Pratique des machines
TP2 : Système de fichier (suite), commandes utiles

am@up8.edu

Septembre 2022

Dans ce TP :
— explication des dossiers contenus à la racine
— `wget, curl, entrées, sorties, pipe`

Le système de fichiers arborescent

Linux a un système de fichiers arborescent, càd. comme un arbre généalogique : un dossier peut contenir d'autres dossiers, etc. La racine de l'arbre est / (ou : pour MacOs). C'est le seul dossier qui n'est pas contenu dans un dossier.

Dossiers contenus à la racine :
— /**bin** (binaries) programmes exécutables essentiels à tous les utilisateurs
— /**boot** instructions de démarrage (non modifiables)
— /**dev** (devices) appareils et périphériques : "Tout est un fichier" (disque SDA et ses partitions,...)
— /**etc** (et ceatera) configurations système spécifiques à la machine
— /**home** fichiers personnels des utilisateurs, configurations personnelles des applications (ex. ∼/.mozilla)
— /**lib** (libraries) bibliothèques partagées essentielles et modules du noyau
— /**media** ou **mnt** (mount) appareils mobiles notamment (clé USB,...)
— /**opt** (optional) autre emplacement de certains logiciels 6/14TD1
— /**proc** (processes) ressources (ex. CPU) et processus du système (identifiés par leur PID) : fichiers virtuels
— /**root** home de l'administrateur
— /**run** informations sur la RAM des processus en exécution
— /**sbin** (system binaries) programmes du système
— /**snap** paquets spéciaux d'Ubuntu
— /**srv** (server) dossiers liés à l'utilisation de la machine en tant que serveur (vide de base)
— /**sys** (system) manière d'interagir avec le noyau (kernel), état des périphériques et des sous-systèmes
— /**tmp** (temporary) fichiers temporaires d'applications
— /**usr** (user) applications installées par les utilisateurs,
  — contient de même **bin**, **lib**,...
  — les applications qui s'installent pour quelques utilisateurs seulement sont dans local
  — les applications installées pour tous se mettent dans share

— `/var` (variable) fichiers qui grandissent dans le temps (logs,...)

Manier le terminal

Quelques astuces et raccourcis clavier pour jongler avec le terminal :
— `Ctrl + Maj + t` : ouvrir un nouvel onglet dans le terminal
— `Ctrl + D` : arrêter le processus en cours ou fermer le terminal (ex : lancez `cat` et arrêtez le processus)
— flèches du haut / bas : se déplacer dans l'historique des commandes lancées
— `Ctrl + a` : aller au début de la ligne
— `Ctrl + e` : aller à la fin de la ligne
— `Ctrl + k` : couper jusqu'à la fin de la ligne
— clic droit "copier" : copier la région en surbrillance
— clic droit "coller" ou clic molette ou `Ctrl + y` (seulement au sein du terminal) : coller
— `Ctrl + c` : arrêter la commande en cours
— `clear` ou `Ctrl + l` : effacer les commandes précédentes de l'interface graphique
— `reset` : réinitialiser l'environnement du shell

1. testez ces commandes
2. la commande `Ctrl + z` suspend le processus de premier plan en cours. Exécutez la séquence suivante :

```
> mousepad (ou gedit)
Ctrl-z
> fg
Ctrl-z
> bg
```

Que se passe-t-il ? À quoi se substitue la séquence `"1) Ctrl-Z 2) >bg"` ?

> La commande se substitue à la suffixation par le caratère '&' d'une commande, qui la lance en background.

Exercice 1 : Téléchargement et archives

Pour télécharger un fichier, utilisez wget.

1. Lancez :

```
$ wget https://alicemillour.github.io/assets/cours/PdM/
   PdM_2223_TP1.zip
```

2. utilisez `unzip` pour décompresser l'archive.
   Il existe aussi la commande `curl` pour télécharger du contenu. `wget` a l'avantage de pouvoir télécharger un dossier récursivement. `curl` a l'avantage d'être compatible avec beaucoup plus de protocoles (ex. SMTP, IMAP,...) que juste HTTP, HTTPS et FTP. `curl` peut aussi envoyer des requêtes, contrairement à `wget`.

3. Recherchez dans la documentation comment utiliser `curl` pour télécharger seulement l'**en-tête** (head ou header) d'un site, et faite-le sur `https://www.univ-paris8.fr/`.

```
$ curl -I https://www.univ-paris8.fr/
```

Exercice 2 : Séquences de commandes, entrées et sorties

Le point-virgule permet d'effectuer deux commandes, l'une à la suite de l'autre.

```
1 $ ls ; mv ..
```

On peut bien sûr enchaîner les point-virgules

```
1 $ touch doc ; echo "Informations:" > doc ; mv doc ..
```

Que fait la commande précédente ?

1. crée un document du nom de `doc`
2. écrit "Informations :" dans `doc`
3. déplace `doc` dans le dossier parent (au-dessus) du dossier courant

Exercice 3 : Sortie et écriture

On peut rediriger la sortie d'une commande avec le chevron droite >.

1. Créez un fichier fichier et écrivez-y le résultat de ls avec

```
1 $ ls > fichier
```

2. Supprimez fichier et relancez la commande. Est-ce qu'elle réussit ? En affichant le contenu de fichier, expliquez pourquoi c'est le cas en décrivant l'ordre des actions que le shell a effectué en exécutant ls > fichier.

   À la question 1, le fichier est déjà créé donc la redirection vers le fichier fonctionne sans surprise.
   À la question 2, on redirige la sortie du `ls` alors que le fichier n'est pas encore créé, et la commande fonctionne tout de même. En effet, lorsqu'on ouvre le contenu du fichier `fichier` créé, `fichier` apparaît dans la liste. On en conclut donc que l'ordre des actions effectuées par le shell est :

   (a) création d'un fichier de nom `fichier`

   (b) liste des fichiers présents dans le répertoire

   (c) redirection de la sortie du `ls` dans `fichier`

3. Imprimez un message dans fichier avec

```
1 $ echo "Un message" > fichier
```

La chaîne de caractère a écrasé ce que contenait fichier. Trouvez un moyen de savoir si > supprime tout ce qui était écrit dans fichier ou alors écrit juste par dessus.

Pour vérifier si > supprime ou bien écrit "par dessus" on peut créer un fichier `deux_lignes.txt` contenant :
`ligne 1`
`ligne 2`
puis exécuter :

```
$ echo "Un message"> deux_lignes.txt
```

4. Pour ajouter à la fin (append), on emploie le double chevron », par exemple

```
1 $ echo "Un autre message" >> fichier
```

Il existe un fichier "poubelle" sur Linux qu'on peut accéder en écriture seulement. Tout ce qui y est écrit est perdu. Il permet de "jeter" la sortie, par exemple :

```
1 $ ls > /dev/null
```

On peut utiliser des parenthèses pour créer un bloc contenant une séquence de commandes, et appliquer une redirection sur ce bloc.

```
1 $ ( ls ; pwd ) > fichier
```

Que fait la commande précédente ?

Si `fichier` n'existe pas, alors il est créé. Puis, la commande écrit dans un fichier de nom `fichier` la liste des fichiers présents dans le répertoire courant suivi du chemin vers le répertoire courant.

Pour spécifier l'entrée d'une commande ou d'un bloc de commandes, on utilise le chevron gauche <, par exemple

```
1 $ cat < fichier
```

Un avantage de < est d'être utilisable après un bloc de commandes, par exemple

```
1 $ ( cat ; ls ) < fichier
```

À retenir : La redirection d'entrée et la redirection de sortie requièrent un nom de fichier (unique). Note : De manière générale, les **commandes qui attendent en argument un nom de fichier** (comme cat) se comportent différemment des **commandes qui attendent un message** (comme echo). Par exemple, la commande suivante ne peut pas afficher le contenu du fichier.

```
1 $ echo < file
```

On peut bien sûr mêler redirection d'entrée et de sortie. Est-ce que les deux commandes suivantes font la même chose ?

```
1 $ cat < toto > tutu
2 $ cat > tutu < toto
```

Non, les deux commandes ne font pas la même chose : la première écrit le contenu de `toto` dans un fichier de nom `tutu`, tandis que la seconde fait l'inverse.