Les filtres et utilitaires

Un **filtre** (ou une commande filtre) est un programme sachant écrire et lire des données par les canaux standards d'entrée et de sortie. Il en modifie ou traite éventuellement le contenu. **wc** est un filtre. Les utilitaires sans être obligatoirement des filtres permettent un certain nombre d'actions sur des fichiers ou leur contenu comme le formatage ou l'impression.

1. Extraction des noms et chemins

La commande **basename** permet d'extraire le nom du fichier dans un chemin.

```
$ basename /tmp/seb/liste
liste
```

La commande **dirname** effectue l'inverse, elle extrait le chemin.

```
$ dirname /tmp/seb/liste
/tmp/seb
```

2. Recherche de lignes

Il s'agit d'extraire des lignes d'un fichier selon divers critères. Pour cela vous disposez de trois commandes **grep**, **egrep** et **fgrep** qui lisent les données soit depuis un fichier d'entrée, soit depuis le canal d'entrée standard.

a. grep

La syntaxe de la commande **grep** est `grep [Options] modèle [Fichier1...]`.

Le modèle se compose de critères de recherche ressemblant beaucoup aux critères déjà exposés pour vi par exemple. Il ne faut pas oublier que ces critères doivent être interprétés par la commande **grep** et pas par le shell. Il faut donc verrouiller tous les caractères.

```
$ cat fic4
Cochon
Veau
Boeuf
rat
Rat
boeuf
$ grep "^[bB]" fic4
Boeuf
boeuf
```

La commande **grep** peut aussi prendre quelques options intéressantes.

- **-v** effectue la recherche inverse : toutes les lignes ne correspondant pas aux critères sont affichées.
- **-c** ne retourne que le nombre de lignes trouvées sans les afficher.
- **-i** ne différencie pas les majuscules et les minuscules.
- **-n** indique le numéro de ligne pour chaque ligne trouvée.
- **-l** dans le cas de fichiers multiples, indique dans quel fichier la ligne a été trouvée.

```
$ grep -i "^b" fic4
Boeuf
boeuf
```

b. egrep

La commande **egrep** étend les critères de recherche et peut accepter un fichier de critères en entrée. Elle est équivalente à un grep -E. Elle emploie comme critères des expressions régulières.

```
egrep -f fichier_critère Fichier_recherche
```

Caractère spécial	Signification
|	Ou logique, l'expression située avant ou après doit apparaître.
(...)	Groupement de caractères.
[...]	Un caractère à cette position parmi ceux indiqués.
. (point)	Un caractère quelconque.
+	Répétition, le caractère placé avant doit apparaître au moins une fois.
*	Répétition, le caractère situé avant doit apparaître de zéro à n fois.
?	Le caractère situé avant doit apparaître une fois au plus.
{n}	Le caractère situé avant doit apparaître exactement n fois.
{n,}	Il apparaît n fois ou plus.
{n,m}	Il apparaît entre n et m fois.
^	En début de chaîne.
$	En fin de chaîne.

Seulement bonjour et bonsoir commençant par une majuscule ou une minuscule s'ils sont seuls sur une ligne :

```
^[bB]on(jour|soir)$
```

Vérification très sommaire de la validité d'une adresse IP :

```
echo $IP | egrep '([0-9]{1,3}\.){3}[0-9]{1,3}')
```

Voici comment cette ligne est décomposée :

- ([0-9]{1,3}\.){3} :www.xxx.yyy.
- [0-9] : un caractère entre 0 et 9
- {1,3} : répété entre une et trois fois, donc x, xx ou xxx
- \. : suivi d'un point
- {3} : le tout trois fois

Puis [0-9]{1,3} : .zzz

- [0-9] : un caractère entre 0 et 9
- {1,3} : répété entre une et trois fois

c. fgrep

La commande **fgrep** est un grep simplifié et rapide (fast grep) et équivaut à un grep -F . Elle accepte aussi un fichier de critères de recherche mais il s'agit là de critères simples, sans caractères spéciaux. Vous saisissez dans le fichier de critères des lignes simples (du texte et des chiffres), une recherche par ligne. Fgrep va alors rechercher dans un fichier cible ou un flux en entrée les lignes correspondant à chacun des critères.

3. sed

L'apprentissage de sed demanderait tout un livre. Sed est un éditeur de flux (Stream Editor) permettant de filtrer et de transformer du texte. C'est un peu comme un éditeur permettant de modifier du texte via des commandes scripts, mais en une passe et sans édition interactive. Il utilise un jeu étendu de commandes issu de l'éditeur ed. Sa syntaxe de base est :

```
sed -e '<cmd>' fic
```

Pour utiliser sed, il faut apprendre et comprendre les expressions rationnelles. Le tableau de la commande **egrep** reprend la syntaxe de base des expressions. Tout ouvrage sur sed part de ces expressions, et réciproquement.

Sed est très souvent utilisé pour remplacer des valeurs par d'autres (substitution) ou supprimer des lignes particulières (bien que grep pourrait être utilisé dans ce cas). La syntaxe basique de substitution est la suivante :

```
s/<ancien>/nouveau/[g]
```

Le `g` final permet de faire un remplacement sur toute la ligne en cas de présence de plusieurs occurrences. Voici un exemple qui remplace `__NOM__ par Toto` :

```
$ echo "Je m'appelle __NOM__. Tu t'appelles __NOM__?" | sed -e 's/__
NOM__/Toto/'
Je m'appelle Toto. Tu t'appelles __NOM ?
$ echo "Je m'appelle __NOM__. Tu t'appelles __NOM__ ?" | sed -e 's/__
NOM__/Toto/g'
Je m'appelle Toto. Tu t'appelles Toto ?
```

Vous pouvez placer une valeur numérique dans le champ nouveau pour préciser, si la recherche comporte plusieurs éléments regroupés par des parenthèses, sur quel élément recherché travailler. Voici un simple exemple qui rajoute des étoiles autour du nom toto :

```
$ echo toto | sed -e "s/\(toto\)/**\1**/"
**toto**
```

Pour supprimer toutes les lignes vides ou ne contenant que des espaces :

```
$ sed -e '/^ *$/d' fichier
```

4. Colonnes et champs

La commande **cut** permet de sélectionner des colonnes et des champs dans un fichier.

a. Colonnes

La syntaxe est la suivante :

```
cut -cColonnes [fic1...]
```

Une colonne est la position d'un caractère dans la ligne. Le premier caractère est la colonne 1, le deuxième la colonne 2, et ainsi de suite. Une ligne de 80 caractères dispose de 80 colonnes. La numérotation commence à 1. C'est la méthode idéale pour des fichiers plats et à format fixe où chaque champ débute et finit à des positions données.

Le format de sélection de colonne est le suivant :

- une colonne seule (ex. -c2 pour la colonne 2) ;
- une plage (ex. -c2-4 pour les colonnes 2, 3 et 4) ;
- une liste de colonnes (ex. -c1,3,6 pour les colonnes 1, 3 et 6) ;
- les trois en même temps (ex. -c1-3,5,6,12-).

```
$ cat liste
Produit prix    quantites
souris  30      15
disque  100     30
ecran   300     20
clavier 45      30

$ cut -c1-5 liste
Produ
souri
disqu
ecran
clavi

$ cut -c1-3,10-12,15
Prorx   quantites
sou0    15
dis0    30
ecr0    20
cla530
```

b. Champs

La commande **cut** permet aussi de sélectionner des champs. Ces champs doivent être par défaut délimités par une tabulation, mais le paramètre `-d` permet de sélectionner un autre caractère (espace, ;). La sélection des champs est identique à celle des colonnes.

Le caractère séparateur doit être unique. Il n'est pas possible d'en mettre deux ou trois, ou une chaîne de séparateurs. Pour éliminer les caractères multiples, utilisez tr. De même le séparateur par défaut est la tabulation. Or par défaut les tabulations sont souvent remplacées par des espaces au sein des éditeurs... `cut -dc -fChamps [fic1...]`

Voici quelques exemples. Le fichier liste contient des champs séparés par des tabulations.

```
$ cat liste
Produit prix    quantites
souris  30      15
dur     100     30
disque  100     30
ecran   300     20
clavier 45      30
carte   45      30

$ cut -f1 liste
Produit
souris
dur
disque
ecran
clavier
carte

$ cut -f1,3 liste
Produit quantites
souris  15
dur     30
```

```
disque  30
ecran   20
clavier 30
carte   30
```

Notez que si vous inversez l'ordre des champs (-f3,1) vous n'obtenez pas l'effet escompté : les champs sortent toujours dans le sens 1,3.

Voici comment isoler les noms d'un groupe et leurs identifiants respectifs :

```
$ cat /etc/group
seb@slyserver:~> cat /etc/group
at:!:25:
audio:x:17:
avahi:!:106:
beagleindex:!:107:
bin:x:1:daemon
cdrom:x:20:
console:x:21:
daemon:x:2:
dialout:x:16:seb,steph,henri,public
disk:x:6:

$ cut -d: -f1,3 /etc/group
at:25
audio:17
avahi:106
beagleindex:107
bin:1
cdrom:20
console:21
daemon:2
dialout:16
disk:6
```

S'il n'y a pas de délimiteur (tabulation ou autre) dans une ligne, cut affiche toute la ligne.

5. Décompte de lignes

La commande **wc** (*word count*) permet de compter les lignes, les mots et les caractères.

`wc [-l] [-c] [-w] [-w] fic1`

- `-l` : compte le nombre de lignes
- `-c` : compte le nombre d'octets
- `-w` : compte le nombre de mots
- `-m` : compte le nombre de caractères

```
$ wc liste
       12        48       234 liste
```

Le fichier liste contient 12 lignes, 48 mots et 234 caractères.

6. Tri de lignes

La commande **sort** permet de trier des lignes. Par défaut le tri s'effectue sur tout le tableau et en ordre croissant. Le

tri est possible sur un ou plusieurs champs. Le séparateur de champs par défaut est la tabulation ou au moins un espace. S'il y a plusieurs espaces, le premier est le séparateur, les autres des caractères du champ.

La syntaxe de sort a évolué depuis quelques années et Linux s'est mis en conformité. Aussi l'ancienne syntaxe basée sur +/- n'est plus utilisée. À la place, il faut utiliser le paramètre `-k`. La numérotation des champs commence à 1.

```
sort [options] [-k pos1[,pos2]] [fic1...]

$ cat liste
souris  optique 30      15
dur     30giga  100     30
dur     70giga  150     30
disque  zip     12      30
disque  souple  10      30
ecran   15      150     20
ecran   17      300     20
ecran   19      500     20
clavier 105     45      30
clavier 115     55      30
carte   son     45      30
carte    video   145     30
```

Voici comment trier par ordre alphabétique sur la première colonne :

```
$ sort -k 1 liste
carte   son     45      30
carte   video   145     30
clavier 105     45      30
clavier 115     55      30
disque  souple  10      30
disque  zip     12      30
dur     30giga  100     30
dur     70giga  150     30
ecran   15      150     20
ecran   17      300     20
ecran   19      500     20
souris  optique 30      15
```

Quelques paramètres

Option	Rôle
-d	Dictionnary sort (tri dictionnaire). Ne prend comme critère de tri que les lettres les chiffres et les espaces.
-n	Tri numérique, idéal pour les colonnes de chiffres.
-b	Ignore les espaces en début de champ.
-f	Pas de différences entre majuscules et minuscules (conversion en minuscules puis tri).
-r	Reverse, tri en ordre décroissant.
-tc	Nouveau délimiteur de champ c.

Exemple, tri numérique sur le prix par produits en ordre décroissant :

```
$ sort -n -r -k 3 liste
ecran   19      500     20
ecran   17      300     20
ecran   15      150     20
dur     70giga  150     30
carte   video   145     30
dur     30giga  100     30
clavier 115     55      30
clavier 105     45      30
```

```
carte   son     45      30
souris  optique 30      15
disque  zip     12      30
disque  souple  10      30
```

Il est aussi possible de démarrer le tri à partir d'un certain caractère d'un champ. Pour cela vous devez spécifier le « .pos » : -k1.3 commencera le tri à partir du troisième caractère du champ 1.

7. Suppression des doublons

La commande **uniq** permet de supprimer les doublons dans des flux en entrée ou des fichiers triés. Par exemple, voici comment sortir uniquement la liste des GID réellement utilisés comme groupe principal des utilisateurs :

```
$ cut -d: -f4 /etc/passwd | sort -n | uniq
0
1
2
7
8
12
13
14
25
49
51
62
...
```

8. Jointure de deux fichiers

a. Sur des champs communs

La commande **join** permet d'effectuer une jointure de deux fichiers en fonction d'un champ commun. Les deux fichiers doivent être triés sur les champs spécifiés pour la jointure.

```
join [-tc] [-1 n] [-2 m] fic1 fic2
```

L'option `-t` indique le séparateur, `-1` le champ du premier fichier et `-2` le champ du second fichier sur lesquels effectuer la jointure. Notez que `join` gère mal les doublons et risque de s'arrêter dans ce cas.

La commande **join** risque de ne pas vous fournir le résultat attendu. C'est que dès qu'elle ne trouve pas une correspondance entre deux lignes, elle s'arrête.

b. Ligne à ligne

La commande **paste** regroupe n fichiers en un. Pour cela elle concatène les lignes de chacun des fichiers en une seule ligne : ligne1 de fic1 avec ligne2 de fic2, ligne3 de fic 3, et ainsi de suite. C'est un peu l'inverse du cut. Le séparateur par défaut est la tabulation mais vous pouvez préciser un délimiteur avec `-d`.

```
$ cat fic1
liste_a
liste_b
liste_c

$ cat fic2
liste_a2
liste_b2
liste_c2

$ paste -d: fic1 fic2
liste_a:liste_a2
```

```
liste_b:liste_b2
liste_c:liste_c2
```

9. Découpage d'un fichier en morceaux

a. Découper

Voici une commande fort pratique, **split**, qui permet de découper un gros fichier en plusieurs morceaux d'une taille donnée. Les systèmes de fichiers ne sont pas tous égaux devant la taille maximale d'un fichier. Sous Linux le problème se pose peu, un système de fichiers de type ext3 supportant de fichiers de 1TO (TB = TeraByte = TeraOctet = 1024 Go) soit l'équivalent de 130 DVDs double couche environ. Mais les bandes magnétiques, ou dans une plus faible mesure les disques amovibles, n'ont pas tous cette possibilité.

Une clé USB ou un disque externe sont généralement « formatés » avec un système de fichiers de type VFAT issu du monde Microsoft. Ce système de fichiers provenant de DOS puis Windows 9x garantit une compatibilité entre tous les systèmes (Unix, Windows, MacOS), qui peut le plus peut le moins. VFAT (ou plutôt FAT16 ou FAT32) ne supporte que des fichiers d'une taille maximum de 4 Go. Une image ISO de DVD ou une archive de sauvegarde ne peut y rentrer d'un seul bloc. Il faut donc découper le fichier en plusieurs morceaux.

```
split [-l n] [-b n[bkm] [fichier [préfixe]]
```

La commande peut fonctionner selon deux modes :

- découpage par lignes avec `-l` : les fichiers en sortie auront tous n lignes de texte (sauf éventuellement le dernier) ;
- découpage à taille fixe avec `-b` : les fichiers auront tous une taille fixe de n octets. Le suffixe b indique une taille de n blocs (512 octets), k indique n ko (1024 octets) et m indique n Mo (1024 ko).

Comme tout filtre split peut prendre un flux en entrée, ce qui est le cas si aucun fichier n'est précisé, ou si un tiret est présent. Un préfixe définit le nom des fichiers en sortie. Voici un fichier de 1 Go à découper en tranches de 150 Mo. Le préfixe est fic. Chaque fichier en sortie s'appelle ficaa, ficab, ficac, ficad, et ainsi de suite.

```
$ ls -l grosfichier
-rw-r--r-- 1 seb users 1073741824 mar 12 19:47 grosfichier
$ split -b 150m grosfichier fic
$ ls -l fic*
-rw-r--r-- 1 seb users 157286400 mar 12 20:15 ficaa
-rw-r--r-- 1 seb users 157286400 mar 12 20:15 ficab
-rw-r--r-- 1 seb users 157286400 mar 12 20:15 ficac
-rw-r--r-- 1 seb users 157286400 mar 12 20:16 ficad
-rw-r--r-- 1 seb users 157286400 mar 12 20:16 ficae
-rw-r--r-- 1 seb users 157286400 mar 12 20:16 ficaf
-rw-r--r-- 1 seb users 130023424 mar 12 20:16 ficag
```

b. Reconstruire

Une ligne suffit pour reconstruire un fichier splité à l'aide des redirections :

```
$ cat fic* > newfic
$ ls -l newfic
-rw-r--r-- 1 seb users 1073741824 mar 12 20:47 newfic
```

10. Remplacement de caractères

a. Liste de caractères

La commande **tr** permet de substituer des caractères à d'autres et n'accepte que des données provenant du canal d'entrée standard, pas les fichiers.

```
tr [options] original destination
```

L'original et la destination représentent un ou plusieurs caractères. Les caractères originaux sont remplacés par les caractères de destination dans l'ordre indiqué. Les crochets permettent de définir des plages.

Par exemple, remplacer le o par le e et le i par le a.

```
$ cat liste | tr "oi" "ea"
Preduat ebjet   prax    quantates
seuras  eptaque 30      15
dur     30gaga  100     30
dur     70gaga  150     30
dasque  zap     12      30
dasque  seuple  10      30
ecran   15      150     20
ecran   17      300     20
ecran   19      500     20
clavaer 105     45      30
clavaer 115     55      30
carte   sen     45      30
carte   vadee   145     30
```

Avec cette commande vous pouvez convertir une chaîne en majuscules ou en minuscules.

```
$ cat liste | tr "[a-z]" "[A-Z]"
PRODUIT OBJET   PRIX    QUANTITES
SOURIS  OPTIQUE 30      15
DUR     30GIGA  100     30
DUR     70GIGA  150     30
DISQUE  ZIP     12      30
DISQUE  SOUPLE  10      30
ECRAN   15      150     20
ECRAN   17      300     20
ECRAN   19      500     20
CLAVIER 105     45      30
CLAVIER 115     55      30
CARTE   SON     45      30
CARTE   VIDEO   145     30
```

Supprimer les répétitions

Surtout, `tr` admet deux paramètres, `-s` (squeeze) et `-d` (delete), qui permettent de supprimer des caractères en doublons ou non. C'est parfait dans le cas de séparateurs multiples. Voici un exemple pratique où l'on cherche à isoler l'adresse IP d'une machine.

```
$ /sbin/ifconfig eth0
eth0      Lien encap:Ethernet  HWaddr 00:13:D3:D7:A4:6C
          inet adr:10.9.238.170  Bcast:10.9.239.255  asque:255.255.252.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:15054381 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4991811 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:4157389034 (3964.7 Mb)  TX bytes:374974072 (357.6 Mb)
          Interruption:22 Adresse de base:0xcc00
```

Seule la deuxième ligne, contenant `inet`, vous intéresse :

```
$ /sbin/ifconfig eth0| grep "inet "
          inet adr:10.9.238.170  Bcast:10.9.239.255  Masque:255.255.252.0
```

Pour isoler l'adresse IP située après « `inet adr:` » le séparateur « : » semble intéressant mais dans ce cas un cut retournerait «`10.9.238.170 Bcast` » ce qui ne convient pas. L'astuce consiste à remplacer tous les espaces par un seul « : ». Le paramètre `-s` remplace une chaîne de n caractères identiques par un seul. S'il n'est pas précisé c'est le même caractère, sinon un caractère de substitution donné.

```
$ /sbin/ifconfig eth0| grep "inet " | tr -s " " ":"
:inet:adr:10.9.238.170:Bcast:10.9.239.255:Masque:255.255.252.0
```

Il n'y a plus qu'à compter : l'adresse IP est en quatrième position (le premier champ avant le premier « : » est vide).

```
$ /sbin/ifconfig eth0| grep "inet " | tr -s " " ":" | cut -d: -f4
10.9.238.170
```

b. Tabulations et espaces

La plupart des éditeurs remplacent les tabulations par des espaces. Or certaines commandes s'attendent à obtenir des tabulations comme délimiteurs de champs (comme cut). S'il est possible de s'en sortir avec tr, deux commandes sont à votre disposition pour ce cas spécifique.

La commande **expand** convertit les tabulations en espaces. La commande **unexpand** convertit les espaces en tabulations. Soit le fichier liste selon le modèle prédécent où les colonnes sont séparées par des espaces au lieu de tabulations. Dans le premier cas le résultat n'est pas du tout celui attendu. La commande **cut** tente de sortir le troisième champ d'un fichier tabulé. Comme il n'y a pas de tabulations, il affiche toute la ligne.

```
$ cut -f1 liste
Produit     objet       prix    quantites
souris      optique     30      15
dur         30giga      100     30
dur         70giga      150     30
disque      zip         12      30
disque      souple      10      30
...
```

La commande **unexpand** avec le paramètre `-a` remplace toutes les séquences d'au moins deux espaces par le nombre nécessaire de tabulations. Cette fois le résultat est correct.

```
$ unexpand -a liste | cut -f1
Produit
souris
dur
dur
disque
disque
...
```

11. Visualisation de texte

a. En pleine page

Rien n'empêche de détourner un quelconque flux pour l'afficher sur l'écran ou l'imprimante. Voici quelques commandes.

- page par page : **pg**, **more**, **less**
- en bloc : **cat**
- à l'envers : **tac**
- en dump hexadécimal : **hexdump**
- création d'une bannière : **banner**
- formatage pour impression : **pr**
- numéroter les lignes : **cat -n** ou **nl**

b. Début d'un fichier

Pour voir le début d'un fichier utilisez la commande **head**.

```
head [-c nbcar] [-n nblignes] [fic1...]
```

Le paramètre `-c` permet de préciser un nombre d'octets d'en-tête à afficher. Par défaut dix lignes sont affichées. Le paramètre `-n` permet d'indiquer le nombre de lignes à afficher. Vous pouvez indiquer directement le nombre de lignes :

```
head [-nblignes] [fic1...]

$ head -3 liste
Produit objet   prix    quantites
souris  optique 30      15
dur     30giga  100     30
```

c. Fin et attente de fichier

Pour voir les dernières lignes d'un fichier, utilisez la commande **tail**.

```
tail [+/-valeur[b/c]] [-f] [fic1...]
```

Comme pour head, par défaut les dix dernières lignes sont affichées. La valeur `-nblignes` permet de modifier cet état. Précisez c pour indiquer un nombre de caractères. Un b indique un nombre de blocs (512 octets par bloc).

Un `+` inverse l'ordre de la commande, et devient un `head (tail +10 <=> head -n 10)`.

Enfin l'option `-f` laisse le fichier ouvert. Ainsi si le fichier continue d'être rempli (par exemple un fichier trace), son contenu s'affichera en continu sur l'écran jusqu'à interruption volontaire par l'utilisateur ([Ctrl] **C**).

```
$ tail -5 liste
ecran   19      500     20
clavier 105     45      30
clavier 115     55      30
carte   son     45      30
carte   video   145     30

$ tail -10c liste
eo      145     30
```

12. Duplication du canal de sortie standard

Dans certains cas, comme par exemple la génération de fichiers traces, il peut être nécessaire de devoir à la fois placer dans un fichier le résultat d'une commande et de filtrer ce même résultat avec une autre commande. Utilisez pour cela la commande **tee** qui permet de dupliquer le flux de données. Elle lit le flux de données provenant d'une autre commande par le canal d'entrée, l'écrit dans un fichier et restitue ce flux à l'identique par le canal de sortie. Par défaut le fichier généré écrase l'ancien s'il existe.

```
tee [-a] nom_fic
```

Le paramètre `-a` signifie append. Dans ce cas le fichier n'est pas écrasé mais complété à la fin. Par exemple, vous voulez obtenir à la fois dans un fichier la liste des noms d'utilisateurs et afficher leur nombre sur écran.

```
$ cat /etc/passwd | cut -d: -f1 | tee users | wc -l
       65
$ cat users
root
nobody
nobodyV
daemon
bin
uucp
uucpa
auth
cron
lp
tcb
```

```
...
```

13. Comparaison de fichiers

Les deux commandes permettant de comparer le contenu de deux fichiers, ou d'un fichier et d'un flux sont les commandes **diff** et **cmp**.

a. diff

La commande **diff** indique les modifications à apporter aux deux fichiers en entrée pour que leur contenu soit identique.

```
diff [-b] [-e] fic1 fic2
```

L'option `-b` permet d'ignorer les espaces (blank), et l'option `-e` permet de générer un script ed (nous ne l'utiliserons pas). Cette commande renvoie trois types de messages :

- `APPEND : ligne1 a ligne3,ligne4, ex 5 a 6,8` veut dire : à la ligne 5 de fic1 il faut raccrocher les lignes 6 à 8 de fic2 pour que leurs contenus soient identiques.
- `DELETE : ligne1,ligne2 d ligne3, ex 7,9 d 6` veut dire : les lignes 7 à 9 de fic1 doivent être supprimées, elles n'existent pas derrière la ligne 6 de fic2.
- `CHANGE : ligne1,ligne2 c ligne3,ligne4, ex 8,12 c 9,13` veut dire : les lignes 8 à 12 de fic1 doivent être échangées contre les lignes 9 à 13 de fic2.

Dans tous les cas, le signe "<" indique les lignes de fic1 concernées, et le signe ">" les lignes de fic2 concernées.

```
$ cat liste
Produit objet   prix    quantites
souris  optique 30      15
dur     30giga  100     30
dur     70giga  150     30
disque  zip     12      30
disque  souple  10      30
ecran   15      150     20
ecran   17      300     20
ecran   19      500     20
clavier 105     45      30
clavier 115     55      30
carte   son     45      30
carte   video   145     30

$ cat liste2
Produit objet   prix    quantites
souris  boutons 30      15
dur     30giga  100     30
dur     70giga  150     30
disque  zip     12      30
disque  souple  10      30
ecran   15      150     20
ecran   17      300     20
ecran   19      500     20
ecran   21      500     20
clavier 105     45      30
clavier 115     55      30
```

Le fichier liste est l'original. Dans liste2, la deuxième ligne a été modifiée, une ligne écran a été ajoutée et les deux dernières lignes ont été supprimées.

```
$ diff liste liste2
2c2
< souris        optique 30      15
---
```

```
> souris        boutons 30      15
9a10
> ecran 21      500     20
12,13d12
< carte son     45      30
< carte video   145     30
```

- `2c2` : les lignes 2 de liste et liste2 doivent être échangées (elles doivent concorder soit en optique, soit en boutons).
- `9a10` : après la ligne 9 de liste (écran 19) il faut ajouter la ligne 10 (écran 21) de liste2.
- `12,13d12` : les lignes 12 et 13 de liste (carte son et vidéo) doivent être supprimés car elles n'existent pas après la ligne 12 de liste2.

b. cmp

La commande **cmp** compare les fichiers caractère par caractère. Par défaut la commande s'arrête dès la première différence rencontrée et indique la position de l'erreur.

```
cmp [-l] [-s] fic1 fic2
```

Le paramètre `-l` détaille toutes les différences en trois colonnes. La première colonne représente le numéro de caractère, la deuxième la valeur octale ASCII du caractère concerné de fic1 et la troisième la valeur octale ASCII du caractère concerné de fic2.

L'option `-s` retourne uniquement le code d'erreur (non visible), accessible par echo $?.

```
$ cmp liste liste2
liste liste2 differ: char 38, line 2
$ cmp -l liste liste2
    38 157 142
    39 160 157
    40 164 165
    41 151 164
    42 161 157
    43 165 156
    44 145 163
   182 143 145
   183 154 143
  ...
```

14. Délai d'attente

La commande **sleep** permet d'attendre le nombre de secondes indiqués. Le script est interrompu durant ce temps. Le nombre de secondes et un entier compris entre 0 et 4 milliards (136 ans).

```
$ sleep 10
```