

[S'INSCRIRE](#)[SE CONNECTER](#)[Accueil](#) ▶ [Cours](#) ▶ [Reprenez le contrôle à l'aide de Linux !](#) ▶ [Extraire, trier et filtrer des données](#)

Reprenez le contrôle à l'aide de Linux !



30 heures Facile

Licence



Extraire, trier et filtrer des données

[Connectez-vous](#) ou [inscrivez-vous](#) gratuitement pour bénéficier de toutes les fonctionnalités de ce cours !

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 découvrirez 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

En naviguant sur OpenClassrooms, vous acceptez notre [politique de cookies](#).

[ACCEPTER](#)

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 livre [*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
```

En naviguant sur OpenClassrooms, vous acceptez notre [politique de cookies](#).

ACCEPTER ✓

```
#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 successive 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
-------------------	---------------

<code>.</code>	Caractère quelconque
----------------	----------------------

En naviguant sur OpenClassrooms, vous acceptez notre [politique de cookies](#).

ACCEPTER

Caractère spécial	Signification
<code>^</code>	Début de ligne
<code>\$</code>	Fin de ligne
<code>[]</code>	Un des caractères entre les crochets
<code>?</code>	L'élément précédent est optionnel (peut être présent 0 ou 1 fois)
<code>*</code>	L'élément précédent peut être présent 0, 1 ou plusieurs fois
<code>+</code>	L'élément précédent doit être présent 1 ou plusieurs fois
<code> </code>	Ou
<code>()</code>	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]alias .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 mon livre PHP [*Concevez votre site web avec PHP et MySQL*](#) (Livre du Zéro) ou sur le Site du Zéro des explications plus complètes.

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



wc : compter le nombre de lignes



uniq : supprimer les doublons



cut : couper une partie du fichier



QUIZ : QUIZ 2

LES FLUX DE REDIRECTION



Le professeur

Mathieu Nebra

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

Découvrez aussi ce cours en...



Livre



PDF

OpenClassrooms

Le blog OpenClassrooms

Recrutement

Devenez mentor

Nous contacter

Professionnels

Affiliation

For Business


En plus

Qui sommes-nous ?

Fonctionnement de nos cours

Conditions Générales d'Utilisation

Politique de Protection des Données Personnelles

 Français ▼



Télécharger dans
l'App Store

