

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.

Recopier chez vous les fichiers `acouper`, `acouper0`, `atrier1`, `blancs_et_tabs`, `caracteres_repetes`, `dates`, `toto` à partir de ce dossier : `/net/autre/cvuchener/if110`.

1 Utilitaires `tr`, `cut`, `sort`

1.1 EXERCICE Tester le filtre `tr` (translate) :

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

N.B. Si les crochets `[ ]` ne marchent pas, essayer avec les accolades `{ }`.

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

```
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 champs 3, et l'ordre des entiers)  
sort -k 2 atrier1  
(trie selon le champs 2, et l'ordre alphabétique).
```

Remplacez toutes les suites d'espacements ('sp' ou 'ht' avec la commande *tr*) par un caractère ':'. Le fichier obtenu s'appelle maintenant **atrier2**.

```
sort -k 3 atrier2  
sort -t : -k 3 atrier2
```

1.6 EXERCICE Ecrire une commande `./renverser.sh <fichier>` qui renverse l'ordre des lignes d'un fichier (on pourra combiner *cat -n*, *sort -r* et *cut*). Veuillez donc créer un fichier **renverser.sh** auquel vous donnerez les droits en execution (`chmod +x renverser.sh`). Pour cela utiliser le code ci-dessous :

```
#!/bin/bash  
commande $1 #le $1 sert à récupérer le premier argument de l'appel du script
```

1.7 EXERCICE (POUR CEUX QUI SONT EN AVANCE) 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)
\{ <i>n</i> , <i>m</i> \}		X		X		$n \leq i \leq m$ occurrences de l'expression précédente
{ <i>n</i> , <i>m</i> }			X		X	<i>idem</i>
+	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					<i>idem</i> emacs
()			X		X	grouper les sous-expressions

EXEMPLES.

motif	signification
mer*	<i>me, mer, merr, merrr, ...</i>
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 Recherchez dans tous les fichiers du dossier `/usr/share/vim-scripts/doc/` dont le suffixe est `.txt` les lignes qui contiennent un des trois mots suivant : “each”, “because”, “new”. Les mots doivent être trouvés même s’ils commencent par une majuscule.

3.1 Utilisation de **grep** sous emacs - Pour ceux qui sont en avance

Lancez emacs. L’utilitaire **grep** peut être lancé à partir d’emacs par la commande **grep** (évidemment) :

M-x grep

Emacs ouvre le mini-buffer qui affiche **grep -n -e** et où il faut taper la suite de commande, c’est-à-dire le motif (après **-e**) ensuite, éventuellement, d’autres options et finalement les noms de fichiers. Après RET emacs affiche les résultats de la recherche dans une nouvelle fenêtre. Vous pouvez parcourir les lignes retrouvées à l’aide de la commande C-x ‘.

3.3 EXERCICE Refaites Exo. 3.1 sous emacs. Parcourez les lignes retrouvées.

3.4 EXERCICE Refaites Exo. 3.2 sous emacs. Parcourez les lignes retrouvées, essayez de modifier certaines lignes, par exemple remplacez certaines occurrences de “because” par “parce que”, “each” par “chaque”, etc. Notez que après avoir modifié les lignes vous pouvez relancer le parcours de lignes retrouvées par **grep** en tapant encore une fois C-x ‘.