# Reprenez le contrôle à l'aide de Linux !

30 heures Facile

Mis à jour le 29/06/2021

## Extraire, trier et filtrer des données

Comme vous le savez déjà, la plupart des commandes de Linux sont basées sur le modèle du système d'exploitation Unix. Ce sont les mêmes. Certaines s'utilisent de la même manière depuis les années 60 ! Avantage pour les informaticiens : pas besoin de réapprendre à utiliser les mêmes commandes tous les trois mois.

Mais la question que vous devez vous poser est la suivante : comment se fait-il que la plupart de ces commandes n'aient pas changé depuis si longtemps ? La réponse vient du fait qu'elles n'ont pas eu besoin de changer. En effet, la plupart des commandes que vous découvrez sont très basiques : elles accomplissent une tâche et le font bien, mais pas plus. Ce sont les « briques de base » du système.

Dans ce chapitre, nous allons découvrir une série de commandes basiques qui permettent d'extraire, trier et filtrer des données dans des fichiers. Vous utiliserez certaines d'entre elles (comme `grep` ) presque tous les jours !

### grep : filtrer des données

La commande `grep` est essentielle. De toutes celles présentées dans ce chapitre, il s'agit probablement de la plus couramment utilisée.

Son rôle est de rechercher un mot dans un fichier et d'afficher les lignes dans lesquelles ce mot a été trouvé. L'avantage de cette commande est qu'elle peut être utilisée de manière très simple ou

plus complexe (mais plus précise) selon les besoins en faisant appel aux expressions régulières.

Les expressions régulières constituent un moyen très puissant de rechercher un texte. On les utilise non seulement dans la ligne de commandes Linux, mais aussi dans des éditeurs de texte avancés et dans de nombreux langages de programmation tels que PHP. Vous trouverez d'ailleurs deux chapitres assez complets au sujet des expressions régulières dans le cours ***Concevez votre site web avec PHP et MySQL*** que j'ai rédigé.

Nous allons commencer par utiliser `grep` de manière très simple ; nous verrons ensuite comment faire des recherches plus poussées avec les expressions régulières.

## Utiliser `grep` simplement

La commande `grep` peut s'utiliser de nombreuses façons différentes. Pour le moment, nous allons suivre le schéma ci-dessous :

```
grep texte nomfichier
```

Le premier paramètre est le texte à rechercher, le second est le nom du fichier dans lequel ce texte doit être recherché.

Essayons par exemple de rechercher le mot « alias » dans notre fichier de configuration `.bashrc` . Rendez-vous dans votre répertoire personnel (en tapant `cd` ) et lancez la commande suivante :

```
grep alias .bashrc
```

Cette commande demande de rechercher le mot « alias » dans le fichier `.bashrc` et affiche toutes les lignes dans lesquelles le mot a été trouvé.

Résultat :

```
$ grep alias .bashrc

#  /.bash_aliases, instead of adding them here directly.
#if [ -f  /.bash_aliases ]; then
#    .  /.bash_aliases
# enable color support of ls and also add handy aliases
    alias ls='ls --color=auto'
    #alias dir='ls --color=auto --format=vertical'
    #alias vdir='ls --color=auto --format=long'
# some more ls aliases
alias ll='ls -lArth'
```

```
#alias la='ls -A'
#alias l='ls -CF'
```

Pas mal, n'est-ce pas ? Comme vous pouvez le voir, `grep` est davantage un outil de filtre qu'un outil de recherche. Son objectif est de vous afficher uniquement les lignes qui contiennent le mot que vous avez demandé.

Notez qu'il n'est pas nécessaire de mettre des guillemets autour du mot à trouver, sauf si vous recherchez une suite de plusieurs mots séparés par des espaces, comme ceci :

```
grep "Site du Zéro" monfichier
```

### `-i` : ne pas tenir compte de la casse (majuscules / minuscules)

Par défaut, `grep` tient compte de la casse : il fait la distinction entre les majuscules et les minuscules. Ainsi, si vous recherchez « alias » et qu'une ligne contient « Alias », `grep` ne la renverra pas.

Pour que `grep` renvoie toutes les lignes qui contiennent « alias », peu importent les majuscules et les minuscules, utilisez l'option `-i` :

```
$ grep -i alias .bashrc

# Alias definitions.
#  /.bash_aliases, instead of adding them here directly.
#if [ -f  /.bash_aliases ]; then
#    .  /.bash_aliases
# enable color support of ls and also add handy aliases
    alias ls='ls --color=auto'
    #alias dir='ls --color=auto --format=vertical'
    #alias vdir='ls --color=auto --format=long'
# some more ls aliases
alias ll='ls -lArth'
#alias la='ls -A'
#alias l='ls -CF'
```

On notera que la première ligne renvoyée n'était pas présente tout à l'heure car le mot « Alias » contenait une majuscule. Avec l'option `-i` on peut désormais la voir.

### `-n` : connaître les numéros des lignes

Vous pouvez afficher les numéros des lignes retournées avec `-n` :

```
$ grep -n alias .bashrc

49:#  /.bash_aliases, instead of adding them here directly.
52:#if [ -f  /.bash_aliases ]; then
53:#    .  /.bash_aliases
56:# enable color support of ls and also add handy aliases
59:    alias ls='ls --color=auto'
60:    #alias dir='ls --color=auto --format=vertical'
61:    #alias vdir='ls --color=auto --format=long'
64:# some more ls aliases
65:alias ll='ls -lArth'
66:#alias la='ls -A'
67:#alias l='ls -CF'
```

### `-v` : inverser la recherche : ignorer un mot

Si, au contraire, vous voulez connaître toutes les lignes qui **ne contiennent pas** un mot donné, utilisez `-v` :

```
$ grep -v alias .bashrc

#  /.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
[ -z "$PS1" ] && return

# don't put duplicate lines in the history. See bash(1) for more options
export HISTCONTROL=ignoredups
# ... and ignore same sucessive entries.
export HISTCONTROL=ignoreboth

# ... (renvoie beaucoup de lignes, je ne mets pas tout ici)
```

Cette fois, on récupère toutes les lignes du fichier `.bashrc` qui ne contiennent pas le mot « alias ».

### `-r` : rechercher dans tous les fichiers et sous-dossiers

Si vous ne savez pas dans quel fichier se trouve le texte que vous recherchez, vous pouvez sortir l'artillerie lourde : l'option `-r` (*recursive*). Cette fois, il faudra indiquer en dernier paramètre le **nom du répertoire** dans lequel la recherche doit être faite (et non pas le nom d'un fichier).

```
grep -r "Site du Zéro" code/
```

... recherchera la chaîne « Site du Zéro » dans tous les fichiers du répertoire `code`, y compris dans les sous-dossiers.

Notez que le « `/` » à la fin n'est pas obligatoire. Sans cela Linux comprendra tout de même très bien qu'il s'agit d'un répertoire.

```
$ grep -r "Site du Zéro" code/

code/intro.html: Nous vous souhaitons la bienvenue sur le Site du Zéro !
code/tpl/define.tpl: Le Site du Zéro
```

Cette fois, le nom du fichier dans lequel la chaîne a été trouvée s'affiche au début de la ligne.

À noter qu'il existe aussi la commande `rgrep` qui est équivalente à `grep -r`.

## Utiliser `grep` avec des expressions régulières

Pour faire des recherches plus poussées – pour ne pas dire des recherches *très poussées* –, vous devez faire appel aux expressions régulières. C'est un ensemble de symboles qui va vous permettre de dire à l'ordinateur très précisément ce que vous recherchez.

Je vous propose dans un premier temps de jeter un oeil au tableau suivante regroupant les principaux caractères spéciaux qu'on utilise dans les expressions régulières.

| Caractère spécial | Signification |
|---|---|
| `.` | Caractère quelconque |
| `^` | Début de ligne |
| `$` | Fin de ligne |
| `[]` | Un des caractères entre les crochets |
| `?` | L'élément précédent est optionnel (peut être présent 0 ou 1 fois) |
| `*` | L'élément précédent peut être présent 0, 1 ou plusieurs fois |

| Caractère spécial | Signification |
| --- | --- |
| `+` | L'élément précédent doit être présent 1 ou plusieurs fois |
| `\|` | Ou |
| `()` | Groupement d'expressions |

Help ! Je n'ai rien compris. :-(

C'est normal. Pour bien faire, il faudrait un ou deux chapitres entiers sur les expressions régulières. Je n'ai pas vraiment la place ici pour faire un « minicours » sur les expressions régulières, je vous propose donc de jeter un oeil à ces quelques lignes pour apprendre par l'exemple.

Tout d'abord, il faut savoir qu'on doit utiliser l'option `-E` pour faire comprendre à `grep` que l'on utilise une expression régulière.

```
$ grep -E Alias .bashrc

# Alias definitions.
```

Notez que vous pouvez aussi utiliser la commande `egrep` qui équivaut à écrire `grep -E` .

C'est une expression régulière très simple. Elle demande de rechercher le mot « Alias » (avec un A majuscule). Si le mot est présent dans une ligne, cette dernière est renvoyée.

Bon, jusque-là, rien de nouveau ; ça fonctionnait comme ça avant qu'on utilise les expressions régulières. Essayons de pimenter cela en faisant précéder « Alias » d'un accent circonflexe qui signifie que le mot doit être placé au début de la ligne :

```
$ grep -E ^Alias .bashrc
```

Résultat : `grep` ne renvoie rien. En effet, la ligne de tout à l'heure commençait par un `#` et non pas par « Alias ».

En revanche, on a un résultat si on fait ceci :

```
$ grep -E ^alias .bashrc

alias ll='ls -lArth'
```

Cette fois, la ligne commençait bien par « alias ». De même, on aurait pu utiliser un `$` à la fin pour demander à ce que la ligne se termine par « alias ».

Quelques autres exemples que vous pouvez tester :

```
grep -E [Aa]lias .bashrc
```

... renvoie toutes les lignes qui contiennent « alias » ou « Alias ».

```
grep -E [0-4] .bashrc
```

... renvoie toutes les lignes qui contiennent un nombre compris entre 0 et 4.

```
grep -E [a-zA-Z] .bashrc
```

... renvoie toutes les lignes qui contiennent un caractère alphabétique compris entre « a » et « z » ou entre « A » et « Z ».
Je vous ai fait là une introduction très rapide mais il y aurait beaucoup à dire. Si vous voulez en savoir plus sur les expressions régulières, vous trouverez dans le cours *Concevez votre site web avec PHP et MySQL* .

Comme vous pourrez le constater, les expressions régulières fonctionnent aussi bien sans le `-E` . Pourquoi ?
Normalement, cette option sert à activer la gestion des expressions régulières les plus complexes. Dans la pratique, le manuel nous dit que la version GNU de `grep` (celle que l'on utilise sous Linux) ne fait pas de différence, que l'option soit présente ou non. Les expressions régulières sont toujours activées. En clair, vous aurez besoin du `-E` si un jour vous utilisez `grep` sur une autre machine de type Unix mais en attendant, vous pouvez très bien vous en passer. Le `-E` a été conservé pour des raisons de compatibilité.

## sort : trier les lignes

La commande `sort` se révèle bien utile lorsqu'on a besoin de trier le contenu d'un fichier.

Pour nos exemples, je vous propose de créer un nouveau fichier (avec `nano` par exemple) appelé `noms.txt` et d'y placer le texte suivant :

```
François
Marcel
Albert
Jean
Stéphane
patrice
Vincent
jonathan
```

Ensuite, exécutez la commande `sort` sur ce fichier :

```
$ sort noms.txt

Albert
François
Jean
jonathan
Marcel
patrice
Stéphane
Vincent
```

Le contenu du fichier est trié alphabétiquement et le résultat est affiché dans la console.
Vous noterez que `sort` ne fait pas attention à la casse (majuscules / minuscules).

### `-o` : écrire le résultat dans un fichier

Le fichier en lui-même n'a pas été modifié lorsque nous avons lancé la commande. Seul le résultat était affiché dans la console.

Vous pouvez faire en sorte que le fichier soit modifié en précisant un nom de fichier avec l'option `-o` :

```
sort -o noms_tries.txt noms.txt
```

... écrira la liste de noms triés dans `noms_tries.txt` .

### `-r` : trier en ordre inverse

L'option `-r` permet d'inverser le tri :

```
$ sort -r  noms.txt

Vincent
```

```
Stéphane
patrice
Marcel
jonathan
Jean
François
Albert
```

### `-R` : trier aléatoirement

Cette option permet de trier aléatoirement les lignes d'un fichier. C'est assez amusant et ça peut se révéler utile dans certains cas :

```
$ sort -R noms.txt

patrice
François
Marcel
jonathan
Jean
Albert
Vincent
Stéphane
```

### `-n` : trier des nombres

Le tri de nombres est un peu particulier. En effet, la commande `sort` ne distingue pas si les caractères sont des nombres et va donc par défaut les trier par ordre alphanumérique, en prenant en compte le premier chiffre uniquement. Par conséquent, le « mot » 129 précèdera 42 alors que ça devrait être l'inverse !

Prenons un exemple. Créez un nouveau fichier `nombres.txt` et placez-y les nombres suivants :

```
36
16
42
129
27
364
```

Triez-les comme vous avez appris à le faire :

```
$ sort nombres.txt

```

```
129
16
27
36
364
42
```

Comme vous voyez, tout ce qui commence par 1 est en premier, puis vient ce qui commence par 2 et ainsi de suite.
Bien sûr, quand on veut trier des nombres, c'est n'importe quoi.

C'est là que l'option `-n` intervient. Elle permet de trier en considérant le texte comme des nombres. Cette fois, le nombre 42 sera bien placé avant 129 !

```
$ sort -n nombres.txt

16
27
36
42
129
364
```

Magique. ;-)

## wc : compter le nombre de lignes

La commande `wc` signifie *word count*. C'est donc a priori un compteur de mots mais en fait, on lui trouve plusieurs autres utilités : compter le nombre de lignes (très fréquent) et compter le nombre de caractères.

Comme les précédentes, la commande `wc` travaille sur un fichier.

Sans paramètre, les résultats renvoyés par `wc` sont un peu obscurs. Voyez plutôt :

```
$ wc noms.txt

 8  8 64 noms.txt
```

Ces trois nombres signifient, dans l'ordre :

1. le nombre de lignes.
2. le nombre de mots.
3. le nombre d'octets.

Il fallait le savoir !

Dans le cas de notre fichier `noms.txt`, il est normal d'avoir autant de lignes que de mots car nous avions mis un seul mot par ligne.

### `-l` : compter le nombre de lignes

Pour avoir uniquement le nombre de lignes, utilisez `-l` :

```
$ wc -l noms.txt

8 noms.txt
```

### `-w` : compter le nombre de mots

Combien de mots différents y a-t-il dans le fichier ?

```
$ wc -w noms.txt

8 noms.txt
```

### `-c` : compter le nombre d'octets

Combien d'octets comporte le fichier ?

```
$ wc -c noms.txt

64 noms.txt
```

### `-m` : compter le nombre de caractères

Ah, voilà une information qui ne nous a pas été donnée lorsque nous avons lancé la commande `wc` sans paramètre.

L'option `-m` renvoie le nombre de caractères :

```
$ wc -m noms.txt

62 noms.txt
```

Comme vous pouvez le voir, le nombre de caractères est différent du nombre d'octets.

## uniq : supprimer les doublons

Parfois, certains fichiers contiennent des lignes en double et on aimerait pouvoir les détecter ou les supprimer. La commande `uniq` est toute indiquée pour cela.

Nous devons travailler sur un fichier **trié**. En effet, la commande `uniq` ne repère que les lignes successives qui sont identiques.
Je vous propose de créer un fichier `doublons.txt` contenant les noms suivants :

```
Albert
François
François
François
Jean
jonathan
Marcel
Marcel
patrice
Stéphane
Vincent
```

Il y a des noms en double (et même en triple) dans ce fichier. Appliquons un petit coup de `uniq` là-dessus pour voir ce qu'il en reste :

```
$ uniq doublons.txt

Albert
François
Jean
jonathan
Marcel
patrice
Stéphane
Vincent
```

La liste de noms sans les doublons s'affiche alors dans la console !

Vous pouvez demander à ce que le résultat sans doublons soit écrit dans un autre fichier plutôt qu'affiché dans la console :

```
uniq doublons.txt sans_doublons.txt
```

La liste sans doublons sera écrite dans `sans_doublons.txt` .

### `-c` : compter le nombre d'occurrences

Avec `-c`, la commande `uniq` vous affiche le nombre de fois que la ligne est présente dans le fichier :

```
$ uniq -c doublons.txt
      1 Albert
      3 François
      1 Jean
      1 jonathan
      2 Marcel
      1 patrice
      1 Stéphane
      1 Vincent
```

On sait ainsi qu'il y a trois fois « François », une fois « Jean », deux fois « Marcel », etc.

### `-d` : afficher uniquement les lignes présentes en double

L'option `-d` demande à afficher uniquement les lignes présentes en double :

```
$ uniq -d doublons.txt

François
Marcel
```

Comme seuls François et Marcel avaient des doublons, on les voit ici s'afficher dans la console.

Comme pour les autres commandes présentées dans ce chapitre, je ne vous ai pas fait la liste de toutes les options disponibles. J'ai choisi de vous présenter celles qui me paraissaient les plus intéressantes ou les plus utiles, mais c'est tout à fait subjectif. Ayez le réflexe d'aller regarder le manuel ( `man uniq` par exemple) pour connaître la liste exhaustive des options de la commande.

## cut : couper une partie du fichier

Vous avez déjà coupé du texte dans un éditeur de texte, non ?
La commande `cut` vous propose de faire cela au sein d'un fichier afin de conserver uniquement une partie de chaque ligne.

### Couper selon le nombre de caractères

Par exemple, si vous souhaitez conserver uniquement les caractères 2 à 5 de chaque ligne du fichier, vous taperez :

```
$ cut -c 2-5 noms.txt

ran
arce
lber
ean
tép
atri
ince
onat
```

`cut` a quelques soucis avec les mots contenant des accents. Comme vous pouvez le voir, certains mots ici coupés ont quatre lettres (comme prévu) alors que d'autres en ont trois. Ceci est dû à l'encodage des caractères, aux accents. La commande `cut` se base sur le nombre d'octets, et comme nous l'avons vu plus tôt, celui-ci n'est pas forcément égal au nombre de caractères. À l'heure actuelle on ne peut rien y faire, c'est la commande `cut` qui devra être mise à jour par les programmeurs.

Pour conserver du 1er au 3ème caractère :

```
$ cut -c -3 noms.txt

Fra
Mar
Alb
Jea
St
pat
Vin
jon
```

Comme vous pouvez le voir, si on ne met pas de chiffre au début, `cut` comprend que vous voulez parler du premier caractère.

De même, pour conserver du 3ème au dernier caractère :

```
$ cut -c 3- noms.txt

ançois
rcel
bert
an
```

```
éphane
trice
ncent
nathan
```

Là encore, pas besoin de donner le numéro du dernier caractère, la commande `cut` comprend comme une grande qu'elle doit couper jusqu'à la fin.

## Couper selon un délimiteur

Faisons maintenant quelque chose de bien plus intéressant. Plutôt que de s'amuser à compter le nombre de caractères, nous allons travailler avec ce que l'on appelle un **délimiteur**.

Prenons un cas pratique : les fichiers CSV (*Comma Separated Values.* Ce sont des fichiers dont les valeurs sont séparées par des virgules. Notez qu'Excel utilise plutôt le point-virgule comme séparateur, mais le principe reste le même.) Vous en avez peut-être déjà vu : ils sont générés par des tableurs — tels qu'Excel ou Calc — pour faciliter l'échange et le traitement de données.

Imaginons que vous ayez une (petite) classe et que vous rendiez les notes du dernier contrôle. Vous avez fait un joli tableur et vous avez enregistré le document au format CSV. Le fichier sur lequel nous allons nous baser sera le suivant :

```
Fabrice,18 / 20,Excellent travail
Mathieu,3 / 20,Nul comme d'hab
Sophie,14 / 20,En nette progression
Mélanie,9 / 20,Allez presque la moyenne !
Corentin,11 / 20,Pas mal mais peut mieux faire
Albert,20 / 20,Toujours parfait
Benoît,5 / 20,En grave chute
```

Comme le nom CSV l'indique, les virgules servent à séparer les colonnes. Ces dernières contiennent, dans l'ordre :

- le prénom ;
- la note ;
- un commentaire.

C'est un exemple tout à fait fictif, bien entendu. ;-)

Créez, avec le texte que je viens de vous donner, un nouveau fichier que vous appellerez par exemple `notes.csv` .

Imaginons que nous souhaitions extraire de ce fichier la liste des prénoms. Comment nous y prendrions-nous ?

On ne peut pas utiliser la technique qu'on vient d'apprendre car les prénoms ne font pas tous la même longueur. Nous allons donc nous servir du fait que nous savons que la virgule sépare les différents champs dans ce fichier.

Vous allez avoir besoin d'utiliser deux paramètres :

- `-d` : indique quel est le délimiteur dans le fichier ;
- `-f` : indique le numéro du ou des champs à couper.

Dans notre cas, le délimiteur qui sépare les champs est la virgule.
Le numéro du champ à couper est 1 (c'est le premier).

Testez donc ceci :

```
$ cut -d , -f 1 notes.csv

Fabrice
Vincent
Sophie
Mélanie
Corentin
Albert
Benoît
```

C'est pas beau, ça ? :-)

Après le `-d`, nous avons indiqué quel était le délimiteur (à savoir la virgule « `,` »).
Après le `-f`, nous avons indiqué le numéro du champ à conserver (le premier).

Si nous voulons juste les commentaires :

```
$ cut -d , -f 3 notes.csv

Excellent travail
Nul comme d'hab
En nette progression
Allez presque la moyenne !
Pas mal mais peut mieux faire
Toujours parfait
En grave chute
```

Pour avoir les champs n°1 et n°3 (le prénom et le commentaire) :

```
$ cut -d , -f 1,3 notes.csv
```

```
Fabrice,Excellent travail
Vincent,Nul comme d'hab
Sophie,En nette progression
Mélanie,Allez presque la moyenne !
Corentin,Pas mal mais peut mieux faire
Albert,Toujours parfait
Benoît,En grave chute
```

De même, il est possible de conserver toute une série de champs avec le tiret comme tout à l'heure : `cut -d , -f 2-4 notes.csv` a pour effet de conserver les champs n°

2, 3 et 4.
D'autre part, `cut -d , -f 3- notes.csv` conserve les champs du n°3 jusqu'à la fin.

Vous êtes bien obligés d'admettre que, quand on sait bien s'en servir, la console de Linux peut vous permettre d'effectuer des opérations vraiment puissantes que vous ne pensiez même pas pouvoir faire aussi simplement jusqu'à présent. ;-)

## En résumé

- `grep` est une commande couramment utilisée pour rechercher un mot dans un fichier.
- On peut utiliser des expressions régulières, un système complexe mais puissant, pour effectuer des recherches précises. On fait dans ce cas appel à la commande `egrep` .
- `sort` trie des lignes de texte par ordre alphabétique. Le paramètre `-n` permet de trier par ordre numérique.
- `wc` compte le nombre de lignes, de mots et d'octets dans un fichier.
- `uniq` supprime les doublons d'un fichier.
- `cut` coupe une partie d'un fichier.

**Que pensez-vous de ce cours ?**

J'AI TERMINÉ CE CHAPITRE ET JE PASSE AU SUIVANT

## Le professeur

**Mathieu Nebra**

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

QUIZ : QUIZ 2

LES FLUX DE REDIRECTION ›

## Découvrez aussi ce cours en...


Livre


PDF

EN PLUS

Boutique

Mentions légales

Conditions générales d'utilisation

Politique de protection des données personnelles

Cookies

Accessibilité

Français

Télécharger dans l'App Store