

ISS - Initiation aux Systèmes d'exploitation et au Shell

LU2IN020

TP 04 – Combinaison de commandes avec les tubes

Julien Sopena

février 2022

Le but de cette quatrième semaine est d’approfondir l’étude des arborescences de processus, d’introduire la notion de variables d’environnement. Nous étudierons aussi les différents types de redirections, ainsi que la substitution de commandes.

Exercice 1 : What does the Tux say ?

Question 1

Pour commencer, vous allez récupérer les données nécessaires à l’ensemble des exercices. Placez-vous dans le répertoire correspondant à ce TP pour y extraire l’archive après l’avoir chargée à l’URL suivante : http://julien.sopena.fr/LU2IN020-TP_04.tgz

Question 2

Comme beaucoup de projets aujourd’hui, le code qui nous intéresse dans cet exercice, et son historique de développement, sont disponibles sur un serveur *git*. Si nous n’étudierons pas dans cette UE son fonctionnement, il est très facile de l’utiliser pour récupérer les sources désirées. Il suffit en effet de *cloner* le dépôt avec la commande suivante :

```
git clone https://github.com/schacon/cowsay.git
```

Attention, ici *git* va utiliser le protocole https pour récupérer les données. Or, à la *PPTI* les requêtes *http* et *https* doivent passer par un proxy. Le processus exécutant *git* devra donc avoir la variable `$https_proxy` dans son environnement avec pour valeur : `proxy.ufr-info-p6.jussieu.fr:3128`.

Question 3

Maintenant que vous avez récupéré les sources, entrez dans le répertoire *cowsay* et lancez le script *install.sh* en lui donnant en paramètre un répertoire d’installation. Classiquement, un utilisateur installe ses propres logiciels dans un répertoire `~/usr`.

Une fois l’installation terminée, vous pouvez effacer le répertoire *cowsay* que vous avez cloné.

Question 4

Si tout c'est bien passé, vous devriez avoir l'exécutable `cowsay` dans le répertoire `~/usr/bin/`. Faites en sorte de pouvoir lancer cet exécutable depuis n'importe quel endroit dans n'importe quel terminal.

Question 5

Dans le répertoire `exo1` vous trouverez un fichier `csv` (Comma-Separated Values) qui contient une liste d'animaux avec le nom de leurs cris en français, puis en anglais.

Écrivez un script `jungle.sh` qui après avoir demandé à l'utilisateur de choisir entre VF et VO, lancera pour toutes les lignes du fichier la commande `cowsay` avec l'option `-f` suivie du nom de l'animal (1^{re} valeur), ainsi qu'un son en paramètre (2^e valeur pour la VF ou 3^e pour la VO).

Une petite aide : lorsque l'on parse un fichier et que l'on veut utiliser plusieurs valeurs se trouvant sur une même ligne, il est inefficace d'utiliser des compositions de commandes. En effet, on devrait alors refaire le parsing pour chaque valeur. Le plus simple est d'utiliser la commande `read` et la variable d'environnement `$IFS` pour fixer le séparateur.

Exercice 2 : Sur vos traces

Dans cet exercice, nous allons étudier les informations contenues dans le fichier `/var/log/wtmp` qui enregistre toutes les connexions sur une machine. N'étant pas textuel, on ne peut pas lire directement son contenu avec un `cat`. Cependant, ses données sont rendues accessibles avec la commande `last`.

Question 1

Pour commencer, créez un fichier `connexions.log` contenant la sortie de la commande `last` avec l'option `-da`

Question 2

Voulez allez maintenant commencer à implémenter un script `traces.sh` que vous compléterez au fur et à mesure des questions. Ce script prendra en paramètre un nom de fichier contenant les logs à analyser.

Dans un premier temps, vérifiez que le script reçoit bien un paramètre et que ce paramètre est un fichier accessible en lecture.

La bonne pratique est d'afficher pour chaque type d'erreur un message spécifique, suivi de l'usage général. Le script retourne alors une valeur différente de 0.

Ces deux dernières actions étant communes à chaque erreur, il est pratique d'utiliser les fonctions du *Bash* pour éviter tout copier/coller (signe de ...). Les fonctions fonctionnent comme un script, à la différence qu'elles sont définies dans le fichier. Dans une fonction, on accède aux paramètres comme dans un script, avec les variables `$1, $2, ..., $n`. Attention, `$0` représente toujours le nom du script (et non de la fonction) qui s'exécute.

Créer une fonction `usage` est un très bon exemple de l'utilisation de fonction *Bash*.

```
function maFonction() {
    # Du code qui peut utiliser $1, $2, ...
    # et toutes les autres variables du script
}
```

```
function usage() {
    echo "Usage : $0 <param1> <param2>"
    exit 1
}
```

Question 3

Pour commencer le traitement de la trace, afficher le nombre d'entrées qu'elle contient. Attention aux dernières lignes qui ne constituent pas des entrées.

Question 4

Ajoutez maintenant les commandes nécessaires pour afficher un message indiquant si vous êtes ou non le dernier à vous être connecté à la machine. Vous pouvez utiliser la commande `whoami` pour récupérer votre nom de login.

Question 5

Comptez toutes vos connexions dans la trace et affichez cette information.

Question 6

Ajoutez un affichage du nombre d'utilisateurs qui se sont connectés au moins une fois sur la machine.

Question 7

Après les utilisateurs, on s'intéresse aux machines utilisées pour se connecter. Compter le nombre de machines distantes présentes dans la trace. Les noms de machine sont placés à la fin de l'affichage (option `-a` de la commande `last`) et la machine locale est référencée par les noms `0.0.0.0` ou `localhost`.

Une petite aide : avec la trace que vous utilisez, il sera difficile d'utiliser les séparateurs pour accéder au nom des machines. Le plus simple est de raisonner en fonction du nombre de caractères. Vous pouvez regarder par exemple le `man` de la commande `cut`.

Question 8

Affichez, pour chaque utilisateur s'étant connecté au moins une fois à distance, la liste des machines depuis lesquels se sont faites ces connexions.

Question 9

Pourquoi ne voit-on que des adresses de la `ppti` ?

Question 10

La commande `ssh` permet de se connecter à une machine distante au travers d'un protocole chiffré. Une fois connecté vous obtenez un shell, comme si vous étiez connecté en local. Plusieurs personnes peuvent être connectées en même temps.

Connectez vous sur la passerelle `ssh.ufr-info-p6.jussieu.fr` et regardez qui y est connecté avec la commande `finger`.

Question 11

Maintenant que vous êtes connecté, générez une nouvelle trace. Attention cela peut prendre du temps, car l'option `-d` fait une résolution `dns` inversée sur chaque ip pour essayer de trouver le nom de la machine. Vous pouvez désactiver cette option pour gagner du temps. Une fois la trace générée, déconnectez-vous avec la commande `exit`.

Question 12

Relancez votre script sur la nouvelle trace. Outre la composition de commandes, vous pouvez retenir de cet exercice qu'on laisse partout des traces de nos activités numériques, traces qu'il est facile d'exploiter avec quelques lignes de *Bash*...

Exercice 3 : Généalogie

Question 1

Dans cet exercice on s'intéresse aux liens de filiation entre processus. Pour commencer, exécutez la commande `ps -eF` pour avoir une vue d'ensemble sur les processus tournant sur votre ordinateur.

Question 2

Nous voulons travailler ici sur ces données. Malheureusement, elles sont structurées pour être facilement lues par l'homme ce qui les rend difficilement interprétables par un script. Pour ce type de traitement, on préfère l'utilisation de séparateur, plutôt que de jolis alignements.

Heureusement, la commande `ps` est très personnalisable. Peut-être même trop ... Pour vous en rendre compte, comptez le nombre de mots du `man` de cette commande et comparez cette longueur avec celui de la commande `mkdir`.

Question 3

Lorsque l'on veut automatiser un traitement de données, il vaut souvent mieux partir de données brutes, plutôt que d'utiliser le rendu d'une commande. Cela n'était pas possible avec le fichier `/var/log/wtmp`, d'où l'utilisation de la commande `last`. Mais pour les processus le système offre un grand nombre de données brutes textuelles au travers du `procfs`.

Ainsi, on trouve pratiquement toutes les informations disponibles sur l'état du système dans le répertoire `/proc`. Celles relatives aux processus se trouvent dans des répertoires dont le nom correspond à chacun des `pid`.

Dans cet exercice on s'intéresse au fichier `stat` et notamment au 4^e champ qui indique le `ppid`. Commencez par ouvrir un *Bash* et utilisez ces informations pour trouver le nom (2^e champ) du père de ce processus. Vous devriez pouvoir vérifier que le processus *Bash* chargé de l'interprétation des commandes est le fils d'un processus correspondant à la partie graphique du terminal.

Question 4

Rendez-vous maintenant dans le répertoire `exo3` et lancer l'exécutable `famille`.

Question 5

Pour mener à bien le travail demandé par l'exécutable, réalisez un script `genealogie.sh` qui :

- vérifie qu'il a bien reçu un paramètre ;
- vérifie que ce paramètre correspond au pid d'un processus existant ;
- utilise le `procfs` pour remonter l'arbre généalogique du processus dont le pid a été passé en paramètre jusqu'à arriver au processus 1, en affichant les pid trouvés.

Question 6

Vérifiez votre résultat en lançant la commande `pstree -cp` avec comme paramètre l'avant-dernier processus trouvé, *i.e.*, le fils du processus 1. Notez que cette dernière filiation peut vous paraître étrange, mais qu'elle vous sera expliquée en cours et en TD dans quelques semaines.