall (pour activer tous les warnings) et « -ansi -pedantic » pour être certain que vous respectez bien les standards...

Modifier votre Makefile et vérifier que votre projet compile toujours. Si besoin faîtes les corrections nécessaires, pour qu'il n'y ait plus de warning ou d'erreur.

N'oublier pas de commiter vos modifications.

2-Création d'une branche:

Pour repartir dans une configuration propre, supprimer la sandbox:

```
rm -fr /tmp/tpsvn/sandbox
```

puis faites un checkout du trunk dans la nouvelle sandbox:

```
svn co file:///tmp/tpsvn/repos/trunk tmp/tpsvn/sandbox
```

Aller dans la sandbox et contrôler votre code...

Vous savez également que d'après les recommandations standards un programme doit renvoyer « EXIT_SUCCESS » en cas de réussite ou « EXIT_FAILURE » en cas d'échec (voir man 7 sdtlib.h en cas de doute), or il semble d'après certaines rumeurs que la release-0.1 ne respecterait pas cette consigne.

Il est urgent de corriger ce bug afin de satisfaire les plus puristes des futures clients...

Pour cela, créer une branche « bugfix-1.0 » dans « branches » sur le dépôt, c'est-à-dire en copiant directement les deux URLs concernées (sans passer par une copie dans la sandbox).

Si le répertoire branches n'existe pas, il faut le créer...

Il faut maintenant rapidement corriger le bug, mais la sandbox contient uniquement le trunk en principe... Or il faut travailler dans la branche bugfix-1.0 que l'on vient de créer.

On pourrait supprimer le contenu de la sandbox et refaire un checkout de la branche...

Mais on peut aussi utiliser la commande « switch » qui permet de mettre jour la copie locale avec une branche du dépôt...

Aller dans « trunk » et faire le switch pour basculer le dépôt de la copie locale.

Corriger le problème de conformité de votre programme.

Valider votre modification avec un commit.

Le bug étant corrigé, il faut créer un nouveau tag « rel-1.1 ». Faire la copie avec la méthode de votre choix.

3-Fusion de branches:

Il reste maintenant à intégrer la correction dans le développement principal: le trunk.

Il faut donc réutiliser la commande « switch » pour basculer dans le trunk.

Pour faire la fusion entre la version corrigée et la version principale, il faut utiliser la commande « merge ». Le principe de la fusion est tout simplement de prendre, les corrections apportées aux fichiers dans une branche, pour les appliquées (sous la forme d'un patch) à l'autre branche.

Mais pour pouvoir utiliser cette commande, il faut les numéros des révisions que l'on veut fusionner.

On utilisera pour ça la commande « log » qui permet de voir les différentes révisions et pour clarifier les choses, on utilisera l'option « --stop-on-copy » qui permet de détecter les créations de branches et de tags.

```
svn log  --verbose --stop-on-copy
file:///tmp/tpsvn/repos/tags/rel-1.1/hello.c

svn log  --verbose --stop-on-copy
file:///tmp/tpsvn/repos/tags/rel-1.0/hello.c
```

Une fois que l'on a récupéré les numéros de révision AvantCorrection (AV) et AprèsCorrection (AP), on réalisera la fusion en utilisant ces numéros (il ne faut pas écrire AV et AP évidemment):

En étant dans le répertoire trunk, celui dans lequel on veut fusionner la branche, taper:

```
svn merge -rAV:AP file:///tmp/tpsvn/repos/tags/rel-1.1
```

Faire un svn status -v pour s'assurer que la fusion a réussie et qu'elle n'a pas généré de conflit...

(si un conflit apparaît le résoudre, comme précédemment.)

Contrôler le fichier fusionné, puis faire le commit pour valider la modification.

Remarques:

La commande « merge » peut également servir à annuler une modification (en supposant que l'erreur est été validée à la révision 12 alors que la 11 était bien) on fait un merge avec les numéros de revision inversés:

```
svn merge -r12:11 file:///tmp/tpsvn/repos/trunk
```

Dans ce cas, on écrase la révision 12 avec la révision 11.

Attention il n'y a pas de d'historique des fusions. Il faut être prudent et marquer dans le message les infos concernant la fusion...

Le développement peut continuer, dans le trunk, jusqu'à la sortie de la release 2.0, que l'on

marquera par un tag... et ainsi de suite.