Pour ce TD, vous travaillerez dans le répertoire `~/1A-OMGL-VCS/TD1`. Définissons une variable bash `TD1` pour y faire référence :

```
export TD1=$HOME/1A-OMGL-VCS/TD1
```

il faut ensuite créer ce répertoire et s'y rendre :

```
mkdir -p $TD1
cd $TD1
```

Imaginons que vous développez un logiciel. Créez un répertoire `depot` dans lequel vous placez le premier fichier de votre projet : ce fichier nommé `salut.c` contient le code suivant :

```
int main()
{
}
```

Pour vous permettre de travailler vous obtenez à présent une copie `travail-moi` du `depot` que vous allez pouvoir modifier :

```
cp -a depot travail-moi
```

Exercice 1. Simple `diff`.

Le code ci-dessus est buggé car il devrait retourner un entier : le status d'exécution du programme (0 en cas de succès, et par exemple -1 en cas d'erreur). Effectuez maintenant la modification suivante dans le fichier `travail-moi/salut.c` :

```
int main()
{
  return 0;
}
```

Nous supposerons que votre répertoire courant est à présent `travail-moi`, c'est à dire votre répertoire de travail. Nous allons à présent étudier le delta entre `../depot/salut.c` et `salut.c`. Celui-ci peut être calculé de différentes manières grâce à `diff`. Etudiez et discutez les résultats des invocations suivantes :

- `diff ../depot/salut.c salut.c`
- `diff -c ../depot/salut.c salut.c`
- `diff -u ../depot/salut.c salut.c`

Vous trouverez la version PDF de la page de manuel en français pour `diff` dans le répertoire `/pub/1A/VCS`.

Exercice 2. Simple `patch`.

Retournons dans le répertoire parent `$TD1` (dont `depot` et `travail-moi` sont tous deux des sous-répertoires) et sauvons le delta dans un fichier :

```
cd $TD1
diff -u depot/salut.c travail-moi/salut.c > patch.1
```

Obtenons maintenant une nouvelle copie du `depot` que nous appellerons `travail-lui` :

```
cp -a depot travail-lui
```

Nous allons transférer dans `travail-lui` les modifications faites dans `travail-moi` et sauvegardées dans `patch.1`.

```
cd travail-lui
patch < ../patch.1
```

Nous pouvons vérifier que nous avons à présent dans `travail-lui` la même chose que dans `travail-moi` :

```
diff salut.c ../travail-moi/salut.c
```

Vous trouverez la version PDF de la page de manuel en français pour `patch` dans le répertoire `/pub/1A/VCS`.

Exercice 3. `patch` inversé.

Nous pouvons également appliquer un patch en sens inverse pour en annuler les effets :

```
patch -R < ../patch.1
```

On peut évidemment combiner les invocations de `diff` et `patch`. Par exemple, pour réappliquer les modifications effectuées dans `moi` :

```
diff -u ../depot/salut.c ../travail-moi/salut.c | patch
```

et pour les réannuler :

```
diff -u ../depot/salut.c ../travail-moi/salut.c | patch -R
```

Exercice 4. Patches en cascade.

Retournons à présent dans le répertoire `travail-moi`. Nous sauvegardons d'abord notre travail dans `depot` (une sorte de *commit*) :

```
cd $TD1/travail-moi
cp salut.c ../depot/
```

et modifions le programme de la manière suivante :

```c
#include <stdio.h>

int main()
{
  printf("salut\n");
  return 0;
}
```

Nous pouvons calculer un nouveau delta par rapport à la version précédente :

```
diff -u ../depot/salut.c salut.c
```

Procédons comme précedemment pour sauvegarder ce delta :

```
cd $TD1
diff -u depot/salut.c travail-moi/salut.c > patch.2
```

Puis retournons dans `travail-lui` et appliquons nos deux deltas l'un après l'autre :

```
cd travail-lui
patch < ../patch.1
patch < ../patch.2
```

Pour annuler ces effets il faut appliquer les deux deltas inversés mais dans l'ordre opposé :

```
patch -R < ../patch.2
patch -R < ../patch.1
```

Exercice 5. Diff récursif.

Retournons dans `travail-moi` et créons un sous répertoire `lib` dans lequel nous aurons le fichier `msg.c` contenant le code ci-dessous :

```c
#include <stdio.h>

void msg()
{
   printf("salut\n");
}
```

Allons maintenant dans le répertoire `travail-lui` pour y créer aussi un sous répertoire `lib` contenant le fichier `msg.c` avec le code ci-dessous :

```c
#include <stdio.h>

void msg()
{
   printf("bonjour\n");
}
```

Maintenant retournons dans le répertoire `$TD1` parent de `travail-moi` et `travail-lui`, et invoquons la commande suivante :

```
diff -u -r travail-moi travail-lui > patch.3
```

Le résultat contient 2 *hunks* : le premier concerne le fichier `lib/msg.c` et le second le fichier `salut.c`. Voici donc un delta impliquant plusieurs fichiers et répertoires dans un projet.

Exercice 6. Patch récursif.

Nous pouvons appliquer ce patch au répertoire `travail-moi` pour qu'il devienne exactement comme le répertoire `travail-lui`. Première tentative :

```
patch --dry-run < patch.3
```

`--dry-run` permet d'exécuter la commande sans réellement changer quoique ce soit : ceci permet de voir ce qui se passerait, mais en toute sécurité, sans rien changer. Dans le cas présent, `patch` nous demande quels fichiers modifier (*le responsable de TD doit expliquer*).

La solution est d'utiliser l'option `-p` (voir la page de manuel) :

```
patch -p0 < patch.3
```

Après quoi on peut vérifier que `travail-moi` est identique à `travail-lui` :

```
diff -r travail-moi travail-lui
```

On peut également annuler l'application du patch de la manière suivante :

```
patch -p0 -R < patch.3
```

Exercice 7. Diff récursif et fichiers nouveaux/manquants.

Nous somme dans le répertoire `$TD1`. Créons deux nouveaux fichiers :

```
echo untel > travail-moi/qui
echo la > travail-lui/ou
```

Considérez (et expliquez) la différence entre :

- `diff -ur  travail-moi travail-lui`
- `diff -urN travail-moi travail-lui`

Exercice 8. Fusion simple.

Il faudrait illustrer si possible l'usage des applications `kdiff3` (excellent), `kompare` (très joli, mais ne supporte la fonctionalité de merge), `meld`, `vimdiff`, `diff3`, `sdiff`, `ediff` (dans GNU Emacs).

Nous sommes toujours dans le répertoire `$TD1`. Faisons en sorte que `depot` et `travail-lui` soient des copies de `travail-moi` :

```
rm -rf depot travail-lui
cp -a travail-moi depot
cp -a travail-moi travail-lui
```

Nous pouvons maintenant faire comme si `travail-moi` et `travail-lui` avaient obtenu une copie du même `depot`. Supposons que `travail-moi` modifie le fichier `salut.c` de la manière suivante :

```
#include <stdio.h>

/*
 * un commentaire inutile
 */

int main()
{
  printf("salut\n");
  return 0;
}
```

et que de son coté `travail-lui` ajoute un `fflush` tout aussi inutile :

```
#include <stdio.h>

int main()
{
  printf("salut\n");
  fflush(stdout);
  return 0;
}
```

On constate que :

```
diff -u travail-moi/salut.c travail-lui/salut.c
```

ne nous permet pas de fusionner les modifications de `travail-moi` et `travail-lui` (discuter pourquoi). Pour connaitre les modifications effectuées par chacun, il faut comparer à la version originale disponible dans `depot` :

```
diff -u depot/salut.c travail-moi/salut.c > patch.moi
diff -u depot/salut.c travail-lui/salut.c > patch.lui
```

on pourrait alors appliquer `patch.moi` à `travail-lui` et `patch.lui` à `travail-moi`. C'est bien compliqué ! il existe un utilitaire, `diff3`, qui aide à effectuer cette fusion :

```
diff3 -m travail-moi/salut.c depot/salut.c travail-lui/salut.c
```

Dans le cas présent, les modifications de chacun sont bien séparées et `diff3` est capable de les fusionner directement (un antislash apparaissant comme dernier caractère d'une ligne indique à bash que la ligne n'est pas vraiment terminée : elle continue en fait sur la ligne suivante. J'utilise cette notation ici parce que la commande est trop longue pour tenir sur une seule ligne de cette page) :

```
diff3 -m travail-moi/salut.c \
         depot/salut.c \
         travail-lui/salut.c \
    > salut.c
```

```
mv salut.c depot/
cp depot/salut.c travail-moi/
cp depot/salut.c travail-lui/
```

Attention de ne pas rediriger directement dans `depot/salut.c` car le fichier serait alors simultanément lu et écrit (danger !).

Exercice 9. Fusion avec conflits.

Supposons que `travail-moi` modifie `salut.c` de la manière suivante (il enlève 2 lignes de code) :

```
#include <stdio.h>

/*
 * un commentaire inutile
 */

int main()
{
  return 0;
}
```

et que `travail-lui` ajoute 2 lignes de code :

```
#include <stdio.h>

/*
 * un commentaire inutile
 */

int main()
{
  printf("salut\n");
  fflush(stdout);
  printf("bye\n");
  fflush(stdout);
  return 0;
}
```

Expliquer pourquoi le transfert de modif ci-dessous de `travail-lui` vers `travail-moi` échoue :

```
cd $TD1/travail-moi
diff -u ../depot/salut.c ../travail-lui/salut.c | patch --dry-run
```

Voyons à présent ce que `diff3` donne :

```
cd $TD1
diff3 -m travail-moi/salut.c depot/salut.c travail-lui/salut.c
```

(discuter le résultat). Si `kdiff3` est disponible essayez :

```
kdiff3 -m depot/salut.c travail-moi/salut.c travail-lui/salut.c \
    --auto -o $TD1/nouveau-salut.c
```

`kdiff3` est parfois capable de résoudre les conflits automatiquement de manière heuristique : il est clair ici, en comparant à la version originale dans `depot`, que `travail-moi` a effacé 2 lignes de code et que `travail-lui` a ajouté 2 lignes de code après celles effacées par `travail-moi`. Les 2 modifications sont donc indépendantes et dans ce cas on peut donc faire les deux. C'est ce que `kdiff3` choisit de faire. On peut également invoquer `kdiff3` de sorte qu'il nous donne une interface graphique dans laquelle on peut explicitement résoudre les conflits à la main :

```
kdiff3 -m depot/salut.c travail-moi/salut.c travail-lui/salut.c \
    -o $TD1/nouveau-salut.c
```