Module administration système et réseau GNU/Linux

Les commandes de base / deuxième partie

Sommaire

Pré-requis

- Avoir une VM avec une distribution GNU/Linux

Objectifs

- Savoir rechercher.
- Savoir filtrer et rediriger les sorties de commandes

A. Rappel

1. L'interpréteur de commande

Un **interpréteur de commande** est un programme appelé **shell**.

L'interpréteur de commande permet d'exécuter des instructions que vous saisissez au clavier ou au sein d'un script et vous en retourne l'accès.

Deux méthodes d'accès au **shell** sont possibles sous GNU/Linux :

- Le **mode console** qui affiche un shell unique en plein écran, c'est l'interface utilisateur de base du système d'exploitation. Sous GNU/Linux les consoles sont en génral au nombre de six par défaut;
- Le **mode terminal** qui émule une console et qui affiche en général le shell dans une fenêtre.

Sous GNU/Linux Il existe différents shells : sh (Bourne Shell), ksh (Korn Shell), Bash (Bourne Again Shell)

Le shell fournit par défaut avec les distributions GNU/Linux répandues est Bash.

2. L'invite de commande

Le **shell** attend des entrées au clavier sur une ligne appelée l'**invite de commande** ou **prompt**.

Le prompt fournit en général des informations sur le terminal et votre position dans le système de fichiers.

Premier exemple :

```
stag@linuxserver:/srv/Documents$
```

Décomposons cette ligne :

- stag : c'est le nom de connexion (login) actuellement connecté au terminal.
- linuxserver : c'est le nom d'hôte (hostname) de la machine raccordé au terminal.
- /srv/Documents : c'est la position actuelle du shell dans l'arborescence.
- « $ » : c'est la terminaison standard du bash pour un utilisateur sans pouvoir.

Deuxième exemple :

```
stag@linuxserver:~$
```

- « ~ » : indique que l'utilisateur est dans son répertoire personnel qui est par défaut /home/*login* (ici /home/stag).

Le cas de l'utilisateur « root » : à la place du symbole « **$** », la terminaison du bash sera le symbôle « **#** »

```
root@linuxserver:~#
```

3. Utiliser le shell

Le clavier s'utilise comme d'habitude. On saisit une commande et on la valide par entrée.

Les flèches droite et gauche du clavier permettent de se déplacer sur la ligne de commande.

Les flèches haut et bas permettent de naviguer dans l'historique des commandes saisies précédemment.

Raccourcis-clavier utiles :

- **[CTRL] D** : permet de quitter le terminal.
- **[CTRL] L** : efface le contenu du terminal.
- **[CTRL] U** : efface la ligne de la position où se situe le curseur sur la ligne de commande jusqu'au début.
- **[CTRL] K** : efface la ligne de la position où se situe le curseur sur la ligne de commande jusqu'à la fin.
- **[CTRL] A** : Amène le curseur au debut de la ligne.
- **[CTRL] E** : Amène le curseur en fin de ligne.

L'auto-complétion est également d'une aide précieuse.

- Elle permet de compléter les commandes, les chemins saisis ou les noms de fichiers.
- Un appui sur la **touche TAB** complète la saisie dans le cas d'une seule solution.
- Sinon, il faudra faire un deuxième appui pour obtenir la liste des possibilités.

Si un double appui sur la touche TAB ne provoque aucune réaction de la part du système, c'est qu'il n'existe aucune solution à la complétion en cours.

B. Les caractères de substitution (wildcards/métacaractères)

Lors de l'utilisation de commandes en rapport avec le système de fichiers, il peut devenir intéressant de filtrer la sortie de noms de fichiers à l'aide de certains critères, par exemple avec la commande **ls**.

Caractère(s)	Rôle
*	Remplace une chaîne de longueur variable, même vide.
?	Remplace un caractère unique quelconque.
[…]	Une série ou une plage de caractères.
[a-b]	Un caractère parmi la plage indiquée (de a à b inclus).
[!…]	Inversion de la recherche.
[^…]	Idem.

Au lieu d'afficher toute la liste des fichiers, on peut filtrer l'affichage à l'aide de divers critères et caractères spéciaux.

Soit le contenu suivant :

```
$ ls
afic   afic2  bfic   bfic2  cfic   cfic2  dfic   dfic2
afic1  afic3  bfic1  bfic3  cfic1  cfic3  dfic1  dfic3
```

Obtenir tous les fichiers commençant par a :

```
$ ls a*
afic1  afic2  afic3
```

Tous les fichiers de quatre caractères commençant par a :

```
$ ls a???
afic
```

Tous les fichiers d'au moins trois caractères et commençant par b :

```
$ ls b??*
bfic  bfic1  bfic2  bfic3
```

Tous les fichiers finissant par 1 ou 2 :

```
$ ls *[12]
afic1  afic2  bfic1  bfic2  cfic1  cfic2  dfic1  dfic2
```

Tous les fichiers commençant par les lettres de a à c, possédant au moins un second caractère avant la terminaison 1 ou 2 :

```
$ ls [a-c]?*[12]
afic1  afic2  bfic1  bfic2  cfic1  cfic2
```

Tous les fichiers ne finissant pas par 3 :

```
$ ls *[!3]
afic  afic1  afic2  bfic  bfic1  bfic2  cfic  cfic1  cfic2  dfic  dfic1  dfic2
```

C. Rechercher des fichiers

1. Commande find

La commande find recherche l'emplacement d'un fichier.

Syntaxe de la commande find

```
$ find répertoire_où_rechercher -name nom_du_fichier_recherché
```

Exemple : Rechercher le fichier fic.txt dans le répertoire Documents

```
$ find Documents -name "fic.txt"
Documents/rep2/fic.txt ← le chemin du fichier relatif au répertoire courant est renvoyé
```

Si le répertoire de recherche n'est pas précisé, la commande **find** cherchera à partir du répertoire courant.

Il est possible d'utiliser l'option **-exec** de la commande **find** pour exécuter une commande à chaque ligne de résultat :

```
$ find /tmp -name "*.txt" -exec rm -f {} \; (respectez les espaces)
```

La commande précédente recherche tous les fichiers du répertoire /tmp se finissant par « ***.txt*** » et les supprime (**Note** : si vous utilisez des wildcards comme l'étoile dans ce cas, vous devez mettre le motif recherché entre guillemets).

Comprendre l'option -exec

Dans l'exemple ci-dessus, la commande **find** va construire une chaîne de caractères représentant la commande à exécuter.

Si la commande **find** trouve trois fichiers nommés log1.txt, test.txt et hjksdf.txt, alors la commande find va construire la chaîne en remplaçant dans la chaîne « rm -f {} \; » les accolades par un des résultats de la recherche, et cela autant de fois qu'il y a de résultats.

Ce qui nous donnerait l'équivalent de :

```
$ rm -f /tmp/log1.txt ; rm -f /tmp/log2.txt ; rm -f /tmp/log3.txt ;
```

Plutôt pratique si vous avez des centaines de fichiers, voire des milliers à supprimer.

Le caractère "**;**" est un caractère spécial du shell qui doit être protégé par un "\" pour éviter son interprétation trop tôt par la commande find.

On peut aussi remplacer **-exec** par **-ok** ce qui permettra de demander une confirmation avant suppression pour chaque résultat retourné dans le cas d'un « rm ».

La commande **find** peut valoir un cours à elle toute seule. Nous avons vu un premier **critère** « -name ». Nous pouvons combiner celui-ci avec d'autres critères (critères courants):

Option	Paramètre	Définition
-name	nom fichier	recherche sur le nom du fichier.
-perm	droit d'accès	recherche sur les droits d'accès du fichier
-user	nom login	recherche sur le propriétaire du fichier
-group	nom du groupe	recherche sur le groupe auquel appartient le fichier
-type	d, f ou l	recherche sur le type (d=répertoire, f=fichier normal, l= lien)
-size	taille minimum	recherche sur la taille du fichier en nombre de blocs (1 bloc=512octets)
-atime	nbjour	recherche par date de dernier accès en lecture du fichier
-mtime	nbjour	recherche par date de dernière modification du fichier exemple :
-ctime	date	recherche par date de création du fichier.

Pour le moment, certains critères vont vous échapper et c'est normal vu que nous sommes qu'au début du module.

On peut combiner ces critères :

Pour combiner le critère1 ET le critère2 (ET logique): **-a**

```
$ find chemin_où_chercher critère1 -a critère2
```

Pour combiner le critère1 OU le critère2 (OU logique) : **-o**

```
$ find chemin_où_chercher critère1 -o critère2
```

On peut aussi utiliser la négation (le NON logique) : **!**

```
$ find chemin_où_chercher ! critère1
```

Voici quelques liens si vous souhaitez en apprendre plus :

https://www.absolinux.net/tutos/ldcunix.html#BODY_1.5.3

http://www.funix.org/fr/unix/grep-find.htm

N'hésitez pas aussi à vous appuyer sur « **man find** ».

2. Commande locate

La commande **locate** permet de trouver très rapidement un fichier. Contrairement à ce que l'on pourrait penser locate ne vas pas chercher le fichier au sein de l'arborescence, mais au sein d'une base de données contenant la liste des fichiers existants.

De ce fait, il se peut que lorsque vous venez de créer un fichier ou un répertoire que ce dernier ne soit pas dans cette base de données, il faut pour cela réactualiser la base de données à l'aide de la commande **updatedb**.

La commande locate fonctionne comme suit :

```
locate nom_de_ficher
```

Exemple :

```
$ locate passwd
/etc/passwd
/etc/passwd-
/etc/cron.daily/passwd
/etc/pam.d/chpasswd
/etc/pam.d/passwd
/etc/security/opasswd
/usr/bin/gpasswd
/usr/bin/grub-mkpasswd-p
```

Mise à jour :

```
$ updatedb ou bien sudo updatedb
```

3. Retrouver des exécutables

a. whereis

whereis - Rechercher les fichiers exécutables, les sources et les pages de manuel d'une commande

```
$  whereis find
find: /usr/bin/find /usr/share/man/man1/find.1.gz /usr/share/info/find.info.gz
```

b. which

Localiser une commande.

```
$ which find
/usr/bin/find
```

D. Redirections

Les redirections permettent de rediriger :

- l'affichage de l'écran vers un fichier ou périphérique
- les messages d'erreur vers un autre fichier
- de remplacer la saisie clavier par le contenu d'un fichier
- de renvoyer la sortie d'une commande vers une autre commande

1. En sortie

Redirige le résultat de la commande dans le fichier : *fichier_sortie*.

```
$ commande > fichier_sortie
```

Si le fichier n'existe pas, il sera créé.

S'il existe, son contenu sera écrasé, même si la commande tapée est incorrecte.

Le shell commence d'abord par créer le fichier « *fichier_sortie* » puis exécute ensuite la commande.

C'est un aspect important des redirections : **les redirections sont interprétées de la droite vers la gauche**, et les redirections sont mises en place AVANT l'exécution des commandes : il faut bien créer le fichier avant de pouvoir y écrire. D'où le fait que même si la commande est fausse, le fichier est créé ou écrasé…

Pour ajouter des données à un fichier existant sans écraser le contenu on utilise « >> » :

```
$ commande >> fichier_sortie
```

Exemple :

```
$ ls -l /bin > listexecutables
$ ls -l /usr/bin >> listexecutables
```

2. En entrée

```
$ commande < fichier.txt
```

Exemple avec la commande **wc** (word count) qui permet de compter le nombre de lignes, de mots et de caractères d'un fichier :

```
$ wc < fichier.txt
4      29      203
```

3. Les canaux standards

On peut considérer un canal comme un fichier, qui possède son propre descripteur par défaut, et dans lequel on peut ou lire ou écrire :

- Canal d'entrée = stdin et porte le descripteur 0.
- Canal de sortie = stdout et porte le descripteur 1.
- Canal d'erreur = stderr et porte le descripteur 2.

Linux comme le langage C sur lequel il est construit possède un flot d'entrée (le clavier), un flot de sortie (l'écran) et un flot d'erreur (l'écran aussi).

Le problème vous l'avez peut-être compris c'est que l'écran affiche non seulement les retours de la commande mais aussi les éventuels messages d'erreur.

Exemple : En tant que simple utilisateur, je veux obtenir l'espace occupé par les répertoires contenus dans /

```
$ du -sh /*
...
du: impossible de lire le répertoire '/boot/efi': Permission denied
du: impossible de lire le répertoire '/boot/lost+found': Permission denied
du: impossible de lire le répertoire '/proc/tty/driver': Permission denied
....
```

La commande retourne des erreurs en plus de la sortie normale rendant difficile la lecture des informations.

Pour améliorer la lisibilité je vais rediriger la sortie erreur vers un fichier spécial **/dev/null** (appelé aussi « trou noir ») :

```
$ du -sh /* 2>/dev/null
0               /1
4,0K            /bin
162M            /boot
0               /dev
105M            /etc
37G             /home
4,0K            /lib
(...)
40K             /tmp
7,0G            /usr
13G             /var
$
```

Toutes les erreurs ont été envoyés dans */dev/null* ainsi je récupère uniquement les

informations utiles.

Vous pouvez rediriger les **deux canaux de sortie dans un seul et même fichier**, en les liant. On utilise pour cela les caractères associés « **>&**».

Les redirections étant en principe en fin de commande, le shell recherche d'abord les caractères <, >, >> en fin de ligne.

Ainsi si vous voulez grouper les deux canaux de sortie et d'erreur dans un même fichier, il faut procéder comme suit :

```
$ commande > sortie_et_erreur.log 2>&1
```

Cette ligne de commande redirige la sortie d'erreur (2) et la sortie standard (1) dans le fichier « *sortie_et_erreur.log* ».

4. Pipelines / tubes

Les « pipes » notés : « | » (en français « tube de communication ») permettent de rediriger la sortie d'une commande vers une autre.

En tapant :

```
$ commande1 argument1 | commande2
```

La commande commande2 prend pour argument(s) le résultat de l'exécution de commande1 argument.

Bien évidement, plusieurs commandes peuvent être combinées avec des pipes :

```
$ commande1 argument1 | commande2 | commande3 | commande4
```

Autant de commandes que vous voulez mais en faisant attention que les résultats de chaque commande existent. Si une des commandes est fausse ou ne donne pas de résultat, toutes les autres commandes derrière le pipe seront inefficaces.

Quelques exemples d'utilisation :

```
$ cat fichier | grep texte_a_chercher
```

Pour faire afficher un fichier très long page par page !

```
$ cat fichier | grep texte_a_chercher | less
```

E. Filtres et utilitaires

Une commande filtre est un programme sachant écrire et lire des données par les canaux standards d'entrée et sortie. Il en modifie ou traite éventuellement le contenu. Les utilitaires sans être obligatoirement des filtres permettent un certains nombres d'actions sur des fichiers ou leur contenu comme le formatage ou l'impression.

1. Les commandes cat, tac, tail, head

a. La commande cat

```
$ cat fichier
```

Elle permet d'afficher le fichier à l'écran; cette commande n'est vraiment intéressante que si le fichier est court.

Affiche à l'écran les trois fichiers à la suite dans l'ordre indiqué :

```
$ cat fich1 fich2 fich3
```

Numérote les lignes du fichier à l'affichage :

```
$ cat -n fichier
```

Fait apparaître les caractères non lisibles à l'écran :

```
$ cat -v fichier
```

b. La commande head

Elle permet d'éditer les 10 premières lignes d'un fichier en commençant par le début, ce qui est fort utile dans le cas des fichiers très longs :

```
$ head fichier
```

Pour afficher les 15 premières lignes:

```
$ head -15 fichier
```

c. La commande tail

```
$ tail fichier
```

Elle permet d'éditer un fichier en commençant par la fin. Ceci est très utile quand on possède un fichier très long et que l'on veut récupérer un résultat qui se trouve à la fin du texte.

Pour obtenir les douze dernières lignes d'un fichier:

```
$ tail -12 fichier
```

Pour avoir en plus le titre du fichier:

```
$ tail -12 -v fichier
```

Sur les serveurs vous devrez savoir **suivre un fichier de log en direct**. La commande tail peut vous y aider. Par exemple vous venez de configurer un serveur web Apache et vous voulez tracer les connexions. Utilisez l'option -f :

```
# tail -f /var/log/apache2/access.log
```

Et pour le debug :

```
# tail -f /var/log/apache2/*.log
```

Qui vous permettra de suivre le fichier access.log et le fichier error.log en direct.

d. La commande grep

La commande **grep** permet de rechercher une chaîne de caractères dans un fichier. Les options sont les suivantes :

- **-v** affiche les lignes ne contenant pas la chaîne
- **-c** compte le nombre de lignes contenant la chaîne
- **-n** chaque ligne contenant la chaîne est numérotée
- **-x** ligne correspondant exactement à la chaîne
- **-l** affiche le nom des fichiers qui contiennent la chaîne

Exemple avec le fichier **carnet-adresse** contenant :

```
marcel:13:0466342233:Gardagnes
myriam:30:0434214452:Nimes
olivier:29:0298333242:Brest
yvonne:92:013344433:Palaiseau
```

```
$ grep Brest carnet-adresse
```

Permet d'obtenir les lignes contenant la chaîne de caractère Brest, soit :

```
olivier:29:0298333242:Brest
```

2. Manipulation de fichiers

a. La commande sort

```
$ sort [option] fichier
```

Option de la commande sort

-r	pour inverser l'ordre
-n	pour trier dans l'ordre numérique
-f	pour ignorer la casse
-b	pour ignorer les blancs en début de ligne
-d	pour utiliser les blancs et l'ordre alphabétique uniquement.

Quelques exemples :

Trier les fichiers par ordre alphabétique (indépendamment de la casse) :

```
$ ls -1 | sort -f
```

Trier les fichiers par taille croissante :

```
$ ls -s | sort -n
```

Trier les fichiers par taille décroissante :

```
$ ls -s | sort -rn
```

Donner la liste des 5 plus gros fichiers :

```
$ ls -s | sort -rn | head -5
```

b. La commande cut

Permet d'extraire certains champs d'un fichier.

Pour illustrer cette partie, nous allons utiliser le fichier carnet.txt (**copiez le contenu suivant et placez-le dans le fichier carnet.txt**):

```
prof:0561001485:ldnr:toulouse
eleve1:48731321:entrep2:paris
eleve2:45454575:entrep2:marseille
eleve3:55555555:entrep8:lyon
eleve4:11111111:entrep9:nancy
```

L'option **-c** extrait les colonnes en se basant sur la position des caractères :

```
$ cut -c1-10 carnet.txt
```

affichera à l'écran les 10 premiers caractères de chaque ligne :

```
prof:05610
eleve1:487
eleve2:454
eleve3:555
eleve4:111
```

```
$ cut -c1,15 carnet.txt
```

Va extraire le premier et le quinzième caractère de chaque ligne :

```
p5
e1
e5
e5
e1
```

L'option **-f** extrait suivant le nombre de champs :

```
$ cut -f1,3 carnet.txt
```

Va extraire le premier et le troisième champ de chaque ligne, **si chaque champ** est séparé par des espaces ou des tabulations.

Cependant notre fichier utilise les « : » pour séparer les champs. Nous allons donc utiliser l'option **-dx.** Le caractère **x** définit le séparateur de champs (« : » dans notre cas).

```
$ cut -d: -f1,3 carnet.txt
```

Va extraire le premier et le troisième champ, :

```
prof:ldnr
eleve1:entrep2
eleve2:entrep2
eleve3:entrep8
eleve4:entrep9
```

```
$ cut -d: -f2- carnet.txt
```

Va extraire du deuxième jusqu'au dernier champ :

```
0561001485:ldnr:toulouse
48731321:entrep2:paris
45454575:entrep2:marseille
55555555:entrep8:lyon
11111111:entrep9:nancy
```

c. La commande tr

La commande « tr » convertit un ensemble de caractères en un autre.

Elle permet par exemple de convertir un texte majuscule ou en minuscule :

```
$ echo "Bonjour $USER"
Bonjour stag
$ echo "Bonjour $USER" | tr [a-z] [A-Z]
BONJOUR STAG
```

Une des options de la commande « tr » est l'option « -s ». Elle permet de réduire une suite consécutive d'un même caractère à un seul. Exemple :

```
$ ls -l
total 10004
drwxrwxr-x.  4 olivier olivier     4096 12 avril 08:48 ansible
drwxrwxr-x.  3 olivier olivier     4096 11 avril 20:06 Arduino
drwxr-xr-x.  2 olivier olivier     4096 11 avril 18:34  Bureau
```

Remplaçons les espaces par le symbole « : ».

```
$ ls -l | tr " " ":"
total:10004
drwxrwxr-x.::4:olivier:olivier:::::4096:12:avril:08:48:ansible
drwxrwxr-x.::3:olivier:olivier:::::4096:11:avril:20:06:Arduino
drwxr-xr-x.::2:olivier:olivier:::::4096:11:avril:18:34:Bureau
```

Si je souhaite faire un « cut » pour dégager certaines colonnes cela risque d'être compliqué. Passons alors l'option « -s » à la commande « tr » :

```
$ ls -l | tr -s " " ":"
total:10004
total:10004
drwxrwxr-x.:4:olivier:olivier:4096:12:avril:08:48:ansible
drwxrwxr-x.:3:olivier:olivier:4096:11:avril:20:06:Arduino
drwxr-xr-x.:2:olivier:olivier:4096:11:avril:18:34:Bureau
```

La lisibilité est clairement améliorée. Au lieu de transformer les espaces en autant de « : », tous les espaces consécutifs ont été convertis en un seul « : ». C'est parfait pour générer des séparateurs.

d. Les afficheurs ("pagers") : more et less

La commande more

La commande more permet d'afficher le contenu d'un fichier page par page.

```
$ more fichier
$ ls /bin | more
```

La commande less

less permet de lire un fichier sans l'éditer et donc sans le modifier. Il est plus puissant que more (même si less signifie « moins » en anglais) mais il ne se trouve pas dans tous les systèmes Linux, il faut parfois le faire rajouter par le root. La commande less, contrairement à more permet de se déplacer dans le fichier avec les flèches de direction et de mettre du texte en surbrillance lors de la recherche d'un mot.

```
$ less fichier
$ ls /bin | less
```

e. sed

Sed est un éditeur de flux permettant de filtrer et de transformer du texte. L'intérêt de sed réside dans son utilisation dans des scripts étant donné qu'il permet une édition de texte en une passe et sans édition interactive. Très pratique quand on souhaite déployer un fichier de configuration générique dont seules les valeurs changent sans devoir ouvrir à chaque fois le fichier de configuration pour les saisir.

Voici une utilisation très courante, la substitution, sachant que sed vaut un livre à lui tout seul.

(**utilisez le fichier carnet.txt**)

```
$ sed -e 's/eleve/stagiaire/' carnet.txt
prof:0561001485:ldnr:toulouse
stagiaire1:48731321:entrep2:paris
stagiaire2:45454575:entrep2:marseille
stagiaire3:55555555:entrep8:lyon
stagiaire4:11111111:entrep9:nancy
```

Cela ne modifie que la sortie et non le fichier. Si l'on veut modifier le fichier en remplaçant eleve par stagiaire :

```
$ sed -i 's/eleve/stagiaire/' carnet.txt
$ cat carnet.txt
prof:0561001485:ldnr:toulouse
stagiaire1:48731321:entrep2:paris
stagiaire2:45454575:entrep2:marseille
stagiaire3:55555555:entrep8:lyon
stagiaire4:11111111:entrep9:nancy
```

sed -i ou -e suivi de 's/motif recherché/motif de remplacement/' remplace uniquement le premier motif trouvé d'une ligne. Dans le fichier carnet.txt remplacez ldnr par prof :

```
$ sed -i 's/ldnr/prof/' carnet.txt
$ cat carnet.txt
prof:0561001485:prof:toulouse
..
$ sed -e 's/prof/formateur/' carnet.txt
formateur:0561001485:prof:toulouse
...
$ sed -e 's/prof/formateur/g' carnet.txt
formateur:0561001485:formateur:toulouse
...
```

Le **g** final fait un remplacement sur toute la ligne.

La suppression

Cette option de sed permet de filtrer les lignes qui satisfont une expression régulière. Ces lignes ne sont pas détruites dans le fichier d'origine, mais sont transmises en sortie écran. Si on veut modifier le fichier, il faut soit rediriger l'affichage vers un fichier de sortie (s'il on veut conserver le fichier original) soit utiliser l'option **-i**.

Pour effacer les 3 premières lignes :

```
$ sed '1,3d' carnet.txt ← modifie la sortie uniquement sur l'affichage
stagiaire3:55555555:entrep8:lyon
stagiaire4:11111111:entrep9:nancy

$ sed '1,3d' carnet.txt > nouveau_carnet.txt ← redirige la sortie vers un nouveau fichier
$ sed -i '1,3d' carnet.txt ← modifie le fichier
```

On efface toutes les lignes **ne commençant pas** par « stagiaire » (à cause du ! devant le d) :

```
$ sed '/^stagiaire/!d' carnet.txt
formateur:0561001485:formateur:toulouse
```

Cet autre exemple montre comment détruire toutes les lignes vides d'un fichier :

```
$ sed /^$/d nom_fichier > nouveau_fichier
```

Vous trouverez de nombreux exemples (plus ou moins avancés, voire très avancés) sur le web.

F. Annexe

1. Les expressions régulières

On a vu auparavant ce qu'étaient les **métacaractères** . Les expressions régulières sont aussi des suites de caractères permettant de faire des sélections. Elles fonctionnent avec certaines commandes comme **egrep**.

Les différentes expressions régulières sont :

Caractère spécial	Signification
|	Ou logique, l'expression située avant ou après doit apparaître.
(...)	Groupement de caractères.
[...]	Un caractère à cette position parmi ceux indiqués.
[^...]	tous les caractères sauf ceux énumérés.
. (point)	Un caractère quelconque.
+	Répétition, le caractère placé avant doit apparaître au moins une fois.
*	Répétition, le caractère situé avant doit apparaître de zéro à n fois.
?	Le caractère situé avant doit apparaître une fois au plus.
{n}	Le caractère situé avant doit apparaître exactement n fois.
{n,}	Il apparaît n fois ou plus.
{n,m}	Il apparaît entre n et m fois.
^	En début de chaîne.
$	En fin de chaîne.

Exemple d'utilisation avec la commande **egrep** :

```
$ cat salutation.txt
Bonjour
bonjour
salut
Coucou
Bonjour bonjour
Bonsoir
hello
bonsoir
bonsoir tout le monde
$
```

Filtre pour obtenir seulement bonjour et bonsoir commençant par une majuscule ou une minuscule s'ils sont seuls sur une ligne :

```
$ egrep "^[bB]on(jour|soir)$" salutation.txt
Bonjour
bonjour
Bonsoir
bonsoir
$
```

Pour ceux qui souhaitent aller plus loin, vous trouverez de nombreux exemples sur le web.

2. Initiation à awk

awk est une commande très puissante, c'est un langage de programmation a elle tout seule qui permet une recherche de chaînes et l'exécution d'actions sur les lignes sélectionnées. Elle est utile pour récupérer de l'information, générer des rapports, transformer des données entre autres.

Un programme **awk** possède la structure suivante: **critère de sélection d'une chaîne {action}**, quand il n'y a pas de critère c'est que l'action s'applique à toutes les lignes du fichier.

Exemple: Afficher le dernier champs du fichier /etc/passwd

```
$ awk -F: '{print $NF}' /etc/passwd
/bin/bash
/usr/sbin/nologin
/usr/sbin/nologin
/usr/sbin/nologin
```

« -F: » correspond au délimiteur de champs (les « : » dans le cas du fichier passwd). Si vous ne définissez pas de délimiteur, awk utilisera les espaces comme délimiteur entre champs.

NF est une variable prédéfinie d'awk, elle est égale au nombre de champs dans une ligne. Si une ligne possède 4 champs, **$NF** est donc égale à 4 qui est aussi le n° du dernier champs.

On peut aussi passer des critères. Par exemple affichons uniquement les lignes de 5 à 10 du fichier passwd :

```
$ awk -F: 'NR == 5, NR == 10 {print NR "  :  "  $0 }' /etc/passwd
5  :  sync:x:4:65534:sync:/bin:/bin/sync
6  :  games:x:5:60:games:/usr/games:/usr/sbin/nologin
7  :  man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
8  :  lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
9  :  mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
10  :  news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

NR correspond au numéro de ligne et **$0** à l'enregistrement complet de la ligne. Si nous avions voulu que le premier champs, à savoir le nom des utilisateurs dans ce cas, nous aurions remplacé **$0** par **$1**.

Si l'on souhaite rediriger la sortie vers un fichier :

```
$ awk -F:  '{print $1":"$6  > "passwd.num"}' /etc/passwd
$ cat passwd.num
root:/root
daemon:/usr/sbin
bin:/bin
sys:/dev
sync:/bin
```