TP 3 : scripts shell

Introduction à l'infrastructure système et réseau

http://lacl.fr/~sivanov/doku.php?id=fr:cours_de_systemes_et_reseaux [*]

sergiu.ivanov@u-pec.fr

Cet énoncé vous demande de concevoir plusieurs scripts shell. Pour un créer un, écrivez votre code dans un fichier `script.sh` (l'extension est facultative) et rendez-le exécutable avec `chmod +x script.sh`. Pour lancer le fichier `script.sh` qui se trouve dans votre répertoire de travail utilisez la syntaxe `./script.sh`.

Rappel ! Certaines des questions proposées pourront apparaître sur les interrogations écrites au milieu ou à la fin du semestre ! N'hésitez donc pas à discuter vos réponses avec le prof.

Exercice 1 : monitoring système

La commande `sleep k` suspend l'exécution d'un script pour `k` secondes.

Écrivez le script `oeil` qui affichera périodiquement la charge du processeur et la taille de mémoire vive disponible, ainsi que l'heure de prélèvement de ces mesures.

Comment feriez-vous pour faire des prélèvements toutes les demi-secondes ?

Exercice 2 : supprimer la cible d'un lien

Écrivez le script `assassin` qui acceptera en argument un lien symbolique et qui supprimera la cible de ce lien sans supprimer le lien lui-même. Faites en sorte que si `assassin` est appelé pour un fichier qui n'est pas un lien symbolique, le script affiche le message « *nom de fichier* n'est pas un lien symbolique ».

Exercice 3 : réimplémenter `xargs -L1`

Écrivez le script `argsninja` qui lancera une commande pour toute ligne de l'entrée standard. Prenons, par exemple, le fichier `liste.txt` avec comme contenu les deux lignes suivantes :

```
quordon
bleut
```

La commande `./argsninja echo "J'ai un" < liste.txt` doit afficher la sortie suivante :

```
J'ai un quordon
J'ai un bleut
```

Exercice 4 : sauvegarde

La commande `read line` lit la ligne suivante de l'entrée standard dans la variable `line`. La commande `read mot1 mot2 reste`, mettra le premier *mot* de cette ligne dans la variable `mot1`, le deuxième mot dans la variable `mot2`, et le reste de la ligne dans la variable `reste`.

Écrivez le script `airbag` qui permettra de sauvegarder des fichiers dans un dossier de sauvegarde. Un fichier d'entrée `liste.conf` pourrait contenir les lignes suivantes :

```
/home/user/backup-target/

/home/user/pictures/  *.png
/home/user/music/     *.ogg
```

[*] Tous les liens sont cliquables.

La commande `airbag liste.conf` doit copier toutes les images PNG du dossier `/home/user/pictures/` dans le dossier `/home/user/backup-target/home-user-pictures/` et tous les fichiers audio OGG du dossier `/home/user/music/` dans le dossier `/home/user/backup-target/home-user-music/`.

Les motifs autorisés pour les noms de fichiers sont les mêmes que ceux qui peuvent être spécifiés dans le prédicat `-name` de la commande `find`.

Exercice 5 : messagerie

Écrivez les scripts `pigeon-envoie` et `pigeon-recoit` qui serviront à envoyer et à réceptionner des messages en local (sur le même ordinateur). On doit pouvoir utiliser les deux scripts de la façon suivante :

```
> echo "Bonjour !" | ./pigeon-envoie
> echo "Vous allez bien !" | ./pigeon-envoie
```

```
> ./pigeon-recoit
Bonjour !
> ./pigeon-recoit
Vous allez bien ?
```

Réflexion. Comment géreriez-vous les situations dans lesquelles plusieurs utilisateurs essaient d'envoyer ou de réceptionner des messages en même temps ? (Ceci est une question de réflexion ; il n'y a pas obligation d'écrire le code que fera la gestion.)

Exercice 6 : messagerie personnalisée

Modifiez les scripts `pigeon-envoie` et `pigeon-recoit` de l'exercice précédent de sorte que l'on puisse envoyer des messages personnalisés : des messages destinés à un utilisateur en particulier. `pigeon-envoie` prendra donc en argument les noms de l'expéditeur et du destinataire ; `pigeon-recoit` prendra en argument le nom de l'utilisateur dont on souhaite voir les messages. Voici un exemple d'échange entre `bob` et `gremlin` :

```
> echo "Bonjour gremlin !" | ./pigeon-envoie bob gremlin
> ./pigeon-recoit gremlin
Message de bob:
Bonjour gremlin !
```

Réflexion. Supposons que les utilisateurs de notre messagerie correspondent aux utilisateurs du système Linux hôte. Comment vous assureriez-vous que les utilisateurs ne lisent que les messages qui leurs ont été envoyés ? (Ceci est une question de réflexion ; il n'y a pas obligation d'écrire le code qui gérera les droits d'accès.)