sed par la méthode structuraliste

- Objet : Décrypter la documentation de sed et dégager la structure de son fonctionnement.
- Niveau requis :
  avisé
- Commentaires : Pour tentative d'application (orienté Linux) de la *théorie de la praxis* de F. de Saussure

“Notre dette des uns aux autres, infiniment, dans une chaîne sans fin ni frontière, dans l’espace comme dans le temps. Cette dette dont la conscience nous fera accueillir l’étrange et l’étranger, l’inconnu comme l’imprévu lorsqu’ils frapperont inopinément à nos portes. A condition cependant que nous ayons appris à vivre en vérité. Et que, par conséquent, nous n’ayons pas renoncé à savoir.”

Michel Foucault

Références

- Illustration de man sed : http://www.scotchlinux.tuxfamily.org/doc/sed.php
- Exemples d'utilisation :
  http://www.system-linux.eu/index.php?post/2008/12/21/La-commande-Sed

Introduction

man

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

L'éditeur de flux sed est lancé en ligne de commandes avec la commande sed.
Il ne faut pas confondre la commande sed reconnue par le shell (qui lance le programme sed) avec les commandes internes du programme sed[1)]

Par défaut (sans option) sed traite le flux d'entrée et affiche sur la sortie, non seulement le flux d'entrée qu'elle enregistre, mais aussi le résultat de son traitement.
Le flux d'entrée peut être ce qui lui est transmis par un pipe en ligne de commandes ou par un fichier donné en argument.

- Méthodes d'appel :

```
cmd[cat|echo|ls|...] [fichier[repertoire]]\
 | sed [options] cmd_de_sed
```

ou

```
sed [options] cmd_de_sed fichier
```

En tant que commande lancée par le shell, sed possède des options :

- Les options sans implication avec les commandes internes de sed

Options	**Signification**
–posix	Désactiver toutes les extensions GNU
-u, –unbuffered	Charger des quantités minimales de données depuis les fichiers d'entrée et libérer les tampons de sortie plus souvent
–help	Afficher cette aide et sortir
–version	Afficher les informations de version du logiciel

- Les options sensibles aux commandes internes de sed

options	**significations**
-e	enchaîner plusieurs commandes
-r	utiliser les expressions régulières étendues dans un script
-n	mode silencieux : permet de ne rien modifier associée au drapeau p (print) (affichage sur la sortie standard)
-f	Les commandes sont lues à partir d'un fichier préalablement rédigé.
-i	Le fichier est édité sur place.

“Options sensibles”

-J'appelle “options sensibles” celles dont l'utilisation dépend des commandes internes de sed, et dont l'usage réclame une connaissance de la “grammaire” de sed.

-J'appelle “grammaire” du programme sed les règles qui gèrent la syntaxe globale d'une ligne de commandes sed relativement aux commandes internes de sed.

En effet, la syntaxe de l'adressage et le choix des options dépendent du choix des commandes internes.

Par conséquent :
1) toutes les options “lues” par le shell qui lance sed, ne sont pas toutes associables entre elles selon les commandes internes utilisées.
2) certains types d'adressage ne fonctionnent qu'avec certaines commandes de sed.
3) toutes les commandes internes ne sont pas toutes associables entre elles.
4) certaines commandes nécessitent la présence d'autres commandes pour que leurs usages prennent sens.

Cette grammaire se résume ainsi :

```
sed [-n [-e commande] [-f script] [-i[.extension]]\
 [l [cesure]] rsu] [<commande>] [<fichier(s)>]
```

→ La compréhension de la documentation et le fonctionnement global de sed sont facilités lorsque l'on maîtrise un double rapport d'implications :

- 1) la mise en relation `[options]` ⇔ `cmd_de_sed` ;
- 2) `/adressages/` ⇔ `cmd_de_sed`

- 3) la mise en relation de différentes commandes

La compréhension globale de sed dépend tout d'abord de la mise en relation de 1 et 2.

Les rapports d'implication "options de sed" <=> "commandes de sed"

Cette association détermine les possibilités de combinaison des options de la commande sed.

Les options -r ; -e ; -n

```
sed -rne '/adressage/cmd_de_sed' fichier
```

sed-options

```
Il est vraiment idiot,
se balade sur un vélo
sans pédale, et porte ses sandales
à son goulot.
il veut pour adresse mail 127.0.0.1@lo
```

- Exemple :

```
sed -rne '4p' sed-options
```

```
à son goulot.
```

> On voit là que la commande (interne de sed) p prévaut sur l'option sed `-n`.
> Dans l'exemple précédent l'option `-e` et `-r` sont inutiles, elles ne le sont plus pour l'exemple suivant :

```
sed -rne '/([[:digit:]].){3}[[:digit:]]@lo/p' -e '4p' sed-options
```

```
à son goulot.
il veut pour adresse mail 127.0.0.1@lo
```

La possibilité d'utiliser telle où telle option dépend de la fonction des commandes de sed.
Ainsi l'option `-r` ne s'utilisera pas avec la commande “y” puisque la translittération n'utilise pas de *E.R.*
L'option `-i` ne s'utilisera qu'avec les commandes qui transforment le flux d'entrée, donc elle ne s'utilisera pas avec les commandes de sed “p”.

Options -i avec -r , -n , -e

Lorsqu'on a besoin d'adresser au moyen d'une expression régulière étendue, et de modifier réellement, il faut associer les options `-r` et `-i`.
La syntaxe est toujours `-ri` , et non pas -ir.
On peut associer d'autres options, après `-ri`

```
sed -rine '/([[:digit:]].){3}[[:digit:]]@lo/d' sed-options
```

ou

```
sed -rien '/([[:digit:]].){3}[[:digit:]]@lo/d' sed-options
```

> L'option `-n` rendant le retour silencieux :

```
cat sed-options
```

```
Il est vraiment idiot,
se balade sur un vélo
sans pédale, et porte ses sandales
à son goulot.
```

L'option f

```
sed [options] -f script [-i] [fichier...]
```

Cette option permettant l'utilisation de script sed, va donc influer sur la possibilité d'utiliser conjointement d'autres options de sed.
Les commandes internes de sed doivent être inscrites dans le script, ce qui influe sur la syntaxe d'appel des commandes internes. L'utilisation du tampon n'étant plus la même avec l'utilisation d'un script sed, l'effet de l'option `-n` va changer.

Elle implique aussi la possibilité d'utiliser les commandes de sed mises au point principalement pour l'usage de scripts sed.
(Voir la troisième partie “scripts sed avancés”)

- **Éléments de syntaxe d'un script sed :**

> Le fait que les commandes internes soient dans un fichier (ou script sed) implique que l'encadrement des commande de sed par ' ou “ ne doit pas être utilisé.
>
> Un adressage pour chaque commande :

```
/adressage1/cmd1_interne
/adressage2/cmd2_interne
```

```
...
```

> Un adressage pour plusieurs commandes :

Utilisation d'une commande spécifiquement utilisée avec les scripts sed : { }.

```
/adressage1/{
    cmd1_interne
    cmd2_interne
    ...
}
```

- sed -f sans autres options :

```
sed -f script-sed fichier
```

1. Soit le fichier d'origine nommé “sed-options”
2. Soit le script sed nommé “script1-sed” :

```
Script1 :
/([[:digit:]].){3}[[:digit:]]@lo/p
4p
```

```
sed -rnf script1-sed sed-options
```

```
à son goulot.
il veut pour adresse mail 127.0.0.1@lo
```

- Option -f avec d'autres options :

1. Soit le fichier d'origine nommé “sed-options”
2. Soit le script sed nommé “script2-sed” :

```
/([[:digit:]].){3}[[:digit:]]@lo/a\bizarre ce Nono
```

Dès lors qu'on utilise l'option -f, l'option -e n'est plus utilisable puisque le script enchaîne les commandes en les plaçant chacune sur une ligne.
Mais l'option -r doit être utilisable puisqu'il peut y avoir dans le script sed une commande dont l'adressage est une *E.R.* étendue.

De même l'option -i doit être utilisable puisque un script sed doit servir à modifier un fichier texte !
Cela se fait ainsi :

```
sed -rf script2-sed -i sed-options
```

Pour visualiser, utilisation d'une commande d'affichage (**cat** par exemple) même si -n n'a pas été utilisée[2)]
Attention l'utilisation de l'option -n est plus qu'inutile !

En l'utilisant, il ne resterait du fichier que ce qui lui aurait été ajouté.
La note ci-dessus en donne exemple.

```
cat sed-options
```

```
Il est vraiment idiot,
se balade sur un vélo
sans pédale, et porte ses sandales
à son goulot.
il veut pour adresse mail 127.0.0.1@lo
bizarre ce Nono
```

Si on ajoute l'option `-n` ;

```
sed -rnf script2-sed -i sed-options
```

on obtient alors :

```
cat sed-options
```

```
bizarre ce Nono
```

La ligne de commandes de “script2-sed”, matche la ligne “il veut pour adresse mail 127.0.0.1@lo”, du fait de l'option `-n`, ni la ligne qui correspond à la *E.R*, ni les autres lignes du fichier ne sont placées dans le tampon de travail de sed ; puis sed applique la commande d'ajout de “script2-sed” ; et inscrit enfin ce résultat dans le fichier “sed-options”.
Par conséquent il ne reste plus dans le fichier que l'ajout “bizarre ce Nono”.

Les rapports d'implication /adressages/ <=> commandes de sed

Analyse de l'implication : `/adressages/` ⇔ `cmd_de_sed`

- Sans adressage, sed applique chaque commande à l'intégralité des lignes du fichier à traiter.
Cela implique donc un rapport entre ce que fait une commande interne et le type d'adressage qu'il est possible d'utiliser pour telle commande.
Autrement dit, ce que permet une commande interne détermine les types d'adressages possibles, adressage simple ou double d'abord.

- De plus, certaines commandes indiquent en elles-mêmes, non pas le type d'adressage, mais un adressage précis, comme par exemple `N`.

Ainsi :

1. Certains adressages sont spécifiques à certaines commandes seulement.
2. Avec la commande s il apparaît l'adressage du numéro d'occurrence sur une ligne.[3)]

- Par ailleurs, l'adressage étant tributaire des commandes internes, on recense deux caractères spéciaux d'adressage.

1. le caractère d'exclusion de l'adressage (!)
2. le caractère d'adressage de la dernière ligne ($).[4)]
   Tous deux sont des caractères spéciaux des commandes internes de sed, et se placent donc devant la commande interne, jamais dans l'adressage !

Les adressages universaux

Il s'agit des adressages qui fonctionnent avec toutes les commandes de sed utilisables en ligne de commandes.

le caractère d'exclusion (!)

sed-options

```
Il est vraiment idiot,
se balade sur un vélo
sans pédale, et porte ses sandales
à son goulot.
il veut pour adresse mail 127.0.0.1@lo
```

- Exemple : afficher toutes les lignes sauf celle de la *E.R*

```
sed -rne '/([[:digit:]].){3}[[:digit:]]@lo/!p' sed-options
```

ou 

```
sed -rne '$!p' sed-options
```

```
Il est vraiment idiot,
se balade sur un vélo
sans pédale, et porte ses sandales
à son goulot.
```

le caractère d'adressage de dernière ligne ($)

- Exemple : afficher la dernière ligne de “sed-options”

```
sed -rne '$p' sed-options
```

```
il veut pour adresse mail 127.0.0.1@lo
```

Adressage simple

- un numéro de ligne (les lignes sont numérotées à partir du chiffre 1);
- le caractère $ désignant la dernière ligne;
- par l'encadrement /ER/cmd : une expression régulière encadrée par des caractères /;

Attention avec l'adressage simple !

- Avec l'adressage simple par numéro de ligne, seule la ligne indiquée par le numéro est traitée.
- Avec l'adressage simple par l'encadrement /, toutes les lignes qui correspondent à /expression régulière/ sont traitées .

Commandes de sed supportant les adressages universaux

i\texte	insérer du texte
a\texte	ajouter du texte

Préparation

sed-automn.txt

```
automne
mon bel automne
dans les vergers
du fond du ciel
les éperviers
de tes yeux planent
tes feuilles
tes fruits, ta pomme
et le cerf, et son brame
à jamais aimés
de l'homme
en ton cheveu
me donnent
et des langueurs
et son adieu
```

"i" : insertion avant la ligne d'adressage

```
sed '1i\Ton sentiment comme atmosphère' sed-automn.txt
```

> Le titre “Ton sentiment comme atmosphère” est ajouté sur la sortie standard.

"a" : ajout après l'adressage

```
sed -e '1a\ ' -e '2,$s/^/ /' sed-automn.txt
```

> Pour ajouter une ligne vide en deuxième ligne avec la commande (a), il faut ajouter au moins un caractère, ici, “espace”.
> Pour désigner une ligne précise aux commandes qui suivent une commande d'ajout de ligne, il peut être plus simple d'utiliser l'adressage par encadrement (par exemple ici, `/^mon/,$s/^/ /` , ou `2,$s/^/ /`).

Utiliser conjointement i et a

Modifier le fichier “sed-automn.txt” :

- en insérant le titre “Ton sentiment comme atmosphère”
- en insérant une ligne ne comportant qu'un espace,
- en ajoutant un espace à partir de la ligne `/^automne/`

```
sed -e '1i\Ton sentiment comme atmosphère'\
 -e '/^automne/i\ '\
 -e '1,$s/^/ /' -i sed-automn.txt
```

On aurait pu utiliser la ligne de commande qui suit, pour obtenir le même résultat.

```
sed -e '1i\Ton sentiment comme atmosphère'\
 -e '1i\ '\
 -e '1,$s/^/ /' -i sed-automn.txt
```

> Il est à noter quels ont été les adressages utilisés pour obtenir ce résultat, par rapport à ceux qui avaient été utilisés précédemment
> (pour les exemples où les commandes i et a étaient utilisées seules).
> Voir la prochaine note.

Résultat

L'option `-i` a été utilisée, le fichier “sed-automn.txt” est maintenant :

```
cat sed-automn.txt
```

```
Ton sentiment comme atmosphère

 automne
```

```
mon bel automne
dans les vergers
du fond du ciel
les éperviers
de tes yeux planent
tes feuilles
tes fruits, ta pomme
et le cerf, et son brame
à jamais aimés
de l'homme
en ton cheveu
me donnent
et des langueurs
et son adieu
```

Attention !

Si on utilise plusieurs commandes d'ajout, la charge de la mémoire tampon est si importante que sed se réfère à la numérotation des lignes du fichier d'origine pour l'adressage par numéro de ligne de toutes les commandes!

Ainsi cette ligne de commandes sed :

```
sed -e '1i\Ton sentiment comme atmosphère'\
 -e '2i\ '\
 -e '2,$s/^/ /' sed-automn.txt
```

```
Ton sentiment comme atmosphère
automne

 mon bel automne
#  J'ai  supprimé  le  reste  du  retour  qui  ne  sert  pas
l'explication.
```

Et celle-ci :

```
sed -e '1i\Ton sentiment comme atmosphère'\
 -e '2i\ '\
 -e '3,$s/^/ /' sed-automn.txt
```

```
Ton sentiment comme atmosphère
automne

mon bel automne
 dans les vergers
#  J'ai  supprimé  le  reste  du  retour  qui  ne  sert  pas
l'explication.
```

Ce point est très important pour la création de scripts sed plus complexes.
Il faut toujours considérer le contenu de la mémoire tampon pour enchaîner plusieurs commandes.

Adressage double

C'est une paire d'adresses simples séparées par une virgule. Toutes les combinaisons sont possibles.

- deux numéros de ligne **n,m**cmd_sed : (**n** et **m** sont des nombres) l'analyseur de sed traite la plage de lignes commençant de la ligne **n** jusqu'à la ligne **m** .
- deux encadrements **/ER1/,/ER2/** : l'analyseur de sed traite la plage de lignes commençant à partir de la première ligne reconnue par *ER1* jusqu'à la ligne reconnue par *ER2*, en prenant le nombre de lignes le plus large[5)]
- combinaison **n,/ER/** : La plage est constituée à partir de la ligne numéro **n** , jusqu'à la première ligne reconnue par le *ER*
- combinaison **/ER/,m** : La plage est constituée à partir de la première ligne vérifiant *ER* jusqu'à la ligne de numéro **m** .

Les commandes de sed supportant les adressages universaux et l'adressage double

Commandes	Significations
d	supprimer
p	afficher avec -n
=	afficher n° de ligne(s)
y/..../..../	translittération
c\texte	changement de texte

"p" : affichage

```
sed -n '/.*,[[:space:]]/,9p' sed-automn.txt
```

```
 tes fruits, ta pomme
 et le cerf, et son brame
```

"d" : suppression

```
sed -e '4,6d' -e '8,14d' sed-automn.txt
```

```
Ton sentiment comme atmosphère

 automne
 les éperviers
 me donnent
 et des langueurs
```

```
 et son adieu
```

"=" numérotation

```
sed '1,$=' sed-automn.txt
```

```
1
Ton sentiment comme atmosphère
2

3
 automne
4
 mon bel automne
5
 dans les vergers
6
 du fond du ciel
7
 les éperviers
8
 de tes yeux planent
9
 tes feuilles
10
 tes fruits, ta pomme
11
 et le cerf, et son brame
12
 à jamais aimés
13
 de l'homme
14
 en ton cheveu
15
 me donnent
16
 et des langueurs
17
 et son adieu
```

Pour n'avoir que la numérotation, on peut utiliser l'option `-n` :

```
sed -n '=' sed-automn.txt
```

"y" : translittération

```
sed '3,4y/a/A/' sed-automn.txt
```

```
Ton sentiment comme atmosphère

 Automne
 mon bel Automne
 dans les vergers
 du fond du ciel
 les éperviers
 de tes yeux planent
 tes feuilles
 tes fruits, ta pomme
 et le cerf, et son brame
 à jamais aimés
 de l'homme
 en ton cheveu
 me donnent
 et des langueurs
 et son adieu
```

"c" : changement de texte

```
sed -e '5,$c\
 fleuve Léthé\
 en ton grand coeur\
 devient Vestale\
' sed-automn.txt
```

```
Ton sentiment comme atmosphère

 automne
 mon bel automne
 fleuve Léthé
 en ton grand coeur
 devient Vestale
```

Toutes ces commandes regroupées en un script

- Créer un script qui utilisera toutes ces commandes
- pour transformer le fichier “sed-automn.txt”

Le script

```
/.*,[[:space:]]/,9p
```

```
4,6d;8,14d
#1,$=
3,4y/a/A/
5,$c\
 fleuve Léthé\
 en ton grand coeur\
 devient Vestale
```

> Le caractère # commente la ligne.

Lancer le script avec l'option -i

```
sed -f script-sed.poesie -i sed-automn.txt
```

```
Ton sentiment comme atmosphère

 Automne
 tes fruits, ta pomme
 et le cerf, et son brame
 fleuve Léthé
 en ton grand coeur
 devient Vestale
```

La commande "s"

Syntaxe globale

```
s/motif/substitut/[options_de_s]
```

À voir : la-fonction-de-substitution-s

Détail : "motif" et "substitut"

```
s/motif/substitut/
    -> si motif = une E.R
    -> alors subtitut = une chaîne de caractère
```

> Pour *E.R*
> On peut utiliser la syntaxe habituelle, ou la syntaxe étendue avec l'option `sed -r`.
> Voir : sed-et-les-expressions-rationnelles

Détail : la ligne de commande "s"

```
  s/motif/substitut/[options_de_s]
```

les séparateurs (/)

```
echo "aaaa BBB cCcC # 12345" > ~/sed1.txt
```

```
sed -i 's+ +\n+g' ~/sed1.txt && cat sed1.txt
```

```
aaaa
BBB
cCcC
#
12345
```

Tout cela revient au même

```
echo "aaaa BBB cCcC # 12345" | sed 's+ +\n+g'
```

```
echo "aaaa BBB cCcC # 12345" | sed 's/ /\n/g'
```

```
echo "aaaa BBB cCcC # 12345" | sed 's_ _\n_g'
```

Détail : "options-de-s" ("flags")

g	global : toutes les occurrences
n	la nième occurrence d'une ligne
p	afficher la ligne modifiée
w	écrire les substitutions effectuées dans un fichier
e	exécution d'une commande
I	ignorer la casse
M	correspondance de E.R. (implémentation GNU, non portable)

g ; n et p

- substituer toutes les occurrences de motif

sed-coincoin.txt

```
coin
coin
foobar
barbar
```

```
coincoin babar coincoin
```

```
sed 's/coincoin//g' sed-coincoin.txt
```

```
coin
coin
foobar
barbar
 babar
```

> Sans g, seul le “coincoin” à gauche de “babar” aurait été substitué par un espace.

- Substituer l'occurrence d'une ligne

```
sed '/babar/s/coin//3' sed-coincoin.txt
```

```
coin
coin
foobar
barbar
coincoin babar coin
```

- Afficher sur la sortie standard la substitution :

```
var="ligne1\nligne2\nligne3\nligne\nligne5"
```

```
echo -e "$var" | sed '4 s/e/e4/p'
```

```
ligne1
ligne2
ligne3
ligne4
ligne4
ligne5
```

> Le doublon est dû à l'affichage du cache, auquel s'ajoute l'affichage de la substitution.
> Il faut utiliser l'option sed -n :

```
echo -e "$var" | sed -n '4 s/e/e4/p'
```

```
ligne4
```

w écrire

```
sed -n 's/barbar/babar/pw fichier-barbar' sed-coincoin.txt
```

```
babar
```

```
cat fichier-barbar
```

```

babar
```

e : exécution d'une commande

```
sed 's/barbar/date/e' sed-coincoin.txt
```

```
coin
coin
foobar
lundi 28 juillet 2014, 12:32:20 (UTC+0200)
coincoin babar coincoin
```

I : ignorer la casse

```
var="COINcoin\nbabar\ncoinCOIN\nbabar\ncoincoin"
```

```
echo -e "$var" | sed 's/coincoin/tointoin/Ig'
```

```
tointoin
babar
tointoin
babar
tointoin
```

Les indicateurs spéciaux

&	chaîne à remplacer
(\n)	sous-chaîne

Le caractère &

& : Représente la chaîne à remplacer.
Si le signe & est présent dans le motif de remplacement, alors, il sera substitué par le motif de recherche en entier.

sed1.txt

```
aaaa
BBB titi
cCcC
#
```

```
12345
BBB
titi cCcC
titi BBB
/coucou/titi
```

- Entourer de crochet la chaîne “12345”

```
sed 's/[0-9][0-9]*$/[&]/' sed1.txt
```

```
aaaa
BBB
cCcC
#
[12345]
BBB
titi cCcC
```

- Faire ensorte que le fichier “sed1.txt” ne comporte deux lignes “titi grominet”

```
sed -re 's/titi/& grosminet/' -e\
 '/titi/!d' -e\
 's/(BBB|cCcC)//' -e\
 '/^ /d' -e\
 '/^\/.*/d' -i\
 sed1.txt
```

```
cat sed1.txt
```

```
titi grosminet
titi grosminet
```

l'indicateur de sous-chaîne (\n)

n est le numéro de champ d'une partie du motif de recherche, chaque partie du motif de recherche étant délimitée ainsi `\(sous-motif\)`.

- Exemple inverser la deuxième ligne de “sed1.txt”

```
sed -e '2s/\(titi\) \(grosminet\)/\2 \1/g' sed1.txt
```

```
titi grosminet
grosminet titi
```

Des adressages spéciaux

La commande s supporte les adressages universaux.
Elle permet en plus de nouveaux types d'adressage.

Remarque sur l'adressages mixtes : 'n,/E.R./' ou '/E.R./,n'

1) 'n,/E.R./s/chaineA/chaineB'

n est un nombre entier. On remplace chaîneA par chaîneB de la ligne n à la ligne comportant /E.R./.

- Préparation : fichier “sed1.txt

```
AAAA 2 fraise
3 BBB orange
AAAA 6 cerise
BBB 1 abricot
9 AAAA pomme
BBB 5 poire
```

- Substitution de la ligne n° 2 jusqu'à la ligne de chaîne “cerise”

```
sed '2,/cerise/s/[[:digit:]]/GAGA/' sed1.txt
```

```
AAAA 2 fraise
GAGA BBB orange
AAAA GAGA cerise
BBB 1 abricot
9 AAAA pomme
BBB 5 poire
```

2) '/E.R./ns/chaîneA/chaîneB/'

- Substitution à partir de la chaîne “orange” jusqu'à la ligne n° 5

```
sed '/orange/,5s/[[:digit:]]/GAGA/' sed1.txt
```

```
AAAA 2 fraise
GAGA BBB orange
AAAA GAGA cerise
BBB GAGA abricot
GAGA AAAA pomme
BBB 5 poire
```

Il faut trois “genres d'objets”. Car si /E.R./ d'adressage correspond à chaîne, alors l'indication n n'arrêtera pas la substitution.

```
sed '/[[:digit:]]/,3s/[[:digit:]]/GAGA/' sed1.txt
```

> Cette ligne ne remplacera pas un chiffre par “GAGA” de la première ligne où il y a un chiffre à la ligne 3.

```
AAAA GAGA fraise
GAGA BBB orange
AAAA GAGA cerise
BBB GAGA abricot
GAGA AAAA pomme
BBB GAGA poire
```

Adressage avec n° d'occurrence (fonctionne sur une ligne)

```
echo "lalalala" | sed -e 's/la/li/2'
```

```
lalilala
```

Adressage du retour chariot

Par exemple, pour matcher une chaîne constituée de deux chaînes égales séparées par un retour à la ligne, du fichier essai :

```
abcd
abcd
coucou coucou
```

```
  sed -r 'N;s/^(.*)\n\1$/X/g' essai
```

```
X
coucou coucou
```

Adressage avec le tilde

man sed

```
first~step
 Match every step' th line starting with line first.
```

Remplaçons “first” et step” par “n” et “m” qui sont toujours des nombres !
L'expression `n ~ m` ou `n~m` signifie simplement :

La E.R sélectionne toutes les nième ligne(s) d'un fichier en sautant un nombre de ligne(s) égal à un écart (~).
Cet écart est égal à la différence entre le nombre `m` avec la première ligne du fichier (c'est-à-dire la position ordinal 1).

On connaît donc le nombre de ligne(s) sautée(s) en faisant toujours `m - 1`.

Et si n est égal à zéro, n devient la position ordinale 1 du fichier, ainsi que l'adressage de début. Donc la *E.R.* sélectionne toutes les `m`èmes lignes à partir de la ligne `n`, et saute m - 1 ligne(s).

- `1 ~ 2` : sélection des lignes impaires : 1ière, 3ième, 5ième, etc.
  En effet, de la première ligne du fichier jusqu'à `m`, il y a 1 (m = 2 et 2 - 1 =1).
  Il sera sauté 1 ligne (non-sélectionnée) et cela à partir de la ligne n°1 (n = 1 = la position ordinale).
  Donc sélection de la première ligne, saut d'une ligne, sélection de la 3ième ligne, saut 1 ligne, etc.
- `2 ~ 3` : sélection les lignes 2 ; 5 ; 8 ; 11
  L'écart sauté sera de 2 car l'écart entre la position 3 (m = 3) et la première ligne du fichier, est de 2 lignes (3 - 1 = 2), et cela commence à la deuxième ligne (n = 2).
  Donc la deuxième ligne est sélectionnée, deux lignes sont sautées (3ième et 4 ième), la 5 ième est sélectionnée, etc.
- `0 ~ 5` : sélection des lignes 5 ; 10 ; 15 etc.
  L'écart est égal à m - 1 = 5 - 1 = 4 ; donc 4 lignes sautées, à partir de la cinquième ligne (m = 5).

- Exemple 1:

```
sed '1~2 s_.*[coucou]_OK_' essai-tilde
```

```
OK/1
/titi/coucou/2
OK/3
/tutu/coucou/4
OK/5
/nono/coucou/6
OK/7
/lili/coucou/8
OK/9
/fofo/coucou/10
OK/11
/bibi/coucou/12
OK/13
/sisi/coucou/14
OK/15
```

- Exemple 2 :

```
sed '2~3 s_.*[coucou]_OK_' essai-tilde
```

```
/toto/coucou/1
OK/2
/tati/coucou/3
```

```
/tutu/coucou/4
OK/5
/nono/coucou/6
/lulu/coucou/7
OK/8
/fifi/coucou/9
/fofo/coucou/10
OK/11
/bibi/coucou/12
/zaza/coucou/13
OK/14
/baba/coucou/15
```

- Exemple 3 :

```
sed '10~5 s_.*[coucou]_OK_' essai-tilde
```

```
/toto/coucou/1
/titi/coucou/2
/tati/coucou/3
/tutu/coucou/4
/nani/coucou/5
/nono/coucou/6
/lulu/coucou/7
/lili/coucou/8
/fifi/coucou/9
OK/10
/dodu/coucou/11
/bibi/coucou/12
/zaza/coucou/13
/sisi/coucou/14
OK/15
```

- Exemple 4 :

```
sed '0~5 s_.*[coucou]_OK_' essai-tilde
```

```
/toto/coucou/1
/titi/coucou/2
/tati/coucou/3
/tutu/coucou/4
OK/5
/nono/coucou/6
/lulu/coucou/7
/lili/coucou/8
/fifi/coucou/9
OK/10
/dodu/coucou/11
/bibi/coucou/12
/zaza/coucou/13
```

```
/sisi/coucou/14
OK/15
```

Commandes indiquant en elle-même un adressage sed

: Commentaire (Aucune adresse autorisée)

Le caractère dièse (#) commence un commentaire et ce jusqu'à la fin de la ligne.
Dans un script sed, il peut se trouver sur la même ligne qu'une commande, elle ne sera pas prise pour un commentaire.
Si les deux premiers caractères d'un script sed sont ”#n“, l'option ”-n“ (no-autoprint) est alors forcée.

q quit : Quitter (une adresse autorisée)

Cette commande permet de quitter sed après le travail sur l'adressage en cours d'exécution.
Elle est utile quand on utilise sed sur de gros fichiers.

Exemple sans commande "q"

reprise du “sed-automn.txt” et script1 contenant :

```
4,5p
8!d
```

```
sed -f script1 sed-automn.txt
```

```
 mon bel automne
 dans les vergers
 de tes yeux planent
```

> Dans ce cas, sed affiche de la ligne 4 à la ligne 5.
> Puis la commande d supprime tout sauf les lignes traitées par p et d.

Utilisation de la commande ''q''

Reprise du fichier “sed-automn.txt” et script2 contenant :

```
# 4,5p
# 8!d

6q;4,5p;8!d
```

```
sed -nf script2 sed-automn.txt
```

```
mon bel automne
dans les vergers
```

> Dans ce cas, la commande q adresse les 6 premières lignes dans la mémoire principale de sed, qui peut alors les utiliser une fois, donc pour une seule commande après q.
> En effet, si l'adressage de la commande suivante “déborde” de celui de la commande q, alors sed affiche la ligne que la commande q a adressée à la mémoire principale.

```
sed '2q;4,5p' sed-automn.txt
```

```
Ton sentiment comme atmosphère
#la deuxième ligne est une ligne vide
```

> Utiliser sans adressage elle affiche la première ligne du fichier ; sed enregistre au moins une ligne pour traiter une commande sans adressage du fichier.

la commande n (next-line)

À ne pas confondre avec l'option de la ligne de commande sed.
La commande n déplace l'adressage de l'espace de travail d'une ligne.

Affichage de lignes sans la commande n

```
echo -e "1A\n2B\n3C" | sed -ne '2p'
```

```
2B
```

Affichage de lignes avec la commande n

- adressage simple :

```
echo -e "1A\n2B\n3C" | sed -n '2 {n;p}'
```

```
3C
```

- adressage double :

```
echo -e "1A\n2B\n3C\n3D" | sed -n '1,3 {n;p}'
```

```
2B
3D
```

la commande { }

Elle permet d'indiquer un seul adressage pour plusieurs commandes.

- Elle s'utilise en ligne de commande voir l'exemple de la commande précédemment considérée ;

```
sed [options] '/adressage/ {cmd1;cmd2;...}'
```

- Elle s'utilise aussi dans un script sed ;

```
adressage{
  cmd1
  cmd2
  ...
}
```

- Elle permet l'imbrication d'adressage ;

```
adressage1{
  adressage1A cmd1A
  adressage1B cmd1B
  ...
  adressage2{
  cmd2A cmd2A
    ...
  }
}
```

La commande r "fichier lu" (read)

Elle adresse à sed un fichier et l'insère dans le fichier de destination.

```
sed '/adressage-pour fichier-modifié/ r fichier-mofifié
```

Il ne doit y avoir qu'un seul espace entre la commande et le nom du “fichier lu”.
Il faut donner le chemin absolu du “fichier lu” si le terminal ne se trouve pas dans le répertoire parent de ce “fichier lu”.

Préparation

- soit le fichier “sed-fich-lu.txt” :

```
texte du fichier lu
deuxième ligne
```

- soit le fichier “sed-coincoin.txt” :

```
coin
coin
foobar
barbar
```

Ajouter le contenu de "sed-fich-lu.txt" dans "sed-coincoin.txt"

```
sed '1,3r sed-fich-lu.txt' sed-coincoin.txt
```

```
coin
texte du fichier lu
deuxième ligne fichier lu
coin
texte du fichier lu
deuxième ligne fichier lu
foobar
texte du fichier lu
deuxième ligne fichier lu
barbar
```

La commande w fichier (write)

Écrit la ligne en cours de traitement dans le fichier spécifié à la suite de la commande “w”.
Le fichier sera créé, même si le traitement est nul en sortie.
Tout comme la commande “r” (lecture), il ne doit y avoir qu'un seul espace entre la commande et le nom du fichier à créer.
Si un fichier du même nom existe déjà, il sera écrasé sans avertissement ni confirmation et ce à chaque invocation du script.
Si plusieurs instructions de la commande “w” sont appelées à écrire dans un même fichier depuis un script, chaque écriture est ajoutée à la fin du fichier.

créer un fichier avec sed et echo :

```
echo "contenu d'un fichier créé" | sed 'w fichier-sed-cree'
```

```
contenu d'un fichier créé
```

```
cat fichier-sed-cree
```

```
contenu d'un fichier créé
```

Les commandes multi-lignes

On les utilise avec d'autres commandes.

Commandes	Significations
N	nouvelle ligne
D	supprimer (en rapport à \n)
P	afficher (en rapport à \n)

La commande N

La commande “N” insère le caractère “nouvelle ligne” (\n) à la fin du contenu de l'espace de travail et ajoute la ligne suivante du flux d'entrée à l'espace de travail.
Une fois le caractère “nouvelle ligne” dans l'espace de travail, il peut être appelé par \n par une commande interne de sed qui peut alors substituer un caractère une nouvelle ligne par un espace.

Exemples

- Sans adressage :

```
echo -e "coucou\ntiti" | sed '/coucou/{N;s/\n/ /}'
```

```
coucou titi
```

> Là, pour la ligne comportant “coucou”, s se sert du caractère de fin de ligne placé dans l'espace de travail pour pouvoir matcher ce retour à la ligne avec “echo”.
> Elle peut alors le remplacer par un espace entre “coucou” et “titi”.

- Avec adressage :

```
echo -e "A\nB\nC\nD\nE" | sed '/B/{N;s/\n/ /}'
```

```
A
B C
D
E
```

C'est l'inverse de ceci !

```
echo "coucou titi" | sed 's/ /\n/'
```

```
coucou
titi
```

Bien penser dans la grammaire de sed ! S'il y a plus deux mots séparés d'espaces, donc plus de deux espaces à substituer en autant de saut de lignes,
il faudrait ajouter l'option “g” de la commande s.
Sinon, sed affiche le résultat et sort après avoir trouvé la première occurrence d'un espace.

D (Delete)

La commande efface l'espace de travail jusqu'à un caractère délimitant une nouvelle ligne.
On en sert en association avec la commande N pour supprimer un saut de ligne consécutif à un saut

de ligne ;
autrement dit, pour remplacer plusieurs sauts de ligne par un seul saut de ligne.

Rappel :

- `/^$/` : une ligne vide
- `/^\n$/` : une ligne comportant un saut de ligne

Préparation

- Soit le fichier “sed-D.txt”

sed-D.txt

```
ligne suivie de deux sauts de ligne


fin
```

Exemple

```
sed '/^$/{N;/^\n$/D;}' sed-D.txt
ligne suivie de deux sauts de ligne

fin
```

> La première ligne vide est validée par se correspondance avec /^$/ ;
> puis, N rend utilisable le caractère de nouvelle ligne (\n);
> puis D supprime de l'espace de travail la ligne vide /^\n$/D enregistrée précédemment;
> puis il passe à la ligne suivant qui est aussi une ligne vide, il fait le même travail;
> enfin, il passe à la ligne “fin” qui ne correspond pas à /^$/, et affiche le résultat avant de sortir.
> Ainsi il peut supprimer ligne à ligne autant de lignes vides successives qu'il y en aurait, afin que toutes les lignes du texte ne soient espacées au plus que d'un seul saut de ligne.

La commande P (print)

La commande “P” affiche le contenu de l'espace de travail jusqu'au premier caractère délimitant une nouvelle ligne (\n).

```
echo -e "ligne1 ligne2 ligne3" | sed '{P;s/ /\n/g}'
```

```
ligne1 ligne2 ligne3
ligne1
ligne2
```

```
ligne3
```

> La commande P affiche le contenu de l'espace de travail “ligne1 ligne2 ligne3” sur lequel elle applique la substitution de la fin de l'espace en nouvelle ligne \n.

Toutes commandes présentées jusqu'ici,

- celles regroupées en fonction des adressages universaux
- la commande ''s''
- les commandes adressant en-elle-même un adressage à sed
- Les commande fonctionnant en rapport avec d'autres commandes

utilisent principalement le mécanisme suivant :

> Lecture d'une ligne du fichier d'entrée dans l'espace de travail à laquelle est appliquée chaque commande du script séquentiellement.
> Lorsque la fin d'une ligne de commande sed est atteinte, la ligne enregistrée et travaillée est alors envoyée sur la sortie standard,
> puis l'espace de travail est effacé ;
>
> Autrement dit, leur fonctionnement se base sur une utilisation simple du *“pattern space”* à partir duquel un travail ligne à ligne est effectué.

La notion de “pattern space”

C'est la mémoire principale (tampon) qui permet à sed de travailler ; on la nomme aussi “espace de travail”. C'est là que sont enregistrées les données.

Mais sed possède deux sortes de mémoire tampon. La deuxième est appelée “espace annexe” ou “hold space”.
Cette dernière est un espace mémoire où les données (la ou les ligne(s)) peuvent être enregistrées provisoirement.
Certaines commandes permettent de modifier l'adressage des lignes tel qu'il est enregistré dans l'une de ces mémoires.
D'autres, permettent aussi un échange de données d'une mémoire à l'autre.

Il reste à présenter :

- Les commandes utilisant la mémoire annexe (h,H,g,G,x)
- Les commandes de tests faisant appel à des étiquettes (:,b,t,T)

---

1)

y ; s ; a\ ; i\ ; p ; d ; c\ ; w ; q ...

2)

Lors de l'utilisation d'un script sed sur un fichier, pour ce qui concerne les commandes d'insertion (i), d'ajout (a) et de changement (c), fort heureusement, par défaut, l'affichage sur la sortie standard du tampon de travail de sed est annulé.

Ce n'est bien sûr pas le cas pour la commande de suppression `-s`.
3)

Les deux derniers points seront précisés au paragraphe ci-après, “commandes en fonction de leur adressage”.
4)

Ne pas confondre le caractère spéciale $ de commandes sed, avec $ des *E.R.*, et le caractère ^ placé sur une commande pour signifier “première ligne”, n'existe pas !
5)

c'est-à-dire, qu'il peut y avoir une ligne avec ER1 dans la plage **`n,/ER/`**.