Introduction à Linux RAQ - Cours 02

Eric Normandeau - 2015-02-05

Plan de cours

1 - Introduction

Dans ce deuxième cours d'introduction à Linux, nous allons poursuivre notre exploration des commandes importantes. Je vais de plus en plus considérer que la matière déjà vue est acquise, mais n'hésitez surtout pas si vous avez besoin d'un rappel sur une fonction, un paramètre, une manière de faire les choses, etc.

Nous allons aussi parler d'entrée standard, de sortie standard et d'erreur standard (**`standard input`**, **`standard_output`** et **`standard_error`**) et voir plus d'exemples de redirection de sortie de programmes. Nous verrons comment rediriger la sortie d'un programme directement dans un autre programme, ce qu'on appelle un **`pipeline`**, ce qui nous permettra de commencer à manipuler des fichiers texte.

1.1 - Retour sur les exercices

Réponse aux questions relatives aux exercices de la semaine passée.

1.2 - Importer le matériel du cours 02

Nous allons copier un dossier déjà préparé pour le cours 02 avec la commande **`cp`**, que nous allons revoir plus tard :

```
cd  # Pour retourner dans notre dossier d'utilisateur
cp -r /cours_intro_linux/cours_02 .
```

Toutes les commandes du cours utilisant des fichiers sont lancées à partir du dossier **`/home/username/cours_02/01_fichiers`**.

2 - Dossiers importants

Voici un très court résumé des dossiers spéciaux que nous avons déjà vu et des dossiers système.

2.1 - Dossiers spéciaux

- **`.`** : Dossier dans lequel vous vous trouvez présentement.
- **`..`** : Dossier parent de celui dans lequel vous vous trouvez.
- **`~`** : Votre dossier d'utilisateur (eg : **`/home/user02`**)
- **`/`** : Le dossier racine du système.
- **`-`** : Le dossier où vous vous trouviez juste précédemment.

2.2 - Dossiers du système

Il est intéressant de savoir un peu ce que contiennent les dossier qui se trouvent à la racine, mais ça n'est pas essentiel pour utiliser Linux.

- **`/`** : Racine du système
- **`/bin`** : Programmes pour les utilisateurs (*binaries*)
- **`/dev`** : Liens vers composantes physiques (*devices*)
- **`/etc`** : Fichiers de configuration Linux et logiciels
- **`/home`** : Dossiers des utilisateurs (*user home folders*)
- **`/lib`** : Librairies pour logiciels
- **`/media`** : Disques durs externes ou réseau...
- **`/mnt`** : Vieux nom pour **`/media`**
- **`/opt`** : Applications optionnelles
- **`/proc`** : Information des processus et du système
- **`/root`** : Dossier de l'utilisateur **`root`** (administrateur)
- **`/run`** : Services en cours d'utilisation (eg : serveur **`ssh`**)
- **`/sbin`** : Programmes du système (*binaries*)
- **`/srv`** : Services (eg : serveur **`ssh`**)
- **`/sys`** : Fichiers utiles au système
- **`/tmp`** : Fichiers temporaires
- **`/usr`** : Programmes des utilisateurs
- **`/var`** : Fichiers divers (logs, packages, bases de données...)

3 - Commandes système utiles

Certaines commandes nous rendent la vie plus facile. Elles ne nous permettent pas par elles-mêmes d'accomplir des tâches mais font en sorte qu'on peut être plus efficace et mieux suivre ce qui se passe sur le système. Voici quelques-unes de ces commandes :

3.1 - La commande history

La commande **history** liste les commandes qu'on a utilisé dans le passé en ordre chronologique. Cette commande peut être pratique en combinaison avec la commande **grep** que nous allons aborder plus loin pour trouver des commandes complexes que nous avons lancées dans le passé. Lancée seule et sans options, **history** n'est pas toujours très pratique si notre historique de commandes est long.

```
history
```

3.2 - La commande Ctrl-R

Une autre façon de trouver des commandes lancées auparavant est d'utiliser la commande **Ctrl-R**, qui lance une recherche à partir des commandes les plus récentes. Après avoir appuyé sur **Ctrl-R**, il suffit de taper un bout de commande et le terminal cherche en ordre chronologique inverse la première commande qui correspond au texte cherché. Si le terminal trouve une commande qui correspond, vous pouvez chercher plus loin dans le temps en appuyant à nouveau sur **Ctrl-R**. Lorsqu'une commande est affichée, vous pouvez la lancer directement en appuyant sur **Enter** ou la modifier en vous déplaçant dedans avec les flèches à gauche et à droite. Finalement, vous pouvez annuler la recherche avec **Ctrl-C**.

3.3 - La commande Ctrl-C

La commande **Ctrl-C** vous permet de annuler une commande en cours d'exécution, alors que la commande **Ctrl-D** arrête un programme, incluant le terminal **bash** lui-même. Vous pouvez donc utiliser **Ctrl-D** pour quitter votre session de connexion **ssh** au serveur et même pour fermer le terminal. Sur Mac, il est possible que la fenêtre du terminal ainsi *fermé* demeure en fait ouverte. Ce comportement peut être modifié dans les options du terminal.

3.4 - La commande echo

La commande **echo** affiche le texte qu'on lui passe. Avec les bonnes options, on peut composer des messages à l'écran.

```
# Sans aucune option
echo "Bienvenu au deuxième cours"

# Sans retour de ligne
echo -n "Nous sommes le "; date -I
```

3.5 - La commande man

La commande **man** sert à consulter le manuel des commandes disponibles. Le manuel est affiché grâce à la commande **less**.

```
# Chercher les options '-n' et '-e'
man echo
```

3.6 - La commande top

La commande **top** affiche les processus qui utilisent le plus de ressources sur le système. On peut trier les processus en fonction de leur utilisation de différentes ressources (mémoire, processeurs, etc.). On peut même limiter l'affichage à un utilisateur particulier en tapant **u** et en entrant le nom de l'utilisateur. Lancez **top** et tentez d'afficher seulement les processus lancés par votre utilisateur.

```
top
```

Pour sortir de la commande **top**, vous pressez la touche **q**, comme pour sortir de la commande **less**.

3.7 - La commande clear

La commande **clear** permet de vider le terminal et de remettre le curseur en haut. On peut également utiliser **Ctrl-L** pour arriver au même effet.

3.8 - La commande sleep

Finalement, la commande **sleep** sert à marquer une pause en secondes, ce qui peut être utile dans des scripts. On peut également utiliser les options pour que la pause soit plutôt en minutes, heures, etc.

```
sleep 3   # Attendre 3 secondes
sleep 3s  # Identique à la commande précédente
sleep 3m  # Attendre 3 minutes
sleep 3h  # Attendre 3 heures

sleep 3; echo "Hello"
```

3.8 - Expansion automatique avec '*' (globing)

On peut parfois utiliser le symbole ***** pour dire *n'importe quelle chaîne de caractères*. Par exemple, si on veut lister la taille de tous les fichiers fasta d'un dossier, on peut utiliser la commande suivante.

```
ls -lh *.fasta

# Voici ce que la commande 'ls' reçoit en fait:
echo *.fasta
```

4 - Entrées et sorties

4.1 - Sortie standard (standard output)

Il y a différents types d'entrées et de sorties dans le terminal. Par exemple, le résultat de plusieurs commandes est affiché directement dans le terminal. On appelle cette sortie la sortie standard (**`standard output`**). Nous pouvons décider de rediriger le **`standard output`** dans un fichier en utilisant le symbole **>**.

```
# Afficher la sortie à l'écran
echo "Texte de sortie standard"

# Rediriger la sortie dans un fichier
echo "Texte de sortie standard" > echo_output.temp

cat echo_output.temp
```

Il faut noter que si on utilise le symbole **>**, le fichier dans lequel nous redirigeons la sortie sera créé au besoin. Par contre, si il existe déjà, **son contenu sera remplacé !**. Il faut donc être prudent quand nous utilisons la redirection vers un fichier.

Si nous souhaitons plutôt *ajouter* la sortie à la fin d'un fichier existant, nous pouvons utiliser le double symbole **>>**. Encore une fois, le fichier sera créé au besoin mais si il est déjà présent, la sortie standard sera ajoutée à la fin du fichier.

```
echo "Date:" > la_date.temp  # Un seul '>' pour vider le fichier
date >> la_date.temp
echo "..." >> la_date.temp
echo "La date 3 secondes plus tard" >> la_date.temp
sleep 3; date >> la_date.temp

cat la_date.temp
```

4.2 - Erreur standard (standard error)

Afin de différencier les sorties de programmes et les messages d'erreurs, ces derniers sont envoyés vers l'erreur standard (**standard error**). Par exemple, si on lance la commande **cat** avec un nom de fichier inexistant, on obtient une erreur. Il est parfois difficile de différencier les sorties standards (**standard output**) et les erreurs standards (**standard error**) car elles sont toutes deux affichées à l'écran. Cependant, l'erreur standard peut être redirigée en utilisant un autre symbole : **2>**.

```
cat la_date.temp  # Pas d'erreur = sortie standard
cat inexistant.temp  # Erreur = erreur standard
```

Nous allons essayer de différencier la sortie standard et l'erreur standard en les redirigeant dans deux fichiers différents.

```
cat la_date.temp > output.temp 2> error.temp
echo -e "\n==> Begin Output"; cat output.temp; echo -e "==> End Output\n"
echo -e "\n==> Begin Error"; cat error.temp; echo -e "==> End Error\n"
```

Cette fois ci, nous allons donner à la commande **cat** un nom de fichier qui n'existe pas pour voir ce qui arrive avec la sortie et l'erreur standard.

```
cat inexistant.temp > output.temp 2> error.temp
echo -e "\n==> Begin Output"; cat output.temp; echo -e "==> End Output\n"
echo -e "\n==> Begin Error"; cat error.temp; echo -e "==> End Error\n"
```

Finalement, pour rediriger à la fois la sortie standard et l'erreur standard, nous devons utiliser **&>** pour rediriger vers un fichier et **2>&1** pour traiter l'erreur standard comme de la sortie standard. Ce dernier truc nous servira dans la section 4.4 qui porte sur les pipelines.

4.3 - Entrée standard (standard input)

On peut passer l'entrée standard (**standard input**) à une commande en lui donnant un nom de fichier mais également en utilisant le symbole **<**. Cette méthode n'est pas utilisée souvent.

```
cat < la_date.temp
```

4.4 - Pipelines

Les pipelines sont une des formes de redirection les plus importantes. On construit un pipeline en utilisant le symbole **|** (appelé *pipe* et prononcé comme en anglais). Le pipeline permet de passer la sortie standard d'une commande directement dans l'entrée standard de la commande suivante. De cette façon, on peut traiter la sortie d'une commande avec un ou plusieurs autres commandes sans avoir à écrire le résultat dans un fichier intermédiaire. Les pipelines seront particulièrement utiles lorsque nous allons extraire ou de formater l'information contenue dans des fichiers texte.

Voici un exemple très simple de pipeline avec seulement deux commandes. Nous allons compter le nombre de fois où on retrouve le mot **Alice** dans le fichier **alice.txt** avec les commandes **grep** et **wc** :

```
grep -o Alice alice.txt | wc -l
```

La première commande extrait toutes les occurrences du patron de recherche (ici c'est simplement 'Alice') et les met une par ligne. Cette sortie est alors directement passée à la commande suivante comme s'il s'agissait d'un fichier contenant du texte et celle-ci nous retourne le nombre de lignes. Cette même formule peut être utilisée pour différentes applications. Par exemple, nous pourrions être intéressé à savoir combien de fois la séquence **ACTG** se retrouve dans un fichier de séquences.

```
grep -o ACTG sequences_01.txt | wc -l
```

En variant un peu cette recette, on peut répondre rapidement à des questions plus intéressantes à propos de nos jeux de données, même si ils sont contenus dans de très gros fichiers.

5 - Manipulation de texte

Les ordinateurs et les besoins des utilisateurs ont beaucoup changés au fil des années. Cependant, une tendances demeure : nous dépendons des fichiers texte (par opposition aux fichiers binaires, comme les fichiers produits par les logiciels de la suite Microsoft Office, par exemple). Nos données brutes sont dans des fichiers texte. Les paramètres, entrées et sorties des programmes sont dans des fichiers texte. Les fichiers de configuration sont des fichiers textes. La raison est simple : il est facile de produire, éditer, modifier, chercher dans et lire des fichiers texte. Toute la notion de pipeline repose sur le fait que les commandes UNIX utilisent comme entrées des fichiers texte et produisent en sortie des fichiers texte.

Dans cette section, nous allons explorer quelques commandes utiles pour manipuler du texte. Pour présenter les différentes commandes, nous allons utiliser des exemples qui produisent des résultats qui sont peu utiles en eux-mêmes mais qui permettent de mieux comprendre les commandes et la construction de pipelines.

5.1 - La commande grep

La commande **grep** permet de trouver des lignes qui contiennent soit un mot ou un patron de recherche, qui peut être une expression régulière complexe. Voici quelques exemples assez simples. Nous pourrons accomplir des recherches plus complexes quand nous maîtriserons les expressions régulières, un mini langage spécialisé dans la recherche et le remplacement de texte, que nous verrons lors du prochain cours.

```
grep Alice alice.txt  # Cherche les lignes contenant 'Alice'
grep Alice alice.txt | wc -l  # Compte le nombre de ces lignes
```

La commande **grep** possède plusieurs options utiles qui sont présentées brièvement dans les deux prochaines sous-sections.

5.1.1 - Quelques options très utiles

- **-c** : Afficher le nombre de lignes trouvées plutôt que les lignes
- **-v** : Rechercher les lignes où le patron n'est **pas** trouvé
- **-i** : Rechercher en minuscules ou en majuscules
- **-E** : Utiliser des expressions régulières complexes
- **-o** : Afficher seulement la partie qui correspond au patron

Quelques exemples :

```
# Compter le nombre de lignes avec Alice
grep -c Alice alice.txt

# Compter le nombre de lignes sans Alice
grep -vc Alice alice.txt

# Compter un mot sans se soucier des majuscules
grep -c " the " alice.txt  # Compter " the "
grep -c " The " alice.txt  # Compter " The "
grep -ci " the " alice.txt  # Compter " the " et " The "

# Trouver les noms de chapitre
grep -iEo "^chapter.*" alice.txt
```

Attention avec les fichiers fasta !

Le symbole **>** est important pour différencier les séquences dans un fichier fasta. Or, ce même symbole est aussi utilisé par le terminal pour signifier la redirection de la sortie standard. Lorsqu'on veut rechercher les noms des séquences contenues dans un fichier fasta avec la

commande **grep**, **il est crucial de mettre le symbole > entre guillemets** ! Si vous ne le faites pas, le systèmes croira que vous souhaitez écrire dans le fichier en question et il sera écrasé. Voici la syntaxe à utiliser.

```
# Compter les séquences dans un fichier fasta
grep -c ">" sequences_01.fasta

# Compter les séquences pour TOUS les fichiers fasta
grep -c ">" *.fasta
```

5.1.2 - Quelques options moins fréquentes mais utiles

- **-A** : Afficher la ligne plus n lignes **après** celle-ci
- **-B** : Afficher la ligne plus n lignes **avant** celle-ci
- **-C** : Afficher la ligne plus n lignes **avant et après** celle-ci
- **-f** : Rechercher les patrons qui se trouvent dans un fichier
- **-R** : Rechercher dans tous les fichiers d'un dossier

Quelques exemples :

```
# Lignes avec patron plus 3 lignes de contexte avant et après
grep -C 3 "Alice said nothing" alice.txt

# Lignes avec un des patrons dans le fichier 'patterns_for_grep'
grep -i -f patterns_for_grep alice.txt | wc -l
```

5.2 - La commande sort

La commande **sort** permet de trier des lignes. On peut trier en ordre croissant ou décroissant, alphabétiquement ou numériquement, etc.

```
cat patterns_for_grep
sort patterns_for_grep
```

5.2.1 - Quelques options utiles

- **-r** : Trier en ordre inverse (reverse)
- **-u** : Trier et enlever les lignes identiques
- **-n** : Trier en ordre numérique
- **-h** : Trier tailles de fichiers (eg : 10Ko < 10Mo < 10Go)
- **-V** : Trier numéros de version (eg : v1.2 < v10.1)

Les options **-h -V** ne sont pas disponibles sur tous les systèmes, entre autres sur MacOS.

Quelques exemples :

```
sort -r patterns_for_grep

# Compter le nombre de lignes uniques
sort -u alice.txt | wc -l

# Afficher les fichiers et dossier triés par taille
du -sh * | sort -hr
```

5.3 - La commande uniq

Cette commande enlève les lignes répétées. En utilisant son option **-c** en combinaison avec la commande **sort**, elle retourne le nombre de fois où chaque ligne a été trouvée.

Comparer le résultat des deux commandes suivantes

```
# Nombre de lignes dans le fichier
wc -l alice

# Trouver le nombre de lignes uniques
sort -u alice.txt | wc -l

# Trouver lignes les plus fréquentes
sort alice.txt | uniq -c | sort -nr | head -n 20

# Trouver les codons les plus fréquents
grep -v ">" codons.fasta | sort | uniq -c | sort -nr | head
```

5.4 - La commande cut

La commande **cut** permet d'extraire des parties de lignes, soit par colonnes ou par position des caractères dans la ligne.

5.4.1 - Quelques options utiles

- **-f** : Lister les colonnes (field) à garder
- **-d** : Spécifier le caractère séparateur de colonnes
- **-c** : Spécifier les caractères à garder

```
# Garder seulement la deuxième colonne
cut -f 2 work.csv
```

```
# Garder les colonnes 2 et 3
cut -f 2,3 work.csv

# Garder les caractères 1, 3 et 5
cut -c 1,3,5 alice.txt | head -20

# Garder les caractères 10 à 20
cut -c 10-20 alice.txt | head -20

# Garder les caractères du début de la ligne jusqu'au 15ième
cut -c -15 alice.txt | head -20
```

6 - Mot de la fin

6.1 - Aujourd'hui, nous avons vu :

- Révision rapide des dossiers importants
- Quelques commandes système utiles
- Les entrées, les sorties et la redirection
- Les pipelines
- Quelques commandes pour manipuler du texte

Nous avons vu quelques commandes qui permettent de manipuler du texte mais pour l'instant nos capacités sont limitées par le fait que nous n'avons pas encore vu les expressions régulières (**`regular expressions`** ou **`regex`**) ni comment utiliser la commande **`perl`** pour rechercher et remplacer du texte. Ces deux outils seront au coeur du prochain cours. Il faudra alors bien maîtriser les commandes déjà vues ainsi que comment construire des pipelines.

6.2 - Au prochain cours, nous verrons :

- Manipulation de texte plus avancée
- Expressions régulières
- Plus de magie avec perl, sed et awk
- Compression et décompression de fichiers

6.3 - Questions et suggestions

N'hésitez pas à me poser vos questions durant les cours ou par courriel. Je vais tenter d'y répondre durant les cours. Je vais aussi prendre vos suggestions en note pour tenter d'améliorer le cours.

7 - Exercices

7.1 - Commandes utiles

- Afficher l'historique des commandes (commande : **`history`**)
- Rechercher la dernière commande où vous avez utilisé **`cp -r`** (commande : **`Ctrl-R`**)
- Lancer la commande **`cat`** sans option et arrêtez la (commande : **`Ctrl-C`**)
- Afficher votre nom suivi de la date dans le terminal (commande : **`echo`**)
- Afficher le manuel de la commande **`echo`** (commande : **`man`**)
- Afficher les processus courants (commande : **`top`**)
- Vider le terminal du texte affiché (commande : **`clear`** ou **`Ctrl-L`**)
- Faire une pause de 3 secondes (commande : **`sleep`**)
- Afficher votre nom mais envoyer le résultat dans un fichier (commande : **`>`**)
- Ajouter la date à la fin de ce fichier (commande : **`>>`**)
- Afficher les 10 dernières commandes lancées grâce à un pipeline (commande : **`history`**, **`|`**, **`tail`**)

7.2 - grep

- Compter le nombre de séquences dans tous les fichiers .fasta (commande : **`grep`**)
- Trouver tous les noms de séquences du fichier sequences_02.fasta (commande : **`grep`**)
- Trouver toutes les séquences du fichier sequences_02.fasta (commande : **`grep`**)

7.3 - sort

- Trier en ordre alphabétique les lignes du fichier **`work.csv`** (commande : **`sort`**)
- Trier en ordre alphabétique inverse les lignes du fichier **`work.csv`** (commande : **`sort`**)
- Bonus : Trier en ordre numérique de la colonne 2 les lignes du fichier **`work.csv`** (commande : **`sort`**)

7.4 - uniq

- Compter le nombre de lignes dans le fichier **`repeated_lines.txt`** (commande : **`wc`**)
- Compter le nombre de lignes après avoir enlevé les lignes consécutives identiques (commande : **`uniq`** et **`wc`**)
- Compter le nombre de lignes uniques (commande : **`sort`** et **`wc`**)
- Compter le nombre de fois où chaque ligne est représentée (commande : **`sort`**, **`uniq`**)

7.5 - cut

- Extraire la 3ième colonne du fichier **`work.csv`** (commande : **`cut`**)

- Extraire la 1ère et la 2ième colonne du fichier **work.csv** (commande : **cut**)
- Extraire les 20 premiers caractères de chaque ligne du fichier **repeated_lines.txt** (commande : **cut**)

7.6 - Pipelines

- Afficher les 10 dernières commandes lancées grâce à un pipeline (commande : **history**, **|**, **tail**)
- Afficher les 20 première lignes de **alice.txt** et choisissez un mot fréquent (commande : **head** et **less**)
- Compter le nombre de lignes où ce mot apparaît dans le même fichier (commande : **grep** et **wc**)
- Trouver combien de séquences contiennent **ACCGTA** dans **sequences_02.fasta** (commande : **grep** et **wc**)
- Trouver combien de séquences débutent par **ATT** dans **sequences_02.fasta** (commande : **grep** et **wc**)

8 - Liste de commandes importantes

Voici une courte liste des commandes que nous avons utilisée aujourd'hui. Entre parenthèses, vous trouverez le nom en anglais de la commande (pour vous aider à retenir la commande). Entre crochets, vous trouverez les options les plus souvent utilisées :

8.1 - Commandes système utiles

- **`history`** : Afficher historique des commandes
- **`echo`** : Imprimer texte à l'écran **`[-n, -e]`**
- **`man`** : Afficher le manuel des commandes (manual)
- **`top`** : Afficher les processus gourmands
- **`sleep`** : Faire une pause **`[s, m, h]`**

8.2 - Entrées et sorties

- **`>`** : Rediriger la sortie standard (redirect standard output)
- **`>>`** : Ajouter la sortie standard à la fin d'un fichier (append)
- **`2>`** : Rediriger l'erreur standard (redirect standard error)
- **`&>`** : Rediriger à la fois la sortie et l'erreur standard
- **`2>&1`** : Joindre l'erreur standard et la sortie standard
- **`<`** : Spécifier l'entrée standard (standard input)
- **`|`** : Caractère pour créer un pipeline (pipe)

8.3 - Manipuler du texte

- **`wc`** : Afficher le nombre de lignes d'un fichier (word count) **`[-l]`**
- **`cat`** : Afficher le contenu d'un fichier (concatenate)
- **`head`** : Afficher le début d'un fichier **`[-n]`**
- **`tail`** : Afficher la fin d'un fichier **`[-n]`**
- **`less`** : Lire un fichier
- **`grep`** : Chercher du texte (get regular expression) **`[-c, -v, -i, -E, -o, -A, -B, -C, -f, -R]`**
- **`sort`** : Trier des lignes **`[-r, -u, -n, -h, -V]`**
- **`uniq`** : Enlever lignes répétées (unique) **`[-c]`**
- **`cut`** : Extraire colonnes ou caractères **`[-f, -d, -c]`**