TP N° 3 - Shell bash

Vous rédigerez un compte rendu, sur lequel vous indiquerez la réponse à chaque question, vos explications et commentaires (interprétation du résultat), et le cas échéant la ou les commandes utilisées.

EXERCICE 1 - *méta-caractères*

Créez un repertoire nommé `html` et placez-vous à l'intérieur. Avec la commande `touch` (qui créé des fichiers vides, voir `man touch`) et en utilisant les *méta-caractères*, créez 100 fichiers nommés :

```
00.html, 01.html, 02.html, ..., 09.html
10.html, 11.html, 12.html, ..., 19.html
...
90.html, 91.html, 92.html, ..., 99.html
```

(Vous devez taper une toute petite commande qui tiendra largement sur une ligne.)

Répétez l'opération pour faire la même chose mais avec l'extension `.xml` au lieu de `.html`. Vous aurez donc 200 fichiers au total dans votre répertoire `html`.

1- Effacez les 200 fichiers (en une seule commande, en évitant les confirmations) et recréez-les (les 200 à la fois) avec *une seule* commande `touch` (toujours très courte).

2- Listez tous les fichiers :

1. commençant par 0 ou par 1 (c'est-à-dire les 20 premiers) ;
2. dont le deuxième caractère est soit 7 soit 8 soit 9 ;
3. dont le deuxième caractère est soit 7 soit 8 soit 9 et l'extension est `xml`.

Recommencer en spécifiant un intervalle (de 7 à 9).

3- Effacez tous les fichiers `html` dont le nom contient un zéro (comme premier et/ou comme second caractère).

Suggestion : faites vos essais avec `ls` puis, une fois que vous aurez trouvé le motif correct, utilisez `rm` (toujours en évitant les confirmations).

EXERCICE 2 - *Commandes avec variables*

L'option `-i` de la commande `ls` permet d'afficher le numéro d'inode de chaque fichier (l'inode est une structure de donnée associée à chaque fichier par le système).

1- Affecter à une variable `INODES` la liste des inodes (séparés par des blancs) des fichiers présents dans le répertoire courant.

Remarque : lorsque les champs d'une ligne sont délimités par un nombre variable de blancs et/ou tabulations, vous pouvez utiliser le filtre `tr` avec l'option `squeeze` pour régulariser ces délimiteurs (en traduisant plusieurs délimiteurs consécutifs en un seul).

2- Essayez d'obtenir le même résultat par la commande `awk`.

(Par exemple, pour extraire le champ n°5 d'un ensemble de lignes, la commande à utiliser serait : `awk '{print $5}'`.)

EXERCICE 3 - *script*

Créer un script `test-fichier`, qui précisera le type du fichier passé en paramètre, ses permissions d'accès pour l'utilisateur

Exemple de résultats :

```
Le fichier /etc est un répertoire
"/etc" est accessible par root en lecture écriture exécution

Le fichier /etc/smb.conf est un fichier ordinaire qui n'est pas vide
"/etc/smb.conf" est accessible par jean en lecture.
```

EXERCICE 4 - Afficher le contenu d'un répertoire

Écrire un script bash `listedir.sh` permettant d'afficher le contenu d'un répertoire en séparant les fichiers et les (sous)répertoires.

Exemple d'utilisation :

```
$ ./listdir.sh
```

affichera :

```
--------------   Fichiers dans /etc/rc.d --------------------
rc
rc.local
rc.sysinit
--------------   Repertoires dans /etc/rc.d --------------------
init.d
rc0.d
rc1.d
rc2.d
rc3.d
rc4.d
rc5.d
rc6.d
```

EXERCICE 5 - Lister les utilisateurs

Écrire un script bash affichant la liste des noms de login des utilisateurs définis dans `/etc/passwd` ayant un UID supérieur à 500.

Indication : `for in $(cat /etc/passwd)` permet de parcourir les lignes du dit fichier.

EXERCICE 6 - lecture au clavier

La commande bash `read` permet de lire une chaîne au clavier et de l'affecter à une variable. exemple :

```
echo -n "Entrer votre nom: "
read nom
echo "Votre nom est $nom"
```

La commande `file` affiche des informations sur le contenu d'un fichier (elle applique des règles basées sur l'examen rapide du contenu du fichier).

Les fichier de texte peuvent être affiché page par page avec la commande `more`.

1- Tester ces trois commandes ;

2- Écrire un script qui propose à l'utilisateur de visualiser page par page chaque fichier texte du répertoire spécifié en argument. Le script affichera pour chaque fichier texte (et seulement ceux là) la question "voulez vous visualiser le fichier machintruc ?". En cas de réponse positive, il lancera `more`, avant de passer à l'examen du fichier suivant.

EXERCICE 7 - itération, chaînes de caractères, expressions

1- On a un répertoire peuplé de fichiers dont les noms sont de la forme `dcp_1234.jpg` ou `DCP_1234.JPG`, ou encore `DCP_1234.jpg`, etc, où 1234 est une suite de chiffres quelconques.

Écrire un shell script qui renomme tous ces fichiers, pour obtenir `photo_124.jpg` (toujours en minuscules). Les script prendra les noms des fichiers à traiter en argument sur la ligne de commande.

2- On peut lancer un émulateur de terminal coloré avec la commande

```
xterm -bg nom_de_couleur
```

On crée un fichier `colors.txt` contenant des noms de couleurs standards (voir par exemple `/usr/X11R6/lib/X11/rgb.txt`), un nom par ligne.

La variable spéciale de bash `$RANDOM` permet de générer un nombre entier aléatoire entre 0 et 32767 (voir `man bash`).

Écrire un script qui ouvre une fenêtre terminal avec un couleur de fond aléatoire, choisie dans la liste `colors.txt`.

Ajouter un bouton à la barre d'outils qui lance un terminal coloré via votre script. Tester.

EXERCICE 8 - *Boucles for, &&, ||, if-then-else, case*

Utilisez les 200 fichiers (100 html et 100 xml) créés au cours du TP précédent (si vous ne le avez plus, recréez-lez).

1- Éliminez tous les fichiers `.html` dont le nom contient un chiffre dans l'ensemble 0,1,2, et tous les fichiers `.xml` dont le nom contient un chiffre dans l'ensemble $7, 8, 9$ (en utilisant `rm` et les méta-caractères).

2- En utilisant une boucle `for`, créez un script qui renomme (avec `mv`) l'extension .html en .xml si le fichier .xml correspondant n'existe pas et, vice-versa, transforme l'extension .xml en .html si le fichier html correspondant n'existe pas.

Suggestion : utiliser la commande `basename`.

1. Donner une solution qui utilise seulement `&&` et/ou `||` ;
2. donner une solution qui utilise if-then-else ;
3. donner une solution qui fabrique une variable `CAS` telle que `CAS=html` s'il n'y a pas un correspondant xml, telle que `CAS=xml` s'il n'y a pas un correspondant html, et telle `CAS=html-xml` si les deux existent, puis exécuter l'action correspondante par une instruction `case`.

3- En utilisant une boucle `for`, créez un script qui renvoie la concaténation de tous les fichiers (normaux) présents dans le répertoire courant.

EXERCICE 9 - Ecrire un script qui découpe un fichier en autant de morceaux que nécessaire pour que chacun des morceaux rentre dans une disquette (1,4 Mo) :

```
decouper <fichier>
```

Exemple :

```
decouper toto.zip
```

construira des fichiers `toto.zip.aa`, `toto.zip.ab`, `toto.zip.ac` tels que la commande

```
cat toto.zip.* >toto.zip.toutneuf
```

reconstitue exactement le fichier d'origine.

Note : pour contruire un fichier `toto.zip` sur lequel faire vos tests, vous pouvez faire, par exemple :

```
tar cf - /bin/ | zip toto -
```

(commande qui construit une archive contenant tout le repertoire `/bin` ; cette archive est passée en entrée à la commande `zip` qui crée le fichier compressé, nommé `toto.zip`).

EXERCICE 10 -

1- Écrire une fonction `backup` qui sauvegarde une *liste de répertoires* se trouvant dans un fichier dont le nom est donné en argument.

La sauvegarde est faite dans une archive tar zippée :

```
backup <fichier> <destination>
```

Exemple :

```
backup liste_rep toto.zip
```

Les répertoires seront indiqués ligne par ligne dans le fichier en question. Par exemple, le contenu du fichier

```
liste_rep
```

pourrait être :

```
Desktop/
.kde/
bin
.gnome
```

Indication : on pourra utiliser substitution de commande (i.e. `$(commande)`) pour transformer le contenu du fichier en une liste de répertoires séparés par des blancs.

2- Écrire un script exécutable utilisant le deux fonctions précédentes pour permettre à l'utilisateur de sauvegarder dans un jeu de disquettes le contenu d'un ensemble de répertoires :

```
tofloppy <fichier>
```

Exemple :

```
tofloppy liste_rep
```

Le script construira premièrement une archive tar zippée (cf. fonction backup), il la découpera en plusieurs morceaux (cf. fonction decouper), puis il demandera à l'utilisateur d'insérer une par une les disquettes nécessaires à la sauvegarde :

```
Insérez la disquette n°1 et tapez <entrée>

sauvegarde en cours...
```



```
Insérez la disquette n°2 et tapez <entrée>

sauvegarde en cours...

...

Sauvegarde terminée, merci d'avoir utiliser les
fabuleux services (gratuits) de ce script de sauvegarde!
```

Pour écrire sur une disquette vous choisirez entre faire un montage sur le système de fichier (commande `mount`), puis une copie (commande `cp`), ou bien copier octet par octet (sans montage) le fichier dans le floppy par la commande `dd if=fichier of=/dev/fd0`.

Pour conclure l'exercice, placez le script tofloppy dans le répertoire des binaires personnel (`~/bin`). Assurez-vous que la variable `PATH` contienne ce répertoire (sinon modifiez-la, en l'exportant), et testez le script en l'appelant simplement par son nom à partir d'un autre répertoire.

3- Écrire le script `fromfloppy` faisant le travail inverse de `tofloppy`, c'est-à-dire faisant la restauration de tous les répertoires présents dans un jeu de disquettes :

```
Insérez la disquette n°1 et tapez <entrée>

Autre disquette (o/n) ? o

Insérez la disquette n°2 et tapez <entrée>

Autre disquette (o/n) ? n

Tous vos répertoires seront restaurés à partir
du répertoire courant. Confirmez (o/n) ? o

Restauration terminée.
```