TP d'Administration

TP1 : Commandes BASH

Benoît Da Mota, André Rossi, Vincent Vigneron

Janvier 2018

1 Navigation dans l'arborescence

Sous Linux comme avec la majorité des systèmes d'exploitation, l'utilisateur accède à ses données grâce à un système de gestion de fichiers sous forme arborescente. Le disque dur est partitionné en partitions, et dans chaque partition se trouve une arborescence de répertoires dans lesquels se trouvent les fichiers. Il faut donc tout d'abord apprendre à naviguer dans l'arborescence. Dans notre terminal nous avons un interpréteur de programmes (nous utiliserons l'interpréteur BASH) qui interprète un grand nombre de commandes et qui peut exécuter tous les programmes présents sur le système. BASH est installé par défaut sur la plupart des distributions de Linux, mais nous verrons plus tard comment s'en assurer. Une commande est une instruction directement interprétable par BASH, un peu comme un mot clé d'un langage de programmation, un programme comme son nom l'indique est un programme installé sur le système et peut donc être absent si l'utilisateur a décidé de s'en passer. La première commande à connaître est la commande d'aide : `help`. En tapant `help` dans un terminal nous obtenons la liste des commandes internes de l'interpréteur BASH :

```
etudiant@25B:~$ help
...
 job_spec [&]                    history [-c] [-d décalage] [n] ou history -a>
 (( expression ))                if COMMANDES; then COMMANDES; [ elif COMMAND>
 . nom_fichier [arguments]       jobs [-lnprs] [jobspec ...] ou jobs -x comma>
 :                               kill [-s sigspec | -n signum | -sigspec] pid>
 [ arg... ]                      let arg [arg ...]
...
```

Pour avoir l'aide sur une commande en particulier, il suffit de taper `help` puis le nom de la commande. Dans la liste de commande, `cd` et `pwd` vont nous être utiles rapidement. Nous allons aussi utiliser les programmes `man`, `which`, `who`, `df`, `du`, `whoami`, `ls`, `ln`, `touch`, `mkdir`, `rmdir`, `chmod`, `cp`, `mv`, `rm`, `sort`, `uniq`, `wc`. L'aide sur un programme ne s'obtient pas avec la commande `help` mais grâce à un autre programme : `man`. Ainsi, taper : `man sort` affiche l'aide du programme `sort`. De plus, un grand nombre de ces commandes et programmes peuvent rediriger leur flux de sortie (ou changer leur flux d'entrée) grâce aux instructions de redirection : `<` et `>`. La commande pipe `|` peut également être utilisée.

Question 1.1 À partir de votre home directory (`/home/etudiant`), créez un répertoire `LINUX`.

```
mkdir LINUX
```

Question 1.2 Entrez dans ce répertoire.

```
cd LINUX
```

Question 1.3 Accédez au dossier parent, puis revenez dans `LINUX`.

```
cd ..
cd LINUX
```

Question 1.4 Comment connaître le chemin absolu du répertoire dans lequel vous êtes actuellement ?

```
pwd
```

Question 1.5 Créez un fichier `fichier1` contenant le nom des fichiers et répertoires de la racine.

```
ls / > fichier1
```

Question 1.6 Créez un fichier caché vide `fichier2`. (Utilisez le caractère spécial représentant votre home.)

```
touch ~/.fichier2
```

Question 1.7 Rajoutez dans `fichier2` le nom des fichiers et répertoires de votre home et de la racine.

```
ls / >>.fichier2
ls ~ >>.fichier2
```

Question 1.8 Comment revenir à votre home directory depuis n'importe lequel des répertoires de l'arborescence de fichiers ?

```
cd
```

Question 1.9 Affichez le contenu du répertoire courant au format long.

```
ls -l
```

Question 1.10 Affichez également les fichiers cachés.

```
ls -a
```

Question 1.11 Affichez tous les fichiers (cachés ou non) au format long.

```
ls -la
```

Question 1.12 Créez un sous-répertoire nommé `TP1` dans le dossier courant.

```
mkdir TP1
```

Question 1.13 Copiez `fichier1` vers le sous-répertoire `TP1` en l'appelant `fichier1_copie`.

```
cp fichier1 TP1/fichier1_copie
```

Question 1.14 Déplacez `fichier2` dans le répertoire `LINUX/TP1`.

```
mv .fichier2 ~/LINUX/TP1
```

Question 1.15 Rendez `fichier2` non caché.

```
mv ~/LINUX/TP1/.fichier2 ~/LINUX/TP1/fichier2
```

Question 1.16 Créez un répertoire `UNIX` situé au même niveau dans l'arborescence de fichiers que `LINUX`.

```
mkdir ~/UNIX
```

Question 1.17 Copiez tous les fichiers et sous-répertoires de `LINUX` dans `UNIX`.

```
cp -r ~/LINUX/* ../UNIX/
```

Question 1.18 Créez dans `LINUX` un lien physique `fichier1_ref_phys` et un lien symbolique `fichier1_ref_symb` de `UNIX/fichier1`.

```
ln ../UNIX/fichier1 fichier1_ref_phys
ln -s ../UNIX/fichier1 fichier1_ref_symb
```

Question 1.19 Affichez l'`inode` de `fichier1_ref_phys`, `fichier1_ref_symb` et de `UNIX/fichier1`. Que constate-t-on ?

```
ls -i ../UNIX/fichier1 fichier1_ref_phys fichier1_ref_symb
echo "On voit que ../UNIX/fichier1 et fichier1_ref_phys ont le même inode"
```

Question 1.20 Supprimez `fichier2`.

```
rm .fichier2
```

Question 1.21 Supprimez le répertoire `UNIX` et ses sous-répertoires.

```
rm -r ~/UNIX
```

Question 1.22 Quelles sont les conséquences pour les fichiers `fichier1_ref_phys` et `fichier1_ref_symb` du répertoire `LINUX` ?

```
echo "fichier1_ref_symb est désormais un lien mort, mais fichier1_ref_phys n'
   est pas affecté."
```

Question 1.23 Affichez l'`inode` puis le contenu de `fichier1_ref_phys` et `fichier1_ref_symb`. Que constate-t-on ?

```
ls -i fichier1_ref_phys fichier1_ref_symb
echo "inodes inchangés"
cat fichier1_ref_phys
echo "contenu inchangé"
cat fichier1_ref_symb
echo "message d'erreur : lien mort"
```

Question 1.24 Affichez le chemin d'un programme accessible depuis n'importe quel répertoire. Par exemple, vous pourrez chercher où se situe le programme `grep`.

```
which grep
```

Question 1.25 Affichez le login avec lequel vous êtes connecté.

```
echo "$USER"
whoami
```

Question 1.26 Affichez l'aide du programme `grep`.

```
man grep
```

Question 1.27 Affichez la liste des utilisateurs actuellement connectés.

```
who -a
```

Question 1.28 Affichez la taille occupée par tous les fichiers du répertoire courant (cela inclut la taille de tous les sous-répertoires) sous forme lisible (en méga-octets par exemple).

```
du -h
```

Question 1.29 Affichez la taille et l'espace libre de tous les disques sous forme lisible (en giga-octets par exemple).

```
df -h
```

Question 1.30 Rendre un fichier modifiable par n'importe quel utilisateur.

```
chmod a+w fichier1
```

Question 1.31 Écrire de manière symbolique (`rwxrwxrwx`), les permissions suivantes : 644, 755, 000, 711, 700, 777, 555, 111, 600, 731.

```
echo "644 = rw-r--r--"
echo "755 = rwxr-xr-x"
echo "000 = ---------"
echo "711 = rwx--x--x"
echo "700 = rwx------"
echo "777 = rwxrwxrwx"
echo "555 = r-xr-xr-x"
echo "111 = --x--x--x"
echo "600 = rw-------"
echo "731 = rwx-wx--x"
```

Question 1.32 Enlever les permissions d'exécution sur le répertoire `LINUX`. Que constate-t-on ?

```
cd
chmod a-x LINUX
echo "Plus la permission de faire cd LINUX"
```

2 Analyse d'un fichier (grep)

L'analyse d'un fichier est une opération courante : il s'agit d'extraire du fichier les parties qui nous intéressent. Les programmes `head` et `tail` permettent respectivement de récupérer les lignes du début (ou de la fin) du fichier. Par exemple, `head -2 fichier` renvoie les deux premières lignes de `fichier`.

Le programme `grep` est un outil plus avancé qui permet d'extraire une ou plusieurs lignes (voire même des portions de lignes) d'un ou plusieurs fichiers en fonction d'expressions régulières renseignées par l'utilisateur. C'est un

outil puissant qu'il faut connaître. Une utilisation courante de `grep` est l'analyse de logs. Pour les exercices suivants, nous nous placerons dans le répertoire `/var/log` contenant les fichiers de log du système et nous analyserons le contenu de ces fichiers avec les programmes `grep`, `head` et `tail`.

Le programme `grep` est souvent utilisé avec des expressions régulières. Une expression régulière est une écriture compacte pour représenter une séquence de caractères. Ce sont des « patrons » permettant de retrouver des motifs présents dans des chaînes de caractères. Elles peuvent être combinées pour former des expressions régulières complexes. Voici quelques exemples d'expressions régulières simples :

- `.` représente n'importe quel caractère
- `*` l'élément précédent peut être rencontré zéro ou plusieurs fois
- `|` représente un choix entre deux expressions
- `( )` permet de grouper une expression complexe pour la rentre atomique et ainsi la combiner avec une autre (un `*` ou un `|` par exemple)
- `[ ]` représente une liste de caractères au choix, par exemple `[0-9]` représente un caractère numérique quelconque.

Par exemple, l'expression régulière `[a-zA-Z]([a-zA-Z0-9]\|_\)*` représente toutes les chaînes de caractères commençant par une lettre (minuscule ou majuscule) puis composées de lettres, de chiffres ou du symbole « _ ». La chaîne « `cons_0truct` » est conforme à cette expression régulière, mais pas la chaîne « `est-il` ».

La commande `dmesg` affiche un fichier de log rempli à chaque démarrage de la machine. Il contient des informations intéressantes que nous allons manipuler.

Question 2.1 Affichez les dernières lignes (les 20 dernières) du fichier `dmesg`.

```
dmesg | tail -20
```

Question 2.2 Extraire les lignes du fichier retourné par `dmesg` contenant le mot « linux » quelle que soit la casse du mot. Affichez les numéros de lignes pour chaque occurrence trouvée. Affichez ensuite le nombre de lignes contenant au moins une occurrence de « linux ».

```
dmesg | grep -i 'linux'
dmesg | grep -i -n 'linux'
dmesg | grep -i -c 'linux'
```

Question 2.3 Affichez toutes les adresses IP contenues dans tous les fichiers log pour lesquels vous avez les droits. Affichez le nombre d'adresses ip trouvées. Attention, certains caractères spéciaux comme les accolades ou les parenthèses doivent être protégés par un anti-slash.

```
grep -sho '\([0-9]\{1,3\}\.\)\{3\}[0-9]\{1,3\}' /var/log/* | wc -l
echo "-s means silent (don't report errors like access denied)"
echo "-h suppresses the prefixing of file names on output"
echo "-o prints only the matched (non-empty) parts of a matching line, with
   each such part on a separate output line."
```

Question 2.4 Affichez-les maintenant de manière ordonnée en éliminant les doublons.

```
grep -sho '\([0-9]\{1,3\}\.\)\{3\}[0-9]\{1,3\}' /var/log/* | sort | uniq
```

3 Compression de fichiers et de répertoires (tar)

Le programme `tar` permet de compresser des fichiers et des répertoires en une archive. C'est un programme très complet avec lequel il est possible de compresser vers plusieurs formats, de tester les archives etc. Par convention, on ajoutera l'extension « `.tar` » à toute archive non compressée réalisée avec `tar`. Si l'archive est compressée au format `gzip`, on ajoutera « `.gz` », on obtiendra donc « `.tar.gz` » (l'extension `.tgz` est synonyme et souvent employée). Si l'archive est compressée au format `bzip2`, on utilisera l'extension « `.tar.bz2` ». Ces extensions doivent être données au moment où l'on appelle le programme `tar` en les ajoutant à la fin du nom du fichier archive.

Question 3.1 Créez une archive comprenant tous les fichiers se terminant par « `.log` » présents dans le répertoire `/var/log` et pour lesquels vous avez les droits. On créera une archive compressée au format `gzip`.

```
tar zcvf ~/archive.tar.gz $(ls /var/log/*.log)
```

Question 3.2 Vérifiez le contenu de cette archive sans l'extraire.

```
tar tf archive.tar.gz
```

Question 3.3 Quelle est la taille de cette archive (en kilo-octets) ?

```
du archive.tar.gz
```

Question 3.4 Changer le mode de compression de l'archive pour la compresser au format `bzip2`. Attention, pour changer le mode de compression il ne faut pas extraire les fichiers ! Il faut décompresser l'archive au format `gzip` en utilisant le programme du même nom (`gzip` et `gunzip`) puis la recompresser avec le programme `bzip2`.

```
gunzip archive.tar.gz
bzip2 archive.tar
```

Question 3.5 Extraire les fichiers avec `tar` directement dans votre répertoire « home ».

```
tar xvf archive.tar.bz2 -C ~
```

4 Recherche de fichiers (find)

Le programme `find` permet de rechercher des fichiers dans une sous-arborescence de la partition. Il ne se base que sur le nom ou les attributs du fichier, il est donc complémentaire de `grep`. La chaîne de caractères fournie à `find` peut aussi, comme pour `grep` être une expression régulière. Il est également possible d'appliquer un programme spécifique à chaque nom de fichier résultant de la recherche.

Question 4.1 Cherchez toutes les images (`.jpg`) présentes sur votre machine.

```
find / -type f -name "*.jpg"
```

Question 4.2 Cherchez toutes les images (`.jpg` et `.png`) dont la taille est supérieure à 10 kilo-octets.

```
find / -type f \( -name "*.jpg" -o -name "*.png" \) -size +10k
```

Question 4.3 Relancer la même recherche en excluant les messages d'erreur avec une redirection de la sortie standard d'erreur vers `/dev/null`.

```
find / -type f \( -name "*.jpg" -o -name "*.png" \) -size +10k 2>/dev/null
```

Question 4.4 Cherchez toutes les images (`.jpg` et `.png`) dont la taille est supérieure à 10 kilo-octets et dont le propriétaire n'est pas « `root` ».

```
find / -type f \( -name "*.jpg" -o -name "*.png" \) -size +10k -! -user "root"
   2>/dev/null
```

Question 4.5 Cherchez toutes les images (`.jpg` et `.png`) dont la taille est supérieure à 10 kilo-octets en excluant le répertoire « `/usr/share/icons` ».

```
find / -path /usr/share/icons -prune -o -type f \( -name "*.jpg" -o -name "*.
   png" \) -size +10k 2>/dev/null
```

Question 4.6 Copier toutes ces images dans un répertoire (que vous aurez créé au préalable).

```
mkdir ~/IMG
find / -path /usr/share/icons -prune -o -type f \( -name "*.jpg" -o -name "*.
   png" \) -size +10k -exec cp {} ~/IMG \; 2>/dev/null
```