

TP n° 1: Introduction au Shell

Exercice 1.

Shell we start?

Le *shell* est un langage de programmation interprété : il est donc possible d'écrire des programmes plus complexes qu'une simple ligne.

Pour la facilité de lecture et de réutilisation, les programmes sont souvent écrits dans un fichier. Généralement, la première ligne d'un script *shell* indique le nom du programme qui doit l'interpréter à travers la syntaxe

```
#!/chemin/vers/le/programme
```

Par exemple `#!/bin/sh`.

Ainsi, à titre d'illustration, voici le Hello World en shell :

```
#!/bin/sh
echo "Hello, World!"
```

1. Que font les commandes suivantes : `man`, `cd`, `ls`, `pwd`, `mkdir`, `rm`, `mv`, `cp`, `file`, `echo`, `cat`, `head`, `tail`, `touch`, `history`, `cal`, `grep`? En utilisant la commande `type`, déterminer si les commandes précédentes sont des programmes ou des fonctions du shell.
2. Écrire une boucle affichant les entiers divisibles par 3 entre 1 et 100 (utiliser un `if`).
3. Tester le "hello world" : copiez le contenu dans un fichier `hello`; donnez-lui les permissions d'exécutables; exécutez `hello`.
4. La commande `wc` permet de compter le nombre de mots d'un fichier. L'option `-l` compte à la place le nombre de lignes. Lancez la commande `ls | wc -l`. À quoi correspond le résultat?
Le caractère `|` (pipe, ou tuyau) prend la sortie de la commande à sa gauche, et la donne en entrée à la commande à sa droite. À l'aide de `grep`, `cat` et du pipe, afficher le modèle du CPU (regarder le fichier `/proc/cpuinfo`).
5. Modifier le script pour qu'il affiche son premier argument.
6. Modifier le script pour qu'il affiche **tous** ses arguments.
7. Écrire un script shell qui affiche tous ses paramètres, mais **un** par ligne.
8. Écrire un script shell qui prend en paramètres deux noms de fichiers : `file1 file2` et qui supprime `file2` seulement si `file1` existe.

Attention : `file1` et `file2` doivent être des fichiers, et non pas des répertoires.

Exercice 2.

Un peu de géographie

1. Télécharger le fichier `https://remy.grunblatt.org/teaching/OSMTowns.txt` contenant les coordonnées GPS des villes de France, en utilisant la commande `wget` ou `curl`.
2. Compter le nombre de villes.
3. Trouver la ville la plus à l'Ouest de la France et la ville la plus à l'Est de la France.¹
4. En utilisant ce fichier, écrire un script shell `coordoonees` qui prend en argument le nom d'une ville et qui renvoie ses coordoonees GPS.

1. Il est possible d'utiliser la commande `cut` pour découper un fichier selon des délimiteurs. Par exemple, `echo "A,B,C,D,E" | cut --delimiter="," --fields 3` renvoie `C`.

5. Toujours en utilisant ce fichier, écrire un script shell qui prend en argument le nom d'une ville et renvoie un lien du type `https://www.openstreetmap.org/#map=18/45.72980/4.82712` (en remplaçant 45.72980 par la latitude de la ville et 4.82712 par sa longitude).
6. Utiliser la commande `grep` pour afficher l'ensemble des villes contenant "chien" dans leur nom. En une seule utilisation de `grep`, trouver l'ensemble des villes contenant "Chat" ou "chat".

Exercice 3.

Curling. - Difficile

Le but de cet exercice est d'écrire un script qui prendra en paramètre le nom d'une ville quelconque, effectuera une recherche sur Wikipedia pour trouver une image correspondant à cette ville, puis à GraphHopper (un service de navigation pour Open Street Map) pour trouver la distance entre Lyon et la ville donnée. Ensuite il téléchargera l'image et rajoutera le nom de la ville et la distance à Lyon sur l'image.

Les étapes à suivre :

1. Découvrez *GraphHopper Directions API (Application Programming Interface)*² pour récupérer la distance entre Lyon et une ville donnée. Dans un premier temps un navigateur web peut être utilisé pour trouver la bonne requête.

Pour récupérer une clef d'API, voir <https://graphhopper.com/dashboard/#/register>.

2. En utilisant cette API, écrire un script shell `distance` qui prend en argument le nom d'une ville et qui renvoie sa distance à Lyon. Utiliser `curl` pour télécharger la page web et utiliser des expressions régulières avec `grep` et `sed` pour récupérer les parties intéressantes.

Exemple :

```
$ ./distance "Porcelelette"
496
```

3. En utilisant *Mediawiki API*³, écrire un script shell `image` pour afficher le lien de la première image correspondant à cette ville.

```
$ ./image "Porcelelette"
https://upload.wikimedia.org/wikipedia/commons/2/24/Vue_du_village_de_Porcelelette.JPG
```

4. En combinant les deux scripts précédents, écrire un troisième script `descr` qui prend en argument un nom de ville et crée une image qui contient l'image de la ville avec en dessous un texte indiquant sa distance à Lyon dans un fichier `image.jpg`. Par exemple `./descr "Porcelelette"` crée l'image suivante



2. <https://graphhopper.com/api/1/docs/>

3. <https://fr.wikipedia.org/w/api.php>

5. (Optionnel) Modifier le script `image` pour prendre un entier supplémentaire pour sélectionner non pas la première image dans les résultat mais les n premières images.
☞ Utiliser l'API de commons et non celle de wikipédia : <https://commons.wikimedia.org/w/api.php>
6. (Optionnelle) Écrire un script `collage` qui prend en argument n^2 images et qui réalise un collage sous forme de mosaïque.
 - (a) (Optionnelle) Rajouter une description sous les image (par exemple le *nom de base* de l'image).
7. (Optionnelle) Utiliser les scripts précédents pour écrire un script qui prend un mot au hasard (`/usr/share/dict/*`) et réalise un collage avec les images issues de Wikipedia.