Dans ce TD, vous allez commencer à vous familiariser avec SVN (Subversion). La commande principale pour cet utilitaire est `svn`. Les fonctionalités qu'il offre sont accessibles par des sous-commandes.

```
export TD4=$HOME/1A-OMGL-VCS/TD4
mkdir $TD4
cd $TD4
```

Exercice 1. Création d'un repo que nous appelerons `$TD4/TD-SVN`. Ceci se fait grace à la commande `svnadmin`. Pour comprendre comment l'utiliser :

```
svnadmin --help
```

On découvre ainsi la sous-commande `create` :

```
svnadmin create --help
```

On sait à présent comment créer notre repo :

```
svnadmin create $TD4/TD-SVN
```

Exercice 2. Commençons par créer un petit projet :

```
mkdir salut
cat > salut/README <<EOF
ce programme dit bonjour en plusieurs langues
EOF
cat > salut/salut.c <<EOF
int main()
{
}
EOF
```

Avant d'importer ce projet dans SVN, nous allons créer dans le repo l'arborescence administrative usuelle aux projets gérés par SVN. Il n'est pas obligatoire de suivre cette organisation, mais elle a fait ses preuves et est donc recommendée :

```
svn mkdir -m "creation␣du␣projet␣salut" \
  file://$TD4/TD-SVN/salut{,/trunk,/branches,/tags}
```

Nous venons de créer, dans le repo, un projet `salut` ayant 3 sous-répertoires :

- `trunk` dans lequel résidera le tronc principal de dévelopement
- `branches` dans lequel nous créerons des branches de dévelopement parallèles

- `tags` dans lequel nous créerons des tags, c'est à dire des références à des révisions particulières du projet (correspondant par exemples à des livraisons)

Notez que la commande `svn mkdir` accède au repo au travers d'un URL. En général, `svn` utilise (la syntaxe) des chemins locaux pour accéder à une copie de travail et (la syntaxe) des URL pour accéder au repo. Lorsqu'on accède au repo *localement*, on utilise un URL du type `file:///path...`

Exercice 3. Il est temps d'importer les fichiers initiaux du projet. On se renseigne d'abord sur l'opération d'importation :

```
svn import --help
```

Puis on la met en pratique :

```
cd salut
svn import -m "import␣initial" file://$TD4/TD-SVN/salut/trunk
```

Exercice 4. On peut alors se débasser du répertoire qui a servi à faire l'import (il n'est pas sous contrôle de SVN ; il a juste servi à faire l'import), puis obtenir une copie de travail à partir du répo grâce à un checkout. On s'informe d'abord sur la sous-commande `checkout` :

```
svn checkout --help
```

Puis on la met en pratique :

```
cd ..
rm -rf salut
svn checkout file://$TD4/TD-SVN/salut/trunk salut
```

Exercice 5. Nous remarquons que notre projet contient un bug : le `main` ne retourne pas 0 comme il devrait. Pour réparer un bug, une bonne méthodologie rendue aisée par SVN est de créer une branche. Pour créer une branche on utilise simplement l'opération de copie : on va copier le `trunk` dans `branches/bug-1`. On se renseigne d'abord sur l'opération `copy` :

```
svn copy --help
```

Puis on la met en pratique :

```
svn copy -m "creation␣d'une␣branche␣pour␣le␣bug␣1" \
  file://$TD4/TD-SVN/salut/{trunk,branches/bug-1}
```

Exercice 6. Nous pouvons à présent obtenir un copie de travail de cette nouvelle branche :

```
svn checkout file://$TD4/TD-SVN/salut/branches/bug-1 salut-bug-1
```

Allons dans ce répertoire de travail :

```
cd salut-bug-1
```

et corrigeons le bug en ajoutant `return 0` dans `main`. Nous pouvons voir les fichiers modifiés dans notre copie de travail grâce à la sous-commande `status` :

```
svn status
```

Nous pouvons à présent faire un commit. On s'informe d'abord sur la sous-commande `commit` :

```
svn commit --help
```

Puis on la met en pratique :

```
svn commit -m "bug-1␣fixed"
```

Notez que, puisque nous somme dans notre répertoire de travail, il n'est pas nécessaire de spécifier le répo sur la ligne de commande. En effet, cet information est enregistrée dans le sous-répertoire administratif appelé `.svn` qui joue pour SVN un rôle assez similaire au sous-répertoire `CVS` pour CVS.

Exercice 7. Maintenant que, sur notre branche, nous avons développé un "bugfix" qui est bien au point, nous allons fusionner les changements correspondants dans le tronc. Pour déterminer le changements effectués sur la branche, il faut comparer l'état de la branche au moment où elle a été créée et son état final après notre travail. L'état de la branche a un moment donné est identifié par un numéro de révision. Examinons tout d'abord l'historique de la branche :

```
svn log
```

Notez que cela inclus aussi l'historique avant la création de la branche : c'est l'historique du projet depuis son origine jusqu'au bout de cette branche. On se renseigne sur la sous-commande `log` :

```
svn log --help
```

On découvre ainsi une option interessante `--stop-on-copy` qui permet de ne pas allez au-delà du moment où on a copié (le tronc) pour créer la branche :

```
svn log --stop-on-copy
```

On découvre ainsi que le numéro de révision pour la création de la branche est 3 (indiqué par `r3`).

Exercice 8. Maintenant allons dans la copie de travail du tronc pour fusionner les changements effectués sur la branche :

```
cd ../salut
```

On se renseigne sur la sous-commande `merge`

```
svn merge --help
```

Puis on la met en pratique. Une bonne idée avant de véritablement effectuer une fusion "pour de vrai" c'est de faire "comme si" pour voir se qui se passerait et vérifier qu'il n'y aurait pas de problèmes. Pour cela on utilise l'option `--dry-run` :

```
svn merge --dry-run -r 3:HEAD \
   file://$TD4/TD-SVN/salut/branches/bug-1
```

La notation `-r 3:HEAD` indique que nous voulons tous les changements entre la version 3 (création de la branche) et la version `HEAD` (la plus récente sur la branche). Tout semble se passer correctement. Nous n'avons pas de conflits. Nous pouvons donc faire cette fusion pour de vrai :

```
svn merge -r 3:HEAD file://$TD4/TD-SVN/salut/branches/bug-1
svn commit -m "fusion de branches/bug-1"
```

Exercice 9. Notez que le commit crée une nouvelle révision (r5) dans le repo. Mais notre copie de travail n'est pas encore informée :

```
svn info
```

Pour synchroniser notre copie de travail avec le repo, il faut faire un update :

```
svn update
svn info
```

Exercice 10. On notera une limitation de SVN : lorsqu'on effectue un merge entre branches, on fusionne les changements correspondants, mais l'historique associé à ces changements n'est pas propagé :

```
svn log
```

On voit ci-dessus que le commit correspondant au bugfix proprement dit, bien qu'il soit enregistré dans la branche, n'apparait pas dans le tronc après fusion. La raison est que l'opération `merge` de SVN et CVS ne fusionne que les changements et non pas les historiques. Les messages de log correspondants à ces changements ne sont donc pas transférés. C'est dommage ! Cette limitation n'existe plus avec le système de versionage `bzr`.

Pour palier à cette limitation, nous avons deux options principales :

- rédiger pour le commit du merge un nouveau message de log qui incorpore toutes les informations utiles présentent dans les messages de log sur la branche
- plus simplement, insérer l'output de la commande suivante dans le message de log :

```
svn log --stop-on-copy file://$TD4/TD-SVN/salut/branches/bug-1
```

Exercice 11. Nous pouvons observer les différences entre deux versions (ou la copie de travail et une version) grâce à la sous-commande `diff` :

```
svn diff --help
svn diff -r 2:5
```

Exercice 12. Nous allons restructurer notre projet pour faire apparaitre un sous-répertoire `src` pour le code source et un sous-répertoire `doc` pour la documentation. Commençons par le premier qui va illustrer la première façon de procéder :

```
mkdir src
svn add src
svn mv salut.c src/salut.c
svn commit -m "code␣source␣dans␣src"
```

Avec cette méthode, nous avons travaillé dans la copie de travail. Nous pouvons vérifier que l'historique associé au fichier `salut.c` a été préservé par l'opération `svn mv` :

```
svn log src/salut.c
```

Pour le répertoire `doc`, nous allons utiliser une seconde méthode qui opère directement sur le répo :

```
svn mkdir -m "creation␣du␣repertoire␣doc" \
  file://$TD4/TD-SVN/salut/trunk/doc
```

Nous pouvons observer que notre copie de trvail n'est plus à jour par rapport au répo :

```
svn status --show-updates
```

Il nous faut donc mettre à jour :

```
svn update
```

Pour illustrer une troisième méthode, nous allons aussi ajouter un sous-répertoire `include`. Nous utiliserons à nouveau `svn mkdir`, mais cette fois-ci avec en argument un chemin dans la copie de travail :

```
svn mkdir include
svn commit -m "creation␣du␣repertoire␣include"
```

Maintenant déplaçons le fichier `README` dans le répertoire `doc` :

```
svn mv README doc/README
svn commit -m "documentation␣dans␣doc"
```

Nous n'avons pas vraiment besoin du sous-répertoire `include`, nous allons donc le supprimer :

```
svn rm include
svn commit -m "suppression␣de␣include"
```

Exercice 13. Une fonctionalité interessante de SVN, c'est de pouvoir annoter chaque ligne d'un fichier avec la version dans laquelle cette ligne a été le plus récemment modifiée, et l'utilisateur responsable :

```
svn blame src/salut.c
```

Cette fonctionalité est également présente dans CVS grâce à la sous-commande `annotate`. La sous-commande `blame` peut-être invoquée dans Emacs. Visitons le fichier `src/salut.c` avec Emacs, puis utilisons le menu `Tools > Version Control > Annotate` (que l'on peut également invoquer avec '`C-x v g`').

Exercice 14. SVN a un grand nombre de paramètres qui peuvent être personalisés par chaque utilisateur. Visitez le fichier `~/.subversion/config`. Celui-ci est divisé en sections comme `[auth]`, `[helpers]`, `[tunnels]` etc... Dans la section `[helpers]`, ajoutez la ligne :

```
editor-cmd = emacs
```

Dorénavant, SVN saura que vous voulez éditer vos message de log avec Emacs (vous pouvez bien entendu stipuler un autre éditeur).

Exercice 15. Ajoutez un `#include <stdio.h>` en haut de `src/salut.c` et un `printf("bonjour\n");` dans le `main`. Puis ajoutez la ligne :

```
* francais
```

dans `doc/README`.

```
svn status
svn diff
svn commit
```

Quand nous invoquons la sous-commande `commit`, notre éditeur est utilisé pour nous permettre de rédiger le message de log : nous rédigeons, nous sauvegardons, et nous quittons l'éditeur. Le commit termine et créer une nouvelle révision dans le répo. Rappelons nous que notre copie de travail n'est pas encore synchronisée avec le répo :

```
svn info
```

Plus précisemment, notre copie de travail possède bien en effet toutes les modifications enregistrées dans le répo, mais ne sait pas encore que cela veut dire qu'elle peut utiliser la nouvelle révision (r11) comme version de base. Notre copie pense encore que sa version de base est la revision 7. Il suffit de faire un update :

```
svn update
svn info
```

Exercice 16. Il existe plusieurs méthodes pour accéder à un répo avec SVN : directement dans le système de fichiers (c'est ce que nous avons fait jusqu'à présent avec des URLs `file:///path`), en utilisant ssh avec des URLs `svn+ssh://host/path`, au travers de HTTP, et au travers d'un serveur dédié avec des URLs `svn://host/path`. Nous allons tester cette dernière méthode :

```
svnserve --listen-port=8080 --root=$TD4/TD-SVN -d --foreground
```

Retournons dans le répertoire parent de `salut` :

```
svn co svn://localhost:8080/salut/trunk SALUT
```