

INITIATION À L'ENVIRONNEMENT UNIX : TP N° 4

septembre 2024 — Pierre Rousselin

1 Révisions

Exercice 1 : Révisions

Vous devriez être capable de faire cet exercice sans aide. Si ce n'est pas le cas, il faut revenir à des exercices similaires dans les sujets de TP précédents.

- Créer un répertoire `rep_exo1` et dedans l'arborescence suivante, où les fichiers `a`, `b`, ..., `i` sont des répertoires et `foo.txt` et `bar.c` sont des fichiers réguliers. La commande `tree` depuis `rep_exo1` doit donner la sortie suivante :

```

.
├── a
│   ├── b
│   │   ├── foo.txt
│   │   └── c
│   │       └── bar.c
│   └── d
│       ├── e
│       └── f
└── g
    ├── h
    └── i
  
```

9 directories, 2 files

- Éditer les fichiers `foo.txt` et `bar.c` avec un éditeur dans le terminal. Les sorties de ces commandes `cat` depuis `rep_exo1` doivent être les suivantes :


```

$ cat a/b/foo.txt
bar
baz
$ cat a/c/bar.c
int main() {
    return 0;
}
      
```
- Faire une copie de `foo.txt` dans le répertoire `g`.
- Se déplacer dans `g` et supprimer le fichier `foo.txt` du répertoire `b`.
- Copier tous les fichiers du répertoire `/usr/include` dont le nom commence par `sys` et se termine par `.h` dans le répertoire `f`. Qui est le propriétaire de ces fichiers ? Et des fichiers dans `/usr/include` ?
- Changement d'avis, les déplacer dans `e`.
- Renommer `bar.c` en `my_true.c`.
- La commande `true` est-elle une primitive du `shell` ou un programme externe ? Voir son aide et l'exécuter.
- Quelles sont toutes les commandes `true` disponibles (voir l'aide de `type` si besoin) ? À quoi correspond la page de manuel de `true` ? Qu'y est-il écrit dans `NOTE` ?
- Avec un compilateur C (par exemple `gcc`, s'il est installé), compiler `my_true.c`, appeler `my_true` l'exécutable produit. Si besoin, se référer au cours de programmation 1 pour la compilation d'un programme en C.

11. Exécuter le programme `my_true` :
 - a) depuis le répertoire `c` ;
 - b) depuis le répertoire `a` (c'est-à-dire en se déplaçant d'abord dans `a`) ;
 - c) depuis votre répertoire personnel.
12. Faire une sauvegarde de votre `bashrc` (par exemple, appelée `bashrc_svg`). Ajouter le répertoire contenant `my_true` au `PATH` dans votre `bashrc` puis sourcer le `bashrc`. Les commandes suivantes devraient fonctionner depuis n'importe quel répertoire :

```
$ my_true
$ echo $?
0
$ if my_true; then printf 'lol\n'; fi
lol
```

13. On va masquer `my_true`, avec un script shell cette fois. Dans le répertoire `i`, créer un fichier `my_true` contenant un script shell qui se contente d'afficher `coucou` sur le terminal. Le rendre exécutable et l'exécuter.
14. Ajouter le répertoire `i` à la variable `PATH` dans votre `bashrc` de façon à ce que les commandes suivantes produisent les sorties suivantes (depuis n'importe quel répertoire) :

```
$ my_true
coucou
$ echo $?
0
$ if my_true; then printf 'lol\n'; fi
coucou
lol
$ while my_true; do sleep 1; done
coucou
coucou
coucou
^C
```

Pour tuer la dernière commande, taper `Ctrl`+`c`.

15. Ménage : supprimer le répertoire `rep_exo1` et tout ce qu'il contient. Revenir à votre `bashrc` d'origine.

--- * ---

2 Utilisateur et groupe

Exercice 2 :

1. À l'aide du manuel de la commande `id`, chercher comment écrire sur le terminal :
 - a) votre numéro d'utilisateur (`uid`) ;
 - b) votre nom d'utilisateur ;
 - c) le numéro de groupe (`gid`) de votre groupe courant ;
 - d) le nom de votre groupe courant ;
 - e) les noms et numéros des autres groupes auxquels vous appartenez.
2. Créez un nouveau fichier `truc` et regardez avec `ls -l` son utilisateur et son groupe propriétaire.
3. À l'aide de la commande `chgrp` (dont vous irez voir la page de manuel) changer le groupe propriétaire du fichier `truc`, utiliser un des groupes auxquels vous appartenez.
4. Le répertoire `COMMUN` est dans le répertoire parent de votre `HOME`. Créez-y un petit fichier texte de quelques lignes et faites en sorte que seuls les étudiants de L1 puissent le lire et le modifier, pas vos enseignants.

5. Utilisez la commande `newgrp` avec l'un des groupes auquel vous appartenez, mais pas le groupe courant. Ensuite créez un fichier et visualisez son groupe propriétaire. Remarque : il est possible que ça ne fonctionne pas dans la sous-arborescence issue de votre `home` (à cause du NFS, *Network File System*). Dans ce cas, essayer cette manipulation dans le répertoire `/tmp`. Revenez à votre shell précédent avec `Ctrl`+`d`.

--- * ---

3 Contenus des fichiers

Exercice 3 : Du texte

1. Créer un répertoire `contenu_fichier` pour cet exercice et s'y déplacer.
2. En utilisant un éditeur de texte, entrer le texte suivant dans le fichier `Mirabeau.txt`
Vienne la nuit sonne l'heure
Les jours s'en vont je demeure
3. Voir le type de ce fichier avec la commande `file`.
4. La commande `od -t x1 -t c Mirabeau.txt` affiche le contenu de `Mirabeau.txt` octet par octet, chaque octet représenté sous forme hexadécimale (option `-t x1`), c'est-à-dire en base 16, où deux chiffres représentent un octet, et interprété comme un caractère (option `-t c`). Voir le contenu du fichier `Mirabeau.txt` et répondre aux questions suivantes :
 - a) Quelle est la taille (en octets) du fichier ? Voir aussi la sortie de `wc -c Mirabeau.txt` et de `ls -l Mirabeau.txt`.
 - b) Quels octets (en hexadécimal) correspondent à la lettre B, à l'espace, à la fin de ligne ?
 - c) Retrouver ces caractères dans la page `man ascii`.
5. Avec le logiciel de traitement de texte `LibreOffice`, écrire le fichier `Mirabeau.ods` en y entrant les mêmes deux lignes que précédemment.
Voir avec `od` son contenu et avec `wc -c` sa taille.
Le fichier `Mirabeau.ods` est-il lisible par un humain ? Discuter des avantages et inconvénients des formats texte et OpenDocument.
6. Ajouter la ligne suivante à la fin du fichier `Mirabeau.txt`
Les mains dans les mains restons face à face
Le type du fichier a-t-il changé ? Voir son contenu avec `od`. Est-ce qu'il y a autant d'octets que de caractères dans le fichier ? Si non pourquoi ? Observer la différence entre `wc -c` et `wc -m`.
7. En éditant un nouveau fichier texte, dire sur combien d'octets sont codés, en UTF-8, les caractères © (copyright), μ (la lettre grecque mu) et €.
8. BONUS : pour ceux qui sont bien avancés et ont déjà bien compris le binaire, voir la page `man utf-8` et comprendre comment passer du point de code unicode de â, de € et de © au codage en UTF-8. Chercher les points de code unicode sur Wikipedia.
9. Avant que le format UTF-8 soit bien répandu, les symboles les plus courants utilisés en français étaient codés sur un seul octet, en étendant `ascii`. Ce format s'appelle ISO/CEI 8859-1 ou encore latin-1. La commande suivante permet de convertir un fichier `utf-8` en `latin-1`.
`$ iconv -f UTF-8 -t LATIN1 Mirabeau.txt >Mirabeau_latin1.txt`
Visualiser `Mirabeau_latin1.txt` dans un éditeur de texte. A-t-il compris qu'il y avait un caractère â ? Voir le type du fichier avec `file`, son contenu avec `od`, compter ses caractères et ses octets.
Discuter des avantages et inconvénients de `latin-1` par rapport à `utf-8`.
10. À l'aide du manuel, trouver comment afficher tous les encodages de fichiers supportés par `iconv`.

--- * ---

Exercice 4 : Des fins de ligne

Le fichier `Mirabeau_windows.txt` a été écrit sous windows. Il est disponible à l'adresse https://www.math.univ-paris13.fr/~rousselin/unix/Mirabeau_windows.txt

1. Téléchargez-le en ligne de commande.
2. Voyez son type avec `file` et son contenu avec `od`. Qu'y a-t-il de spécial ? Consulter : https://fr.wikipedia.org/wiki/Fin_de_ligne
3. La commande

```
$ tr IL_MANQUE_DES_CHOSES_ICI <Mirabeau_windows.txt >Mirabeau_unix.txt
```

va nous permettre d'enlever les caractères retour chariot. À vous, en consultant le manuel, de trouver par quoi remplacer `IL_MANQUE_DES_CHOSES_ICI`.
4. Vérifier avec `file` et `od` que ces retour chariot ont bien disparu.
5. BONUS : avec simplement les commandes `printf` et `cat`, créer, à partir de `Mirabeau_unix.txt`, le fichier `Mirabeau_windows2.txt` qui a les fins de ligne windows. Indices : changer l'IFS et utiliser une substitution de commandes.
6. BONUS2 : Faire la même chose avec `awk`. La page wikipedia <https://en.wikipedia.org/wiki/AWK> contient beaucoup d'exemples.
7. Si vous êtes sur votre machine personnelle, cherchez comment installer les commandes `dos2unix` et `unix2dos` si ce n'est pas déjà fait et voir leur manuel.

--- * ---

4 Fichiers : inodes, liens physiques et symboliques

Exercice 5 : Dates associées aux fichiers

1. Dans le répertoire de votre choix, créer un fichier (par exemple `bar`) contenant une ligne de texte de votre choix.
2. Noter l'heure affichée par `ls -l bar`.
3. Afficher le contenu du fichier, puis revoir l'heure affichée par `ls -l`. A-t-elle changée ?
4. Modifier le contenu de `bar`, par exemple en ajoutant une ligne de texte et voir la sortie de `ls -l`. Parmi les 3 dates vues en cours (accès, modification, changement) quelle est celle qui est affichée par défaut avec `ls -l` ?
5. Chercher à l'aide de `man ls` l'option permettant d'afficher la date de dernier accès plutôt que celle de dernière modification et l'essayer sur votre fichier.
6. Afficher le numéro d'inode correspondant à `bar`.
7. Renommer le fichier `baz`. À l'aide des sorties de

```
$ ls -lid . baz
```

dire qui a été modifié : le fichier ? le répertoire qui le contient ?
8. Changer les permissions du fichier en enlevant à tous les autres utilisateurs la permission de le lire et de le modifier puis voir la sortie de la commande `stat`. Qui a été modifié, le fichier ou son inode ?

--- * ---

Exercice 6 :

1. Créer l'arborescence suivante :

```
rep_liens/  
└── rep1  
    └── fich1  
1 directories, 1 file
```

où **rep1** est un répertoire et **fich1** est un fichier normal contenant une ligne de texte (de votre choix). Déplacez-vous dans **rep_liens/**

2. Voir le numéro d'inode de **rep1/fich1** et son nombre de liens à l'aide de **ls -li**.
3. La commande **du -sh .** donne l'occupation totale sur le disque par les fichiers normaux dans l'arborescence issue du répertoire courant. Qu'écrit-elle ici ? (Remarque : la mémoire du disque est sans doute organisée en blocs de 4 Kio, ce qui fait qu'un fichier, même tout petit, occupe tout de même 4 Kio).
4. En utilisant la commande **ln**, créer dans **rep_liens** un lien physique nommé **fich** vers le fichier **rep1/fich1**. Voir la sortie de

```
$ ls -li fich rep1/fich1
```

en faisant particulièrement attention au numéro d'inode. Afficher le contenu de **fich** et **rep1/fich1**.
5. Combien d'espace en mémoire occupent les fichiers normaux de l'arborescence issue de **rep_liens** ? Essayez de deviner avant de lancer **du** pour le vérifier.
6. Ajouter une ligne au fichier **rep1/fich1**. Voir le contenu de **fich**.
7. Créer une *copie*, nommée **copie**, de **fich**. Voir son nombre de liens et son numéro d'inode. Puis modifier son contenu et comparer avec celui de **fich**. Voir l'occupation en mémoire avec **du**.
8. Créer dans **rep_liens** un *lien symbolique* nommé **symlink** vers **rep1/fich1** avec **ln -s** (voir le manuel de **ln**). Afficher le contenu de **symlink** avec **cat**, ses attributs avec **ls -li** et l'occupation en mémoire de **rep_liens** avec **du**. Qu'est-ce qui distingue un lien symbolique d'un lien physique ?
9. Enlever la permission **x** sur le répertoire **rep1**. Que se passe-t-il lorsque vous entrez les commandes

```
$ cat fich
$ cat symlink
```

et pourquoi d'après-vous ?
10. Redonnez-vous la permission **x** sur **rep1** et observez l'inode de **rep1/fich1** avec la commande **stat rep1/fich1**. Renommez **rep1/fich1** en **rep1/fichier1**. Le numéro d'inode de **fichier1** a-t-il changé ? Et l'inode lui-même ?
11. Que se passe-t-il maintenant lorsque vous entrez les commandes suivantes ?

```
$ cat fich
$ cat symlink
```

À l'aide de **ln -is**, « réparez » le lien cassé **symlink**. Vérifiez en affichant le contenu de **symlink**.
12. Ouvrez **symlink** dans un éditeur de texte pour en modifier le contenu. Celui de **rep1/fichier1** et de **fich** ont-ils changé ? Pour votre éditeur de texte, y a-t-il une différence entre ouvrir le lien symbolique ou l'un des liens physiques ?
13. Supprimer **rep1/fichier1** et essayez d'ouvrir (avec **cat** ou un éditeur de texte) les fichiers **symlink** et **fich**. Y a-t-il une différence ?
14. **Le plus important :** grand ménage, on supprime le répertoire **rep_liens** et tout ce qu'il contient et surtout on **prend des notes** sur ce qu'on a appris ici :
 - a) Comment créer un lien physique ? un lien symbolique ?
 - b) Qu'est-ce qui différencie les deux ?
 - c) Quelle est la différence entre une copie et un lien (physique) ?

--- * ---

Exercice 7 : Liens et répertoires

1. Recréez l'arborescence du début de l'exercice précédent. Créer un second répertoire **rep2** dans **rep_liens**.
2. Est-il possible de créer un lien physique vers le répertoire **rep_lien/rep1** ? vers un répertoire en général ?

3. Déplacez-vous dans `rep_liens/rep2` et créez-y un lien symbolique `symrep` vers le répertoire `../rep1`. La commande `tree` sur `rep_liens` devrait donner :

```
rep_liens/
├── rep1
│   └── fich1
└── rep2
    └── symrep -> ../rep1/
3 directories, 1 file
```

4. Entrez les commandes suivantes depuis `rep_liens`

```
$ ls rep2/symrep/
$ ls -lid rep2/symrep
$ ls -lid rep2/symrep/
$ cat rep2/symrep/fich1
$ cat rep1/fich1
```

Finalement, dans `rep2/symrep`, tout se passe comme si ... ?

5. Vous trouvez ça compliqué ? Vous n'avez encore rien vu... Essayer la succession de commandes :

```
$ cd rep2/symrep
$ pwd
$ pwd -P
$ ls ..
$ cat ../fich1
$ cd ..
$ ls # oui, c'est pire...
$ cd - # on retourne au répertoire précédent
$ pwd
$ pwd -P # comme "physique"
$ cd -P .. # idem
$ pwd
```

Vous n'avez pas bien compris ? C'est normal... En fait, `pwd` et `cd` se comportent différemment des autres commandes avec les liens symboliques vers les répertoires.

Quand on a entré `ls ..` depuis `rep2/symrep`, tout s'est passé comme si on l'avait fait depuis `rep1` (normal car `rep2/symrep` est un lien symbolique vers `rep1`).

Idem pour `cat ../fich1` qui affiche bien le `fich1` de `rep1`, comme il se doit.

Le problème est (comme souvent) le shell, qui, pour `pwd`, affiche le chemin avec le lien symbolique (et pas le « vrai » répertoire) et avec `cd ..` se contente (sans l'option `-P`) de se déplacer dans le répertoire qui précède celui donné par `pwd`...

--- * ---

5 Scripts shell : construction case

Nous allons maintenant voir la construction `case` qui est la construction la plus utile dans les scripts shell.

Exercice 8 : Construction `case` et commande `read`

1. Dans un éditeur de texte, écrire le script suivant que vous appellerez `qui_est_ce`

```
#!/bin/sh
echo 'Bonjour, qui est-ce ?'
read nom
case $nom in
```

```

$USER | [mM]oi)
    echo "Suis-je bête, c'est moi !"
    ;;
*[Vv]irus*)
    echo "Sors de mon terminal, méchant virus !"
    ;;
*)
    echo "Ravi de vous rencontrer, $nom !"
esac

```

2. Rendre ce script exécutable et l'exécuter plusieurs fois, en donnant les réponses suivantes :

- a) votre nom d'utilisateur ;
- b) Le coronavirus ;
- c) Alice et Bob ;
- d) Moi ;
- e) moi ;
- f) Virus.

3. Modifier votre script, de manière à ce qu'il traite les cas suivants :

- a) Si c'est l'une de vos trois personnalités préférées, le script affiche
Je vous adore <NOM> !
où <NOM> est remplacé par le nom de cette personnalité. Remarque : dans les motifs de la construction **case**, les espaces (et autres caractères spéciaux) doivent être inhibés, par exemple par une contre-oblique.
- b) Si la réponse contient le mot **vendeur** ou **vendeuse**, le script affiche
Désolé, mais je ne suis pas intéressé, au revoir !
- c) Que se passe-t-il si la réponse est **Un vendeur de virus** ?

--- * ---

Exercice 9 :

Écrire le script shell **week-end**. Ce script affiche **Au boulot** : (ou bien **Week-end** :) selon que le jour actuel est un jour entre lundi et vendredi ou bien samedi ou dimanche. Vous utiliserez une construction **case** et une substitution de commande.

--- * ---

6 Scripts shell : construction for

Exercice 10 : Construction **for** et longueur des chaînes

1. Dans un éditeur de texte, écrire le script suivant que vous appellerez **corbeau**

```

#!/bin/sh
total=0
for chaine in Maître corbeau 'sur un arbre' perché \
    tenait 'en son bec' 'un fromage'
do
    printf '%s\n' "$chaine : longueur ${#chaine}"
    total=$(( $total + ${#chaine} ))
done

printf 'longueur totale : %d\n' $total

```

Rendre ce script exécutable et l'exécuter.

2. Modifier ce script de manière à ce qu'à la fin soit affiché :

La dernière chaîne est <chaîne>

où <chaîne> est à remplacer par la dernière chaîne de la liste de la construction **for**.

3. Modifier ce script de manière à ce qu'avant la chaîne, soit affiché chaîne numéro <NUM> : . Par exemple :

chaîne numéro 3 : perché : longueur 6

--- * ---

Exercice 11 : Boum !

Écrire le script **boum** qui produit l'affichage suivant :

```
5
4
3
2
1
BOUM !
```

avec une seconde d'attente entre chaque affichage. On utilisera une construction **for** et la commande **sleep** (voir sa page de manuel).

--- * ---

Exercice 12 : mes_fichiers

Écrire le script **mes_fichiers** qui affiche le nom des fichiers (non cachés) du répertoire courant en les numérotant puis affiche le nombre total de fichiers.

Par exemple, si le répertoire courant contient les fichiers **a.out** **b.c** et **réflexions.txt** le script doit afficher (pas nécessairement dans cet ordre)

```
fichier numéro 1 : a.out
fichier numéro 2 : b.c
fichier numéro 3 : réflexions.txt
Nombre total de fichiers : 3
```

Indice : **for** fichier **in** *

--- * ---

Exercice 13 : type_fichiers

1. Écrire le script **type_fichiers** qui, pour chaque fichier (non caché) du répertoire courant, affiche le nom du fichier, suivi de son type :

- Fichier source C si l'extension est **.c**
- Fichier HTML si l'extension est **.htm** ou **.html**
- Image JPEG si l'extension est **.jpg** ou **.jpeg** ou **.JPG** ou **.JPEG**
- Type de fichier inconnu dans les autres cas.

Par exemple, si le répertoire courant contient les fichiers **a.c** **b.JPG** **index.htm** et **lolcats.gif** le script doit afficher (pas nécessairement dans cet ordre)

```
a.c: Fichier source C
b.JPG: Image JPEG
index.htm: Fichier HTML
lolcats.gif: Type de fichier inconnu
```

2. Enfin, modifier votre script de manière à ce qu'à la fin soit affiché le nombre de fichiers de chaque type (C, JPEG, HTML et inconnu).

--- * ---