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 du répertoire `~jcaval/Utils`.

1 Utilitaires tr, cut, sort

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

```
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
```

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 Ecrire une commande `comprime <fichier>` qui remplace toutes les séquences de caractères d'espacement (sp,ht) par un seul espace (sp). Vérifier avec od que la commande fonctionne !

1.4 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 ?

1.5 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`
Reprendre ces expériences avec le fichier `acouper0`. Comment expliquer ces résultats ? Utiliser la commande `comprime` de l'exercice ci-dessus pour construire un fichier `acouper1` sur lequel cut agisse "convenalement". Exécuter `cut -d ' ' -f 2 acouper1`

1.6 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). Introduisez un blanc supplémentaire entre "Rene" et "Sapri" (3ieme ligne de `atrier1`) et recommencer (le fichier obtenu s'appelle maintenant atrier2) :
`sort  -k 2 atrier2`
Comment expliquez-vous ce changement dans le classement ? Remplacer 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`

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

1.8 Exercice On souhaiterait trier un fichier dont les lignes ont la forme
`<champ1>␣<champ2>...␣NUMERO␣<entier>␣...`
par ordre croissant des champs `<entier>`.
Exemple : le fichier
```
cette␣ligne␣a␣le␣NUMERO␣2␣mais␣elle␣est␣ici
ce␣NUMERO␣1␣a␣bonne␣tete
j'ai␣le␣␣NUMERO␣3
```
sera transformé en :
```
ce␣NUMERO␣1␣a␣bonne␣tete
cette␣ligne␣a␣le␣NUMERO␣2␣mais␣elle␣est␣ici
j'ai␣le␣␣NUMERO␣3
```
Peut-on effectuer simplement ce tri à l'aide de la commande `sort` ? Revenir à cette question après avoir vu la commande `sed`.

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

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

3.2 Exercice Rechercher dans tous les fichiers avec le suffixe `.man` les lignes qui contiennent un de trois mots : "each", "because", "new". Les mots doivent être trouvés même s'ils commencent par une majuscule.

3.1 Utilisation de grep sous emacs

Lancer 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 Refaire Exo. 3.1 sous emacs. Parcourir les lignes retrouvées.

3.4 Exercice Refaire 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 ‘`.

4 sed - stream oriented editor

Sed opère selon les règles suivantes :

- Chaque ligne est copiée dans un buffer spécial (pattern space).
- Toutes les commandes sont appliquées une à une dans l'ordre sur les textes dans le buffer.
- Si une commande change le contenu du buffer la commande suivante est appliquée au texte modifié et non pas à la ligne originale.
- Le fichier d'entrée original ne change pas, le résultat d'édition sort sur la sortie standard et peut être rédirigé vers un fichier.

La syntaxe de sed :

`sed [`*options*`]` *commande fichier*

Si le *fichier* est absent alors `sed` lit sur l'entrée standard. Deux options nous seront utiles :

`-e` *commande*	où *commande* est une commande, cette option est utile si nous avons plusieurs commandes à exécuter.
`-n`	supprime la sortie par défaut ; `sed` affiche seulement les lignes spécifiées par la commande `p` ou par l'option `p` de la commande `s`.

Les commandes de sed ont la forme suivante :

`[`*adresse1* `[,`*adresse2*`]][!]`*commande* `[`*arguments*`]`

Comme nous pouvons voir ci-dessus, une commande sed peut être précédée par 0, 1 ou 2 *adresses*. Une adresse peut avoir la forme :

n	où *n* est un entier et désigne la ligne numéro *n*.
$	désigne la dernière ligne.
/*motif*/	où *motif* est une expression régulière (entre deux caractères *slash*).
\%*motif*%	*idem*, mais le séparateur / peut être remplacé par un autre caractère (ici par %).

si la commande est précédée par :	**alors elle s'applique à :**
aucune adresse	chaque ligne
une adresse	chaque ligne qui corresponde à l'adresse
deux adresses	chaque ligne à partir de la première ligne qui correspond à *adresse1* jusqu'à la ligne qui correspond à l'adresse *adresse2*
une adresse suivie de !	chaque ligne qui ne correspond pas à l'adresse

EXEMPLES. Les exemples qui suivent utilisent la commande `d` (delete).

`/^BEGIN/,/^END/d`	Supprime toutes les lignes entre une ligne qui commence par BEGIN et la première ligne qui suit et qui commence par END (inclusivement). Si aucune ligne qui suit BEGIN ne commence par END alors toute les lignes jusqu'à la fin du fichier sont supprimées.
`/SAVE/!d`	Supprime les lignes qui ne contiennent pas SAVE.
`1,$!d`	Supprime toutes les lignes sauf la première et la dernière.

4.1 Quelques commandes de sed

Sauf mention contraire, vous allez tester vos commandes `sed` en utilisant le fichier `toto`.

4.1.1 Commande d

[*adresse1* [,*adresse2*]][!]`d`

La commande `d` supprime la ligne (où les lignes) spécifiées par l'adresse. Ces lignes ne passent pas vers la sortie standard. Une nouvelle ligne est lue et l'édition recommence avec la première commande du script.

4.1 EXERCICE Supprimer les lignes vides du fichier *toto*.

4.2 EXERCICE Supprimer les lignes vides ou composées uniquement des caractères `TAB` et `SPC`.

REMARQUE 10 Pour entre un caractère spéciale interprété par bash[22] dans la ligne de commande on le tape précédé par `C-v`.

[22]Comme `TAB`, le caractère de nouvelle ligne (`RET`), `C-a`, `C-e`, `C-p` et d'autres.

4.1.2 Commande p

$\boxed{[\textit{adresse1}\,[,\textit{adresse2}]][\,!]\texttt{p}}$

La commande `p` envoie le contenu du buffer (pattern space) vers le sortie standard. Une nouvelle ligne est lue. Cette commande s'utilise uniquement si sed était invoqué avec l'option `-n`.

4.3 Exercice Sortir uniquement les lignes 19-21 du fichier toto.

4.4 Exercice Sortir les lignes 19 et 21 uniquement.

4.5 Exercice Sortir les lignes qui contiennent au moins trois chiffres consécutifs.

4.6 Exercice Sortir les lignes plus longues que 65 caractères.

4.1.3 Commande r

$\boxed{[\textit{adresse}]\texttt{r}\ \textit{fichier}}$

La commande `r` lit le *fichier* et l'ajoute au buffer d'édition. Il faut exactement un espace entre `r` et le nom de fichier

4.7 Exercice Ajouter le fichier *items* dans le fichier toto après la ligne qui contient "Liste des items". (Vous pouvez utiliser le fichier *items* qui se trouve chez `~jcaval/Utils/items`).

4.1.4 Commande s

$\boxed{[\textit{adresse1}\,[,\textit{adresse2}]][\,!]\texttt{s}/\textit{motif}\,/\textit{subst}\,/\textit{options}}$

La commande `s` est la principale commande de sed. Elle provoque une substitution de *motif* par *subst*.

Les options suivantes peuvent être specifiéees :

- *i* où *i* est un entier, remplace i$^{\text{ème}}$ occurence du *motif*. Par défaut seulement la première occurence du motif sur la ligne est remplacée.
- `g` Toutes les occurences du *motif* seront remplacées, pas seulement la première.
- `p` Si la ligne dans le buffer a subi une substitution alors elle sera envoyée vers la sortie standard. Cette option est à utiliser seulement si sed a été invoqué avec l'option `-n`.

4.8 Exercice Substituer "foo" par "tar" seulement si la ligne contient "baz".

4.9 Exercice Ajouter 4 espaces au début de chaque ligne du fichier toto.

4.10 Exercice Sortir uniquement les lignes contenant la lettre "a" en remplaçant chaque occurrence de "a" par "aaa".

4.11 Exercice Supprimer de chaque ligne de toto les caractères blancs (`SPC` et `TAB`) au début de la ligne.

4.12 Exercice Supprimer de chaque ligne de toto les caractères blancs (`SPC` et `TAB`) au début et à la fin de chaque ligne.

4.13 Exercice Supprimer les lignes vides qui se trouvent au début du fichier toto (et seulement celles au début).

4.14 Exercice Remplacer la première occurrence de "toto" par "bobo" (la première dans le fichier et non pas la première dans chaque ligne).

4.15 EXERCICE Sur chaque ligne supprimer le premier mot entre les accolades. Par exemple toto contient la ligne :

```
{un}   {deux}  {trois}
```

qui à la sortie de votre script doit avoir la forme :

```
{}   {deux}  {trois}
```

4.16 EXERCICE Deviner quel est le résultat de la commande suivante :
`echo foo | sed 's/o*/AAA/'`

Exécuter la commande pour le vérifier.

Maintenant il devait être clair quelle sera la sortie de la commande :
`echo foo | sed 's/o*/AAA/g'`
(Dans le cas contraire exécuter-la.)

REMARQUE 11 Jusqu'à maintenant la partie *subst* de la commande `s` était indépendante du *motif*. Il y a deux possibilités pour réutiliser le motif entier ou les sous-motifs dans la substitution :
- Le caractère & utilisé dans *subst* désigne tout le motif.
- Les sous-motifs peuvent être entourés par `\(` et `\)`. Dans ce cas la notation \n, où n est un entier, $1 \leq n \leq 9$, utilisée dans *subst* désigne le n$^{\text{ème}}$ sous-motif.

4.17 EXERCICE Essayer : `sed 's/.*/(&)/' toto`

4.18 EXERCICE Le fichier `toto` possède une ligne qui contient le mot " Prenom" que vous pouvez voir avec `grep Prenom toto`. Exécuter la commande

```
sed 's/\(Nom : [A-Z][a-z]*\)\(.*\)\(Prenom : [A-Z][a-z]*\)/\3\2\1/' toto
```

et vérifiez comment elle a agi sur cette ligne.

4.19 EXERCICE Remplacer dans toto chaque occurrence d'une suite de chiffres "α" par "item α : ".

4.20 EXERCICE Écrire une commande sed qui, utilisée comme un filtre, simule la commande `basename` de bash. Par exemple,

`echo '/usr/local/bolo' |` *votre_commande_sed*

et

`echo '/usr/local/bolo/' |` *votre_commande_sed*

donneront à la sortie : le mot "bolo".