Quelques commandes utiles en bash

M. Sevaux (inspired from the web)

Février 2020

Les commandes sur les fichiers

cat, less, tail, head, tee : afficher|créér des fichiers

Ces commandes ont pratiquement toutes la même syntaxe :

```
commande [option(s)] [fichier(s)]
```

et peuvent être utilisées dans un tube (pipe en Anglais "|"). Toutes sont utilisées pour imprimer une partie d'un fichier selon certains critères.

Attention, dans certain systèmes, vous devez faire précéder les commandes par `sudo` pour obtenir les droits de visualiser les fichiers.

cat|less - affichage|création de fichiers

L'utilitaire `cat` agglomère des fichiers et les imprime sur la sortie standard. C'est une des commandes les plus utilisées. Vous pouvez utiliser :

```
cat /etc/passwd
```

pour afficher, par exemple, le contenu du fichier des passwords sur la sortie standard. la commande `cat` a une option très utile (`-n`) qui permet de numéroter les lignes affichées.

La commande suivante va concaténer `f1, f2 ..` dans le nouveau fichier `f`

```
cat f1 f2 .. > f
```

Le même résultat est obtenu avec

```
less f1 f2 .. > f
```

On peut aussi ajouter le contenu d'un fichier à la fin d'un fichier existant avec :

```
less f3 >> f
```

Cette commande ajoute le contenu du fichier `f3` à la suite du contenu du fichier `f` (comme `cat`)

On peut enfin créer un fichier directement dans la console avec la commande

```
cat > monfichier
```

Cette commande créé le fichier `monfichier`, il suffit de taper le contenu du fichier directement dans la console, puis de finaliser la création du fichier avec `CTRL D`.

tail - les dernières lignes

Certains fichiers, comme les fichiers journaux des démons (s'ils sont utilisés), sont souvent de taille énorme et les afficher dans leur intégralité à l'écran n'a aucun intérêt. Vous voudrez souvent ne consulter que quelques lignes d'un fichier. Vous pouvez par exemple utiliser la commande `tail` pour n'afficher que les dernières lignes de `/var/log/mail.info` :

```
tail /var/log/mail.info (ou sudo tail /var/log/mail.info)
```

Elle affichera les dix dernières lignes du fichier `/var/log/mail.info`. Si vous souhaitez n'afficher que les deux premières lignes vous pouvez utiliser l'option `-n` :

```
tail -n2 /var/log/mail.info
```

head - les première lignes

La commande `head` est similaire à `tail`, mais affiche les premières lignes d'un fichier :

```
head /var/log/mail.info
```

Cette commande sans option affiche les 10 premières lignes de `/var/log/mail.info`. Comme avec `tail` vous pouvez imprimer les deux premières lignes à l'aide de l'option `-n` :

```
head -n2 /var/log/mail.info
```

Combiner les commandes head et tail

Vous pouvez aussi utiliser ces commandes de concert. Par exemple, si vous souhaitez seulement afficher les lignes 9 et 10, vous pouvez lancer la commande `head` qui sélectionnera les 10 premières lignes du fichier, les passera à la commande `tail` au travers d'un tube (|) :

```
head /var/log/mail.info | tail -n2
```

La dernière partie de la commande sélectionnera les 2 dernières lignes et les imprimera à l'écran.

De la même façon, si vous voulez afficher la 20ème ligne en partant de la fin, vous pouvez sélectionner les 20 dernières lignes et n'afficher que la première de ces 20 lignes :

```
tail -n20 /var/log/mail.info |head -n1
```

Dans cet exemple, nous disons à `tail` de sélectionner les 20 dernières lignes du fichier, puis de les passer à `head`. Cette dernière en affiche alors la première ligne des lignes qu'elle reçoit à travers le tube.

tee - afficher et sauvegarder dans un fichier

Supposons maintenant que nous souhaitions à la fois afficher à l'écran et enregistrer le résultat de la commande précédente dans le fichier `resultats.txt`. L'utilitaire `tee` va nous y aider. Sa syntaxe est :

```
tee [option(s)] [file]
```

Nous changeons alors la commande précédente :

```
tail -n20 /var/log/mail.info |head -n1|tee resultats.txt
```

Prenons un autre exemple. Nous voulons sélectionner les 20 dernières lignes, les enregistrer dans `resultats.txt`, mais afficher à l'écran seulement les premières de ces 20 lignes. Nous écrivons alors :

```
tail -n20 /var/log/mail.info |tee resultats.txt |head -n1
```

La commande `tee` a une option très utile (`-a`) qui permet d'ajouter les informations reçues à un fichier existant au lieu de l'écraser.

Et pour les fichiers qui sont constament modifiés ?

Revenons à la commande `tail`. Les fichiers comme les journaux changent souvent car les démons y rajoutent sans cesse de l'information au sujet de leur activité. Si vous voulez alors surveiller de manière interactive des changements sur ces journaux, vous pouvez tirer avantage de l'une des options les plus utiles de `tail` : `-f`:

```
tail -f /var/log/mail.info
```

Dans ce cas, tous les changements dans le fichier `/var/log/mail.info` sont affichés immédiatement à l'écran. Cette option est très utile pour surveiller l'activité de votre système. Par exemple, en regardant dans le journal `/var/log/messages`, vous pouvez surveiller les messages du système et plusieurs démons. Cette option peut aussi être utilisée pour surveiller n'importe quel autre fichier.

Par exemple, vous pouvez surveiller le fichier `/var/log/message`, et activer/desactiver le wifi. Vous verrez les opérations qui sont ajoutées dans ce fichier.

```
tail -f /var/log/message
```

Les commandes de filtrage de fichiers

Généralités

Un filtre est une commande qui lit les données sur l'entrée standard, effectue des traitements sur les lignes reçues et écrit le résultat sur la sortie standard.

Bien sûr les entrées/sorties peuvent être redirigées, et enchainées avec des tubes.

A noter que le caractère d'indirection < en entrée n'est pas obligatoire pour les filtres. Ainsi, dans `cat /etc/*.conf > tous.conf`, `cat` va bien lire les fichiers qui correspondent au modèle `/etc/*.conf` et les contaténer dans le fichier `tous.conf`

Dans ce chapitre, on va revoir ou découvrir les principaux filtres utilisés dans le monde UNIX

- grep
- cut
- wc
- tr
- sed

La commande grep : sélection de lignes

Cet utilitaire (General Regular Expression Parser, analyseur général d'expression régulière) sélectionne toutes les lignes qui satisfont une expression régulière (ou rationnelle).

Syntaxe

```
grep [options] expreg [fichiers]
```

Cette commande recherche dans les fichiers ou sur son entrée standard des lignes de texte qui satisfont l'expression régulière expreg indiquée. Sa sortie peut être redirigée dans un fichier.

options

- -c donne seulement le nombre de lignes trouvées obéissant au critère
- -l donne seulement le nom des fichiers où le critère a été trouvé
- -v donne les lignes où le critère n'a pas été trouvé
- -i ne pas tenir compte de la casse (ne pas différencier majuscules minuscules)
- -n pour n'afficher que les numéros des lignes trouvées
- -w pour imposer que le motif corresponde à un mot entier d'une ligne

constructions

`grep` est souvent inclus dans un tube qui lui fournit en entrée le fichier à étudier.

Exemple : placer dans un fichier `sortie` tous les utilisateurs de mon pc/réseau qui se prénoment jean :

```
cat /etc/passwd | cut -d: -f1 | grep -w "jean" > sortie
```

Expressions reconnues

`grep` ne reconnait pas toutes les expressions rationnelles étendues.

Voici la liste des symboles utilisables par grep : `. * [ ] [^ ] ^ $`

- `.` signifie un caractère quelconque
- `*` répétition multiple du caractère situé devant
- `^` début de ligne
- `$` fin d'une ligne (donc "e$" lignes se terminant par e)
- `[...]` contient une liste ou un intervalle de caractères cherchés
- `[^..]` caractères interdits.

Attention

Pour éviter une confusion entre les interprétations de ces symboles spéciaux par `grep` ou par le shell, il est indispensable de "verrouiller" expreg en plaçant l'expression entre guillemets `" "` (et non entre quotes !).

Exemples

Etudier et commenter les commandes suivantes :

- cherche dans `fichier`, les lignes dont la 1ère lettre est quelconque et la 2ème doit être `o`

```
grep "^.o" fichier
```

- cherche dans le fichier `passwd` les lignes commençant par `t`

```
grep "^t" /etc/passwd
```

- cherche les lignes ne commençant pas commençant par `t`

```
grep -v "^t" /etc/passwd
```

- cherche les lignes contenant les mots entiers suivant le modèle `T.t.`

```
grep -w "T.t." /etc/passwd
```

- cherche dans le fichier des groupes, ceux qui commencent par `a` ou `b` .. ou `j`

```
less /etc/group | grep "^[a-j]"
```

- pour lister les sous-répertoires du rép. `/etc`

```
ls -l /etc | grep "^d"
```

- compter les lignes saisies au clavier qui se termine par `a`

```
grep -c "a$"
```

- afficher les lignes des fichiers `essai?.txt` qui contiennent `a`, `b` ou `c`

```
grep [abc] "essai?.txt"
```

- détourne le flot de sortie de la commande pour l'envoyer sur l'entrée de `wc` pour compter ces lignes.

```
grep [abc] "essai?.txt" | wc -l
```

cut : sélection de colonnes

La commande `cut` présente 2 formes suivant que l'on sélectionne des colonnes de caractères ou qu'on distingue des champs séparés par un caractère précis.

sélection colonne

```
cut -c(sélection_colonnes) [fichiers]
```

Exemples

- affiche le 5ième caractère

```
cut -c5 fichier
```

- affiche du 5ième au 10ème caractères

```
cut -c5-10 fichier
```

- affiche le 5ième et le 10ème caractères

```
cut -c5,10 fichier
```

- affiche à partir du 5ième (jusqu'à la fin)

```
cut -c5- fichier
```

sélection champs

```
cut -d(séparateur) -f(sélection_champs) [fichiers]
```

Exemples

- affiche la 5ième colonne d'un fichier où les colonnes sont séparées par des tabulations

```
cut -f5 fichier
```

- affiche la 3ième colonne d'un fichier séparé par des virgules

```
cut -d, -f3 fichier
```

- affiche de la 3ième à la 6ième colonne d'un fichier séparé par des espaces

```
cut -d' ' -f3-6 fichier
```

La commande wc

La commande `wc` affiche à l'écran le nombre de lignes, de mots et de caractères des fichiers passés en paramètres.

Exemples

Pour compter les titulaires d'un compte pouvant se connecter avec le login shell

```
cat /etc/paswwd | grep /bin/bash/ | wc -l
```

La commande tr

la commande `tr`=Translate, est un filtre ne reconnaissant pas les expressions régulières. Cette commande est le plus souvent associée à des redirections Les caractères entrés sont traités et le résultat est envoyé sur la sortie standard. On peut utiliser les intervalles du type `a-z` et les codes ASCII des caractères en notation octale `\0xx`

Syntaxe

```
tr [options] ch1 ch2 <fich1 >fich2
```

Remplace toutes les occurrences de TOUS les caractères de ch1 par le caractère de ch2, de même rang, dans le flot d'entrée.

Exemple

Pour convertir et afficher la ligne saisie au clavier en minuscules

```
read ligne; echo $ligne | tr 'A-Z' 'a-z'
```

La commande `tr -c chaine car` remplace tout caractère NON INCLUS dans la chaine `chaine` par le caractère `car`.

La commande suivante remplace tous les caractères différents de a,b, ..z par un espace

```
echo $ligne | tr -c a-z ' '
```

La commande `tr -d chaine` supprime tout caractère entré, appartenant à la chaine `chaine`

Donc la commande suivante supprime toutes les minuscules non accentuées

```
echo $ligne | tr -d a-z
```

La commande `tr -s chaine` supprime toute répétition des caractères contenus dans `chaine`

Donc la commande suivante supprime les espaces multiples entre les mots

```
echo $ligne | tr -s ' '
```

L'utilitaire sed

Il s'agit d'un utilitaire (sed = "Stream EDitor") qui sélectionne les lignes d'un fichier texte (ou d'un flot provenant d'un pipe) vérifiant une expression régulière et qui leur applique un traitement ou un remplacement.

Syntaxe

```
sed [-n] [-e script] [-f fichier-commandes] fichier-source
```

L'option `-n` empêche la sortie à l'écran du résultat (souvent associé à l'option p). Le fichier source est traité ligne par ligne conformément à la liste des commandes (-e) ou au fichier de commandes (-f).

Commande de substitution

La commande `s` permet d'effectuer des substitutions suivant la syntaxe :

```
sed [adresse]s/expr-régulière/remplacement/options
```

Attention ! contrairement à ce que l'on pourrait attendre, cette commande laisse passer toutes les lignes et ne sélectionne pas celles qui ont satisfait l'expression régulière et donc subi la substitution. Pour sélectionner, voir la commande de destruction.

Options - Sans précision, la commande ne s'applique qu'à la 1ère occurence de chaque ligne. - `0…9` : indique que la substitution ne s'applique qu'à la nième occurence - `g` : effectue les modifications sur toutes les occurences trouvées.

Exemple : `sed s/moi/toi/g fich.moi > fich.toi` le fichier `fich.moi` est parcouru, à chaque occurrence de "moi", ce mot est remplacé par "toi" et le nouveau fichier est sauvegardé sous le nom `fich.toi`

Délimiteur

Le slash / étant très utilisé au niveau du shell comme séparateur de niveau de répertoire, il est possible d'utiliser à la place tout autre caractère comme #

```
sed s#/home#/rep_perso#g /etc/passwd > /tmp/passwd.new
```

Destruction ou sélection

Cette option permet de filtrer les lignes qui satisfont une expression régulière. Ces lignes ne sont pas détruites dans le fichier d'origine, mais ne sont pas transmise en sortie.

Comment modifier alors le fichier à traiter ?

```
cp fichier copie
sed /.../d copie
```

Par exemple, pour détruire toutes les lignes vide d'un fichier :

```
sed /^$/d
```

Ajout, insertion et modification

Pour utiliser ces commandes, il est nécessaire de les saisir sur plusieurs lignes

```
sed [adresse] commande\
expression
```

La commande peut être :

- a pour ajout ;
- i pour insertion ;
- c pour modification.

Trier et combiner

sort - trier les lignes

La commande `sort` permet de trier les lignes d'un fichier. Les caractères "+" et "-" permettent de spécifier de quelle colonne à quelle colonne le tri doit s'effectuer (1ere colonne pour 0, 2eme colonne pour 1…) :

```
sort +1 -2 /etc/passwd
```

Si on spécifie plusieurs critères, le tri se fera d'abord sur le premier champ, puis sur le second si le tri sur le premier champ n'a pas suffit à départager certaines lignes, et ainsi de suite…Il existe diverses options :

Options

- -b Saute les colonnes constituées de blancs.
- -d Trie de type dictionnaire.
- -n Trie par ordre numérique.
- -f Aucune différentiation n'est faite entre minuscules et majuscules.
- -b Ignore les espaces placés en début de champ.
- -r Trie inverse.
- -M Trie chronologiquement les mois.
- -t: Trie suivants les champs séparés par les caractères deux points (":").
- -u ne conserve qu'un seul exemplaire des lignes identiques.

On peut spécifier la recherche sur un caractère situé à une position particulière, par exemple à la 2eme position du 6eme champ :

```
sort -t: +5.1 /etc/passwd
```

Pour plusieurs critères de recherche, il faut spécifier derrière chaque champ le type de tri à mettre en oeuvre pour ce critère. Par exemple :

```
sort -t: +0d -1 +2nr -3
```

triera le 1er champ par ordre dictionnaire, et le 3eme champ par ordre numérique inverse, et

```
sort -t: +4.3n -4.5 +4.0n -4.2
```

triera du 4eme au 6eme caractère du 5eme champ par ordre numérique, et du 1er au 3eme caractère du 5eme champ par ordre numérique, si le premier tri s'est avéré insuffisant.

Bien sur, on peut combiner les commandes cut et sort. Par exemple :

```
cut -d: -f3 /etc/passwd | sort -n > Nombres
```

uniq - filtrer des données en double, triple, ...

La commande `uniq` permet de dédoublonner les lignes d'un fichier. Seules les lignes **identiques et consécutives** sont traitées. Elle s'utilise le plus souvent à la suite de la commande `sort`. Le résultat est stocké dans un fichier de sortie, si celui-ci est spécifié, sur la sortie standard dans le cas contraire.

Syntaxe

```
uniq [options] [fichier_entree [fichier_sortie]]
```

Principales options :

- -d : Affichage des doublons
- -c : Comptage des doublons
- -u : Affichage des lignes uniques

Exemples

Le fichier `fic20` contient des lignes en double

```
$ cat fic20
ceci est un test
ceci est un test
ceci est un test
ceci est un test
ceci est un autre test
ceci est un test
ceci est un autre test
ceci    est     un      test
ceci est un     test
et un   autre test
ceci est un fichier
$
```

Le fichier doit être trié pour que les lignes identiques soient consécutives :

```
$ sort fic20
ceci est un autre test
ceci est un autre test
ceci est un fichier
ceci est un test
ceci est un test
ceci est un test
ceci est un test
ceci est un test
ceci est un     test
ceci    est     un      test
et un   autre test
$
```

Et enfin, suppression des doublons :

```
$ sort fic20 | uniq
ceci est un autre test
ceci est un fichier
ceci est un test
ceci est un     test
ceci    est     un      test
et un   autre test
$
```

Résultat identique avec la commande `sort` et l'option `-u` :

```
$ sort -u fic20
ceci est un autre test
ceci est un fichier
ceci est un test
ceci est un     test
ceci    est     un      test
et un   autre test
$
```

Afficher devant chaque ligne son nombre d'occurences dans le fichier :

```
$ sort fic20 | uniq -c
    2 ceci est un autre test
    1 ceci est un fichier
    5 ceci est un test
    1 ceci est un     test
    1 ceci    est     un      test
    1 et un   autre test
$
```

Afficher uniquement les lignes ayant des doublons :

```
$ sort fic20 | uniq -d
ceci est un autre test
ceci est un test
$
```

Combiner les options -c et -d :

```
$ sort fic20 | uniq -cd
    2 ceci est un autre test
    5 ceci est un test
$
```

Afficher les lignes uniques :

```
$ sort fic20 | uniq -u
ceci est un fichier
ceci est un     test
ceci    est     un      test
et un   autre test
$ sort fic20 | uniq -cu
    1 ceci est un fichier
    1 ceci est un     test
    1 ceci    est     un      test
    1 et un   autre test
$
```