TP : Prise en main du shell bash

Ce TP s'inspire largement de l'excellent travail de mon collègue Nicolas Pécheux.

Un compte-rendu sur feuille (ou dans un fichier) de ce TP est demandé. Certains compte-rendus seront ramassés. Si une question ne comporte pas de phrase interrogative, faire un compte-rendu des manipulations demandées. Les possesseurs d'un Mac ou les utilisateurs d'une machine virtuelle sur Windows risquent de ne pas avoir de réponse à fournir à certaines questions.

Prise en main

Exercice 1.

1. L'instruction `man` *`nomDeCommande`* donne le manuel de la commande. Entrer `man ls`, naviguer avec les flèches ↑ et ↓, quitter en entrant **q+Enter**.

   Que fait l'option `-t` ?
2. Entrer la commande `exit`. Que se passe-t-il ?
3. Que renvoie la commande `whoami` ?
4. Nom de machine :
   - Utiliser `man` pour comprendre ce que fait la commande `hostname` avec ou sans argument.
   - Changer le nom de votre machine en `monordi` (ne pas s'inquiéter, c'est provisoire jusqu'au prochain redémarrage).

     Il semble ne s'être rien passé. Ouvrir un nouveau terminal et constater que le nom de machine a bien changé.
5. Entrer `echo coucou`. Puis lancer `history`.
   - Relancer la dernière commande utilisée avec `!!`.
   - Relancer la commande d'affichage de « coucou » par `!num de commande` où **num de commande** est une information à trouver dans l'historique.
6. A quoi sert la commande `gcc` ? Utiliser le manuel pour le savoir.
7. Avec toutes ces tests, votre terminal est encombré par les résultats des commandes précédentes. Entrer `clear` pour y remédier.
8. La commande `wc` retourne le nombre de lignes, de mots et d'octets d'un fichier.
   (a) trouver l'option qui ne donne que le nombre de lignes
   (b) Si j'arrive à « brancher » la sortie de `ls` sur l'entrée de `wc`, je vais pouvoir compter le nombre de fils de la racine de mon arborescence. Ce branchement de la sortie d'une commande sur l'entrée d'une autre s'appelle un pipe.

   A l'aide d'un pipe, compter le nombre de fils de la racine.

Architecture

Attention, pour les commandes suivantes, il peut être utile de les exécuter plutôt avec les droits du superuser (`sudo maCommande`).

Exercice 2.

1. Avec la commande `lscpu` donner :
   (a) L'architecture du processeur ;
   (b) Son nom de modèle ;
   (c) Son constructeur ;
   (d) Sa vitesse en MHz ;
   (e) Votre processeur est-il en *Little Endian* ou *Big Endian* ? Chercher sur le web ce que ça veut dire.
2. La commande `lshw` affiche des informations très détaillées (trop) sur les périphériques d'un ordinateur.
   Entrer `lshw` . Cela donne beaucoup d'informations.
   On se concentre sur la mémoire. On cherche
   (a) quel type de barrette mémoire est utilisé sur votre machine (on cherche DIMM (PC actuels), SO-DIMM (PC portables) ou SIMM (PC un peu anciens)
   (b) une information sur le standard (DDR3, DDR4).
   (c) le nombre de barettes mémoire sur votre carte mère, leur taille (à priori en GiB -gibibytes : 1024^3 octets-) et le fabricant.
   Entrer `lshw -class memory` et donner ces informations
3. La commande `free` donne des informations sur l'utilisation de la mémoire de votre ordinateur. A l'aide du manuel, trouver les options pour afficher en Gigabytes la mémoire puis en Gibibytes.
   Dans chaque cas donner la mémoire totale installée sur votre machine ainsi que celle qui est disponible.
4. La commande `df -h` (*Display Free* « h » pour « human readable ») affiche l'espace mémoire disponible sur les périphériques.
   (a) Sur quelle *partition* (partie -réelle ou virtuelle- d'un disque dur) est monté le BIOS/UEFI ? Quelle est sa taille ?
   (b) Sur quelle partition est installé votre système ? (une indication : le système est monté sur la racine `/`).
5. La commande `lspci` permet de connaître la liste du matériel (cartes, chipsets, contrôleurs, etc.) de votre PC utilisant l'interface PCI. Les cartes graphiques sont reconnaissables au sigle VGA.
   — A l'aide de cette commande donner la ou les cartes graphiques installées sur votre ordinateur (sur mon PC de bureau, j'en ai une, sur mon PC portable, deux).
   — Les processeurs modernes sont équipés d'une carte graphique « basique ». Identifiez la.

Arborescence des fichiers

Exercice 3. 1. Dans quel répertoire êtes-vous (quelle commande utiliser) ?

2. Qu'y a-t-il dedans ? (quelle commande utiliser)
3. Créer avec `mkdir` et `cp` dans votre répertoire de travail un répertoire **Perso** dont l'arborescence est indiquée figure **??**. Les fichiers **1.txt, 2.txt, 3.txt** sont tous des fichiers vides.

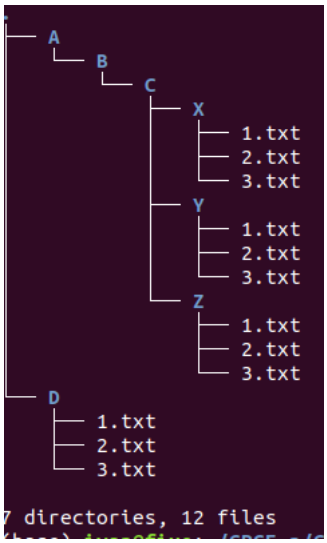

FIGURE 1 – Le répertoire Perso et ses descendants à créer.

Une fois le travail effectué, l'afficher avec `tree` (si la commande n'existe pas, l'installer avec `sudo apt install tree`.

4. Dans le répertoire parent de **Perso**, créer une copie **Perso2** (option récursive de `cp`).
5. Créer un répertoire vide **Perso3**. Le supprimer avec `rmdir`.
6. Utiliser `rmdir` pour supprimer **Perso2**. Un problème ? Consulter `man rmdir`.
7. Changeons de méthode : Supprimer (récursivement) **Perso2**. Vérifier par `ls`.
8. Revenons à **Perso**. Déplacer **X** et son contenu dans **D**. Dessiner sur une feuille l'arborescence obtenue. Vérifier avec **tree**

Processus et mémoire

Exercice 4. Un processus est un programme en cours d'exécution. On utilise l'utilitaire de gestion de processus `htop` qu'on peut installer avec `sudo apt-get install htop`

1. Lancer la commande `htop`. (`q` pour quitter).
   (a) Combien y at-il de processus en cours d'exécution (chercher **Tasks** en haut de l'écran) ? Un autre nom pour *processus* est *tâche*.

(b) Combien de mémoire vive est installé ? Combien est utilisé ? (Regarder en haut de l'écran)

(c) Il y a beaucoup de processus. Cherchons ceux associés à votre compte utilisateur.
En bas de l'écran vous voyez les actions associées aux touches **F1,F2,F3...**. Appuyons sur **F4** (Filtrer) et s'en servir pour trouver quel numéro de processus (PID) est associé à votre shell bash (il peut y en avoir plusieurs).

Quitter l'environnement **htop** en cliquant sur **Quitter** en bas de l'écran.

2. Sous **emacs**, écrire

```
i, t = 1, []
while i > 0:
    t.append(i)
```

Enregistrer ce programme comme **idiot.py** dans le répertoire de ce TP. Quel problème posera ce programme ?

Lancez-le en arrière-plan pour ne pas bloquer votre console (avec l'esperluette) : `python idiot.py &`.

A partir de là, il va falloir faire vite :

(a) relancer `htop` et observer la mémoire consommée grossir (mais ne tardez pas trop, le système va planter).

(b) Filtrer pour trouver les programmes **python** qui sont lancés. Vous devriez trouver votre programme comme sur la figure **??**.

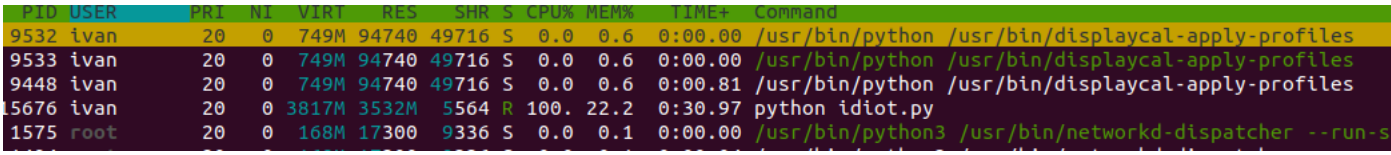

FIGURE 2 – le processus lancé par la commande **python idiot.py**

(c) Votre processus grignote la mémoire. On va le tuer. Choisir la touche F9 (KILL). On peut maintenant envoyer un signal au processus. Sélectionner SIGKILL (numéro 9) et appuyer sur entrée. Le processus est tué.

Gestion des droits

Exercice 5. 1. Ajoutons un nouvel utilisateur **toto** à notre système : `sudo adduser toto`. Choisir « toto » comme mot de passe. Mettre n'importe quoi aux informations demandées puis, à la question « ...Ces informations sont-elles correctes ? [O/n] » répondre **O** et entrer.

Entrer `ls /home/`. Vous devriez constater l'apparition d'un nouveau répertoire dans /**home**.

2. Changeons d'utilisateur : `su toto`. On vous demande d'entrer le mot de passe de **toto**.
3. Ca y est ! Vous êtes **toto**.

(a) Aller dans votre répertoire personnel (comment déjà) ?

(b) Créer un fichier vide **file1**

(c) Redevenez vous-même `su isabelle` (pu mieux : **exit**) si vous vous appelez Isabelle. Et revenez dans votre **home** personnel.

4. Donc **toto** existe et il a un fichier dans son **home**.
   (a) En restant vous-même, aller dans le **home** de **toto** par un `cd`. Puis revenez chez vous.
   (b) Depuis votre **home** lister le contenu de celui de **toto**.
   (c) Entrer `ls -l /home/toto`.
   On va changer ces droits.
   (d) Changeons les droits de **/home** de **toto**. Comme super-utilisateur mettre les droits de **home/toto** sous la forme **rwx r−− r−−**.
      i. Sous votre identité tenter d'afficher le contenu de **/home/toto** avec les droits et les fichiers cachés. Que se passe-t-il ?
      ii. Sous votre identité tenter d'aller dans **/home/toto**. Que se passe-t-il ?
      iii. Tenter les même commandes comme superuser.
      iv. Comme superuser, supprimer les droits en lecture de **/home/toto** au reste du monde.
      Sous votre identité tenter d'afficher le contenu de **/home/toto** avec les droits et les fichiers cachés. Que se passe-t-il ?
5. Supprimons cet utilisateur `sudo toto userdel -r toto` (il faudra peut-être fermer d'abord votre terminal puis en redémarrer un autre).
   Vérifier maintenant les sous-répertoires de **/home** en mode superuser.
6. Allez, un petit exo pour la route. Traduire :

r-x r– r–	544
	777
rwx -w- –x	
	602

Exercice 6. Pour faire cet exercice, créer un répertoire **myRep** et s'y rendre.

```
$ mkdir myRep
$ cd myRep
```

Amusons nous maintenant à écrire un petit script **bash**. Il va afficher une invitation à dire « oui » jusqu'à ce que l'utilisateur entre effectivement « oui ».

L'important n'est pas ici de comprendre la syntaxe des scripts ni le fonctionnement des variables du **bash** mais plutôt de rendre un fichier exécutable

Mais enfin, toute occasion d'apprendre étant bienvenue, nous expliquons un peu le contenu du script.

Lancer la commande `emacs oui.sh &`. Une fenêtre **emacs** s'affiche.

Un script (**bash** ou **Python** ou encore **Perl**, commence toujours par la mention du programme chargé de l'interpréter. Cette première ligne s'appelle la ligne « shebang » :

```
#!/bin/bash
```

C'est donc le programme **bash** du répertoire **bin** qui va interpréter notre script.

Nous introduisons ensuite la variable **reponse** :

```
reponse='non'
```

On s'en doute, cela signifie que la valeur de **reponse** est la chaîne de caractères **'non'**. Les guillemets (droits) sont ceux de la touche 4.

Commençons notre boucle **while** :

```
while [ -z $reponse ] || [ $reponse !='oui' ]
do
```

Les tests en **bash** s'écrivent entre crochets. Nous décidons d'entrer dans la boucle si la variable **reponse** n'est pas vide (avec `[ -z $reponse ]`) ou alors si **reponse** est différente de **'oui'** (avec `[ $reponse !='oui' ]`). Le mot clé `do` indique le début du corps de la boucle, comme en OCaml.

La fonction **read** affiche unne invite et attend que l'utilisateur ait entré une chaîne de caractères. Cette dernière remplace l'ancien contenu de **reponse** :

```
    read -p 'Dites oui : ' reponse
done
```

L'option `-p` permet d'afficher le message d'invite avant la zone de saisie. Le mot clé `done` indique la fin du corps de la boucle, comme en OCaml.

Voici donc notre fichier complet :

```
#!/bin/bash
reponse='non'
while [ -z $reponse ] || [ $reponse != 'oui' ]
do
        read -p 'Dites oui : ' reponse
done
```

Enregistrons le sous le nom **oui.sh** dans le répertoire **myRep**.

Ci-dessous on donne une trace d'exécution où l'utilisateur a entré **'non'** puis la chaîne vide et enfin **'oui'**.

```
Dites oui : non
Dites oui :
Dites oui : oui
```

Questions :

1. Tentons d'exécuter ce programme. Entrons `oui.sh`. Que se passe-t-il ?
2. Tentons autre chose : on indique au système que la commande **oui.sh** se trouve dans le répertoire courant. Entrons `./oui.sh`. Que se passe-t-il ?
3. Afficher les droits du fichier. Qu'observe-t-on ?
4. Pour remédier au problème on va déclarer **oui.sh** exécutable par l'utilisateur. Un bon `chmod` doit changer ça.
5. Recommencer `./oui.sh`. Bon amusement !
6. C'est embêtant ces `./` pour exécuter notre programme **oui.sh**. On va s'en passer :
   (a) Donner l'affichage de `echo $PATH`. On obtient tous les chemins que le système regarde pour trouver l'endroit où est stocké le fichier d'une commande. A priori le fichier **myRep** n'est pas dedans.
   (b) Ajoutons **myRep** provisoirement (pour cette session) au **$PATH** par la commande

   ```
   $ export PATH=$PATH:~/chemin/jusqu'à/votre/répertoire/myRep
   ```

   Bien sûr, il faut entrer le chemin depuis la racine de votre **home** jusqu'à votre **myRep**
   (c) Afficher le nouveau contenu de **$PATH**. Puis entrer `oui.sh`. Top bono !

Divers

Exercice 7. Toutes les recherches s'effectuent dans le répertoire `/dev`

1. Lister tous les périphériques qui sont en accès par bloc.
2. Compter le nombre de périphériques en mode caractère.
3. Tous les fichiers dont le nom de comporte aucune des lettres **a,w,u,h,o,s,t**.

Pour inspecter avec `grep` récursivement un répertoire, on utilise l'option `-r`. Avec l'option `--include "*tex"`, on ne filtre par exemple que les fichiers **.tex**.

4. Dans un répertoire où vous savez qu'il y a beaucoup de fichiers **.c** (par exemple, votre **home**) :
   (a) Afficher pour chaque fichier du répertoire et ses descendants, le nombre de fois ou le mot « affiche » apparaît.
   (b) Chercher les fichiers **.c** qui comportent un nombre non nul de fois le mot « affiche ».

Avec la commande `cut` on découpe une ligne par champs. On précise le délimiteur de champs avec l'option `-d` . Cela a pour effet de séparer ce qu'il y a avant et après le délimiteur. Avec l'option `-f` suivi du numéro de champ, on récupère le champ correspondant.

Dans un fichier **toto**, écrivons :

```
asup.c
td.pdf
arg.c
```

Alors

```
$ cut -d '.' -f 1 toto
asup
td
arg
```

De même

```
$ cut -d '.' -f 2 toto
c
pdf
c
```

5. En explorant récursivement votre **home**, récupérer le nombre maximal d'occurrences du mot « affiche » dans un fichier **.c**.

   Indication. J'ai utilisé dans le désordre **tail,cut,sort,grep**.