Méthologie de la Programmation
TP1 : Terminal, shell, éditeur de texte, exemples de commandes

am@up8.edu

Septembre 2022

Dans ce TP :
— Se familiariser avec l'environnement informatique des salles de TP (terminal, shell, éditeur de texte).
— Lorsque c'est pertinent, notez les réponses aux questions posées (commande ou explication) dans un fichier texte intitulé L1_MdP_TP1_VOTRENOM.txt.

Exercice 1

Un **terminal** (ou émulateur de terminal, ou console) est un logiciel qui sert de point d'accès de communication entre l'humain et la machine. Le plus souvent, on entend par terminal une fenêtre dans laquelle est lancé un shell.

Un **shell** est un interpréteur de commande. On lui envoie des commandes au clavier, le plus souvent un programme à exécuter et les arguments à lui donner en entrée, et le shell exécute la commande et sert au programme à la fois d'entrée standard et de sortie standard (c'est à dire que par défaut le programme va lire ce qu'on tape dans le shell et ce qu'il va écrire sera affiché dans le shell).

Dans cet exercice on va voir comment utiliser quelques commandes de base pour explorer l'arborescence de répertoires et de fichiers.

1. → Ouvrir un terminal.
2. Par défaut, vous êtes dans votre répertoire personnel (« *home directory* »), c'est la racine de la partie du système de fichier *qui appartient à votre compte*.

   La vraie racine du système correspond au répertoire qui se trouve à l'adresse « / ». La commande pwd (pour « *print working directory* ») affiche le chemin du **répertoire de travail**, c'est-à-dire le répertoire dans lequel vous vous trouvez quand vous exécutez l'instruction.
   ´ Dans chaque répertoire il y a deux répertoires cachés :
   — . est le répertoire lui-même, et
   — .. est le **répertoire parent**, c'est à dire celui qui est juste au dessus dans l'arborescence.

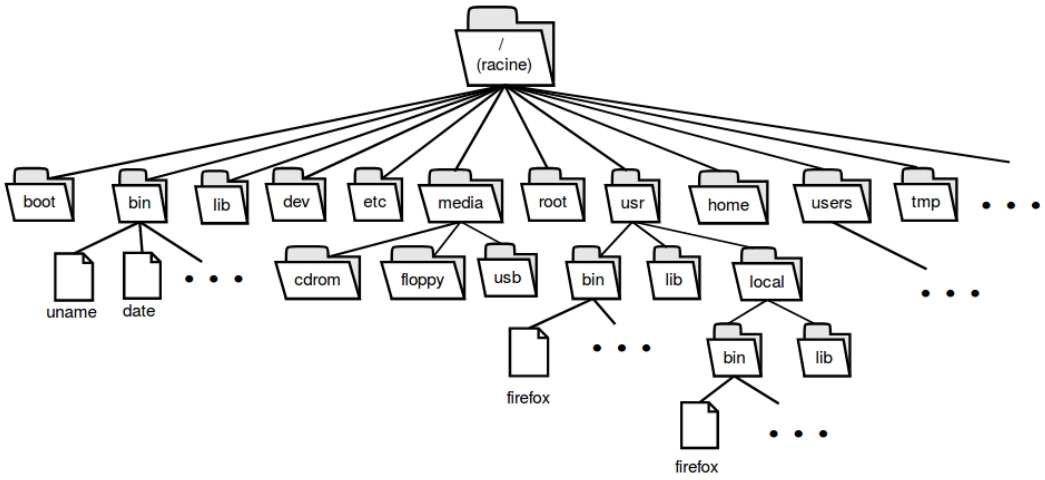

FIGURE 1 – arborescence des fichiers dans linux

Un **chemin** contient des noms de répertoires séparés par des /. chaque / veut dire que l'on entre dans un sous-répertoire.

Si un chemin commence par un /, on dit qu'il est *absolu*, c'est à dire qu'il commence à la racine du système de fichier.

Par exemple :

— le chemin absolu du répertoire `lib` de la Figure 1 est /usr/local/lib.

— le chemin absolu du fichier `uname` de la Figure 1 est /bin/uname.

Sinon, on dit qu'il est *relatif*, et il commence dans le répertoire de travail.

(a) → Afficher le chemin de votre répertoire de travail avec la commande `pwd`.

```
$> pwd
/home/am
```

`pwd` renvoie le chemin absolu vers le répertoire courant.

(b) → Quel est le chemin absolu du répertoire parent de votre répertoire maison ?

```
$> cd ..
$> pwd
/home
```

Le répertoire parent est celui qui se trouve juste avant le répertoire courant dans le chemin absolu : "/home".

3. Une ligne de commande est composée d'une `commande`, d'éventuelles `options` et d'éventuels `arguments` :

   `commande -option1 -option2 -option3 argument1 argument2`

   Le programme `echo` affiche simplement sur sa sortie standard ce qu'on lui a donné en argument.

   → Exécuter la commande `echo "coucou"`.

   votre invite de commande (ou prompt). Par défaut, le prompt affiche votre login, puis un @, puis le nom de la machine sur laquelle vous êtes connecté, puis un :, puis le répertoire de travail, et enfin un $ qui signifie que vous pouvez entrer une commande.

→ Vérifier que la commande `echo ~` donne bien la même chose que `pwd`.
Si les deux commandes ne donnent pas le même résultat c'est que vous ne vous trouvez pas dans votre répertoire maison, essayez :

```
$> cd ~
$> pwd
/home/am
```

4. Pour afficher la liste des fichiers présent dans un répertoire, la commande est `ls` chemin-du-répertoire (pour « list »). Par défaut, `ls` liste les fichiers du répertoire de travail.

   La commande `ls` accepte en argument des *options*, par exemple :
   — `-all` ou `-a` permet de lister aussi les fichiers cachés (ceux qui commencent par un .),
   — `-l` donne plein d'informations en plus sur chaque fichier (on ne s'attardera pas sur leur signification).
   → Afficher la liste des fichiers présent dans votre répertoire maison, puis afficher cette même liste mais avec les fichiers cachés.

5. Pour créer un répertoire, on utilise la commande `mkdir` chemin-du-répertoire (pour « *make directory* »).
   → Créer un répertoire `mdlp` dans lequel on mettra tout ce qui est lié à ce cours. Vérifier avec `ls` que le répertoire est bien là.

6. On peut changer de répertoire de travail avec avec la commande `cd chemin-du-répertoire` (pour « change directory »).

   (a) → Aller dans le répertoire `mdlp`.

   (b) → Afficher tout le contenu (y compris caché) du répertoire `mdlp`. Quelle commande permet de revenir dans le répertoire parent ?

   ```
   $> cd ..
   ou
   $> cd /home/am
   ou
   $> cd
   ```

   (c) → Dans le répertoire `mdlp`, créer un répertoire pour le travail de ce TP (vous pouvez l'appeler comme vous voulez, par exemple tp1) puis aller dedans.

7. La commande `rmdir` chemin-du-répertoire (pour "remove directory") permet de supprimer un dossier, mais refuse par sécurité de le faire si celui-ci n'est pas vide.
   → Créer un dossier `foo` et dedans un dossier `bar`, puis tenter de supprimer `foo` directement, puis faire ce qu'il faut pour supprimer `foo`.

8. Pour créer un fichier, il suffit d'ouvrir un fichier qui n'existe pas. Par exemple, on peut lancer l'éditeur de texte Mousepad et lui demander d'ouvrir le fichier essai.txt avec la commande `mousepad essai.txt` (si mousepad n'est pas installé, `gedit` à la place).
   Avant de poursuivre, merci de quitter complètement Mousepad (fermer toutes les instances ouvertes du logiciel).

   (a) → Lancer `Mousepad` sur le fichier `essai.txt`, et écrire par exemple « coucou » dans le fichier, et l'enregistrer.

(b) → Revenir sur le terminal sans quitter `Mousepad`, que constatez-vous ?

(c) → Quitter `Mousepad` et revenir sur le terminal.

Lorsqu'on exécute une commande dans le terminal, on démarre un processus. **Par défaut, le shell Unix attend que le processus ait fini de s'exécuter pour proposer à nouveau l'invite de commande**. Si vous tentez de lancer d'autres commandes (par exemple `ls`) celle-ci rentre dans la « file d'attente » et ne s'exécutera que lorsque Mousepad aura été fermé.
NB : si vous avez tenté de relancer Mousepad avec la commande :
$> mousepad essai.txt
il est possible que mousepad vous signale que le fichier a été ouvert plusieurs fois et modifié entre temps. Sauvegardez le fichier et fermez les instances de Mousepad.

9. Pour lancer une commande en arrière plan on peut la suffixer avec &.

   (a) → Relancer `Mousepad` sur le même fichier mais cette fois-ci en arrière plan.

   ```
   $> mousepad essai.txt &
   ```

   (b) Revenir sur le terminal sans quitter `Mousepad`, que constatez-vous ?
   L'utilisation du & à la fin de la ligne de commande signale au terminal que le processus peut être lancé en « **tâche de fond** » et qu'il peut vous rendre la main, c'est à dire proposer l'invite de commande à nouveau.

10. On peut afficher le contenu d'un fichier sur la sortie standard avec la commande `cat chemin-du-fichier` (pour *catenate*, parce qu'elle peut prendre plusieurs fichiers d'un coup et les concaténer).
    → Afficher le contenu de essai.txt, modifier le dans mousepad, puis le réafficher.

11. On peut déplacer des fichiers avec la commande `mv` (pour « move ») :
    `mv chemin-du-fichier repertoire-de-destination` Déplacez le fichier essai.txt dans le répertoire parent du répertoire dans lequel il se trouve.

    ```
    $> mv essai.txt ..
    ```

12. On peut supprimer des fichiers avec la commande `rm chemin-du-fichier` (pour « remove »). **Attention** la suppression est définitive, ça ne va pas dans une « corbeille ». → Supprimer le fichier essai.txt.

    ```
    Option 1 : supprimer depuis votre répertoire courant :
    $> rm ../essai.txt
    Option 2 : remonter dans le répertoire parent et supprimer :
    $> cd ..
    $> rm essai.txt
    ```

Exercice 2 : Exécuter l'interpréteur et le compilateur

Téléchargez les fichiers de démo python et C et vérifiez le comportement des deux programmes (`https://alicemillour.github.io/mdp/`).

Exécution de l'interpréteur :
$> python MdP_demo.py
Compilation du code C :
$> gcc MdP_demo.c -o MdP_demo
Exécution du binaire :
$> ./MdP_demo.c

Exercice 3 : Apprendre à utiliser le manuel

Pour avoir de l'aide sur une commande, il est possible de consulter son manuel d'utilisation avec la commande `man` commande (pour « manual »), par exemple :
`man cp`
vous donne l'aide de la commande de copie de fichier.

1. À quoi servent les commandes suivantes ?
   — `wc`
   — `find`
   — `sort`
   — `uniq`
2. Combien d'arguments prennent les commandes suivantes ?
   — `pwd`
   — `cp`
   — `grep`
3. Quelles sont les options à utiliser
   — pour afficher sur la sortie standard un texte dont les lignes ont été classées par ordre décroissant (`sort`)
   — pour afficher sur la sortie standard un texte dont les lignes ont été classées aléatoirement (`sort`)
   — pour lister les fichiers du dossier courant en affichant leurs tailles (`ls`)
   — pour lister les fichiers du dossier courant en affichant leurs tailles de manière « lisible par un humain (*human-readable*) » (`ls`)

Exercice 4 : manipulation élémentaire de fichiers

1. Déplacez-vous dans le répertoire que vous avec créé pour ce cours et créez-y un fichier `premier_fichier.txt` grâce à la commande `touch`.
   On peut **rediriger la sortie standard** vers :
   — un fichier en l'écrasant avec l'opérande >
   — un fichier en concaténant la sortie au contenu existant avec >>
   — une nouvelle commande avec |
2. essayez les commandes suivantes en vérifiant le contenu du fichier au fur et à mesure.

```
echo "première phrase"
echo "première phrase" > premier_fichier.txt
cat premier_fichier.txt
```

```
echo "première phrase" >> premier_fichier.txt
cat premier_fichier.txt
echo "deuxième phrase" >> premier_fichier.txt
cat premier_fichier.txt
echo "troisième phrase" > premier_fichier.txt
cat premier_fichier.txt
```

3. Dans le répertoire créé pour ce cours, créez un nouveau répertoire `tests` et déplacez-y le fichier `premier_fichier.txt`.

Récupérez sur la page `http://alicemillour.github.io/teachings/mp` l'archive `MdP_2223_TP1.zip`

1. téléchargez l'archive et enregistrez-la dans un répertoire de travail adapté.
2. utilisez la commande `unzip` pour décompresser l'archive.

Manipuler les fichiers

la commande `grep`

1. À quoi sert la commande `grep` ?
2. Vous savez que vous avez écrit le numéro de la **salle** du cours dans un fichier, mais vous ne savez plus lequel... placez-vous dans le répertoire `fichiers_perso` et utilisez `grep` pour la retrouver. Donnez le chemin absolu du fichier dans lequel vous avez trouvé le numéro de salle.

```
$> grep -r "A162"
```

L'option -r correspond à une recherche récursive dans tous les sous-dossiers et fichiers du répertoire courant.

la commande `wc` Le répertoire `littérature` contient deux fichiers : le fichier `proust.txt` contient des extraits de *Du côté de chez Swann*, `camus.txt` contient des extraits de *L'Étranger*.

Utilisez la commande `wc` pour vérifier qu'en moyenne, les phrases écrites par Marcel Proust sont plus longues que celles écrites par Albert Camus.

```
$> wc -l camus.txt
$> wc -w camus.txt
$> wc -l proust.txt
$> wc -w proust.txt
```

Les phrases d'Albert Camus font en moyenne 11 mots, celles de Marcel Proust 29.

la commande `tr`

1. À quoi sert la commande `tr` ?
2. Vous souhaitez classer par ordre alphabétique les mots du texte `proust.txt` pour créer un lexique proustien.

(a) Commencez par obtenir une liste de mots, en utilisant la commande `tr` pour remplacer les caractères espace (' ') par des retours à la ligne ('\n'). Redirigez la sortie obtenue grâce à l'opérateur | pour classer les mots par ordre alphabétique et enregistrez le résultat dans le fichier `mots_proust.txt`.

```
$> cat proust.txt | tr " " "\n" | sort > mots_proust.txt
ou
$> tr " " "\n" < proust.txt | sort > mots_proust.txt
```

Cette solution n'est pas satisfaisante : il reste les signes de ponctuation collés à certains mots. On peut utiliser `tr` pour les supprimer :

```
$> cat proust.txt | tr " " "\n" | tr -d [ :punct :] | sort > mots_proust.txt
```

(b) Le fichier `mots_proust.txt` contient des répétitions. Utilisez les commandes `uniq`, `sort` et leurs options pour trouver les 10 mots les plus fréquents dans `proust.txt`.

```
$> cat mots_proust.txt | uniq -ic | sort -h
```

— l'option -i de `uniq` permet d'ignorer la casse des mots (majuscule ou minuscule)
— l'option -c de `uniq` permet d'afficher le nombre de fois qu'apparaît une ligne (qui correspond ici à un mot)
— l'option -h de `sort` indique qu'il faut classer les lignes selon l'ordre numérique

la commande `split`

1. À quoi sert la commande `split` ?
2. Vous avez commencé à écrire un texte : `mon_texte.txt` et vous souhaitez le faire corriger. Pour gagner du temps, vous divisez le texte en trois pour le faire corriger par trois amis. Utilisez la commande `split` pour créer les fichiers correspondants. Utilisez le paramètre optionnel `PREFIX` pour préfixez les fichiers avec `mon_texte_`.

   On peut faire encore mieux en ajoutant le suffixe .txt aux trois textes créés, et utiliser l'option -d qui utilise des suffixes numériques.

   ```
   $> split -n 3 -d mon_texte.txt mon_texte_ –additional-suffix=.txt
   ```

   Résultat :
   mon_texte_01.txt
   mon_texte_02.txt
   mon_texte_03.txt