Exercice 1. Nous continuons aujourd'hui avec le projet que vous aviez commencé la fois dernière. Utilisez le logiciel Cervisia (s'il est installé) de KDE pour explorer votre projet : cervisia donne un interface graphique plus conviviale pour interagir avec CVS. Si vous éditez un fichier avec Kate, vous trouverez également l'accès aux fonctionalité de Cervisia dans le menu "Outils".

Exercice 2. Le support pour CVS est également bien intégré dans Emacs. Vous trouverez certaine de ces fonctionalités dans le menu "`Tools > Version Control`". Pour faire un diff avec la version la plus récente du repo : "`C-x v =`". Pour faire un commit : "`C-x C-q`", ceci ouvrira un buffer pour vous permettre d'éditer le message de log ; finissez l'édition par "`C-c C-c`" et le commit s'effectuera alors. Pour obtenir un historique des commits affectant le fichier : "`C-x v l`".

Exercice 3. Maintenant, voici un exercice pour illustrer quelque chose dont il faut se méfier. Choisissez un fichier de votre projet contenant pas mal de lignes consécutives : un membre A de votre groupe va modifier sa copie en reindentant ces lignes (e.g. en ajoutant 4 espaces au début de chacune de ces lignes) ; un autre membre B de votre groupe devra éditer sa copie de la manière suivante : il choisit juste une ligne dans laquelle il fait une petite modif.

Maintenant A fait un commit de sa modif. Si B tente de faire un commit, CVS lui indiquera que sa copie n'est pas à jour. S'il effectue un update : sa copie du fichier contiendra à présent un énorme conflit qu'il sera bien difficile de résoudre à la main. Le problème, en effet, est que, dans la partie affectée, les deux versions n'ont plus rien en commun car toutes les lignes diffèrent. Il n'y a donc plus de possibilité de fusion intelligente s'appuyant sur la reconnaissance de contexte commun puisque justement il n'y a plus de contexte commun : tout a changé.

Avec juste quelques lignes impliquées, on peut, par inspection, fusionner les modifications apportées par A et B, mais c'est déjà beaucoup de travail. Imaginez la cata, s'il s'agissait de centaines de lignes.

La première leçon à tirer de cet exercice, c'est qu'il ne faut jamais effectuer de reformattage sans concertation préalable. Si un membre du projet veut effectuer un reformattage dans un fichier, il faut qu'il se concerte avec tous les autres membre du projet pour s'assurer, qu'au moins temporairement, il est le seul à effectuer des modifications sur ce fichier. Une fois celles-ci finies et commitées, il informe les autres membres pour qu'ils fassent un update avant de procéder à des modifications sur le fichier en question. Ce conseil est à prendre encore plus à coeur lorsque le document contient du texte : on reformatte souvent un paragraphe sans réfléchir, et on s'en mord ensuite les doigts car les copies des divers membres du groupes ont alors trop divergé pour permettre une fusion raisonable.

Exercice 4. Maintenant, nous allons voir comment récuper une version précédente du projet. Cet exercice démontre également un manque dans CVS (qui heureusement est rectifié dans SVN que nous étudierons plus tard) : dans CVS, chaque fichier est versionné séparément, donc il n'y a pas de numéro de version global permettant d'identifier une version du projet. Une façon de récupérer une

version du projet, c'est de se baser sur une date. Plaçons nous dans le répertoire parent du répertoire de travail `salut` et puisque, à part les modifs effectuées ci-dessus, votre projet n'a (probablement) pas été changé depuis 1 semaine, nous allons demander la version d'il y a une semaine :[1]

```
cvs -d ~/1A-OMGL-VCS/TD-CVS get -D'1 week ago' \
          -d salut-il-y-a-1-semaine salut
```

Puisque nous ne sommes pas dans une copie de travail, il faut à nouveau indiquer explicitement à CVS le répo.

Exercice 5. Une autre façon de faire référence à une version antérieure, c'est de lui donner un nom : c'est ce qu'on appelle un tag.

```
cd salut-il-y-a-1-semaine
cvs tag salut-0
```

Exercice 6. On peut maintenant se servir de ce tag pour récupérer une copie de la version correspondante :

```
cd ..
cvs -d ~/1A-OMGL-VCS/TD-CVS get -r salut-0 \
          -d salut-vieux salut
```

Un usage des tags est pour marquer les révisions du projet correspondant à des livraisons.

Exercice 7. Reprenez votre projet (dans un état à peu près réparé). Nous allons à présent supposer qu'il y a un bug à corriger et pour cela nous allons créer un branche du projet sur laquelle nous allons corriger ce bug :

```
cd ../salut
cvs tag -b salut-bug-1
cd ..
cvs  -d ~/1A-OMGL-VCS/TD-CVS get -r salut-bug-1 \
           -d salut-bug-1 salut
```

[1] au lieu de `~/1A-OMGL-VCS/TD-CVS`, utilisez le chemin réel pour votre dépôt.