Introduction à Linux RAQ - Cours 05

Eric Normandeau - 2015-03-09

Plan de cours

1 - Introduction

1.1 - Annonces

Le cours d'aujourd'hui sera le dernier de la série d'introduction à Linux.

1.2 - Retour sur screen

Certaines personnes ont eu de la difficulté avec **`screen`**. Une des choses importantes à garder en tête est qu'il est préférable de sortir d'une session avant d'en créer une nouvelle. Sinon, on crée des sessions imbriquées une dans l'autre et il est plus difficile de s'y reconnecter.

1.3 - Retour sur les exercices

Réponse aux questions relatives aux exercices de la semaine passée.

Créer fichier haiku

- Taper le texte suivant dans **`haiku.txt`**

```
Chaque fleur qui tombe
Fait vieillir d'avantage
Les branches du prunier

- Yosa Buson (1716 - 1783)
```

Écrire script bash

- **`print_haiku.sh <nom> <fichier>`**
- Imprime **`Bonjour <nom>, voici un haiku:`**
- Imprime ensuite le poème
- Rendre exécutable
- Installer dans dossier **`~/programmes`**

Créer alias

- Créer un alias **`haiku`** dans **`~/.bashrc`**
- Utilise **`print_haiku.sh`** et variable $USER
- Source **`~/.bashrc`**
- Tester l'alias

1.4 - Importer le matériel du cours 05

Nous allons copier un dossier déjà préparé pour le cours 05 avec la commande **`cp`**, que nous allons revoir plus tard :

```
cd  # Pour retourner dans notre dossier d'utilisateur
cp -r /cours_intro_linux/cours_05 .
```

Toutes les commandes du cours utilisant des fichiers sont lancées à partir du dossier **`/home/username/cours_05/`**.

2 - Transfers et téléchargements

Il existe deux commandes principales pour transférer des données entre des ordinateurs UNIX : **scp** et **rsync**. Les deux s'utilisent de façon similaire à la commande **cp**. Les commandes **wget** et **curl** servent à télécharger des données à partir d'Internet.

Dans cette section, nous allons simuler le transfert entre serveurs puisque vous n'avez pas tous accès à d'autres serveurs à partir desquels ou vers lesquels transférer des données. La première fois où vous transfèrerez des données, il est possible que le serveur vous demande de confirmer (en tapant **yes**) que vous faites confiance au serveur où vous vous connectez.

2.1 - scp

La commande **scp**, pour *secure copy*, utilise le protocole **ssh** pour transférer les données. Ces données sont donc encodées durant le transfert.

Similaire à la commande cp

La syntaxe de base de la commande **scp** est très similaire à celle de la commande **cp**.

```
cd
cp /cours_intro_linux/fichier.txt ~
scp fichier copie_fichier
```

La source ou la destination peuvent être sur un autre serveur

La différence est que la source, la destination, ou même les deux, peuvent se trouver sur des serveurs à distance.

D'ici vers un autre serveur

```
# Effacer le fichier copie
rm copie_fichier

# Copier
scp fichier user31@raq.ibis.ulaval.ca:/home/user31/copie_fichier

# Ou
scp fichier user31@raq.ibis.ulaval.ca:~/copie_fichier
```

D'un autre serveur vers ici

```
# Effacer le fichier copie
rm copie_fichier

# Copier
scp user31@raq.ibis.ulaval.ca:/home/user31/fichier copie_fichier

# Ou
scp user31@raq.ibis.ulaval.ca:~/fichier copie_fichier
```

La source **ET** la destination peuvent les deux être sur des serveurs à distance. Il suffit d'utiliser la syntaxe pour spécifier le serveur et le chemin du fichier sur le serveur.

2.2 - rsync

La commande **rsync** joue le même rôle que la commande **scp** mais ajoute des fonctionnalités intéressantes. Par exemple, on peut demander à ce que les données soient compressées durant le transfert, ce qui réduit le temps de transfert de larges fichiers non compressés. La commande **rsync** est également capable de reprendre le transfert là où il était rendu en cas d'interruption accidentelle ou volontaire.

Options à toujours utiliser

```
rsync -avzP Source Destination
```

L'option **a** veut dire *archive*. Elle permet entre autre de garder les permissions des fichiers identiques. L'option **v** est pour *verbose*. La commande affichera des détails de son fonctionnement durant le transfert. L'option **z** veut dire de compresser les fichiers durant le transfert. Si les fichiers sont déjà compressés, elle n'est pas nécessaire. Finalement, l'option **P** veut dire à la fois *partial* et *progress*. La commande va conserver les fichiers qui on été partiellement transférés pour pouvoir relancer le transfert exactement là où il était rendu et elle va afficher le progrès de transfert de chaque fichier.

D'ici vers un autre serveur

```
# Effacer le fichier copie
rm copie_fichier

# Faire une copie
rsync -avzP fichier user31@raq.ibis.ulaval.ca:~/copie_fichier
```

D'un autre serveur vers ici

```
# Effacer le fichier copie
rm copie_fichier

# Faire une copie
rsync -avzP user31@raq.ibis.ulaval.ca:~/fichier copie_fichier
```

D'un serveur 1 à un serveur 2

```
# Effacer le fichier copie
rm copie_fichier

# Faire une copie
rsync -avzP user31@raq.ibis.ulaval.ca:~/fichier \
    user31@raq.ibis.ulaval.ca:~/copie_fichier
```

Transfert de dossiers avec rsync

La commande **rsync** peut transférer soit un dossier soit son contenu. La différence au niveau de la syntaxe est minime (présence ou absence du **/**) mais le résultat peut être très différent. En effet, si on transfère seulement le contenu, tous les dossiers et fichiers du dossier à transférer seront placés librement (pas contenu dans un dossier) dans le dossier de destination. Ceci peut être embêtant si le dossier contenait un grand nombre de fichiers.

```
# Pour transférer le dossier et son contenu
rsync -avzP /cours_intro_linux/dossier_avec_100_fichiers Destination

# Pour transférer seulement le contenu (ajouté un /)
rm -r ~/dossier_avec_100_fichiers
rsync -avzP /cours_intro_linux/dossier_avec_100_fichiers/ Destination
```

2.3 - wget

La commande **wget**, pour *web get*, permet de télécharger des fichiers à partir d'un URL (adresse web). Bien que **wget** est très puissant et chargé en options, il est très facile à utiliser dans le cas où nous avons l'adresse URL d'un fichier. Par exemple, pour télécharger le fichier d'archive d'installation de l'éditeur de texte **joe**, on peut utiliser la commande suivante :

```
wget http://downloads.sourceforge.net/project/joe-editor/\
    JOE%20sources/joe-3.7/joe-3.7.tar.gz
```

2.4 - curl

La commande **curl** a largement été remplacée par **wget** mais il est possible que vous la rencontriez à l'occasion.

3 - Installation de programmes

Nous allons maintenant chacun installer la suite d'outils **blastplus** de NCBI. Nous allons également installer notre propre version de l'éditeur de texte **joe**. La procédure est similaire à ce que nous avons déjà fait pour installer nos scripts.

3.1 - Télécharger

Nous utilisons wget pour télécharger les fichiers désirés. Normalement, il faudrait chercher leur adresse dans un navigateur internet et copier le lien.

```
# Outils de la suite blastplus
wget ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/\
    LATEST/ncbi-blast-2.2.30+-x64-linux.tar.gz

# Éditeur de texte joe
wget http://downloads.sourceforge.net/project/joe-editor/\
    JOE%20sources/joe-3.7/joe-3.7.tar.gz
```

3.2 - Décompresser

Nous utilisons maintenant la commande **tar** pour décompresser les deux archives.

```
# Joe
tar xvfz joe-3.7.tar.gz

# Blastplus
tar xvfz ncbi-blast-2.2.30+-x64-linux.tar.gz
```

3.3 - Compiler

Certains programmes, comme **joe**, on besoin d'être compilés pour être installés. La compilation crée des fichiers exécutables qu'on peut par la suite utiliser. D'autre programmes viennent avec des fichier exécutables déjà compilés pour plusieurs plateformes. C'est le cas de **blastplus**. Il suffit alors de choisir les exécutables qui conviennent à l'ordinateur où nous souhaitons installer le programme.

```
# Aller dans le dossier décompressé
cd joe-3.7

# Afficher le contenu
ls

# Lire les fichiers README et INSTALL
less README
less INSTALL

# Compiler
./configure
make

# Trouver les exécutables
ls

# Tester notre logiciel
./joe
```

3.4 - Installer

Il existe quelques façon d'installer des programmes sur un système où on n'est pas administrateur, mais la plus simple est de mettre les fichiers exécutables dans un dossier et d'ajouter ce dossier à la variable **PATH** en éditant le fichier **~/.profile**.

```
# Joe
cp joe ~/programmes

# Blastplus
cd ~/cd ncbi-blast-2.2.30+
ls
cd bin
cp * ~/programmes
```

3.5 - Vérifier l'installation

Nos deux nouveaux programmes sont maintenant installés. Il reste seulement à tester si on peut les lancer.

```
# Retourner dans notre dossier home
cd
```

```
# Joe
joe
which joe

# Blastplus
blastn -h
which blastn
```

Les deux programmes semblent bien installés et nous pourrons les utiliser.

4 - Recherche de séquences similaires avec Blast

Les utilitaires de la suite **blastplus** permettent de rechercher des séquences qui se resemble. On peut ainsi trouver à quel gène notre séquence correspond ou de quelle espèce elle provient.

4.1 - Créer une base de données

Afin d'utiliser **blast**, nous aurons besoin d'une base de données de séquences sur lesquelles faire notre recherche. La suite **blastplus** contient l'outil **makeblastdb** qui sert à créer une database à partir d'un fichier fasta.

```
# Créer la database
makeblastdb -in coi_especes_quebecoises.fasta -title coi \
    -dbtype nucl -out ./blast_databases/coi
```

4.2 - Lancer une recherche

Maintenant que nous avons notre base de donnée de séquences sur lesquelles faire notre recherche, nous allons utiliser blast pour trouver quelles à séquences de la base de données nos séquences d'intérêt ressemblent. La suite **blastplus** contient plusieurs programmes pour chercher la similarité entre différent types de séquences. Comme nous avons des séquences nucléotidiques et que notre base de données contient elle aussi des séquences nucléotidiques, nous utiliserons **blastn**.

```
# Lancer le blast
blastn -db ./blast_databases/coi -query sequences_mystere_12.fasta \
    -evalue 1e-10 -outfmt 0 -max_target_seqs 1 > sequences_mystere_12.coi0
```

4.3 - Formats de sortie

Les résultats de blast peuvent être formatés de différentes manières pour faciliter leur utilisation. Les formats les plus fréquemment utilisés sont les formats 0 et 6. Nous avons utilisé le format 0, qui est le format par défaut de blast. Il est plus long et montre les détails de l'alignement. On peut visualiser le résultat avec la commande **less**.

```
less sequences_mystere_12.coi0
```

Le format 6 est parfois plus intéressant car il est très bref et il est plus facile d'en extraire les informations essentielles.

```
# Lancer le blast
blastn -db ./blast_databases/coi -query sequences_mystere_12.fasta \
    -evalue 1e-10 -outfmt 6 -max_target_seqs 1 > sequences_mystere_12.coi6

# Visualiser le format de sortie
cat sequences_mystere_12.coi6

# Reformater pour mieux voir
column -t sequences_mystere_12.coi6
```

Est-ce que toutes nos séquences d'intérêt sont similaires à une séquence de la base de données ? Quelles sont les espèces représentées ?

```
# Nombre de séquences mystère
grep -c ">" sequences_mystere_12.fasta

# Nombre de séquences qui blastent
wc -l sequences_mystere_12.coi6

# Décompte par espèce
cut -f 2 sequences_mystere_12.coi6 | cut -d "|" -f 1 | sort | uniq -c
```

5 - Boucles et trucs bash

Il est utile de pouvoir lancer une commande ou un script sur un ensemble de fichiers. Une des façons de faire est d'utiliser une boucle et de traiter chaque fichier un après l'autre.

5.1 - Boucle for

La syntaxe générale d'une boucle est la suivante :

```
for i in ITEMS; do ACTION; done
```

Par exemple, pour afficher les items de la série **`1 2 3`**, on peut lancer la boucle suivante.

```
for i in 1 2 3; do echo $i; done
```

5.2 - Boucle while

Nous allons chercher les mots présents dans le fichier **`mots.txt`** dans le fichier **`alice.txt`** et afficher leur compte.

```
cat mots.txt | while read i; do \
    echo -ne "$i:\t"; grep -o "$i" alice.txt | wc -l; done
```

5.3 Utiliser la sortie d'une commande

Il est parfois utile d'insérer la sortie d'une commande directement dans une autre commande. Le terminal **`bash`** accepte deux syntaxe (**`‘cmd‘`** et **`$(cmd)`**). Cependant, la deuxième est de loin préférée car elle est plus facile à lire et on peut imbriquer plusieurs niveaux un dans l'autre.

```
# Utilisé dans echo
cat mots.txt | while read i; do \
    echo -e "$i:\t$(grep -o "$i" alice.txt | wc -l)"; done

# Utilisé pour initialiser un boucle
for i in $(seq 10); do echo "$i seconde(s)"; sleep 1; done
```

6 - Mot de la fin

6.1 - Aujourd'hui, nous avons vu :

- Transfers et télécharger des fichiers et dossiers
- Installer des programmes
- Rechercher de séquences similaires avec Blast
- Boucles et trucs bash

Il s'agissait du dernier cours de la série d'introduction à Linux. Vous aurez appris les bases de l'utilisation du terminal et vous serez prêts à mieux utiliser les ressources de calcul qui sont à votre disposition.

6.2 - 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 les cours que je donnerai dans le future.

7 - 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 :

- **`scp`** : Copier des fichiers d'un serveur à un autre. **`[-r]`**
- **`rsync`** : Copier des fichiers d'un serveur à un autre. **`[-avzP]`**
- **`wget`** : Télécharger des fichiers.
- **`curl`** : Télécharger des fichiers.
- **`make`** : Compiler certains programmes.
- **`makeblastdb`** : Créer une base de données blast. **`[--help]`**
- **`blastn`** : Faire une recherche de similarité entre séquences. **`[--help]`**
- **`for`** : Faire un boucle.
- **`while`** : Faire un boucle.
- **`$(cmd)`** : Extraire la sortie d'une commande.
- **`‘cmd‘`** : Extraire la sortie d'une commande.