Sed

C’est l’outil absolu pour modifier du texte en le passant par un pipe ! Ou pour effectuer des changements en masses sur un fichier sans l’ouvrir. Bref, comme le dit l’adage : « Sed, c’est bien » □□

Il est possible de faire des trucs de tarés avec (hey, c’est pas juste un truc pour faire des substitutions à coup d’expressions rationnelles, c’est un vrai éditeur de texte, on peut se balader dans le texte, faire des copier/coller, etc).

Syntaxe de base

Avec un fichier :

```
sed <commande> fichier.txt [fichier2.txt] [fichier3.txt]
```

En utilisant la sortie d’une autre commande :

```
find . -name \*.txt | sed <commande>
```

NB : `sed`, par défaut, ne modifie pas le fichier utilisé. Il affichera sur la sortie standard le fichier modifié par la commande passée à `sed`. Si on souhaite que le fichier soit modifié, on utilise l’option `--in-place` ou son abbréviation `-i` (on peut indifféremment placer l’option avant ou après la commande) :

```
sed <commande> --in-place fichier.txt
sed -i <commande> fichier.txt
```

Expression régulières

Si `sed` est un éditeur de texte (son nom veut dire Stream EDitor) et qu’il est utilisable en lui donnant des commandes équivalentes à « Va à ligne 3, supprime 4 caractères, descend à la ligne suivante… », on l’utilise souvent avec des expressions régulières.

Rappel sur les expressions régulières

Les expressions régulières permettent de rechercher des correspondances avec un motif, écrit avec une syntaxe spéciale.

Les éléments de syntaxe suivants appartiennent aux *Perl Compatible Regular Expression* (PCRE), qui sont le standard de la très grande majorité des langages de programmation. `sed` utilisant une syntaxe légèrement différente, certains caractères devront être échappés (voir plus bas).

NB : j'ai placé les expressions régulières de cette section entre des `/` pour les distinguer des chaînes de caractères simples.

- `.` : correspond à un caractère, n'importe lequel. `/./` correspondra à tout sauf à une chaîne vide
- `?` : quantificateur, modifie la correspondance du caractère qui le précède : celui ci peut être présent zéro ou une fois. `/a?/` correspondra à une chaîne vide ou à `a`
- `+` : quantificateur : le caractère précédent sera présent une ou plusieurs fois. `/a+/` correspondra à `a`, `aa`, `aaa`…
- `*` : quantificateur : le caractère précédent sera présent zéro ou plusieurs fois. `/a*/` correspondra à une chaîne vide, à `a`, `aa`, `aaa`…
- `{n}` : quantificateur : le caractère précédent sera présent `n` fois. `/a{3}/` correspondra à `aaa`
- `{n,m}` : quantificateur : le caractère précédent sera présent de `n` à `m` fois. `/a{3,5}/` correspondra à `aaa`, `aaaa` ou `aaaaa`
- `{n,}` : quantificateur : le caractère précédent sera présent au moins `n` fois. `/a{3,}/` correspondra à `aaa`, `aaaa`, `aaaaa`, `aaaaaa`…
- `|` : séparateur d'expression. `/bonjour|hello/` correspondra à `bonjour` ou à `hello`
- `[^liste]` : correspond aux caractères n'étant pas entre crochets. `/[^ae]/` correspondra à n'importe quel caractère sauf à `a` et à `e`
- `[liste]` : correspond à un des caractères entre crochets. `/[ae]/` correspondra à `a` ou à `e`. Pour que le caractère `^` soit un choix possible, il faut le placer à une autre place que la première place : `[liste^]`, `[lis^te]`… On peut aussi spécifier des plages de caractères : `[0-9a-zA-Z]`
- `^` : ancre, correspond au début de la ligne. `/^a/` correspondra à `a` si celui-ci est le premier caractère de la ligne
- `$` : ancre, correspond à la fin de la ligne. `/a$/` correspondra à `a` si celui-ci est le dernier caractère de la ligne
- `(…)` : groupe l'expression. On peut s'en servir, par exemple, pour capturer des éléments (ce qui permet de les réutiliser plus tard) ou faire des sous-expressions. `/Bonjour (foo|bar), ça va \?/` correspondra à `Bonjour foo, ça va ?` et à `Bonjour bar, ça va ?`

Pour utiliser les caractères de manière littérale (exemple : pour correspondre à un point), on les échappera avec un `\`. `/\./` correspondra au caractère point (`.`).

Caractères à échapper dans sed

La version GNU de `sed` utilise les *Basic Regular Expression* (BRE), qui ont une syntaxe légèrement différentes des PCRE.

Certains caractères doivent donc être échappés dans les BRE, qui sont utilisables tels quels pour des PCRE :

- le quantificateur `+`. Exemple : `/a\+/`
- le quantificateur `?`. Exemple : `/a\?/`
- le quantificateur `{i}`. Exemple : `/a\{5\}/`
- les parenthèses `(…)`. Exemple : `/\(a\)/`
- le séparateur d'expressions régulières `|`. Exemple : `/a\|b/`

Effectuer une substitution de texte

La commande à utiliser est `'s/expression régulière/substitution/'` :

```
sed 's/foo/bar/' foo.txt
```

Cette commande remplacera la **1ère occurrence** de `foo` de **chaque ligne** du fichier par `bar`.

Si on souhaite remplacer toutes les occurrences de chaque ligne, on emploie le modificateur `g` :

```
sed 's/foo/bar/g' foo.txt
```

Si on ne souhaite remplacer que l'occurrence n°X de chaque ligne :

```
sed 's/foo/bar/X' foo.txt
## Exemple avec la 2e occurrence :
sed 's/foo/bar/2' foo.txt
```

Si on ne souhaite remplacer l'occurrence n°X de chaque ligne, ainsi que les suivantes :

```
sed 's/foo/bar/gX' foo.txt
## Exemple avec la 2e occurrence et les suivantes :
sed 's/foo/bar/g2' foo.txt
```

Ajouter quelque chose au début de chaque ligne :

```
sed 's/^/FooBar /' foo.txt
```

Ajouter quelque chose à la fin de chaque ligne :

```
sed 's/$/ BazQux/' foo.txt
```

Si on souhaite que l'expression régulière soit insensible à la casse (ex : `s` qui correspond aussi à `S`), on emploie le modificateur `i` :

```
sed 's/foo/bar/i' foo.txt
```

On peut utiliser plusieurs modificateurs en même temps :

```
sed 's/foo/bar/gi' foo.txt
```

Pour réutiliser ce qui a correspondu à l'expression régulière dans la chaîne de substitution, on utilise le caractère `&` :

```
sed 's/foo/& et bar/' foo.txt
```

Pour réutiliser ce qui a correspondu à un groupe d'expression, on utilise `\1` pour le 1er groupe, `\2` pour le 2e groupe... :

```
sed 's/Bonjour (foo|bar). Il fait beau./Bonsoir \1. À demain./' foo.txt
```

NB : on peut utiliser d'autres séparateurs que le caractère `/`, ce qui rend l'écriture d'une commande plus simple si l'expression régulière ou la substitution comportent des `/` : plus besoin de les échapper. Exemple :

```
sed 's@/home/foo@/var/bar@' foo.txt
```

Supprimer des lignes

Supprimer la ligne `n` :

```
sed 'nd' foo.txt
## Exemple avec la 3e ligne :
sed '3d' foo.txt
```

Supprimer les lignes `n` à `m` :

```
sed 'n,md' foo.txt
## Exemple :
sed '3,5d' foo.txt
```

Supprimer toutes les lignes sauf la ligne `n` :

```
sed 'n!d' foo.txt
## Exemple :
sed '6!d' foo.txt
```

NB : attention, le caractère `!` est un caractère spécial pour `bash` (et les autres shells). Si vous utilisez des apostrophes (`'`) pour votre commande `sed`, tout ira bien, mais si vous utilisez des guillemets doubles (`"`), il faut l'échapper avec un `\`.

Supprimer toutes les lignes sauf les lignes `n` à `m` :

```
sed 'n,m!d' foo.txt
## Exemple :
sed '6,8!d' foo.txt
```

Supprimer les lignes qui correspondent à une expression régulière :

```
sed '/regex/d' foo.txt
## Exemple :
sed '/foobar/d' foo.txt
```

Pour supprimer les lignes vides, d'un fichier, il suffit d'utiliser l'expression régulière qui signifie que la ligne est vide :

```
sed '/^$/d' foo.txt
```

Pour supprimer la première ligne correspondant à une expression régulière, ainsi que toutes les lignes suivantes :

```
sed '/regex/,$d' foo.txt
## Exemple :
sed '/foobar/,$d' foo.txt
```

NB : attention encore une fois aux guillemets doubles, il faudrait échapper `$` dans cette commande car le shell essayerait d'interpréter `$d` comme une variable.

Révision #5
Créé 2020-01-23 17:10:26 CET par Luc
Mis à jour 2024-10-30 12:09:38 CET par Luc