Introduction à Linux RAQ - Cours 04

Eric Normandeau - 2015-02-23

Plan de cours

1 - Introduction

1.1 - Annonces

Il n'y aura **pas de cours la semaine prochaine (semaine du 2 mars)** puisqu'il s'agit de la semaine de relâche. Les cours reprendrons mercredi le 11 mars. Il devrait y avoir deux autres cours après celui d'aujourd'hui à moins qu'il n'y ait des demandes spéciales pour couvrir des sujets spécifiques.

Les deux derniers cours ont été plutôt dense en information, donc le cours d'aujourd'hui sera plus léger pour compenser.

1.2 - Retour sur les exercices

Réponse aux questions relatives aux exercices de la semaine passée.

Dans la **`section 6.4`** des exercices, on demandait de trouver les mots les plus fréquent se terminant en **`ing`** dans **`alice.txt`**. Voici la solution :

```
grep -Po '\W\w+ing\W' alice.txt | \
    perl -pe 's/[^\w\n]//g' | \
    sort | uniq -c | sort -nr | head
```

On demandait également de trouver les mots de 8 lettres les plus communs. La solution est très similaire; Seul le patron de recherche est différent :

```
grep -Po '\W\w{8}\W' alice.txt | \
    perl -pe 's/[^\w\n]//g' | \
    sort | uniq -c | sort -nr | head
```

1.3 - Importer le matériel du cours 04

Nous allons copier un dossier déjà préparé pour le cours 04 avec la commande **cp**, que nous allons revoir plus tard :

```
cd  # Pour retourner dans notre dossier d'utilisateur
cp -r /cours_intro_linux/cours_04 .
```

Toutes les commandes du cours utilisant des fichiers sont lancées à partir du dossier **/home/username/cours_04/**.

2 - Éditeurs de texte

Il est utile de connaître au moins un éditeur de texte qu'on peut utiliser dans le terminal. Cela permet d'écrire ou d'éditer des scripts et des fichiers de configuration sur les serveurs où nous travaillons. Il existe des éditeurs de texte faciles à apprendre, tels **nano** et **joe**; Nous allons nous concentrer sur ces deux éditeurs. D'autres éditeurs, comme **vim** et **emacs**, sont beaucoup plus difficiles à apprendre mais sont également beaucoup plus puissants. Nous n'allons pas les voir durant le cours.

2.1 - nano

Le programme **nano** est un éditeur de texte simple à utiliser dans le terminal. Pour lancer **nano**, on utilise la commande suivante :

```
nano fichier_nano.txt
```

Nous sommes ensuite en mode d'édition de texte et il suffit de taper du texte et de se déplacer avec les flèches ou les commandes de raccourci. Par exemple, tapez le texte suivant :

```
Ceci est un fichier de texte
écrit dans 'nano'. Nous allons
également voir 'joe' dans un
instant.
```

Dans le bas du terminal, **nano** affiche certaines commandes utiles. On voit ainsi qu'on peut sortir de **nano** avec **^X**, c'est à dire **Ctrl-x**. Lorsqu'on sort, le programme nous propose de sauvegarder et ensuite nous demande de confirmer le nom du fichier.

Ceci complète le tour d'horizon de **nano**. Essayez de ré-ouvrir votre fichier (**fichier_nano.txt**), de le modifier puis de le sauvegarder. Utilisez la commande **cat** pour vous assurer que le fichier a bien été modifié. Lorsqu'on utilise **nano** et **joe**, il vaut la peine de laisser une ligne vide à la fin du fichier. De cette façon, la commande **cat** insérera une nouvelle ligne avant d'afficher le prompt.

2.2 - joe

Le programme **joe** est un autre éditeur de texte simple à utiliser. Il est un peu plus puissant que **nano**. Pour lancer **joe**, on utilise la commande suivante :

```
joe fichier_joe.txt
```

Le programme nous affiche alors :

```
New File
```

Nous sommes maintenant en mode d'édition de texte et il suffit de taper du texte et de se déplacer avec les flèches ou les commandes de raccourci si vous les apprenez. Par exemple, tapez le texte suivant :

```
J'écris maintenant dans un fichier
texte avec 'joe', qui est plus
puissant que 'nano' mais tout de
même facile à utiliser.
```

Dans le haut du terminal, **joe** nous indique qu'on peut avoir accès à de l'aide en tapant **Ctrl-k h**. Les commandes de **joe** débutent souvent par **Ctrl-k** suivi d'une autre touche. Affichons l'aide et nous voyons un plus grand nombre de commandes utiles. Cette liste de commandes utiles reste affichée pendant qu'on tape, ce qui en fait une référence pratique. On voit ainsi qu'on peut sortir de **joe** en avec la commande **^KX**, c'est à dire **Ctrl-k x**. Le programme sauvegarde alors automatiquement nos changements dans le fichier que nous avons ouvert.

Le programme **joe** n'est pas installé par défaut sur tous les systèmes UNIX mais il est facile à installer. Ceci complète le tour d'horizon de **joe**. Essayez de ré-ouvrir votre fichier (**fichier_nano.txt**), de le modifier puis de le sauvegarder.

3 - Écrire des scripts

Maintenant que nous savons comment éditer des fichiers texte avec **nano** et **joe**, nous pouvons écrire des scripts. Un script est un ensemble de commandes ou d'instructions écrites dans un langage spécifique qui peuvent être lues par un interpréteur pour accomplir des tâches. Par exemple, un script **bash** contient une suite de commandes qui seront exécutées par l'interpréteur **bash**.

Dans cette section, nous allons voir comment écrire et exécuter des scripts **bash** simples. Nous allons aussi voir comment nous pouvons ***installer*** nos scripts pour pouvoir les utiliser comme les autres commandes du système.

3.1 - Définir le problème

Nous allons écrire un script pour compter le nombre de séquences dans un fichier fasta. La première étape est de tester la commande qui nous permettra de compter les séquences. La commande suivante devrait faire l'affaire :

```
grep -c ">" sequences.fasta
```

3.2 - Écrire un script simple

Comme un script est une suite d'instructions dans un langage, la commande ci-haut devrait suffire comme point de départ pour notre script. Nous allons utiliser l'éditeur de texte **joe** pour taper notre commande dans un fichier texte nommé **count_sequences.sh**. L'extension **.sh** est habituellement utilisée pour dénoter les scripts **bash**.

```
joe count_sequences.sh
```

Il suffit maintenant de retaper la commande ci-haut dans le fichier. Nous ajoutons également des commentaires afin de mieux comprendre ce que fait le script. Les commentaires sont une part importante de n'importe quel script ou programme. Ils permettent aux gens qui les relierons de les comprendre beaucoup plus facilement.

Sagesse du jour:

```
Votre collaborateur principal est vous dans le futur
Et le vous d'il y a 6 mois ne répond pas au courriels.
```

3.3 - Lancer le script

Pour lancer notre script, il faut un moyen de dire au système qu'il contient des commandes dans le langage **bash**. En lançant la commande suivante, le systèmes sait qu'il doit lire le script avec l'interpréteur **bash**.

```
bash count_sequences.sh
```

Notre script fonctionne tel que prévu. Cependant, nous pouvons l'améliorer. Voici quelques idées pour le rendre plus utile :

- Pouvoir compter les séquences de n'importe quel fichier fasta.
- Le rendre exécutable (plus besoin d'écrire **bash** devant le nom du script).
- L'installer pour ne plus avoir besoin de dire où il se trouve.

3.4 - Traiter n'importe quel fichier fasta

Présentement, notre script fonctionne seulement avec le fichier fasta nommé **sequences.fasta**. Pour le rendre plus utile, nous allons permettre de spécifier quel fichier utiliser lorsque le script est lancé. Pour cela, corrigez votre fichier de script (**count_sequences.sh**) pour qu'il contienne les lignes suivantes :

```
# Count sequences in a fasta file

# Specify input file
FILE=$1

# Print name of input file
echo "--> $FILE:"

# Count number of sequences
grep -c ">" $FILE
```

Le truc est d'utiliser une variable, ici **FILE**, pour contenir le nom du fichier à traiter. La variable **$1** est une variable **bash** qui contient le premier argument fournit à notre script. On peut donc maintenant utiliser le script avec la commande suivante :

```
bash count_sequences.sh sequences.fasta
```

3.5 - Rendre le script exécutable

Nous souhaitons que notre script ressemble le plus possible aux autres commandes du système (comme **cat**, **grep**, etc.). Il faudrait donc pouvoir le lancer sans avoir à spécifier à chaque fois qu'il s'agit d'un script **bash**. Pour cela, nous allons ajouter une première ligne dans le fichier qui avertira le système que les lignes du script doivent être interprétées par **bash**. Il suffit d'ajouter la ligne suivant tout en haut de votre script :

```
#!/bin/bash
```

Cette ligne est un commentaire spécial. On peut voir qu'il s'agit d'un commentaire puisque la ligne débute par le caractère **#**. Cependant, la combinaison de **#** suivi du point d'exclamation **!** (qui se prononce **shebang**) dit au systèmes d'utiliser le programme **/bin/bash** (l'interpréteur **bash**) pour exécuter les lignes du fichier.

Pour pouvoir lancer notre script, il faut également rendre le fichier exécutable. Regardons les permissions des fichiers dans notre dossier avec la commande suivante.

```
ls -lh
```

On peut voir que les lignes débutent par un code qui ressembles à **-rw-r--r--**. Il s'agit en fait d'information sur les permissions des fichiers. Les permissions possibles sont *read* (**r**), *write* (**w**) et *execute* (**x**). Il y a également trois groupes d'utilisateurs qui peuvent chacun avoir des permissions différentes sur un fichier. Le groupes d'utilisateurs sont *owner* (le propriétaire du fichier), *group* (les utilisateurs faisant partie du même groupe d'utilisateurs que le propriétaire) et *other* (tous les autres utilisateurs). Les lettres des permissions sont donc regroupées par trois (**rwx**), avec un groupe de permissions pour chacun des groupes d'utilisateurs (*owner*, *group* et *other*). Les dossiers on une permission spécial **d** au début. Par exemple, l'ensemble de permission suivant (**-rw-r--r--**) veut dire que le propriétaire peut lire et écrire (ou effacer) le fichier, alors que les membres de son groupe d'utilisateurs, ainsi que tous les autres utilisateurs, peuvent seulement le lire. Voici un autre exemple (**-rwxr-xr-x**) où tout le monde peut lire et exécuter le fichier mais seulement le propriétaire peut y écrire ou l'effacer. Nous allons revoir rapidement les permissions et comment les modifier au prochain cours.

Pour rendre notre fichier exécutable, nous allons utiliser la commande **chmod**, qui veut dire *change mode*. Nous allons rendre le fichier exécutable pour tout le monde.

```
chmod +x count_sequences.sh

# Vérifions le changement
ls -lh
```

La lettre **x** apparaît maintenant pour les trois groupes d'utilisateurs et le nom du fichier est en vert pâle, la couleur réservée pour les fichiers exécutables. Il ne reste plus qu'à lancer le script. Pour cela, il faut dire au système où il se trouve. Comme il est dans le dossier courant, on peut utiliser le raccourci **./**.

```
./count_sequences.sh sequences.fasta
```

3.6 - PATH

Il nous reste un dernier truc à faire pour que notre script soit traité comme les autres programmes disponibles sur le système; Il faut que le système sache où le trouver sans qu'on ait à lui dire à chaque fois.

Le système utilise une variable spéciale pour savoir où chercher les programmes. Il s'agit de la variable **PATH**. Pour voir ce qu'il y a dans la variable **PATH**, il suffit de l'afficher avec la commande **echo** et de mettre le symbole **$** devant le nom de la variable :

```
echo $PATH
```

Cette variable contient en fait une série de chemin de dossiers séparés par des caractères deux-points **:**. Comme nous connaissons des commandes pour manipuler du texte, nous allons afficher chaque dossier sur une seule ligne pour améliorer la lisibilité.

```
echo $PATH | perl -pe 's/:/\n/g'
```

On voit que le système cherche ses programmes dans plusieurs dossiers, dont la plupart se terminent par **/bin**. Cette abréviation signifie ***binary file***, ou ***fichier binaire***, qui est un équivalent de ***programme***.

En tant qu'utilisateurs normaux sur le serveur (c'est à dire sans droits d'administrateur), nous ne pouvons pas écrire dans ces dossiers. Nous ne pouvons donc pas copier notre script, par exemple, dans **/usr/local/bin**. Par contre, nous pouvons nous créer un dossier **/home/user##/programmes** qui nous appartient et modifier la variable PATH pour que le système inclut ce dossier lorsqu'il recherche des programmes.

```
# Créer le dossier de programmes
mkdir ~/programmes

# Copier le script dans le dossier
cp count_sequences.sh ~/programmes

# Lancer la commande suivante SANS MODIFICATION
# pour ajouter ~/programmes à la variable PATH
echo -e "\n# Path\n"'export PATH=~/programmes:$PATH' >> ~/.profile

# Appliquer le changement
source ~/.profile

# Vérifier que la variable PATH est modifée
echo $PATH | perl -pe 's/:/\n/g'
```

Vous devriez voir que la première ligne contient **/home/user##/programmes**. Pour installer des scripts ou des programmes sur un système où vous n'avez pas les droits d'administrateur, il suffit donc de placer les fichiers exécutables dans un dossier qui est inclut dans la variable **PATH**. Nous allons revoir cette approche lorsque nous installerons des programmes au prochain cours.

Il reste à tester que l'installation a bien fonctionné en lançant le script.

```
count_sequences.sh sequences.fasta
```

Voilà, notre script fonctionne maintenant comme les autres programmes disponibles dur le système. C'est-à-dire, presque.

3.7 - Tester avec tous les fichiers fasta

Le fait d'avoir installé notre script le rend plus intéressant, mais nous ne pouvons pas l'utiliser directement sur tous les fichiers fasta d'un dossier en lançant **count_sequences.sh *.fasta**. Si on l'essaie, il nous retourne seulement le nombre de séquences du premier fichier qu'il trouve. C'est normal puisqu'on lui a dit spécifiquement d'utiliser seulement le premier argument avec la variable **$1**. Pour lancer notre script sur tous les fichiers fasta, il faudra faire une boucle dans le terminal, ce que nous verrons plus en détail la semaine prochaine mais voici un exemple.

```
for i in *.fasta; do count_sequences.sh $i; done
```

Si on souhaite ajouter des scripts **bash** ou autres (par exemple des scripts **perl** ou **python**), on utilise la même approche; On les met dans un dossier et on s'assure que ce dossier est inclut dans la variable **PATH**. Au fil du temps, on installe des programmes et on écrit des scripts qui nous sont utiles. Il devient alors plus rapide et facile de faire les analyses souhaitées.

4 - Fichiers de configuration

Lorsqu'on se connecte à un serveur Linux ou qu'on ouvre un terminal sur Linux ou MacOS, le système lit des fichiers de configuration. Sur notre serveur, il s'agit des fichiers **.bashrc** et **.profile**, tous deux trouvés dans votre dossier d'utilisateur (**/home/user##**). Ces fichiers définissent des options par défaut de certaines commandes, les dossiers où chercher des programmes ainsi que des aliases qui peuvent être utilisés dans le terminal. On peut éditer ces fichiers (**prudemment**) pour modifier les options, ajouter des dossiers de programmes et des aliases. Il est cependant important de faire une copie de sauvegarde **AVANT** de modifier ces fichiers puisque si nous faisons des erreurs, par exemple en effaçant le contenu de la variable PATH, il se pourrait que nous ne puissions plus lancer de commandes.

Nous allons donc commencer par nous déplacer dans notre dossier d'utilisateur et faire des copies de sauvegarde de ces deux fichiers.

```
# Move to your user home folder
cd

# Faire des copies des fichiers .bashrc et .profile
cp ~/.bashrc ~/.bashrc.backup
cp ~/.profile ~/.profile.backup
```

4.1 - Fichier ~/.profile

Nous allons maintenant regarder le contenu de ces deux fichiers en commençant par le fichier **.profile**.

```
joe ~/.profile
```

Ce fichier contient des lignes de commentaire au début, suivit de deux sections **if** qui servent à charger le fichier **.bashrc** et à ajouter automatiquement le dossier **/home/user##/bin** à la variable PATH si il existe. Noter que la personne qui a ajouté ces sections a mis des commentaires qui nous permettent de rapidement comprendre le code.

Si vous avez lancé les commandes de la section **3.6 - PATH** plus haut, vous avez maintenant une troisième section dans le fichier **.profile** qui ajoute le dossier **~/programmes** à la variable **PATH**. Si vous avez besoin d'ajouter d'autres dossiers à votre **PATH**, copiez la syntaxe de cette ligne **en faisant bien attention** car une erreur pourrait rendre la variable **PATH** inutilisable.

4.2 - Fichier ~/.bashrc

Le fichier **~/.bashrc** est chargé chaque fois que vous ouvrez un terminal **bash**. Sur le serveur, cela correspond la plupart du temps à votre connexion. Nous allons visualiser son contenu.

```
joe ~/.bashrc
```

Encore une fois, il y a différentes section avec des commentaires mais ce fichier est plus long. Les lignes 19 et 20 (dans **joe**, vous pouvez voir à quelle ligne vous vous trouvez en haut du terminal avec la section **Row ##**) contiennent deux variables qui nous intéressent. Elles permettent de spécifier combien le nombre de commandes à garder dans l'historique de commandes. Nous allons ajouter un zéro à chacune de ces lignes.

Nous allons ensuite nous rendre à la ligne 87 pour voir comment on peut définir des aliases dans le terminal. Un alias est comme une nouvelle commande qui sera automatiquement remplacée par **bash** lors de son exécution. On utilise surtout les aliases pour avoir des raccourcis de noms de commandes. Par example, les lignes 88 à 90 définissent trois aliases pour la commande **ls**. Ainsi, on peut taper simplement **l** pour avoir l'équivalent de **ls -CF --group-directories-first**. Pour lister les aliases existants, on peut utiliser la commande **alias** sans argument.

Nous allons nous déplacer à la fin du fichier **~/.bashrc** pour ajouter les lignes suivantes, qui nous permettrons d'utiliser **lh** à la place de **ls -lh** et **p** à la place de **pwd**.

```
# Aliases added by the user
alias lh='ls -lh'
alias p='pwd'
```

Vous pouvez maintenant sauvegarder les changements faits au fichier et sortir avec **Ctrl-k x**. Afin que ces nouveaux aliases soient en fonction, il faut que le fichier **~/.bashrc** soit chargé à nouveau, ce que vous pouvez faire soit en vous déconnectant du serveur et en vous reconnectant ou, encore mieux, avec la commande suivante :

```
source ~/.bashrc
```

Il reste à tester que les nouveaux aliases fonctionnent.

```
# Afficher le dossier courant
p

# Lister les fichiers
lh
```

5 - Screen

On travaille habituellement de façon interactive dans le terminal. Il faut l'ouvrir et lancer des commandes, puis attendre que les commandes se terminent avant de pouvoir faire autre chose. Cette façon de faire est la plus simple mais ne suffit pas toujours. Parfois, il faudrait pouvoir se déconnecter d'un serveur tout en laissant nos tâches continuer à tourner. Les programmes **screen** et **tmux** ont été créés justement pour répondre à ce besoin. Nous allons nous concentrer sur **screen** car il est installé par défaut sur la plupart des systèmes Linux mais **tmux**, un programme plus récent, joue le même rôle et gagne en popularité.

5.1 - Utilité

Lorsqu'on utilise un serveur de calcul, on se connecte grâce au terminal pour lancer nos commandes. Ces commandes permettent de préparer nos donnée et nos scripts et de lancer les analyses. Ces analyses durent parfois plusieurs heures ou même plusieurs jours. Il n'est pas pratique de devoir laisser le terminal ouvert durant tout ce temps, surtout si on travaille sur notre ordinateur portable personnel. Par exemple, il n'est pas pratique de devoir laisser son portable tourner toute la fin de semaine au laboratoire pour que le serveur puisse terminer nos analyses.

Il serait plus pratique de se connecter au serveur, de lancer nos analyses et de pouvoir nous déconnecter tout en laissant le serveur travailler jusqu'à ce qu'on puisse se reconnecter pour voir si les analyses sont terminées ou pour inspecter les résultats. C'est exactement ce que **screen** permet de faire.

5.2 - Fonctionnement

Pour lancer une session **screen**, il suffit de lancer la commande :

```
screen
```

Un message est affiché que vous pouvez faire disparaître en appuyant sur la touche **Enter**. Vous vous trouvez maintenant dans une nouvelle session **screen**. De la même façon que les commandes pour interagir avec **joe** débutent par `Ctrl-k`, les commandes pour interagir avec **screen** débutent par **Ctrl-a**. Par exemple, pour se détacher d'une instance **screen** sans la fermer, on utilises **Ctrl-a d**. Lorsque vous vous détachez, vous verrez un message similaire à celui-ci :

```
[detached from 7906.pts-0.raq]
```

La dernière partie de ce message vous donne le numéro de votre instance (avant le point) ainsi que son nom (après le point), ce qui vous servira à vous y rattacher. Nous allons premièrement lister les instances de **screen** que vous avez lancées :

```
screen -ls
```

Cette commande vous affiche les instances et leur statut (**Detached** ou **Attached**). Vous pouvez vous reconnecter à une instance avec le statut **Detached** avec :

```
screen -r <ID>
```

Où la partie **<ID>** est soit le numéro ou le nom de l'instance. Comme il est plus facile de se souvenir de noms que de numéros, je vous suggère de nommer vos instances quand vous les créez :

```
screen -S <nom>
```

Où vous remplacez **<nom>** par le nom que vous souhaitez donner à la session. Par example, lancez la commande suivante :

```
screen -S bioinfo
```

Maintenant, vous pouvez vous déconnecter (**Ctrl-a d**) et lister les instances (**screen -ls**). Vous allez voir que l'identifiant d'une des sessions ressemble à **8045.bioinfo**. Vous pouvez vous rattacher à une session en spécifiant soit son numéro (partie avant le point) ou son nom (partie après le point). Par exemple, essayez :

```
screen -r bioinfo

# Lister les instances
screen -ls
```

Vous voyez maintenant que la session **bioinfo** a le statut **Attached**. Pour sortir et fermer un session, il suffit de taper **exit** ou d'utiliser la commande **Ctrl-d**. Fermez votre session courante, puis rattachez vous aux autres sessions existantes s'il y en a et fermez les également.

5.3 - Lancer une commande et y revenir plus tard

Nous allons maintenant nous pratiquer à lancer une commande qui prend beaucoup de temps à s'exécuter puis à nous déconnecter du serveur pendant que la commande continue à s'exécuter. Pour cet exemple, nous allons utiliser la commande **sleep** pour simuler une commande qui prendrait beaucoup de temps avant de terminer.

Premièrement, créez une nouvelle instance de **screen** qui a pour nom **sleep**. Ensuite, tapez la commande **sleep 9999**, qui mettrait environ 2h45 avant de terminer. Maintenant, déconnectez vous de votre session **screen** avec **Ctrl-a d** et déconnectez vous du serveur. Nous allons maintenant nous reconnecter au serveur, lister les instances de **screen** qui sont

actives et nous reconnecter à la session **sleep**. Vous verrez alors que la commande sleep tourne toujours.

Normalement, vous vous déconnecteriez à nouveau pour revenir vérifier plus tard si la commande a terminée son exécution. Pour l'instant, vous pouvez arrêter votre commande avec **Ctrl-c** et terminer votre session **screen**.

Un des désavantages de **screen** est qu'on ne peut pas aussi facilement remonter pour voir plus haut dans le terminal. On peut cependant y arriver si on commencer par lancer la commande **Ctrl-a [**. On peut alors remonter en utilisant les flèches ou la roulette de la souris. Il faut faire **Ctrl-c** pour sortir de ce mode dit **scrollback**.

5.4 - Liste de commandes screen

Si vous êtes à l'aise avec l'utilisation de base de **screen**, voici une liste de commandes qui inclut quelques commandes plus avancée.

- Se détacher d'une instance : **Ctrl-a d**
- Terminer une instance : **exit** ou **Ctrl-d**
- Créer de nouvelles fenêtres sur la même instance : **Ctrl-a c**
- Aller à la fenêtre suivante : **Ctrl-a n**
- Aller à la fenêtre précédente : **Ctrl-a p**
- Aller à la fenêtre précédemment affichée : **Ctrl-a Ctrl-a**
- Aller à la fenêtre numéro 'n' (0-9) : **Ctrl-a 'n'**
- Renommer la fenêtre courante : **Ctrl-a A**
- Afficher les fenêtres ouvertes : **Ctrl-a "**
- Terminer la fenêtre courante : **Ctrl-a k**
- Entrer en mode **scrollback** : **Ctrl-a [**
- Bloquer votre terminal (demandera votre mot de passe) : **Ctrl-a x**

6 - Mot de la fin

6.1 - Aujourd'hui, nous avons vu :

- Éditer fichiers textes
- Écrire, installer et lancer scripts
- La variable `PATH`
- Fichiers de configuration `~/.bashrc` et `~/.profile`
- Utiliser screen

Nous avons à présent vu la grande majorité des commandes et options qui sont utilisées fréquemment sous Linux. Avec ces outils, nous sommes capables de préparer et lancer des analyses simples. Il nous reste cependant à voir comment transférer nos données sur le serveur.

6.2 - Au prochain cours, nous verrons :

- Transfert de données
- Télécharger de fichiers à partir du terminal
- Boucles for et autres trucs pour `bash`
- Installation de programmes
- Analyse : blaster des séquences

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 - Éditeurs de texte

- Créer et éditer un nouveau fichier nommé **haiku.txt** avec **nano**.
- Taper le poème japonais suivant dans le fichier :

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

- Sauvegarder le fichier.
- L'afficher avec la commande **cat**.
- Réouvrir le fichier avec **joe**.
- Ajouter l'auteur du poème :

```
- Yosa Buson (1716 - 1783)
```

- Sauvegarder et afficher avec la commande **cat**

7.2 - Scripts

- Écrire un script **bash** nommé **print_haiku.sh** qui prend deux arguments (un nom de personne et un fichier contenant un haiku) et qui imprime **Bonjour <NOM>, voici un haiku:**, suivi du poème. Voici un exemple de son utilisation :

```
./print_haiku.sh Eric haiku.txt
```

- Installer le script dans votre dossier **~/programmes** et lancez le sans avoir à spécifier le chemin du script (vous devrez quand même spécifier le chemin vers le fichier texte).

7.3 - Fichiers de configuration

- Ajouter un alias **haiku** à la fin de votre fichier **~/.bashrc** qui utilise votre nouveau script pour imprimer votre nom d'utilisateur (contenu dans la variable **$USER**) et votre fichier de poème **haiku**.
- Utiliser la commande **source** pour que le nouvel alias soit disponible.
- Taper : **haiku** pour valider que l'alias fonctionne.

7.4 - Screen

- Créer une nouvelle instance de **`screen`** nommée **`cours04`**.
- S'en déconnecter avec **`Ctrl-a d`**.
- Créer une deuxième instance de **`screen`** nommée **`script`**.
- S'en déconnecter avec **`Ctrl-a d`**.
- Lister les instances existantes.
- Se reconnecter à l'instance **`cours04`** et l'arrêter.
- Se déconnecter du serveur et s'y reconnecter.
- Se reconnecter à l'instance **`script`** et l'arrêter.

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 - Éditeurs de texte

- **`nano`** : Éditeur de texte.
- **`joe`** : Éditeur de texte.

8.2 - Scripts

- **`bash`** : Interpréteur du langage **`bash`**.
- **`chmod`** : Changer les permissions d'un fichier (change mode) **`[+x]`**.
- **`#!/bin/bash`** : Indiquer quel interpréteur utiliser pour exécuter le fichier courant (shebang).
- Variable **`PATH`** : Contient la liste des dossiers où le système cherche des programmes.

8.3 - Fichiers de configuration

- **`alias`** : Commande pour définir des alias.
- **`source`** : Recharger un fichier de configuration.

8.4 - Screen

(Copiées de la **`section 5.4`**) :

- Se détacher d'une instance : **`Ctrl-a d`**
- Terminer une instance : **`exit`** ou **`Ctrl-d`**
- Créer de nouvelles fenêtres sur la même instance : **`Ctrl-a c`**
- Aller à la fenêtre suivante : **`Ctrl-a n`**
- Aller à la fenêtre précédente : **`Ctrl-a p`**
- Aller à la fenêtre précédemment affichée : **`Ctrl-a Ctrl-a`**
- Aller à la fenêtre numéro 'n' (0-9) : **`Ctrl-a 'n'`**
- Renommer la fenêtre courante : **`Ctrl-a A`**
- Afficher les fenêtres ouvertes : **`Ctrl-a "`**
- Terminer la fenêtre courante : **`Ctrl-a k`**
- Entrer en mode **`scrollback`** : **`Ctrl-a [`**
- Bloquer votre terminal (demandera votre mot de passe) : **`Ctrl-a x`**