# Restructurer un fichier de texte

Une des tâches les plus communes et les plus ennuyeuses, pour un administrateur de système, c'est de devoir modifier un fichier de texte pour l'utiliser avec une autre structure, surtout si ce fichier contient des milliers de lignes. Le programme qui fait cela s'appelle une moulinette.

Par exemple, on a un fichier d'utilisateurs avec des virgules pour séparateur et l'ordre Prénom, Nom, Rue, Ville, et il faut en faire un fichier avec des tabulations pour séparateur et l'ordre Nom, Prénom, Rue, Ville.

Pour créer la moulinette qui automatise cette opération, on peut utiliser un langage « normal » comme C/C++ ou Java, un langage de scripting comme Ruby ou Python, un outil Unix comme awk, etc.

La solution diffère radicalement selon le moyen choisi : avec certains d'entre eux, une seule ligne suffit. Avec d'autres, il faut une page de code.

## Fichier d'origine

```
Viviane,Dupont,20 rue de l'Ouest,Lyon
Jean-Pierre,Durand,12 rue Chevrolet,Paris
Emilie,Martin,4 rue de la Cure,Lille
```

## Fichier à obtenir

```
Dupont    Viviane    20 rue de l'Ouest    Lyon
Durand    Daniel     12 rue Chevrolet     Paris
Martin    Emilie     4 rue de la Cure     Lille
```

## Solution avec awk

```
awk -F, '{ print $2 "\t" $1 "\t" $3 "\t" $4 }' file_src > file_dst
```

L'option ***F,*** (lettre *F* majuscule et virgule) indique que le séparateur de champ (*field*) est la virgule. Le fichier source est *file_src*, le fichier cible *file_dst*.

Merci à Guillaume pour cette solution.

## Ruby

```
ruby -F, -a -i -ne 'puts $F,values_at(1,0,2,3).join("\t")' file_src
```

L'option ***F*** indique que le séparateur de champ (*field*) est la virgule et l'option ***a*** qu'on travaille avec un tableau (*array*). L'option ***i*** spécifie qu'on travaille en place (*in place*) : le fichier cible est le fichier source.

## Python

```
def moulinette(source, destination):
   file_src = open(source, 'r')
   file_dst = open(destination, 'w')
   for ligne in file_src:
       data = ligne.rstrip('\n\r').split(",")
       file_dst.write("%s\t%s\t%s\t%s\n" % data[1],data[0],data[2],data[3]) )
   file_src.close()
   file_dst.close()
```

L'indentation est obligatoire en Python car elle sert à délimiter les blocs. Cela permet de rendre le code à la fois plus file_srccompact et plus lisible. Le fichier source est *file_src*, le fichier cible *file_dst*.

Merci à Adrian pour cette solution.

## Java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.channels.FileChannel;

public class Application {
  /**
  * @param args
  * @throws IOException
  */
  public static void main(String[] args) throws IOException {
    // TODO Auto-generated method stub

    FileWriter writer = null;
```

```
try{
  writer = new FileWriter("file_src", true);

  String filePath = "d:\\file_src";
  try{
    BufferedReader buff = new BufferedReader(new FileReader(filePath));
    try {
      String line;
      while ((line = buff.readLine()) != null) {
        writer.write(moulinette(line)+ "\r\n",0,moulinette(line).length()+1);
      }
    } finally {
      buff.close();
    }
  } catch (IOException ioe) {
    System.out.println("Erreur --" + ioe.toString());
  }
}catch(IOException ex){
  ex.printStackTrace();
} finally {
  if(writer != null){
    writer.close();
  }
}

FileChannel in = null; // canal d'entrÈe
FileChannel out = null; // canal de sortie
try {
  // Init
  in = new FileInputStream("file_src").getChannel();
  out = new FileOutputStream("d:\\file_src").getChannel();
  in.transferTo(0, in.size(), out);
} catch (Exception e) {
  e.printStackTrace(); // n'importe quelle exception
} finally { // finalement on ferme
  if(in != null) {
    try {
      in.close();
    } catch (IOException e) {}
  }
  if(out != null) {
    try {
      out.close();
    } catch (IOException e) {}
```

```
      }
    }
    File MyFile = new File("C:\\users\\type\\workspace\\Moulinette\\file_src");
    MyFile.delete();
  }

  public static String moulinette(String phrase)
  {
    String tb[] = {"a","b","c","d"};
    tb = phrase.split(",");
    phrase = tb[1] + "\t"+ tb[0]+ "\t"+  tb[2]+ "\t"+ tb[3];
    return phrase;
  }
  static public boolean deleteDirectory(File path) {
    boolean resultat = true;

    if( path.exists() ) {
      File[] files = path.listFiles();
      for(int i=0; i<files.length; i++) {
        if(files[i].isDirectory()) {
          resultat &= deleteDirectory(files[i]);
        }
        else {
          resultat &= files[i].delete();
        }
      }
    }
    resultat &= path.delete();
    return( resultat );
  }
}
```

Merci à Johan pour cette solution.