TP n°4 : Scripts Shell

Exercice 1. Promptitude
L'invite de commande (`prompt`) du shell `bash` est définie à travers les variables d'environnement `PS1, PS2, PS3, PS4` et `PROMPT_COMMAND`. Lire leur définition ainsi que la section `PROMPTING` du manuel de `bash`.

Écrire un script qui, lorsqu'on le source avec la commande . (ou source), modifie l'invite de commande pour qu'elle affiche entre crochets la date suivie du nom d'utilisateur et du nom du répertoire courant, le tout suivi d'un prompt $.

Exercice 2. Aussi `seq`
Écrire un script `wet` qui accepte de un à trois arguments entiers. Dans le cas où le nombre d'arguments est incorrect, le script affiche un message d'aide à l'utilisation.

- avec 1 argument `n`, le script affiche les entiers de 1 à `n` ;
- avec 2 arguments `m` et `n`, le script affiche les entiers de `m` à `n` ;
- avec 3 arguments `m, k` et `n`, le script affiche les entiers de `m` à `n` par pas de `k`.

Voici un exemple d'utilisation :

```
$ wet 2 5 28
2
7
12
17
22
27
```

Exercice 3. Le juste prix
Écrire un script `justeprix` qui calcule le coût total des ingrédients nécessaires à la réalisation d'une recette à partir d'un fichier `recette` et d'un fichier `prix` donnés en arguments. On suppose que les fichiers sont formatés comme les exemples contenus dans le répertoire `data/cuisine`. Sur ces exemples, votre script doit se comporter ainsi :

```
$ ./justeprix recette prix
4.7090
```

Il pourra être utile pour le faire de redécouvrir à l'aide du manuel l'utilisation des commandes shell `head, tail, sort, join` et `bc`.

Exercice 4. Fichiers

1. Un fichier est tel que chaque ligne contient un nombre (ou un mot). Écrire un script `bash` en passant le nom du fichier en paramètre pour savoir s'il y a un doublon dans le fichier. Dans le cas de doublons, donner le nombre/mot qui en a le plus.
2. Deux fichiers contiennent les coordonnées de points 3D. Il est possible qu'ils contiennent exactement les mêmes points mais pas nécessairement sur la même ligne. Comment détecter ce cas ? Écrire le script correspondant.

Exercice 5. Lignes Tatin
Écrire un script `tatin` qui prend son entrée soit dans un fichier dont le nom est passé en argument, soit sur son entrée standard s'il ne reçoit pas d'argument, et qui écrit sur la sortie les lignes de l'entrée avec tous les mots dans l'ordre inverse de leur apparition dans la ligne. Par exemple :

```
$ ./tatin << EOF
! tarte la de pas C'est
: galactique voyageur du guide le dit le Comme
>> towel. a carry and Panic Don't <<
EOF
C'est pas de la tarte !
Comme le dit le guide du voyageur galactique :
<< Don't panic and carry a towel.
```

Exercice 6. Fichiers conteneurs

Un fichier conteneur est un fichier texte dans lequel apparaît le contenu d'autres fichiers, dits inclus. Le fichier `tp4/conteneur` est un exemple de conteneur qui contient deux fichiers inclus : `lapin.txt` et `hello`. On notera qu'un fichier inclus commence et termine par `### BEGIN` et `### END` suivis du nom du fichier inclus et que chaque ligne est précédée des deux caractères # et espace.

1. Écrire un script `extrait` qui, étant donné le nom d'un conteneur et d'un fichier inclus, recrée le fichier inclus avec son contenu :

   ```
   $ cat hello
   cat: hello: No such file or directory
   $ ./extrait conteneur hello
   $ cat hello
   #!/bin/bash
   echo Bonjour le monde
   ```

2. Écrire un script `insere` qui, étant donné le nom d'un conteneur et d'un fichier inclus, met à jour dans le conteneur le contenu du fichier inclus (l'ancien contenu est effacé). Si le fichier n'est pas encore inclus dans le conteneur, il est inséré en fin de fichier.

Exercice 7. Dans un répertoire `notes` chaque fichier contient les notes d'un module, par exemple `3IF02.csv, 3IF04.csv`, etc. Dans chaque fichier chaque ligne est du format `nom étudiant; note du module`.

1. Ecrire un script shell qui affiche le nom du module suivi par le plus grand nombre d'étudiants.
2. Ecrire un script shell qui affiche pour chaque module la note la plus élevée et la note la plus faible.
3. Ecrire un script shell qui crée un seul fichier `Total.csv` pour tous les modules. Chaque ligne du fichier est du format `nom étudiant ; module ; note du module`. Le fichier Total.csv doit être organisé de sorte que toutes les notes d'un même étudiant soient en lignes consécutives.

Exercice 8. Gestion d'une base de contrôle d'accès

Sous Unix, la base de contrôle d'accès, qui définit quels sont les utilisateurs, les groupes d'utilisateurs et qui permet de gérer l'accès aux ressources est répartie traditionnellement entre plusieurs fichiers stockés dans le répertoires `/etc`. Dans cette série d'exercices, nous vous proposons d'écrire des scripts shell qui reproduisent le comportement d'outils standard du système mais en travaillant sur des fichiers `passwd, group` et `shadow` situés dans le répertoire courant. Vous trouverez dans le répertoire `tp4/etc` des fichiers d'exemple pour faciliter votre travail.

Le fichier `passwd` décrit sur chaque ligne un utilisateur du système sous la forme de plusieurs champs, séparés par le caractère :. Le premier champ contient le login de l'utilisateur. Le second champ contient historiquement le mot de passe de l'utilisateur, de nos jour le symbole `x` indique que le mot de passe se trouve dans le fichier `shadow`. Les troisième et quatrième champs indiquent les `uid` et `gid`, identifiants numériques de l'utilisateur et de son groupe principal. Le cinquième champ décrit qui est l'utilisateur (texte libre). Le sixième champ contient le répertoire racine de l'utilisateur. Enfin, le dernier champ contient le chemin d'accès au programme shell de l'utilisateur.

Le fichier `shadow` décrit sur chaque ligne un utilisateur du système sous la forme de plusieurs champs, séparés par le caractère :. Le premier champ contient le login de l'utilisateur. Le second champ contient le mot de passe crypté de l'utilisateur, ou parfois les symboles * ou ! pour indiquer que l'utilisateur ne possède pas de mot de passe. Le troisième champ contient la date de dernière modification du mot de passe en nombre de jours écoulés depuis le 1er janvier 1970 UTC. Le quatrième champ contient l'âge minimum en nombre de jours pour pouvoir à nouveau changer le mot de passe. Le cinquième champ contient l'âge maximum en nombre de jours avant que le mot de passe soit périmé. Le sixième champ contient le nombre de jours avant expiration à partir duquel le système doit avertir l'utilisateur. Les autres champs resteront vides dans ces exercices.

Le fichier `group` décrit sur chaque ligne un groupe d'utilisateurs du système sous la forme de plusieurs champs, séparés par le caractère :. Le premier champ contient le nom du groupe. Le second champ contient historiquement le mot de passe du groupe, de nos jours `x` indique que celui-ci est dans `gpasswd`. Le troisième champ indique le `gid`, identifiant numérique du groupe. Le quatrième champ est une liste, séparée par des virgules, des utilisateurs qui sont membres du groupe (en plus de ceux pour qui il s'agit du groupe principal).

1. Écrire un script `listuser` qui affiche la liste des login des utilisateurs avec leur description entre parenthèses en les triant par ordre alphabétique du login et termine par le nombre total d'utilisateurs.

   ```
   $ ./listuser
   bin (bin)
   daemon (daemon)
   lapin (Jeannot Lapin)
   lp (lp)
   mail (mail)
   man (man)
   poule (Am Poule)
   root (root)
   sync (sync)
   sys (sys)
   10 users
   ```

2. Écrire un script `groups` qui prend en argument le login d'un utilisateur et affiche la liste des groupes auquel appartient cet utilisateur, en commençant par son groupe principal.

   ```
   $ ./groups lapin
   lapin : lapin adm cdrom staff
   ```

3. Écrire un script `id` qui prend en argument le login d'un utilisateur et affiche l'uid et le nom correspondant, le gid et le nom correspondant ainsi que la liste des groupes auxquels appartient cet utilisateur en donnant leur gid et leur nom, comme sur l'exemple suivant :

   ```
   $ ./id lapin
   uid=54(lapin) gid=54(lapin) groupes=54(lapin),4(adm),24(cdrom),50(staff)
   ```

4. Écrire un script `adduser` qui prend en argument un login, un uid et une description, et qui demande à l'utilisateur de saisir un mot de passe sans l'afficher, puis modifie les fichiers pour y ajouter un nouvel utilisateur comme dans l'exemple suivant :

   ```
   $ ./adduser black 666 "Joe Black"
   Passwd:
   Apr‘es la commande :
   $ grep ’^black’ passwd
   black:x:666:666:Joe Black:/home/black:/bin/bash
   $ grep ’^black’ shadow
   black:$1$666$tgxXRtU2ErowpyzOaeF.01:15511:0:365:7:::
   $ grep ’^black’ group
   black:x:666:
   ```

Indication : pour crypter un mot de passe `monsecret` saisi au clavier, il faut spécifier une valeur de sel (on prendra l'uid) et utiliser soit la commande

```
md5pass monsecret 666
```

soit si elle n'est pas disponible

```
openssl passwd -1 -salt 666 monsecret
```

5. Écrire un script `testlogin` qui demande la saisie d'un login puis d'un mot de passe et qui vérifie si le mot de passe est correct. Si c'est le cas, le programme affiche `Ok` et s'arrête, sinon le programme redemande le mot de passe. Au bout de trois échecs, le programme s'arrête en affichant `Fail`.

```
$ ./testlogin
login: lapin
Passwd:
Invalid password!
Passwd:
Ok
```

Indication : pour retrouver le sel du mot de passe, remarquez que le mot de passe est écrit sous la forme `$1$sel$blabla`.

6. Écrire un script `expiration` qui affiche la liste des utilisateurs dont le mot de passe expire dans moins de 1000 jours avec le nombre de jours avant expiration.

```
$ ./expiration
lapin expires in 363 days.
poule expires in 364 days.
black expires in 365 days.
```