Dans ce TD, vous allez commencer à vous familiariser avec BZR. A la différence de CVS et SVN qui sont des systèmes de versionage centralisés (c'est à dire où le partage d'un même répo est une condition sine-qua-non pour la collaboration), BZR est un système de versionage décentralisé (c'est à dire qui permet de collaborer tout en autorisant chacun avoir son propre répo). La commande principale pour cet utilitaire est `bzr`. Les fonctionalités qu'il offre sont accessibles par des sous-commandes.

Le site web de `bzr` est `http://bazaar-vcs.org/`. J'ai installé la version 1.0 de `bzr` sous :

`/pub/1A/VCS/bzr-1.0`

Il faut donc ajouter ce répertoire à votre `PATH` :

```
export PATH=/pub/1A/VCS/bzr-1.0:$PATH
```

Comme pour les TD précédents, nous allons créer et travailler dans un répertoire spécifique :

```
export TD5=$HOME/1A-OMGL-VCS/TD5
mkdir $TD5
cd $TD5
```

Exercice 1. La première prise de contact avec un VCS est toujours de voir l'aide offerte par la commande :

```
bzr --help
```

qu'on peut aussi obtenir par `bzr help`, c'est à dire la sous-commande `help`. Ceci énumère quelques commandes de base assez similaires à celles offertes par CVS et SVN. Ceci nous indique également qu'on peut obtenir la liste de toutes les commandes de la manière suivante :

```
bzr help commands
```

Exercice 2. La seconde chose à faire avec `bzr`, c'est de vous munir d'une identité (celle-ci sera utilisée dans vos commits) grâce à la sous-commande `whoami` (qui-suis-je). D'abord on s'informe sur la commande :

```
bzr whoami --help
```

Puis on la met en application :

```
bzr whoami 'Frank Chu <fchu@example.com>'
```

où vous remplacerez `Frank Chu` par votre nom et `<fchu@example.com>` par votre addresse email. On peut évidemment mentir, mais le but de cette identification est de permettre (1) de vous identifier dans vos contributions (2) de vous contacter le cas échéant.

Exercice 3. Comme dans les TD précédents, nous allons commencer 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
```

On peut mettre ce répertoire sous le contrôle de `bzr` grâce à la commande `bzr init` dont on peut obtenir l'aide grâce à l'une ou l'autre des commandes suivantes :

```
bzr help init
bzr init --help
```

Le texte d'aide nous explique une des démarches possibles pour importer un projet dans `bzr`. Suivons cette recette :

```
cd salut
bzr init
bzr add .
bzr status
```

La sous-commande `status` nous informe que les fichiers `README` et `salut.c` ont été ajoutés (`added`) dans notre checkout. Pour finaliser ces ajouts, il nous faut maintenant faire un commit. On se renseigne d'abord sur la sous-commande `commit` :

```
bzr commit --help
```

Puis on la met en pratique :

```
bzr commit -m "import␣initial"
```

Notez que contrairement à CVS et SVN qui, après un `import`, nécessitent de faire un `checkout`, la démarche ci-dessus met directement le répertoire sous contrôle de BZR. Il n'y a aussi pas besoin de créer et référer à un répo. Un répo est créé automatiquement : le contenu, l'historique et les informations administratives relatives au répo, à la branche et au checkout se trouvent dans le sous-répertoire `.bzr`. Bien entendu, il ne faut pas aller bricoler dans ce sous-répertoire.

Exercice 4. Nous remarquons que notre projet contient un bug : le `main` ne retourne pas 0 comme il devrait. Pour réparer ce bug, nous suivrons la méthodologie déjà adoptée dans le TD sur SVN : nous allons faire une branche :

```
cd $TD5
bzr branch salut salut.bug-1
```

Allons dans cette branche et obtenons les infos associées à cette branche grâce à la sous-commande `info`. D'abord on s'informe sur la commande :

```
cd salut.bug-1
bzr info --help
```

puis on la met en pratique :

```
bzr info
```

Exercice 5. 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`. D'abord on s'informe sur cette commande :

```
bzr status --help
```

puis on la met en pratique :

```
bzr status
```

Nous pouvons à présent faire un commit en précisant le message de log comme avec SVN :

```
bzr commit -m "bug-1␣fixed"
```

Le répo et toutes les infos historiques et administratives sont enregistrées dans le sous-répertoire `.bzr`. Celui joue un rôle similaire à `CVS` pour CVS et `.svn` pour SVN, mais contient de plus son propre répo.

Exercice 6. Nous pouvons observer l'historique du projet grâce à la sous-commande `log`. On s'informe d'abord sur cette commande :

```
bzr log --help
```

puis on la met en pratique :

```
bzr log
bzr log README
bzr log salut.c
```

quand on demande le log d'un fichier, seules les révisions ayant modifié ce fichier sont listées.

Exercice 7. Maintenant que, sur notre branche, nous avons développé un "bugfix" qui est bien au point, nous allons le fusionner dans le tronc. Retournons donc dans celui-ci :

```
cd $TD5/salut
```

On va commencer par s'informer, grâce à la sous-commande `missing` sur les révisions effectuées dans la branche `salut.bug-1` et qui ne sont pas dans le tronc. On s'informe d'abord sur cette commande :

```
bzr missing --help
```

puis on la met en pratique :

```
bzr missing ../salut.bug-1
```

On peut également visualiser les différences grâce à la sous-commande `diff`. On s'informe d'abord sur cette commande :

```
bzr diff --help
```

puis on la met en pratique :

```
bzr diff . ../salut.bug-1
```

Notez qu'il y a deux arguments dans la commande ci-dessus : le premier fait référence au checkout courant et le second à la branche du bugfix. Bien ! tout est dans l'ordre et nous procédons à la fusion grâce à la sous-commande `merge`. On s'informe d'abord sur cette commande :

```
bzr merge --help
```

puis on la met en pratique :

```
bzr merge ../salut.bug-1
```

Contrairement à CVS et SVN, nous n'avons ici pas besoin d'identifier les révisions à fusionner : comme la commande `missing` l'a illustré plus haut, les révisions non-présentes sont automatiquement identifiées. La sous-commande `status` nous indique que le fichier `salut.c` a été modifié et liste les log des révisions qui viennent d'être fusionnées dans notre copie de travail :

```
bzr status
```

Nous pouvons alors faire un commit pour enregistrer cette fusion :

```
bzr commit -m "fusion␣de␣salut.bug-1"
```

Exercice 8. Avec CVS et SVN, la fusion d'une branche dans une autre ne transfère pas automatiquement l'historique. En particulier, elle n'incorpore pas les messages de log correspondant aux modifs que l'on fusionne. Avec `bzr`, ce transfert est automatique. On peut le vérifier avec la commande de `log` :

```
bzr log
bzr log salut.c
bzr log README
```

Exercice 9. Nous allons restructurer notre projet pour faire apparaitre un sous-répertoire `src` pour le code source et un sous-répertoire `doc` pour la documentation.

```
mkdir src
bzr add src
mkdir doc
bzr add doc
```

`bzr` supporte le renommage (comme SVN) grâce à la sous-commande `mv`. On s'informe d'abord sur cette commande :

```
bzr mv --help
```

puis on la met en pratique :

```
bzr mv salut.c src
bzr mv README doc
```

On peut observer les modifs qui attendent d'être commitées :

```
bzr status
```

On peut aussi tenter un `diff` :

```
bzr diff
```

On peut à présent faire notre commit :

```
bzr commit -m "restructuration␣avec␣src/␣et␣doc/"
```

Exercice 10. Comme CVS et SVN, `bzr` peut également annoter chaque ligne d'un fichier avec la version dans laquelle cette ligne a été le plus récemment modifiée, et l'utilisateur responsable. D'abord on s'informe sur la commande :

```
bzr annotate --help
```

puis on la met en pratique :

```
bzr annotate src/salut.c
bzr annotate --long src/salut.c
```

Exercice 11. Nous avons réparé le bug-1, mais nous aurions peut-être dû ajouter un commentaire dans le code. Retournons dans la branche `salut.bug-1` :

```
cd $TD5/salut.bug-1
```

Dans le `main`, avant le **`return`**, ajoutez le commentaire idiot "*`/* retourne status = succes */`*". Puis faites un commit :

```
bzr commit -m "documentation␣du␣return"
```

Exercice 12. A présent retournons dans la branche `salut` :

```
cd $TD5/salut
```

et regardons ce qu'il nous manque comme révisions par rapport à la branche `salut.bug-1` :

```
bzr missing ../salut.bug-1
```

ceci nous informe que nous avons 2 révisions non-présentes dans `salut.bug-1`, et qu'il nous manque une révision non-présente dans `salut`. Rappelez-vous que maintenant, dans la branche `salut` le fichier `salut.c` se trouve dans le sous-répertoire `src` alors que ça n'est pas le cas dans la branche `salut.bug-1`. Malgré ces différences, nous pouvons quand même fusionner la branche `salut.bug-1` dans la branche `salut` car `bzr` gère le renommage et sait donc que le fichier `salut.c` du premier correspond au fichier `src/salut.c` du second :

```
bzr merge ../salut.bug-1
bzr status
bzr commit -m "fusion␣de␣salut.bug-1␣(mise␣a␣jour)"
bzr log
```

Exercice 13. `bzr` a un certain nombre de paramètres qui peuvent être personalisés soit globalement, soit par branche. Le fichier de configuration `~/.bazaar/bazaar.conf` contient une section nommée `[DEFAULT]` (attention, les majuscules sont importantes). Cette section devrait contenir un paramètre `email` ayant la valeur que vous avez spécifié avec la commande `bzr whoami`. Rajoutez la ligne suivante :

```
editor=emacs
```

pour indiquer que vous voulez éditez vos message de log avec emacs (vous pouvez bien entendu choisir un autre éditeur).

Exercice 14. 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`.

```
bzr status
bzr diff
bzr 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.

Exercice 15. Dans cet exercice nous allons mettre en œuvre la notion de répo partagé par plusieurs branches : au lieu que chaque branche ait son propre répo (ce qui conduit à une duplication des révisions communes dans chaque répo), on peut faire en sorte qu'elles partagent toutes le même répo. Tout d'abord retournons dans le répertoire `TD5` :

```
cd $TD5
```

La création de répo se fait grâce à la sous-commande `init-repo`. On s'informe d'abord sur cette commande :

```
bzr init-repo --help
```

puis on la met en pratique :

```
bzr init-repo SALUT
```

ceci crée un répertoire `SALUT` qui contient un sous-répertoire `.bzr` qui va fonctionner comme un répo partagé par toutes les branches que nous créerons sous SALUT. Nous allons adopter l'organisation suggérée pour SVN et créer des sous-répertoires `trunk`, `branches` et `tags`.

```
cd SALUT
mkdir branches
mkdir tags
bzr branch ../salut trunk
```

voilà : nous avons créés les répertoires `branches` et `tags`, et importé la branche `salut`, sur laquelle nous travaillions précédemment, dans la branche `trunk`.

Exercice 16. Nous allons faire une branche dans laquelle nous allons ajouter le support pour l'Allemand :

```
cd $TD5/SALUT
bzr branch trunk branches/allemand
```

Allons dans le checkout de cette branche :

```
cd branches/allemand
```

et modifions `src/salut.c` en ajoutant `printf("guten␣Tag\n");` avant l'autre `printf`. Puis faisons un commit :

```
bzr commit -m "support␣pour␣l'allemand"
```

Exercice 17. Retournons dans le `trunk` :

```
cd $TD5/SALUT/trunk
```

et modifions `src/salut.c` en ajoutant `printf{"hello\n");` avant l'autre `printf`. Puis faisons un commit :

```
bzr commit -m "support␣pour␣l'anglais"
```

Exercice 18. Nous allons à présent fusionner `branches/allemand` dans `trunk` :

```
bzr merge ../branches/allemand
```

ceci nous informe qu'un conflit a été rencontré. Vous pouvez vérifier que le répertoire `trunk/src` contient les fichiers `salut.c.BASE`, `salut.c.OTHER` et `salut.c.THIS`. Nous pouvons résoudre ce conflit manuellement :

```
cd src
kdiff3 salut.c.{BASE,THIS,OTHER} -o salut.c
```

Dans l'interface graphique, nous pouvons choisir d'adopter à la fois les modifs de `salut.c.THIS` et ceux de `salut.c.OTHER`. On sauve. On quite `kdiff3`. Ensuite il faudrait invoquer les commandes suivantes (**mais ne le faisons pas**) :

```
bzr resolve salut.c
bzr commit -m "fusion␣du␣support␣pour␣l'allemand"
```

Exercice 19. Au lieu d'effectuer la fusion en invoquant `kdiff3` manuellement etc... nous allons utiliser un plugin pour bzr. Ce plugin s'appelle `extmerge`. Il est mentionné sur la page des plugins de `bzr` qui nous informe que sa branche est disponible à l'url suivant :

`http://erik.bagfors.nu/bzr-plugins/extmerge`

Les plugins doivent être installé dans le répertoire `~/.bazaar/plugins` :

```
cd ~/.bazaar
mkdir plugins
cd plugins
bzr branch http://erik.bagfors.nu/bzr-plugins/extmerge
```

Voilà, c'est tout ! le plugin est installé ! Il ajoute la sous-commande `extmerge` à `bzr`. Retournons dans la branche `trunk` et invoquons ce plugin. D'abord on s'informe sur cette commande :

```
bzr extmerge --help
```

puis on la met en pratique :

```
bzr extmerge src/salut.c
bzr resolve src/salut.c
bzr commit -m "fusion␣du␣support␣pour␣l'allemand"
```

Exercice 20. Maintenant nous allons prétendre que `AILLEURS` vous vouliez créer une branche `AILLEURS/salut.commentaires` *liée* à `SALUT/trunk`. Ceci veux dire que tous les commits sur cette branche sont d'abord effectués sur `SALUT/trunk` avant de l'être sur la branche. Dans cette branche, nous ajouterons simplement des commentaires pour les nouveaux `printf` :

```
cd $TD5
mkdir AILLEURS
cd AILLEURS
bzr checkout ../SALUT/trunk salut.commentaires
cd salut.commentaires
```

Dans cette nouvelle branche, modifions d'abord `src/salut.c` pour ajouter la ligne `/* Francais */` juste au dessus du `printf` en Français.

```
bzr commit -m "commentaire␣du␣francais"
```

Exercice 21. Maintenant retournons dans `SALUT/trunk` :

```
cd $TD5/SALUT/trunk
bzr status
```

Nous sommes informé que la copie de travail n'est pas à jour par rapport à la branche. En effet :

```
cat src/salut.c
```

nous démontre que le commentaire `/* Francais */` n'apparait pas encore dans notre copie de travail (le checkout). Il nous faut invoquer `bzr update` :

```
bzr update
cat src/salut.c
```

Maintenant nous sommes à jour.

Exercice 22. Retournons dans `AILLEURS/salut.commentaires` :

```
cd $TD5/AILLEURS/salut.commentaires
```

Supposons à présent que `AILLEURS` soit temporairement déconnecté et ne puisse plus avoir accès à `SALUT`. Modifions `src/salut.c` pour ajouter la ligne `{/* Anglais */` juste au-dessus du `printf` Anglais. Puisque nous ne pouvons plus accéder à `SALUT`, un commit échouera car la branche `AILLEURS/salut.commentaires` est liée à la branche `SALUT/trunk` et donc un commit sur la première doit d'abord réussir sur la seconde. Pour pouvoir quand même effectuer un commit, il va falloir tout d'abord défaire le lien qui unit la première branche à la seconde :

```
bzr unbind
```

A présent, nous pouvons faire un commit qui ne s'effectuera que dans le répo local :

```
bzr commit -m "commentaire␣de␣l'Anglais"
```

Exercice 23. Supposons à présent que nous ayons à nouveau une connection au répo `SALUT`. Nous pouvons reéétablir la liaison :

```
bzr bind $TD5/SALUT/trunk
```

Par contre nos commits locaux n'ont pas encore été propagés dans `SALUT/trunk`. Nous pouvons le faire explicitement de la manière suivante :

```
bzr push $TD5/SALUT/trunk
```

En allant dans `SALUT/trunk`, nous pouvons constater que les changements on été intégrés :

```
cd $TD5/SALUT/trunk
cat src/salut.c
bzr log
```

En principe, il ne faudrait pas travailler dans le checkout d'une branche maitresse (c'est à dire d'une branche à laquelle d'autres peuvent être liées). De cette manière, il ne peut pas y avoir de modifs locales qui pourraient causer des conflits avec nos `push`.