

TP n°1

Utiliser des Expressions Régulières

1 C'est quoi, sed ?

sed est l'éditeur de flux (stream editor) de Linux, qui permet de parser et transformer du texte avec un langage simple et concis. Pour mieux connaître son usage, vous vous référerez, bien entendu, à son manuel. Voici, quelques exemples, pour commencer.

Un des formats d'un appel de **sed** est le suivant:

```
sed [-options] "[adr1[adr2]] [commande ...]" [fichier ...] [> fichier_de_sortie]
```

Si vous appelez **sed** sur un fichier, sans options, adresses ni commandes (**sed** "" toto.txt), il lit *ligne par ligne* toutes les lignes du fichier, il n'en fait rien, et il affiche la ligne ou la copie dans le fichier de sortie, si vous avez précisé celui-ci. Si vous utilisez l'option **-n**, vous arrivez à empêcher même cette écriture. Attention, seulement les éléments entre crochets ([...]) sont optionnels, notamment dans ce cas d'absence de commande, il faut garder les guillemets ("") pour éviter l'ambiguïté entre la commande et le nom du fichier.

Une façon pour restreindre la portée d'une commande, lorsque vous daignez en mettre une, est d'utiliser les adresses. Quand vous utilisez une seule adresse, **sed** n'opère que sur la (ou les) ligne(s) adressée(s). Vous pouvez adresser une ligne par son numéro, ou, plus intéressant, par le fait qu'il contient un motif exprimé sous forme d'*expression régulière* (*basique* par défaut, et *étendue* si vous utilisez l'option **-E**) entre deux slashes : /motif/. Si vous utilisez deux adresses, séparées par une virgule, **sed** opère sur les lignes entre les deux.

Exercice 1 La commande **p** (pour **print**) affiche la ligne lue. Combinez-le avec ce que vous venez de lire pour simuler le comportement de **grep**.

Téléchargez la source **html** de la page web du cours. A l'aide de double adressage et éventuellement la commande **!**, afficher, séparément, une fois le corps (*body*) de la page et une fois tout le reste.

Plus utile que **p**, il y a la commande **s** pour substitution. Vous pouvez l'utiliser avec le format **s/motif/remplacement/flags**. Le programme **sed** remplace les occurrences du **motif**, donné sous forme d'une expression régulière, par **remplacement**. Vous pouvez modifier le comportement par défaut de **sed** par les **flags**.

L'entrée est lue ligne par ligne, et la commande **s** substitue par défaut seulement la première occurrence du **motif** dans chaque ligne. Par rapport aux expressions régulières, **sed** est dit être 'gourmand', c'est-à-dire, il prend la plus longue chaîne de caractères qui correspond à l'expression.

Exercice 2 Commencez par faire **sed** répéter chaque ligne que vous entrez au clavier, en effaçant le premier mot de chaque ligne. Pour que **sed** lise son entrée du clavier, il suffit de laisser vide le champ **fichier**.

Utilisez les *références vers l'arrière* pour répéter l'entrée en inversant le premier et le deuxième mot. Dans les expressions régulières *basiques*, vous pouvez utiliser par exemple la commande

`s/un motif\(\motif 1\)autre motif\(\motif 2\)...\1\2/`, et les références vers l'arrière `\1` et `\2` etc. auront pour valeur les expressions qui ont *matché* respectivement le 1^{er} et le 2nd motifs parenthésés, etc.

Utilisez ensuite le flag `g` (pour `global`) pour remplacer tous les espaces par des tirets bas (`_`) et enregistrez le résultat dans un autre fichier (en ajoutant `> nouveau_fichier` à la fin de la ligne de l'appel de `sed`).

Une dernière commande avant de vous mettre en route pour attaquer les exercices plus sérieux, la commande `N`. Rappelez-vous que `sed` traite son entrée ligne par ligne : `N` vous permet de lire une ligne supplémentaire et l'ajouter à la fin de la ligne déjà lue, séparées, bien entendu, par un caractère de fin de ligne (`\n`). Ainsi, vous pouvez effectuer des opérations sur plusieurs lignes.

Et finalement, sachez que vous pouvez enchaîner plusieurs commandes de `sed` dans un seul appel, comme vous allez voir dans l'exercice suivant:

Exercice 3 Que fait la commande suivant?

```
sed -E "/SKIP3LINES/ {N; N; N; s/SKIP3LINES.*/Les 3 prochaines lignes ignorées./;}"
```

Exercice 4 Quand vous programmez en Java, vous êtes, bien sûr, très pointilleux sur les indentations et compagnie. Notamment, s'il y a une seule commande après un `if`, vous n'utilisez pas d'accolade, mais vous sautez la ligne, vous augmentez l'indentation, et vous écrivez la commande. Par exemple :

```
if (node != NULL)
    return (node.getParent());
```

Mais finalement, vous changez d'avis et décidez d'écrire ce genre de conditionnelle dans une seule ligne :

```
if (node != NULL) return (node.getParent());
```

Utilisez `sed` pour appliquer cette modification au fichier `.java` sur lequel vous travaillez.

2 Nettoyage de textes

Prof de français, vos élèves vous ont rendu 35 poèmes sous forme de fichiers textes. Par expérience, vous savez à l'avance qu'il vaut mieux que vous commenciez par un petit ménage des erreurs classiques afin de vous épargner quelques migraines.

Pour faire cela d'un coup avec un seul appel de `sed`, vous pouvez créer un fichier où vous mettez toutes les commandes, et ensuite appeler `sed` sous le format suivant:

```
sed -f fichier_de_commandes fichier_de_poeme ...
```

Exercice 5 En vous servant de `sed`, automatiser les opérations suivantes :

1. Supprimer tous les espaces et tabulations en début de ligne.
2. Remplacer les doubles (ou triples etc...) espaces au milieu du texte par des espaces simples.
3. Supprimer les répétitions de mots (par ex. *il rentra dans sa sa maison*).
4. Vérifier que toutes les lignes commencent bien par une majuscule (écriture en vers), corriger au besoin.

5. Vérifier qu'il n'y a jamais d'espace avant les virgules (mais un espace après), et qu'il y en a avant/après ";", "!", "?", ":", en corrigeant ce qui doit l'être.

Vous testerez vos commandes sur le fichier *eleve*.

3 Une nouvelle télé !

Cela faisait maintenant un an que le son de votre vieille télé était cassé, et que vous vous étiez habitué à regarder tout vos films avec des sous-titres pour malentendants. Pour Noël, vos parents ont racheté une nouvelle télé, il va donc vous falloir adapter vos fichiers de sous-titres !

Exercice 6 Vous aviez prévu de regarder *Bon Baisers de Bruges* ce soir, il vous faut donc supprimer la partie malentendante du fichiers. Celle-ci contient :

- des bruitages, sous la forme (SOUND EXPLANATION)
- le nom des personnages dans certains dialogues, sous la forme : PERSONNAGE: + texte (toujours en début de ligne et en majuscules)

À l'aide de **sed**, supprimez ces éléments du fichier sous-titres *In.Bruges.srt*, en sachant qu'il n'y a pas d'autres parenthèses que celles des bruitages.

Exercice 7 Votre film commence, le pop-corn est prêt, mais ô désespoir, vous rendez compte que votre télé flambante neuve ne supporte pas les balises html `<i>...</i>` et les affiche explicitement : rien de plus insupportable ! Un rapide coup d'oeil à la documentation vous informe en revanche que ce modèle supporte le Markdown, et qu'un texte en italique doit être encadré par deux étoiles : ***du texte***. Problème, afin d'épargner votre petit frère de la vulgarité du film, certains mots sont masqués avec ce même caractère (entre autres **f*****), ce qui risque de mettre du texte en italique n'importe où. À l'aide de **sed**, remplacez les ***** anti-vulgarités par un autre caractère (par exemple **@**) puis effectuez la substitution des balises italiques par des étoiles (et vite, toute la famille attend dans le canapé !)

Exercice 8 Le film fini, la reste de la famille étant partie se coucher, vous vous proposez de commencer une nouvelle saison de votre série favorite. Vous avez l'habitude de la regarder avec les sous-titres en français, cependant votre nouvelle télé ne supporte pas non plus l'UTF8... À l'aide de la commande **sed y**, remplacez tous les caractères accentués par leur version sans accents dans le fichier *The.Wire.s02e01.srt*.

4 De la musique en pagaille

Après avoir écouté en boucle une playlist sur Internet, vous vous décidez à la télécharger, et voulez l'ajouter à celles que vous avez déjà sur votre disque dur. Problème, les formats de fichiers ne sont pas les mêmes :

-la playlist récupérée sur internet est au format XSPF, dont vous pouvez observer la structure dans le fichier *playlist.xml* -vos playlists sont au format CSV, avec une ligne par morceau sous la forme : numéro de ligne,Artiste,Titre.

Exercice 9 En vous servant de **sed**, créer un fichier au format CSV à partir du fichier *playlist.xml*. Vous veillerez à ce que les noms des artistes ainsi que les titres des chansons aient une majuscule à chaque mot.

Exercice 10 Sauriez-vous ajouter le nom du fichier mp3 correspondant à chaque ligne de votre fichier CSV ?

Exercice 11 Sauriez-vous refaire la même chose à partir du fichier *playlist2.xml* ?

5 À la chasse aux nombres premiers

On se propose dans cet exercice de détecter les nombres premiers à l'aide de `sed`.

Exercice 12 Écrire dans votre langage préféré un script permettant d'énumérer les nombres de 1 à n en unaire, afin de produire un fichier *nombres* qui contiendra 1 sur la première ligne, 11 sur la seconde, 111 sur la troisième, etc.

Exercice 13 Trouver une expression régulière correspondant aux lignes contenant un nombre (écrit en unaire) admettant un diviseur. En déduire une expression régulière pour les lignes contenant un nombre non-premier.

Exercice 14 À l'aide de la commande `sed =`, numéroter les lignes du fichier *nombres*, en séparant le numéro de ligne du contenu par une tabulation. Ainsi, la cinquième ligne du résultat obtenu devrait être :

```
5    11111
```

Exercice 15 En vous utilisant les questions précédentes, écrire un script, qui depuis le fichier *nombres*:

- numérote les lignes
- ne garde que les lignes contenant des nombres premiers
- sur chaque ligne, supprimer la partie unaire.
- regroupe toutes les lignes en une seule, en les séparant par des virgules.

Vous devriez ainsi obtenir la liste des nombres premiers sous la forme 2,3,5,7,11,13,17,19.