# Utilitaires (1)

Un *filtre* est une commande qui lit les données sur l'entrée standard, effectue des traitements sur les lignes reçues et écrit le résultat sur la sortie standard.

Bien sûr, les entrées/sorties peuvent être redirigées, ou enchainées avec des tubes (pipes).

Notez que `cat` est un filtre que vous avez déjà employé.

Recopiez chez vous les fichiers `acouper`, `acouper0`, `atrier1`, `blancs_et_tabs`, `caracteres_repetes`, `dates` et `image.pgm`.

## 1 Utilitaires tr, cut, sort

1.1 Exercice Tester le filtre `tr` (**tr**anslate) :

```
$ tr [a-z] [A-Z]
aaaaaaa bbbbbbb ccc 11111 AAA A22222
AAAAAAA BBBBBBB CCC 11111 AAA A22222
$
$ tr [:lower:] [:upper:]
aaaaaaa bbbbbbb ccc 11111 AAA A22222
AAAAAAA BBBBBBB CCC 11111 AAA A22222
```

L'option -d permet de supprimer (delete) des caractères, l'option -s permet de supprimer des répétitions (squeeze-repeats) de caractères. Essayez :

```
tr -s [a] [a] < caracteres_repetes
tr -s [abc] [abc]< caracteres_repetes
tr -d [a]  < caracteres_repetes
```

1.2 Exercice

```
od -a blancs_et_tabs
od -o blancs_et_tabs
```

(On visualise, sous des formats différents, TOUS les caractères du fichier `blancs_et_tabs`).

```
expand blancs_et_tabs > blancs_et_tabs1
od -a blancs_et_tabs1
```

Quel a été l'effet de `expand` ?

```
unexpand -a blancs_et_tabs1 | od -a
```

Quel est l'effet de `unexpand -a` ?

1.3 Exercice `cut` en sélection de colonnes :

```
cat acouper.
cut -c 4 acouper
cut -c 4,8 acouper
```

Comment faire pour obtenir les colonnes 4 **à** 8 ? les colonnes 4 et suivantes ?

1.4 Exercice `cut` en sélection de champs :
`cat acouper`.
`cut -f 1 acouper`
`cut -f 2 acouper`
On n'obtient pas les champs escomptés (le premier, puis le deuxième) car le séparateur par défaut est TAB. On essaye alors :
`cut -d ' ' -f 1 acouper`
`cut -d ' ' -f 2 acouper`
Reprenez ces expériences avec le fichier `acouper0`. Comment expliquer ces résultats ? Construisez un fichier `acouper1` sur lequel `cut` agisse "convenablement". Exécutez `cut -d ' ' -f 2 acouper1`

1.5 Exercice
`sort -n -k 3 atrier1`
`sort -n -r -k 3 atrier1`
(trie selon le champ 3, et l'ordre des entiers)
`sort -k 2 atrier1`
(trie selon le champ 2, et l'ordre alphabétique).

Remplacez toutes les suites d'espacements ('sp' ou 'ht') par un caractère ' :'. Le fichier obtenu s'appelle maintenant `atrier3`
`sort -k 3 atrier3`
`sort -t : -k 3 atrier3`

1.6 Exercice Ecrire une commande `renverser <fichier>` qui renverse l'ordre des lignes d'un fichier (on pourra combiner `cat -n`, `sort -r` et `cut`)

1.7 Exercice Comment trier, par ordre chronologique, un fichier contenant une date par ligne sous la forme : `jj␣mm␣aaaa` ? Appliquez votre solution au fichier `dates`.

## 2 Expressions rationnelles

| Symbole | emacs | sed | awk | grep | grep -E | signification |
|---|---|---|---|---|---|---|
| `.` | X | X | X | X | X | un caractère quelconque |
| `*` | X | X | X | X | X | expression précédente répétée 0 ou plusieurs fois |
| `^` | X | X | X | X | X | début de ligne |
| `$` | X | X | X | X | X | fin de ligne |
| `\` | X | X | X | X | X | \ placé devant un caractère spécial annule sa signification spéciale |
| `[ ]` | X | X | X | X | X | un ensemble de caractères |
| `[^ ]` | X | X | X | X | X | complément d'un ensemble de caractères |
| `\( \)` | X | X | | | | sauvegarde d'un texte correspondant à un motif (utilisé aussi pour grouper les sous-expression) |
| `\{`$n$`,`$m$`\}` | | X | | X | | $n \leq i \leq m$ occurrences de l'expression précédente |
| `{`$n$`,`$m$`}` | | | X | | X | *idem* |
| + | X | | X | | X | expression précédente répétée une ou plus de fois |
| `?` | X | | X | | X | répétition zéro ou une fois |
| `\|` | | | X | | X | choix entre deux expressions rationnelles |
| `\|` | X | | | | | *idem* **emacs** |
| `( )` | | | X | | X | grouper les sous-expressions |

Exemples.

| motif | **signification** |
|---|---|
| `mer*` | *me, mer, merr, merrr,...* |
| `b[aeiuo]g` | deuxième lettre est une voyelle |
| `^...$` | une ligne contenant exactement trois caractères |
| `[A-Za-z]` | une lettre |
| `[^0-9a-zA-Z]` | tout symbole sauf une lettre ou un chiffre |
| `^[^.]` | le premier caractère n'est pas un point |
| `^\.[a-z]\{2,5\}` | une ligne commençant par un point suivi par $n$ lettres minuscules, $2 \leq n \leq 5$, (`grep` ou `sed`) |
| `0\{5,\}` | cinq ou plus de zéros |

# 3 Utilitaire grep

Cet utilitaire (*General Regular Expression Parser*, analyseur général d'expressions régulières) sélectionne toutes les lignes qui satisfont une expression régulière (ou rationnelle).

`grep` [*options*] *motif fichiers*

Recherche dans un ou plusieurs *fichiers* les lignes qui correspondent à l'expression régulière *motif*. Les valeurs de retour de grep sont : 0 si grep a trouvé des lignes, 1 si grep n'a trouvé aucune ligne, 2 en cas d'erreur. Si les *fichiers* sont absents grep lit l'entrée standard, ce qui permet de l'utiliser comme un filtre.

**Quelques options**

| | |
|---|---|
| `-i` | ignorer la distinction entre les majuscules et les minuscules |
| `-n` | sortir chaque ligne retrouvée précédée par son numéro |
| `-v` | sortir les lignes qui ne correspondent pas au motif |
| `-E` | utiliser les expression régulières étendue (comme `egrep`) |
| `-l` | sortir uniquement les noms de fichiers qui contiennent les lignes recherchées, mais pas les lignes elle-même. |
| `-e` *motif* | le motif peut être précédé par `-e`, utile si *motif* commence par « - ». |
| `-w` | sortir les lignes qui possèdent des mots correspondant au motif. Les caractères différents de lettres, chiffres et le caractère « _ » (souligner) sont considérés comme les séparateurs des mots. |

3.1 Exercice Trouvez parmi vos fichiers dont le nom commence par point ceux qui contiennent le mot `PATH`.

3.2 Exercice Récupérez un fichier python contenant des commentaires et des définitions de fonctions (par exemple `libArray.py` que vous utilisez dans le cours de programmation). Quelle commande permet d'extraire toutes les lignes de commentaires ? Ecrivez une commande qui affiche les noms de toutes les fonctions définies dans le fichier.

3.3 Exercice Un certain nombre de pages de manuel se trouvent dans `/usr/share/man/man1`. Recherchez dans toutes les pages de manuel des commandes qui commencent par "a" les lignes qui contiennent un des trois mots : "each", "because", "new". Les mots doivent être trouvés même s'ils commencent par une majuscule. Attention, les fichiers sont compressés (voir `zgrep`).

3.4 Exercice Récupérer le fichier `exemples_grep.txt`.

- Ecrire une commande qui affiche toutes les lignes du fichier `exemples_grep.txt` qui contiennent le mot `ligne` suivi de deux chiffres.
- Ecrire une commande qui affiche toutes les lignes qui contiennent le symbole `":"` en fin de ligne.
- Ecrire une commande qui affiche toutes les lignes qui contiennent un espace, suivi du symbole `":"`, suivis d'un espace, suivi de cinq caractères quelconques, suivi à nouveau d'un espace.
- Ecrire une commande qui affiche toutes les lignes qui contiennent un espace, suivi d'une virgule, suivie de 0, 1 ou plusieurs caractères quelconques, suivi d'un point.

- Ecrire une commande qui affiche toutes les lignes qui commencent par un caractère différent de l'espace et de `E`.
- Ecrire une commande qui affiche toutes les lignes qui contiennent des mots entiers commençant par `li` et se terminant par un `s`.

## 3.1 Utilitaire sed - Pour ceux qui sont en avance

La commande `sed` permet entre autre de substituer une chaine de caractères par une autre dans un fichier.

3.5 Exercice Visualisez l'image `image.pgm` avec `gimp` par exemple. Créez un nouveau fichier en remplaçant toutes les occurences de la chaine de caractère "251" par la chaine "0". Visualisez à nouveau l'image.