Introduction aux systèmes d'exploitation (IS1)

TP 2 – Organisation du système de fichiers

Téléchargez sur DidEL l´archive `arborescence_TP2.tar` et désarchivez-la dans votre répertoire `~/Cours/IS1`. Placez-vous dans ce répertoire.

Exercice 1 – *consulter d'un coup toute une arborescence*

1. Listez les fichiers et sous-répertoires contenus dans le répertoire `~/Cours/IS1/TP2`, en déterminant la nature de chaque élément.

   « `du` **» (*disk usage*)** affiche des informations concernant la place occupée sur le disque par une arborescence

2. Quelles options de « `du` » permettent de déterminer, avec l'affichage le plus lisible possible, quel sous-répertoire de `~/Cours/IS1/TP2` occupe le plus d'espace sur le disque ?
3. Le caractère « `*` » peut être utilisé dans une ligne de commande pour remplacer n'importe quel morceau (sans caractère « / ») de nom de fichier. Comment l'utiliser pour simplifier la ligne de commande précédente ?
4. En une seule ligne de commande, listez *tous* les fichiers et sous-répertoires situés 2 niveaux en dessous de `~/Cours/IS1/TP2`.
5. En une seule ligne de commande, listez *tout* le contenu de l'arborescence de racine `~/Cours/IS1/TP2`.

Exercice 2 – *chemin relatif, chemin absolu*

En utilisant la commande « `cat` » depuis le répertoire `~/Cours/IS1`, affichez de trois façons différentes le contenu du (seul) fichier de l'arborescence `TP2` dont le nom de base est `milou`.

Déplacez-vous ensuite dans `~/Cours/IS1/TP2/Tutu/Toto` et recommencez.

Placez-vous enfin dans un répertoire où la commande `cat ../Personnages/milou` s'exécute sans erreur (et vérifiez).

Exercice 3 – *fichiers cachés*

Malgré les apparences, le répertoire `~/Cours/IS1/TP2/Duck/CoffreDePicsou` contient plusieurs fichiers. Comment s'appellent-ils ?

Exercice 4 – *modifier d'un coup toute une arborescence*

1. Copiez, en une seule ligne de commande, le répertoire `~/Cours/IS1/TP2/Tintin` (et toute l'arborescence en dessous) sous le nom `~/Cours/IS1/TP2/Sauvegarde`.
2. Vérifiez que la structure de l'arborescence que vous venez de créer est la bonne.
3. La commande « `diff` » permet de comparer le contenu de deux fichiers. Vérifiez que le fichier dont le nom de base est `haddock` a bien été copié.
4. Une option de « `diff` » permet de comparer d'un seul coup deux arborescences. Vérifiez que tout s'est réellement bien passé.
5. Supprimez le répertoire `~/Cours/IS1/TP2/Sauvegarde/Lieux/Reels` et vérifiez que « `diff` » détecte la modification.
6. Supprimez toute l'arborescence `~/Cours/IS1/TP2/Toto`, toujours en une seule ligne de commande. Vérifiez que `~/Cours/IS1/TP2/Toto` n'existe plus.
7. Créez, toujours en une seule ligne de commande, un répertoire `~/Cours/IS1/TP2/Toto/Tutu/Titi`.

Exercice 5 – *liens et inœuds*

1. Affichez les numéros d'inœud des fichiers et sous-répertoires de `~/Cours/IS1/TP2`.
2. Renommez `~/Cours/IS1/TP2/grosminet` en `sylvestre` dans le même répertoire. Qu'est devenu le numéro d'inœud ?
3. Copiez `~/Cours/IS1/TP2/sylvestre` sous le nom `junior` dans le même répertoire. Comparer les numéros d'inœud.
4. Affichez les numéros d'inœud des fichiers contenus dans `~/Cours/IS1/TP2/Duck`. Que remarquez-vous ?

   « `ln` **» (link)** permet de créer un nouveau nom (lien) vers un inœud existant : si *ancien* est un nom valide et *nouveau* un nouveau nom, `ln ancien nouveau` crée un nouveau lien de nom *nouveau* vers le même inœud que *ancien*.

5. Créez dans `~/Cours/IS1/TP2/Duck` un nouveau lien `loulou` vers le même inœud que `riri`.
6. Copiez `riri` sous le nom `gontran` dans le même répertoire.
7. Affichez le contenu des fichiers `riri`, `fifi`, `loulou` et `gontran`.

8. On peut modifier le contenu d'un fichier *fic* pour y placer le texte *nouveau_contenu* par la commande suivante :

```
echo nouveau_contenu > fic
```

Replacez le contenu du fichier `riri` par le texte « *neveu de Donald* » de cette manière.

9. Affichez à nouveau le contenu des fichiers `riri`, `fifi`, `loulou` et `gontran` Expliquez.

10. Supprimez `riri` et listez le contenu de `~/Cours/IS1/TP2/Duck`. Expliquez. Pouvez-vous afficher le contenu de `riri` ? de `fifi` ? Comment recréer `riri` exactement à l'identique ?

Personnaliser son environnement

Certaines commandes sont plutôt longues à taper, notamment lorsqu'il y a des options. Une commande permet de pallier cet inconvénient en créant ses propres commandes.

« `alias` »
- avec un argument de la forme *ma_commande=commande_complète*, définit (ou redéfinit) la commande « `ma_commande` ».
  Attention, si *commande_complète* contient des espaces (par exemple, s'il y a des options), il faut alors la délimiter avec des guillemets.
- sans argument, liste tous les raccourcis qui ont été définis.

Exercice 6 – *définir ses propres commandes*

Créez une commande « `la` » qui liste tous les fichiers d'un répertoire (y compris les fichiers cachés commençant par un point).

Exercice 7 – *changer une commande*

Si *ma_commande* existe déjà, son comportement est redéfini et remplacé par celui décrit ; cela permet en particulier de rajouter systématiquement des options à une commande.

1. Par défaut, « `rm` » ne demande pas de confirmation lorsque vous tentez de supprimer un fichier. Ceci peut se révéler assez dangereux. Trouvez l'option qui permet de demander confirmation, puis changez le fonctionnement de la commande « `rm` » pour qu'elle demande systématiquement confirmation lors d'une suppression.

2. Par défaut, le résultat de la commande « `ls` » est un peu aride. Une option permet de colorer l'affichage pour repérer plus rapidement le type de fichiers présents dans un répertoire. Changez le fonctionnement de la commande « `ls` » afin d'utiliser systématiquement l'affichage coloré.

Exercice 8 – *effectivité des changements*

Testez les raccourcis que vous venez de créer. Puis ouvrez un nouveau terminal et testez-les dans celui-ci. Que constatez-vous ?

Le problème de la commande « `alias` » est qu'elle n'agit que dans le shell avec lequel on est en train d'intéragir : dès qu'on se déconnecte, tout est à refaire. Pour stocker ce genre de paramètres une bonne fois pour toutes, on utilise des *fichiers de configurations* liés au shell – `.bash_profile` et `.bashrc` pour le shell `bash`.

Ce sont des fichiers texte contenant des commandes qui sont lus au moment de l'ouverture d'un nouveau shell. Tout changement dans ces fichiers ne sera donc pris en compte que si on relance le shell (via l'ouverture d'un nouveau terminal par exemple), ou si on force leur lecture par la commande « `source` » (qui prend en argument le nom du fichier à lire).

Exercice 9 – *fichiers de configuration*

Éditez le fichier `.bashrc` situé dans votre répertoire personnel [1] avec l'éditeur « `xemacs` » pour rajouter les commandes des exercices précédents (« `la` », « `ls` » avec couleur, « `rm` » avec confirmation). Vous pouvez également redéfinir « `cp` » et « `mv` » pour qu'elles demandent confirmation en cas d'écrasement du fichier cible.

Vérifiez que les changements sont effectifs à l'ouverture d'un nouveau terminal, et dans les anciens seulement après exécution de « `source .bashrc` ». Mais faites attention à ne pas effacer de fichiers importants si vous voulez tester votre nouvelle version de « `rm` » !

1. ou bien créez-en un nouveau s'il n'existe pas

Manipuler des ensembles de noms de fichier

les *jokers* ou *wildcards* permettent de désigner plusieurs noms de fichier à la fois :

« * » représente n'importe quelle suite de zéro, un ou plusieurs symboles (sauf le « . » en début de mot) ;

« ? » représente exactement un symbole quelconque (sauf le « . » en début de mot ;

« [] » représente exactement un symbole parmi l'ensemble de symboles décrit entre les crochets :
- soit par liste exhaustive, par exemple `[abz]` ;
- soit par intervalle avec le symbole « - », par exemple `[a-f]` ou `[2-7]` ;
- soit par complément avec le symbole « ! », par exemple `[!wxk]`.

Exercice 10 – *filtrer l'affichage*

Affichez la liste de tous les noms de fichier dans le répertoire `/usr/bin`...

1. qui commencent par un `k` ;
2. qui contiennent un `k` ;
3. qui terminent par un `k` ;
4. qui commencent par un `m` et terminent par un `e` ;
5. dont la troisième lettre est un `k` ;
6. qui commencent par un `k` et contiennent exactement 6 caractères ;
7. qui commencent par un `k` et contiennent au moins 6 caractères ;
8. qui commencent par un `k` et contiennent entre 3 et 5 caractères ;
9. qui contiennent un `j` ou un `y` ;
10. qui contiennent un chiffre ;
11. qui contiennent un caractère non alphabétique ;
12. qui contiennent un `k` et un `m` ;
13. qui contiennent un `k` et commencent par une lettre parmi `r`, `s`, `t`, `u`, `v`, `w`, `x` ;
14. qui contiennent un `k` et ne commencent pas par un `m` ;
15. qui contiennent un `k` et ne commencent pas par une lettre parmi `l`, `m`, `n`, `o`, `p`.

Utiliser l'historique

Il arrive parfois que l'on ait à utiliser une même commande plusieurs fois, ou que l'on souhaite corriger une commande tapée précédemment. À cette fin, sur la plupart des *shells* modernes il existe un ensemble de « raccourcis » permettant d'accéder à l'historique des commandes.

Exercice 11 – *naviguer dans l'historique*

En utilisant les flèches haut et bas, on peut faire défiler les commandes précédentes, de la plus récente à la plus ancienne. Utilisez ce mécanisme pour tester les commandes de l'exercice précédent dans d'autres répertoires.

Avec les flèches gauche et droite, vous pouvez également corriger un détail de chaque ligne de commande, ce qui peut être plus rapide que de retaper une longue commande presque identique. Les compositions de touches `Ctrl-a` et `Ctrl-e` permettent de repositionner le curseur au début ou à la fin de la ligne. `Ctrl-k` permet de supprimer la fin de la ligne (au-delà du curseur).

Exercice 12 – *chercher dans l'historique*

La commande `history` affiche une liste des commandes récentes, chacune précédée d'un numéro. En tapant `!`n où n est un numéro dans l'historique, on rappelle la commande correspondante.

La composition de touches `Ctrl-r` permet de faire une recherche dans l'historique.