

# Quelques commandes Unix (très) utiles pour la manipulation de fichiers textes

Franck Sajous (CLLE, CNRS & Université de Toulouse 2)

<http://fsajous.free.fr>

## 1 Préambule

Ce document présente quelques commandes Unix, et les paramètres les plus utiles de ces commandes pour manipuler des fichiers au format texte. Les différentes commandes recensées fonctionnent selon le même principe :

1. lecture ligne par ligne d'un ou plusieurs fichiers (de manière séquentielle), ou de l'entrée standard ;
2. filtrage (sélection) éventuel de certaines lignes selon des critères spécifiés par l'utilisateur ;
3. pour les lignes sélectionnées, restriction éventuelle à certaines informations (*e.g.* selon certains motifs) ;
4. transformation éventuelle des informations sélectionnées ;
5. écriture ligne par ligne du résultat dans un fichier ou sur la sortie standard.

### 1.1 Redirection de la sortie standard dans un fichier

Les commandes présentées dans ce document écrivent le résultat de leur traitement sur la « sortie standard », *i.e.* le résultat est affiché dans le terminal dans lequel on a saisi la commande. Il est possible d'écrire le résultat dans un fichier en « redirigeant la sortie standard ». Il faut pour cela faire suivre la commande d'un chevron fermant (signe supérieur `>`) et du nom (éventuellement précédé d'un chemin) d'un fichier. Par exemple, la commande :

```
head fichier.txt
```

affichera les dix premières lignes de *fichier.txt* dans le terminal, alors que la commande :

```
head fichier.txt > fich10.txt
```

écrira les dix premières lignes de *fichier.txt* dans un fichier *fich10.txt*.

### 1.2 Enchaînement de commandes, ou « pipeline »

Il est possible d'enchaîner les traitements de plusieurs commandes en créant un « pipeline » grâce au caractère | (« pipe »). Si l'on saisit `commande1 fichier.txt | commande2`, la « sortie standard » de `commande1` devient « l'entrée standard » de `commande2`. Au fur et à mesure que `commande1` produit des résultats (*i.e.* écrit des lignes sur sa sortie), `commande2` lit et traite ces lignes, comme la commande le ferait en lisant un fichier. Ainsi, la commande :

```
commande1 fichier.txt | commande2
```

est équivalent à l'enchaînement des commandes suivantes :

```
commande1 fichier.txt > fichierIntermediaire.txt
```

```
commande2 < fichierIntermediaire.txt ou cat fichierIntermediaire.txt | commande2
```

Le « pipeline » évite de créer des fichiers intermédiaires, *e.g.* `sort fichier.txt | uniq -c | sort -brn` (cf. §2.8).

### 1.3 Documentation : man

Les éléments de description donnés ici sont loin d'être exhaustifs. Pour obtenir la documentation complète d'une commande, utilisez dans le terminal la commande `man` (pour *manual*), suivie du nom de la commande. Par exemple `man grep` affichera la documentation de la commande `grep` dans un terminal. Pour faire défiler l'aide, appuyez sur la barre espace. Pour sortir de la page d'aide, appuyez sur la touche `q` (*quit*). Si vous connaissez assez bien une commande et que vous souhaitez simplement afficher ses différents paramètres, vous pouvez le plus souvent saisir le nom de la commande dans le terminal et appuyer sur *Entrée* (*Return*). Par exemple, saisir dans un terminal la commande `grep` et appuyer sur *Entrée* provoque l'affichage de :

```
Usage: grep [OPTION]... MOTIF [FICHIER]...
```

```
Try 'grep --help' for more information.
```



## 2 Commandes

### 2.1 Visualiser un fichier texte : more et less

`less` est une version plus récente que `more` et permet de faire défiler un fichier texte dans un terminal, page par page (une page correspond au nombre de lignes que peut afficher le terminal).

`more fichier` ou `less fichier`

Appuyer sur espace pour passer à la page suivante et sur `q` (*quit*) pour revenir au terminal.

### 2.2 Premières lignes d'un texte : head

NOM

`head` - Afficher le début des fichiers

SYNOPSIS

`head [OPTION]... [FICHIER]...`

DESCRIPTION

Afficher les 10 premières lignes de chaque FICHIER sur la sortie standard.

`-n K, --lines=K`

afficher les K premières lignes au lieu des 10 premières

### 2.3 Dernières lignes d'un texte : tail

Même principe et même syntaxe que `head`.

### 2.4 Sélection des lignes contenant un motif : grep

NOM

`grep` - Afficher les lignes correspondant à un motif donné

SYNOPSIS

`grep [OPTIONS] MOTIF [FICHIER...]`

DESCRIPTION

`grep` recherche dans les FICHIERS indiqués les lignes correspondant à un certain MOTIF.

OPTIONS

`-i, --ignore-case`

Ignorer la casse aussi bien dans le MOTIF que dans les fichiers.

`-v, --invert-match`

Inverser la mise en correspondance, pour sélectionner les lignes ne correspondant pas au motif.

`-h, --no-filename`

Ne pas afficher le nom des fichiers au début des lignes qui correspondent. C'est le comportement par défaut quand il n'y a qu'un fichier dans lequel effectuer la recherche.

`-E, --extended-regexp`

Interpréter le MOTIF comme une expression rationnelle étendue

`-A N, --after-context=N`

Afficher les N lignes qui suivent celle contenant le motif.

`-B N, --before-context=N`

Afficher les N lignes qui précèdent celle qui contient le motif.

`-o, --only-matching`

N'afficher que les parties (non vides) correspondantes des lignes sélectionnées, chaque partie étant affichée sur une ligne séparée.

Pour plus d'information sur le traitement des motifs contenant des expressions régulières, voir la documentation de la commande.



## 2.5 Substitutions de motifs exprimés par expressions régulières « à la Perl » : sed

### NAME

sed - stream editor for filtering and transforming text

### SYNOPSIS

sed [OPTION]... {script-only-if-no-other-script} [input-file]...

### DESCRIPTION

Sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline).

-r, --regexp-extended  
use extended regular expressions in the script.

### COMMAND SYNOPSIS

s/regexp/replacement/  
Attempt to match regexp against the pattern space.  
If successful, replace that portion matched with replacement.  
The special escapes \1 through \9 refer to the corresponding matching sub-expressions in the regexp.

Note 1 : qui peut le plus peut le moins ! Si vous ne savez pas ce que sed considère comme une expression régulière « étendue », n'hésitez pas à utiliser systématiquement l'option -r.

Note 2 : sed est essentiellement utilisée pour effectuer des substitutions de chaînes de caractères. Pour les substitutions complexes, notamment celles qui font intervenir des expressions « non gloutonnes » (cf. section 5.2), mieux vaut recourir à Perl. Il n'est pas nécessaire de connaître ce langage pour effectuer des substitutions, il suffit de connaître les expressions régulières. Dans un terminal, la syntaxe est :

```
perl -pe 's/regexp/replacement/' fichierEntree.txt
```

On peut faire suivre la substitution des modificateurs g et/ou i pour 1) remplacer toutes les occurrences appariées et/ou effectuer les appariement en ignorant la casse : 's/regexp/replacement/gi'

## 2.6 Sélection de champs particuliers d'une ligne : cut

### NOM

cut - Supprimer une partie de chaque ligne d'un fichier

### SYNOPSIS

cut [OPTION]... [FICHIER]...

### DESCRIPTION

Afficher des parties sélectionnées des lignes de chaque FICHIER vers la sortie standard.

-d, --delimiter=DÉLIM  
utiliser le DÉLIMiteur au lieu d'une tabulation comme délimiteur de champs  
-f, --fields=LISTE  
sélectionner seulement ces champs ; afficher également toutes les lignes qui ne contiennent pas de caractère délimiteur, sauf si l'option -s est spécifiée

Utiliser une seule des options -b, -c ou -f. Chaque LISTE se compose d'un intervalle ou de plusieurs intervalles séparés par des virgules. Chaque intervalle se compose de :

N Nième octet, caractère ou champ, compté à partir de  
N- du Nième octet, caractère ou champ jusqu'à la fin de la ligne  
N-M du Nième au Mième octet (inclus), caractère ou champ  
-M du premier au Mième octet (inclus), caractère ou champ

L'entrée standard est lue quand FICHIER est omis ou quand FICHIER vaut « - ».

Note : certains caractères ont un rôle particulier dans l'interpréteur de commande (par exemple : ; | < >). Lorsqu'on veut les utiliser comme délimiteur, il faut les placer entre apostrophes (quotes). Par exemple, pour sélectionner le troisième champ d'une ligne en utilisant le caractère | comme délimiteur : cut -f3 -d'|'. Lorsqu'on veut utiliser le caractère tabulation comme délimiteur, la syntaxe est la suivante : cut -f3 -d\$'\t'.



## 2.7 Trier : sort

NOM

`sort` - Trier les lignes de fichiers texte

SYNOPSIS

`sort [OPTION]... [FICHIER]...`

DESCRIPTION

Afficher sur la sortie standard la concaténation triée de tous les FICHIERS.

- `-b, --ignore-leading-blanks`  
ignorer les blancs de tête
- `-n, --numeric-sort`  
comparer selon la valeur numérique de la chaîne
- `-r, --reverse`  
inverser le résultat des comparaisons
- `-k, --key=POS1[,POS2]`  
utiliser la clé de tri commençant à POS1 (les positions sont comptées à partir de 1) et se terminant à POS2 (la fin de la ligne par défaut). Voir le format de POS ci-dessous.
- `-t, --field-separator=SÉPARATEUR`  
utiliser le SÉPARATEUR à la place d'une transition d'un caractère non blanc vers un caractère blanc.

Par défaut, l'ordre lexicographique est utilisé (la chaîne "12" vient avant la chaîne "3"). Si l'option `-n` est spécifiée, c'est l'ordre numérique qui est utilisé. Si l'on veut trier selon certains champs et non sur la ligne entière, on peut avoir recours à l'option `-k` de la manière suivante :

- `sort -k3,6` : trier sur les valeurs des champs 3 à 6 (le tri ne tient compte ni des champs 1 et 2, ni des champs 7 et suivants)
- `sort -k3,3 -k6,6` : trier sur la valeur du 3<sup>e</sup> champ, puis sur la valeur du 6<sup>e</sup> champ (le tri ne tient compte que de ces deux champs, et aucun autre)

Lorsqu'on trie selon plusieurs champs, l'ordre de tri peut être différent. La documentation stipule que les champs sont délimités par une transition d'un caractère non blanc vers un caractère blanc (*i.e.* espace et tabulation). Pour changer ce comportement, on peut utiliser l'option `-t`. Considérons par exemple la version CSV du lexique PsychoGLAFF (<http://redac.univ-tlse2.fr/lexiques/psychoglaff.html>), dont les champs sont séparés par des caractères ; (point-virgule). Deux lignes (tronquées) sont reproduites ci-dessous :

```
aïeul;Ncms;aïeul;a.jø1;a.j91;153;5.30;362;12.54;173;0.78;540;2.45;684;0.54;2440;1.94;5;4;V.GVC;2; ...
aïeuls;Ncmp;aïeul;a.jø1;a.j91;4;0.13;362;12.54;19;0.08;540;2.45;141;0.11;2440;1.94;6;4;V.GVC;2; ...
```

Si l'on veut afficher les mots du plus fréquent au moins fréquent dans le corpus Frantext (6<sup>e</sup> champ), en affichant les mots d'égale fréquence par ordre alphabétique (forme fléchée = 1<sup>er</sup> champ), on utilisera la commande :

```
sort -k6,6rn -k1,1 -t';' psychoGLAFF-1.0.csv
```

Explications :

1. `-k6,6rn` : trier d'abord sur le 6<sup>e</sup> champ, selon l'ordre numérique inverse;
2. `-k1,1` : trier ensuite sur le 1<sup>er</sup> champ, selon l'ordre par défaut (lexicographique croissant);
3. `-t ';'` : le séparateur de champs est le point-virgule (cf. note de la section 2.6).

## 2.8 Supprimer les lignes identiques successives : uniq

NOM

`uniq` - Signaler ou éliminer les lignes répétées

SYNOPSIS

`uniq [OPTION]... [ENTRÉE [SORTIE]]`

DESCRIPTION

Filtrer les lignes successives identiques du fichier d'ENTRÉE (ou de l'entrée standard), écrire dans le fichier de SORTIE (ou vers la sortie standard).



OPTIONS

- c      préfixer les lignes par le nombre d'occurrences
- i      ignorer les différences de casse

Cette commande n'a pas d'effet si plusieurs lignes identiques ne sont pas successives (*i.e.* si elles sont séparées par des lignes différentes). Exemple :

```

texte.txt    uniq texte.txt    uniq -c texte.txt    uniq -i texte.txt    uniq -i -c texte.txt
AAA
BBB    AAA            1 AAA            AAA            1 AAA
AAA    BBB            1 BBB            BBB            1 BBB
AAA    AAA            2 AAA            AAA            3 AAA
aaa    aaa            1 aaa            CCC            1 CCC
CCC    CCC            1 CCC
    
```

La commande `uniq` est souvent utilisée conjointement avec `sort` (cf. §2.7), pour : d'abord, trier un fichier, puis ne conserver (et éventuellement compter) qu'une instance unique des lignes de ce fichier. Pour trier les lignes d'un fichier par fréquence croissante ou décroissante, on utilise un enchaînement du type : `sort | uniq -c | sort`. En reprenant l'exemple du fichier précédent, la commande `sort texte.txt | uniq -c | sort` donne :

```

AAA    sort    aaa    uniq -c    1 aaa    sort -n    1 aaa
BBB    →      AAA    →      3 AAA    →      1 BBB
AAA         AAA         1 BBB         1 CCC
AAA         AAA         1 CCC         3 AAA
aaa         BBB
CCC         CCC
    
```

Dans la réalité, on remplacera la dernière commande `sort -n` par : `sort -b -n -k1,1`. La commande `uniq`, employée avec l'option `-c`, produit des lignes qui commencent par des espaces (en nombre différents, pour compléter les nombres d'occurrences de longueurs différentes). Or ces caractères "espace" interfèrent avec le tri numérique opéré par `sort`. C'est l'intérêt de l'option `-b` (*ignore leading blanks*). De plus, lors d'un tri par fréquences croissantes ou décroissantes, on souhaite souvent ne trier que sur le premier champ (`-k1,1`).

## 2.9 Compter le nombre de mots/de lignes : wc

NOM

`wc` - Afficher le nombre de lignes, de mots et d'octets d'un fichier

SYNOPSIS

`wc` [OPTION] ... [FICHIER] ...

DESCRIPTION

Afficher le décompte de lignes, de mots et d'octets pour chaque FICHIER

- l, --lines      afficher le nombre de lignes
- w, --words     afficher le nombre de mots

## 2.10 Fusionner deux fichiers : join

Permet d'effectuer la jointure (au sens "bases de données") entre deux fichiers tabulés (plusieurs champs séparés par un caractère donné, pas nécessairement une tabulation) sur la base d'un champ commun. Par défaut, c'est le premier champ qui sert à la fusion et les caractères blancs qui sont considérés comme délimiteurs.

**Important :** les deux fichiers à fusionner doivent être triés sur la valeur du champ qui sert à la fusion.

NAME

`join` - join lines of two files on a common field

SYNOPSIS

`join` [OPTION]... FILE1 FILE2

DESCRIPTION

For each pair of input lines with identical join fields, write a line to standard output. The default join field is the first, delimited by blanks.



Deux options sont essentielles :

1. `-t c`  
utilise `c` comme séparateur (pareil que pour `sort`, pareil aussi que l'option `-d c` de la commande `cut`)
2. `-j f`  
effectue la jointure sur le champ no `f`

Exemple : imaginons le cas où nous aurions construit les tables de fréquence pour deux corpus (on représente des fréquences brutes dans l'exemple; dans la réalité, il conviendrait d'utiliser des fréquences relatives... imaginons donc aussi que les deux corpus ont la même taille).

117;certaines	131;certaines
118;derrière	101;claire
116;porter	157;derrière
116;propre	112;porter
117;responsabilité	108;troisième
116;troisième	...
...	

freq1.txt

freq2.txt

La commande suivante (fusion des fichiers sur le second champ, point-virgule comme séparateur de champs) :

```
join -t';' -j 2 freq1.txt freq2.txt
```

produira la sortie :

```
certaines;117;131
derrière;118;157
porter;116;112
troisième;116;108
```

Par défaut, `join` produit en sorties les lignes pour les valeurs du champ de fusion communes aux deux fichier. Pour produire également les lignes qui comprennent des valeurs spécifiques, il faut utiliser l'option `-a num`.

```
join -t';' -j 2 -a 1 freq1.txt freq2.txt
```

Produit :

```
certaines;117;131
derrière;118;157
porter;116;112
propre;116
responsabilité;117
troisième;116;108
```

Le fichier de sortie contient toutes les lignes du premier fichier, avec les valeurs du second quand elles existent. Il est possible de faire afficher un caractère en cas de valeur absente dans le second fichier (*e.g.* `'-'` ou `'0'`) avec l'option `-e c`, qui s'utilise toujours conjointement avec l'option `-o f`, pour préciser l'utilisation du format `f` en sortie (pour plus de précisions : Google). Vous pouvez tenter l'option `-o auto`.

```
join -t';' -j 2 -a 1 -e '0' -o auto freq1.txt freq2.txt
```

Produit :

```
certaines;117;131
derrière;118;157
porter;116;112
propre;116;0
responsabilité;117;0
troisième;116;108
```

Ça commence à ressembler à quelque chose. Mais il manque les fréquence pour les mots spécifiques au second fichier. On peut doubler l'option `-e a`.

```
join -t';' -j 2 -a 1 -a 2 -e '0' -o auto freq1.txt freq2.txt
```

Produit :



```
certaines;117;131
claire;0;101
derrière;118;157
porter;116;112
propre;116;0
responsabilité;117;0
troisième;116;108
```

## 2.11 Différence entre deux fichiers : diff

NOM

diff - Comparer des fichiers ligne à ligne

SYNOPSIS

diff [option] ... fichiers

DESCRIPTION

Comparer les fichiers ligne à ligne.

### Exemple

Fichier *dormeur1.txt*

```
C'est un trou de verdure où chante une rivière
Ligne intruse 1
Ligne intruse 2
Accrochant follement aux herbes des haillons
D'argent ; où le soleil, de la montagne fière,
Luit : c'est un petit val qui mousse de rayons.
```

Fichier *dormeur2.txt*

```
C'est un trou de verdure où chante une rivière
Accrochant follement aux herbes des haillons
D'argent ; où le soleil, de la montagne fière,
Ligne intruse 3
Ligne intruse 4
Luit : c'est un petit val qui mousse de rayons.
Ligne intruse 5
```

La commande `diff dormeur1.txt dormeur2.txt` produira le résultat suivant :

```
2,3d1
< Ligne intruse 1
< Ligne intruse 2
5a4,5
> Ligne intruse 3
> Ligne intruse 4
6a7
> Ligne intruse 5
```

## 2.12 Connaître l'encodage d'un fichier texte : file

Selon qu'un fichier est encodé en « *latin1* » (ISO-8859-1, par exemple) ou en Unicode (UTF-8, par exemple), et selon la configuration de votre système, il est possible (probable) que l'affichage ou le traitement du fichier pose problème. Pour connaître l'encodage d'un fichier texte, utilisez la commande `file monFichier.txt`. La commande affichera alors dans le terminal une réponse du type : "ASCII text", "ISO-8859 text", "UTF-8 Unicode text, with very long lines", etc.

Note : la commande `file` permet plus largement de déterminer le type d'un fichier : texte, mais pas seulement (binaire, code source, etc.).

## 2.13 Changer l'encodage d'un fichier texte : recode et iconv

`iconv` et `recode` permettent de convertir un fichier d'un encodage donné vers un autre. Selon votre système, une seule de ces deux commandes (ou aucune) peut être installée. Nous décrivons ci-dessous la commande `iconv`. La commande `recode` fonctionne sur le même principe à ceci près qu'elle écrase le fichier initial (voir la documentation).

NOM

iconv - Convertir l'encodage de fichiers d'un encodage vers un autre

SYNOPSIS

iconv -f encodage [-t encodage] [fichier] ...

DESCRIPTION

Le programme `iconv` convertit l'encodage des caractères d'un fichier, ou de l'entrée standard si aucun fichier n'a été spécifié, d'un jeu de caractères vers un autre.



Le résultat est écrit sur la sortie standard à moins que l'option `--output` ait été spécifiée.

```
--from-code, -f encodage
    Convertir les caractères à partir de l'encodage encodage.
--to-code, -t encodage
    Convertir les caractères vers l'encodage encodage.
--output, -o fichier
    Spécifier le fichier de sortie (au lieu de stdout).
--list, -l
    Afficher la liste des jeux de caractères reconnus.
```

Exemple : conversion du fichier *fichierInitial.txt* de latin1 vers UTF-8 et enregistrement du résultat dans le nouveau fichier *fichierConverti.txt* :

```
iconv -f ISO-8859-1 -t UTF-8 fichierInitial.txt -o fichierConverti.txt
```

L'équivalent avec `recode` serait (avec écrasement du fichier initial par sa version recodée) :

```
recode ISO-8859-1..UTF-8 fichierInitial.txt
```

### 3 Application : une table de fréquences... avec les moyens du bord

Construire la table de fréquences d'un texte, ou d'un corpus de textes, consiste à dresser un inventaire des formes de surface, avec leurs fréquences absolues respectives :

46321	de	118	derrière
25850	la	117	responsabilité
18274	le	117	certaines
17245	à	116	troisième
14778	les	116	propre
13996	et	116	porter
...		...	

Le repérage des formes (tokens) est en soi un problème délicat : comment délimiter les unités (mots composés, clitiques, ponctuation fortes et sigles, etc.). Techniquement, on ne peut pas se passer d'un langage de programmation. Cependant, on peut se débrouiller pour obtenir une première approximation avec les commandes Unix que nous avons étudiées.

Dans la suite de cette section, on donne un exemple de construction (un peu fruste) de la table de fréquences d'un corpus contenu dans un fichier texte `corpus.txt`

Première étape : segmenter le fichier au format un token par ligne. Il suffit de remplacer les frontières de mots par un retour à la ligne (sous Unix : `\n`) : `sed 's/ /\n/g' corpus.txt`

Au	l'estimation	été	à
cours	de	mise	cause
de	son	en	de
la	patrimoine	cause	cet
campagne,	a	notamment	appartement.

On voit dans ce premier essai un premier problème : certaines marques de ponctuation collées aux mots. On peut supprimer ces marques placées en fin de ligne :

```
sed 's/ /\n/g' corpus.txt | sed 's/[.,;]$//g'
```

Une autre solution consiste à segmenter à la fois sur les espaces et les marques de ponctuation :

```
sed 's/[ ,;.:!:"]/\n/g' corpus.txt
```

On obtient :

Au	l'estimation	mise	de
cours	de	en	cet
de	son	cause	appartement
la	patrimoine	notamment	.
campagne	a	à	
,	été	cause	



Reste à segmenter sur les apostrophes (on ne peut inclure l’apostrophe dans l’ensemble [ , ; . ! : " ] car l’expression régulière passée à `sed` est déjà entre apostrophes. Une solution est d’utiliser une deuxième commande `sed`, à qui l’on passe entre guillemets le motif à remplacer :

```
sed 's/[ ,;.:!:"]/\n/g' corpus.txt | sed "s/'/\n/g"
```

On obtient :

Au		l		a		à
cours		,		été		cause
de		estimation		mise		de
la		de		en		cet
campagne		son		cause		appartement
,		patrimoine		notamment		.

Pour obtenir la table de fréquences, il ne reste plus qu’à enchaîner avec `sort | uniq -c | sort` (cf. § 2.8) :

```
sed 's/[ ,;.:!:"]/\n/g' corpus.txt | sed "s/'/\n/g" | sort | uniq -c | sort -b -r -n -k1,1
```

3 de		1 mise		1 en		1 appartement
2 cause		1 la		1 cours		1 à
1 son		1 l		1 cet		1 a
1 patrimoine		1 été		1 campagne		
1 notamment		1 estimation		1 Au		

Le résultat ci-dessus est obtenu pour une phrase. Appliqué à tout le fichier, il donne un résultat comme celui présenté en début de section 3.

Quelques manipulations supplémentaires :

— filtrer les hapax : ajouter | `grep -v '^\s*1\s'`

On supprime les lignes qui commencent par une éventuelle suite de blancs, puis 1, puis un autre caractère blanc.

— filtrer les mots de fréquence inférieure à 5 : ajouter | `grep -v '^\s*[1-5]\s'`

## 4 Format CSV

CSV (Comma-separated values) est un format de fichier texte (on peut l’ouvrir et le modifier avec un éditeur de texte, contrairement aux fichiers qui contiennent des données binaires) contenant des données tabulées : plusieurs champs séparés par un caractère donné (virgule, ou autre). C’est un format qu’on peut générer ou manipuler automatiquement (langage de programmation ou commandes Unix). On peut également ouvrir un fichier `.csv` avec un tableur et exporter les données d’un document tableur vers un fichier `.csv` (uniquement les valeurs : pas les formules, ni le formatage, ni les graphiques).

Bien que facile et pratique à utiliser, quelques subtilités tout de même :

— si le séparateur choisi est la virgule et qu’une colonne contient des valeurs textuelles... ces valeurs peuvent contenir des virgules ! Dans ce cas, il faut délimiter ces valeurs par des double-quotes :

```
"ma valeur1, avec des virgules", "ma valeur2, avec des virgules"
```

— si le champ contient le caractère séparateur *et* des double-quotes : on double les double-quotes (si !) :

```
"ma valeur1, avec des ""guillemets"", et des virgules", "ma valeur2"
```

— un tableur sait se débrouiller pour lire l’encodage d’un champ comme ci-dessus (dès lors que l’encodage est correct). En revanche, pour manipuler les lignes contenant ce type de valeurs en ligne de commande... c’est beaucoup moins pratique !

— on peut avoir dans une même colonne des champs délimités par des double-quotes et d’autres non

— lorsqu’on ouvre un fichier `.csv` avec un tableur (un bon tableur : OpenOffice/LibreOffice), il nous est demandé de préciser quels caractères doivent être considérés comme séparateur(s) et comme délimiteur de champs

— on peut spécifier ces caractères lorsqu’on exporte un fichier tableur en CSV

— attention aux blagues du tableur lors de l’export d’un fichier contenant des valeurs décimales (délimiteur = point ou virgule, selon la langue !). À l’ouverture, en plus de préciser les caractères séparateurs et délimiteurs de champs textuels, on peut également spécifier la langue du document, ainsi que la langue et le format (numérique/textuel) pour certaines colonnes : c’est ce qui déterminera l’interprétation qui sera faite du point et de la virgule (caractère textuel ou séparateur décimal ?)



## 5 Expression régulières

### 5.1 Appariement (*matching*)

ouill	les chaînes qui contiennent <i>ouill</i>
^anti	toutes les chaînes qui commencent par <i>anti</i>
eur\$	toutes les chaînes qui finissent par <i>eur</i>
^antigribouilleur\$	la chaîne exacte <i>antigribouilleur</i>
(anti in)	une chaîne qui contient <i>anti</i> ou <i>in</i>
[xyzt]	un des caractères <i>x</i> , <i>y</i> , <i>z</i> , ou <i>t</i>
[a-z]	une lettre en minuscule (sans diacritique)
[A-Z]	une lettre en majuscule (sans diacritique)
[^x]	n'importe quel caractère sauf <i>x</i>
[^a-z]	n'importe quel caractère sauf une lettre en minuscule
[a-z0-9]	une lettre en minuscule ou un chiffre
^[a-z]	une chaîne qui commence par une lettre en minuscule
^[^a-z]	une chaîne qui ne commence pas par une lettre en minuscule
.	n'importe quel caractère
a.b	un chaîne qui contient <i>a</i> , puis n'importe quel caractère, puis <i>b</i>
a*	0 ou plusieurs fois la lettre <i>a</i>
a+	1 ou plusieurs fois la lettre <i>a</i>
a?	0 ou une fois la lettre <i>a</i>
(a b)*	0 ou plusieurs occurrences des lettres <i>a</i> et <i>b</i>
\n	retour à la ligne
\t	tabulation
\s	caractère blanc (espace ou tabulation)
\\	le caractère backslash (déspécialisé)
\	le caractère pipe (déspécialisé)
\*	le caractère * (déspécialisé)
\+	le caractère + (déspécialisé)
\[	le caractère [ (déspécialisé)
\]	le caractère ] (déspécialisé)
\(	le caractère ( (déspécialisé)
\)	le caractère ) (déspécialisé)
\p{L}	une lettre (diacritique compris)... Ne marche qu'avec certains outils, selon encodage, environnement, etc. Voir la doc. de l'outil et des tutoriels sur les "classes de caractères" sur le web (ou ailleurs).

### 5.2 Expressions gloutonnes (*greedy*) vs. non gloutonnes (*non-greedy*)

Les quantifieurs \* et + capturent par défaut la chaîne trouvée la plus longue qui puisse correspondre au motif spécifié. Le modifieur ?, appliqué à ces quantifieurs modifient ce comportement (capture de la chaîne trouvée la plus courte qui puisse correspondre au motif spécifié).

Si le fichier `lapin.txt` contient la ligne suivante :

*un lapin au vin blanc*

La commande

```
sed 's/l.*n/DICTIONNAIRE/g' lapin.txt
```

Produit : *un DICTIONNAIREc*

La commande

```
sed 's/l.*?n/DICTIONNAIRE/g' lapin.txt
```

Produit :

*un DICTIONNAIRE au vin bDICTIONNAIREc*

### 5.3 Parenthèses capturantes

Selon les outils, des variables sont créées, qui correspondent aux expressions « capturées » par les parenthèses. Ces variables sont nommées \$1, \$2, ... \$n ou (c'est le cas de la commande `sed`) \1, \2, ... \n.

```
sed -r 's/l(.*?)n/==\1**/g' lapin.txt
```

produit : *un ==lapin\*\* au vin b==lan\*\*c*

Pour segmenter un texte, en gardant le caractère séparateur (cf. §3) : `sed -r 's/([\t.,;:!?"])/\1\n/g' corpus.txt`

