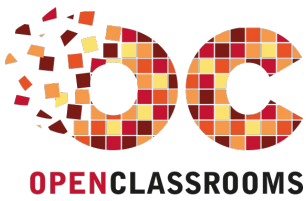
La commande sed

Par yoch



www.openclassrooms.com

Dernière mise à jour le 5/08/2010

Sommaire

Sommaire 2
La commande sed 3
Présentation 3
Premiers pas 4
Adressage 4
Mode silencieux 6
Substitution, translittération 6
Utilisation avancée 7
Commandes groupées 7
Utilisation multiligne 7
Labels et branchements 8
Mini FAQ 9
Partager 9

La commande sed

Par  yoch

Mise à jour : 05/08/2010
Difficulté : Intermédiaire Durée d'étude : 1 jour

Bonjour amis linuxiens,

Aujourd'hui, je vais vous présenter une commande ma foi fort utile, et très puissante : j'ai nommé la commande `sed`.

Comme tous ses cousins du monde UNIX, le mot **sed** a une signification: comprenez "éditeur de flux" *(Stream EDitor)*. Cet outil vous permettra de manipuler vos fichiers textes de façon automatique, de rendre vos scripts (beaucoup) plus puissants, etc.

Pourquoi je prends la peine d'écrire un tuto sur le sujet, alors que le manuel existe ? Eh bien, je croyais comme vous et ai essayé d'apprendre à me servir de **sed** en lisant le manuel: croyez-moi, j'ai plus perdu de temps qu'autre chose.
Bref, cette commande étant assez complexe, je trouve qu'un tuto n'est vraiment pas de trop pour apprendre à s'en servir.

Ce tuto est principalement destiné aux personnes familières avec la ligne de commande. De plus, pour profiter pleinement des possibilités offertes par **sed**, il vaut mieux savoir ce que c'est qu'une **regex**, c'est pourquoi une connaissance du principe des regex est requise pour bien comprendre ce tuto.
Sommaire du tutoriel :

- Présentation
- Premiers pas
- Utilisation avancée
- Mini FAQ

Présentation

Avant de commencer, il est fondamental de bien comprendre le fonctionnement de **sed**. C'est ce qui nous permettra d'en connaitre les limites, savoir ce que l'on peut faire et ne pas faire avec, à quoi est-il adapté, etc.

En premier lieu, il faut retenir que **sed** fonctionne en mode «flux», c'est-à-dire que le flux en entrée (fichier ou autre) est traité ligne par ligne. Ceci garantit de bonnes performances, ainsi qu'une utilisation réduite en mémoire, mais empêche certains types d'utilisations. Donc, à vous de voir si l'utilisation de sed correspond à votre problème ou non.
Retenez bien ce point, nous y reviendrons en détail plus loin.

Ensuite, il convient de noter deux façons d'utiliser **sed** :

- La méthode "classique", qui consiste à appliquer la commande sur le flux d'entrée, et récupérer le flux de sortie. Par exemple, on applique sed sur un fichier, et on redirige la sortie sur un autre fichier.
- La méthode "directe", avec l'option `sed -i`, qui applique la commande directement sur le fichier passé en entrée.

Nous verrons ensemble des exemples d'utilisation avec la première méthode (traitement d'un fichier passé en entrée et affichage à l'écran).

Jetons à présent un coup d'oeil au synopsis :

Citation : MAN page

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

Comme vous pouvez le voir, en plus des options et du flux d'entrée, **sed** reçoit un script. Ce script contiendra toutes les actions à exécuter sur le flux d'entrée.

Notez qu'il existe là encore deux manières de passer un script à sed :

- On peut écrire le script directement dans la ligne de commande, avec l'option `sed -e`. On séparera les commandes avec des point-virgules.
  C'est la façon "uniligne", souvent très pratique.
- On peut passer à sed un fichier externe (par exemple *myscript.sed*) contenant le script, avec `sed -f script-file`.
  Cela assure une meilleure lisibilité pour les gros scripts, et permet aussi de réutiliser un script.

Écrire un script pour **sed** peut devenir assez technique, et c'est seulement en pratiquant que vous arriverez à tirer plein parti de toute la puissance de ce dernier. Aussi, nous allons passer tout de suite à la pratique.

Premiers pas

Commençons par créer un fichier de texte quelconque.

Code : Autre

```
Bonjour,

Ceci est un fichier de test.
Ici la ligne numéro 4.

# ceci pourrait être un commentaire
Ici la ligne numéro 7.

Au revoir
```

Enregistrez ce fichier sous *test.txt*, puis placez vous dans son répertoire.

Nous allons faire nos premiers tests en affichant les résultats de sed dans la console.

Lancez : `sed '' test.txt`

Code : Console

```
$ sed '' test.txt
Bonjour,

Ceci est un fichier de test.
Ici la ligne numéro 4.

# ceci pourrait être un commentaire
Ici la ligne numéro 7.

Au revoir
```

Vous pouvez voir que sed lancé avec un script vide renvoie simplement le contenu du fichier.

Adressage

Adressage par ligne

Lancez : `sed -e '4d; 7d' test.txt`

Code : Console

```
$ sed -e '4 d; 7 d' test.txt
Bonjour,

Ceci est un fichier de test.

# ceci pourrait être un commentaire

Au revoir
```

Ici, nous utilisons l'adressage par ligne. La commande d *(delete)* indique que l'on va supprimer la ligne.

L'option -e permet de passer plusieurs commandes à la suite. Je pense que c'est une bonne habitude de l'ajouter de façon systématique.

On peut aussi utiliser l'adressage par intervalle, comme ceci : `sed '4,7 d' test.txt`

Cette commande va effacer toutes les lignes comprises entre la ligne 4 et la ligne 7.

Code : Console

```
$ sed '4,7 d' test.txt
Bonjour,

Ceci est un fichier de test.

Au revoir
```

Adressage par motif

On peut aussi utiliser des regex pour appliquer la commandes sur toutes les lignes ou le motif est trouvé. On décrit un motif comme ceci : `/regex/`

Exemple : `sed '/^#/ d' test.txt` supprimera toutes les lignes commençant par une dièse (le ^ est un métacaractère signifiant début de ligne).

Code : Console

```
$ sed '/^#/d' test.txt
Bonjour,

Ceci est un fichier de test.
Ici la ligne numéro 4.

Ici la ligne numéro 7.

Au revoir
```

Là encore, on peut utiliser une intervalle avec : `/motif1/,/motif2/`
`sed '/^Bonjour/,/^Au revoir/d' test.txt` supprimera toutes les lignes comprises entre 'Bonjour' et 'Au revoir', donc tout le fichier.

Dans le cas d'adressage par intervalle entre deux motifs, sed va répéter les commandes passées pour cet intervalle à chaque fois que l'intervalle est trouvé.

Si seul /motif1/ est présent, sed va appliquer le script sur tout ce qu'il trouvera après jusqu'à la fin du flux d'entrée.

On peut très bien aussi utiliser un adressage mixte, comme : `sed '/^#/,7 d' test.txt` ou `sed '4,/^#/ d' test.txt`. Je vous laisse deviner ce que font ces commandes...

Ici, nous avons utilisé la commande d *(delete)*, qui accepte l'adressage simple et l'adressage par intervalle. Certaines commandes ne fonctionnent que sur adresse simple, voire n'utilisent pas les adresses. Il faudra toujours vérifier si la commande employée est compatible avec le mode d'adressage choisi.

Mode silencieux

Il existe une autre façon d'utiliser sed, particulièrement intéressante. C'est l'utilisation en mode "silencieux", c'est-à-dire que sed ne doit afficher par défaut aucune ligne. Seules les lignes intéressantes seront affichées, avec la commande p *(print)*. Pour passer en mode "silencieux", il faut utiliser l'option `sed -n`

Lancez : `sed -n '/Ici/p' test.txt`

Code : Console

```
$ sed -n '/Ici/p' test.txt
Ici la ligne numéro 4.
Ici la ligne numéro 7.
```

A noter que la négation d'adresse est possible. Ainsi, cette commande produira exactement le même résultat : `sed '/Ici/!d' test.txt`

Substitution, translittération

Substitution

Bien évidemment, sed permet de remplacer du texte avec des regex. On peut utiliser la syntaxe habituelle, ou la syntaxe étendue avec `sed -r`.

La substitution s'écrit comme ceci : `s/motif/substitut/`
Par défaut, elle s'effectue sur la première occurrence du motif, sauf si on lui ajoute l'option g comme ceci : `s/motif/substitut/g`
On peut aussi choisir l'occurrence voulue, avec par exemple `s/motif/substitut/2` pour la deuxième occurrence.

Si le motif est une regex, le substitut n'en est pas une. Il accepte quand même le métacaractère `&`, ainsi que les références arrières telles que `\1` (de 1 à 9).

Quelques exemples :

- `sed -re 's/^# *//' fichier` décommente les lignes commentées (commençant par une dièse), et supprime les espaces en début de ligne (le * est un métacaractère signifiant 0 ou plus).

- `sed -re 's/\t/    /g' fichier` remplace les tabulations par 4 espaces.

Translittération

La translittération permet d'échanger certains caractères avec d'autres caractères.

On l'écrit comme ceci : `y/liste1/liste2/`.
Par exemple, pour supprimer les accents sur les e dans notre fichier *test.txt*, on fera `sed -re 'y/éèê/eee/' test.txt`

Code : Console

```
$ sed -re 'y/éèê/eee/' test.txt
Bonjour,

Ceci est un fichier de test.
Ici la ligne numero 4.

# ceci pourrait etre un commentaire
Ici la ligne numero 7.

Au revoir
```

Utilisation avancée

Commandes groupées

Sed permet de grouper les commandes avec des accolades. Cela permet de faire plus de choses dans une même commande (et de se rendre compte des limites de l'uniligne, même si nous allons continuer avec).

On peut ainsi faire des trucs comme :

```
sed -
e '/motif 1/ {commande1; commande2} ; /motif_2/ {commande1; commande2; /sous_motif/ {
commande1; commande2} }'
```

Je n'ai pas d'exemple intéressant à vous fournir pour l'instant, nous verrons des exemples plus loin.

Utilisation multiligne

Un des inconvénients principaux pour l'instant est l'impossibilité de travailler sur plus d'une ligne à la fois.

Pour peu que nous restions raisonnables (c'est-à-dire dans l'optique du traitement ligne par ligne), sed nous propose une commande afin de permettre de travailler sur plusieurs lignes . Mais auparavant, un peu de théorie sur le fonctionnement interne de sed s'impose.

Théorie

On pourrait résumer la manière dont sed travaille ainsi :

- Lecture d'une ligne sur le flux d'entrée, et stockage dans l'espace de travail.
- Traitement : exécute séquentiellement les commandes reçues sur l'espace de travail.
- Envoie la ligne au flux de sortie stdout.
- Passe à la ligne suivante...

Cette notion d'espace de travail est importante. Par exemple, la regex `/$/` représente la fin de l'espace de travail, et non la fin de la ligne courante (`\n`). Cela aura un impact dans nos traitements sur plusieurs lignes.

La ligne lue est stockée sans le caractère de fin de ligne, et envoyée au flux de sortie en ajoutant un caractère de fin de ligne. Ce qui fait par exemple que ce script ne fonctionnera pas :

Code : Console

```
$ sed 's/\n//' test.txt
Bonjour,

Ceci est un fichier de test.
Ici la ligne numéro 4.
```

```
# ceci pourrait être un commentaire
Ici la ligne numéro 7.

Au revoir
```

Nous avons essayé ici de supprimer les fins de ligne, mais la manière dont fonctionne sed ne nous permet pas de procéder de la sorte.

La commande N

La solution est d'insérer une ligne supplémentaire dans l'espace de travail. Cette ligne sera ajoutée à la suite d'un caractère de fin de ligne, ce qui va nous autoriser à faire du traitement multiligne.

Il existe une commande pour cela : la commande N.

Maintenant, attention, ce n'est pas tout. Si nous faisons par exemple : `sed -e 'N; s/\n//' test.txt`, on obtiendra

Code : Console

```
$ sed -e 'N; s/\n//' test.txt
Bonjour,
Ceci est un fichier de test.Ici la ligne numéro 4.
# ceci pourrait être un commentaire
Ici la ligne numéro 7.
Au revoir
```

Une fin de ligne sur deux a été supprimée, ce qui est normal. 

Rappelez-vous, sed applique le traitement puis recommence, donc il charge une ligne, charge la seconde, supprime le caractère `\n` (fin de ligne) trouvé, affiche la sortie (ce qui a pour effet de vider l'espace de travail), puis recommence...

La commande D

La commande d *(delete)* dispose aussi d'un équivalent spécialisé pour le traitement multiligne : c'est la commande D, qui efface le contenu de l'espace de travail jusqu'au premier caractère `\n` trouvé. S'il reste des données dans l'espace de travail, sed va relancer son traitement dessus. Sinon, une nouvelle ligne sera chargée.

Exemple d'utilisation : si l'on veut par exemple supprimer les lignes vides d'un fichier, on ne peut pas le faire comme ceci : `sed -re '/^$/ {N; s/\n//}' fichier`, à cause de la particularité évoquée ci-dessus. En effet, si nous avons plusieurs lignes vides à la suite, certaines vont rester.

On pourra par contre le faire comme cela : `sed -re '/^$/ {N; D}' fichier`

Labels et branchements

Une autre fonctionnalité importante et qui ajoute énormément de puissance à sed est la possibilité de branchement dans le script. Il est possible en effet de revenir en arrière dans le script, en créant des labels.

Un label s'écrit : `label`.

Les : font partie du label, ce sont eux qui indiquent que le mot est un label.

Exemple concret d'utilisation

Supposons que vous ayez devant vous un fichier HTML, et que vous voulez supprimer toutes les balises qu'il contient et ne

garder que le texte.

Pourquoi on ne peut pas faire `sed -re 's/<[^>]*>//g' fichier.html` ?

Ce script est très bien, mais le problème est que si vous avez des balises sur plusieurs lignes, votre truc ne va pas marcher...

Ce qu'il faudrait, c'est de vérifier à la fin de chaque traitement s'il ne reste pas une balise ouverte, et si c'est le cas pouvoir charger une nouvelle ligne dans l'espace de travail **et relancer le traitement**.

Pour faire cela, il y a la commande b *(branch)*. 

Voici un script qui marche même avec des balises sur plusieurs lignes : `sed -re ':start s/<[^>]*>//g; /</ {N; b start}' fichier.html`

Mini FAQ

La commande sed existe elle sous Windows ?

En principe non (c'est-à-dire que Windows n'est pas fourni d'office avec). Mais on peut très bien utiliser GNU sed sous Windows. Néanmoins, la syntaxe à utiliser dépendra alors de l'interpréteur de commandes, notamment avec les guillemets...

Au secours, mon script plante pour une raison incompréhensible !?

Attention, certaines options/commandes ne marchent pas avec tous les sed.
Attention aussi à l'ordre dans lequel vous passez les options.
Enfin, vous pouvez toujours demander de l'aide sur un forum dédié (ou non).

Ou puis-je me documenter sur sed ?

Personnellement, je vous conseille cet excellent tuto avec lequel j'ai appris et dont je me suis inspiré pour ce tuto.
Et voici encore une bonne source de réflexion et d'inspiration.
Voilà, ce tutoriel arrive à sa fin.

Ce tuto est loin d'être exhaustif (je pense qu'il faudrait un livre ), il ne présente pas toutes les commandes ni toutes les possibilités offertes par **sed**. Le but était surtout de vous faire découvrir et expérimenter un petit peu ce formidable outil.

J'espère que vous avez bien suivi et pourrez désormais profiter de toute la puissance de **sed**.

@+