Examen de 2ème session de 51IF1IS1

19 juin 2009

Durée : 3h00

A lire attentivement avant de commencer

Pour assurer l'anonymat des copies, nous vous demandons de :
- mettre votre nom et votre numéro d'étudiant dans les caches des copies à en-tête prévus à cet effet et de coller ces derniers,
- écrire un code que vous reporterez sur toutes vos copies.

Les seuls documents autorisés sont les documents papiers autres que les livres. Les documents électroniques sont interdits, en particulier les téléphones, ordinateurs, PDA, etc.

Le barême est donné à titre indicatif.

Exercice 1 : Système de Gestion de Fichiers et redirections (3 points)

1. Expliquer la différence en `bash` entre les redirections `1>` et `2>`.
2. Quels sont les effets possibles des lignes de commande suivantes :
   (a) `ls toto titi 1> tata`
   (b) `ls toto titi 2> tata`
3. Quelles sont les modifications apportées au système de gestion de fichiers par la ligne de commande `mv toto titi; mv titi toto`
4. Quel est le rôle de l'`umask` ? Comment s'en sert-on ?
5. Quel intérêt y a-t-il à laisser les droits en exécution et pas en lecture pour les autres utilisateurs sur un répertoire donné ?
6. Que risque-t-on à laisser les droits en écriture pour les autres sur un de ses répertoires ?

Exercice 2 : Processus et petits scripts (4,5 points)

On suppose que notre repertoire courant contienne le fichier toto mais pas le fichier titi.

1. Que font les commandes
   (a) `ls titi 2> tata && ls toto 1>> tata`
   (b) `ls titi 2> tata || ls toto 1>> tata || echo Bonjour`
2. Quels sont les rôles des commandes `bg` et `fg` ?
3. Que fait la ligne de commande `ls -l /etc | grep "^[^-]" | cut -c47-` ?
   Pour rappel les lignes de réponse de la commande `ls -l` sont de la forme :

   ```
   -rw-r--r-- 1 root root    585 Dec 1  2006 yp.conf
   ```

4. Ecrire une commande qui affiche dans l'ordre alphabétique la liste des utilisateurs qui possèdent un fichier dans le répertoire courant, sans répétition.
5. Ecrire un script qui compte la différence entre deux dates, en nombre de jour. Les deux dates seront des arguments du script, à vous de choisir leur format.

Université Paris Diderot L1 Informatique Année 2008 – 2009

Exercice 3 : La commande `find` (5,5 points)

Vous trouverez dans l'annexe, des extraits de la page de manuel de la commande `find`, traduits en français.

1. Que font les lignes de commandes suivantes ?
   (a)
   ```
   find ~ -maxdepth 1 -a -name "*.ml" -print
   ```
   (b)
   ```
   find / -regex ".*\.[jJ][pP][gG]" -exec display {} \;
   ```
   (c)
   ```
   find /home -name "*~" -mmin +240 -exec rm {} \;
   ```
2. En utilisant la commande `find`, écrivez une ligne de commandes listant tous les fichiers réguliers de votre répertoire personnel qui ne vous appartiennent pas.
3. En utilisant la commande `find`, écrivez une ligne de commandes effaçant tous les répertoires vides vous appartenant (et auxquels vous avez accès).
   Faites-en sorte qu'il y ait une demande de confirmation avant l'effacement de chaque fichier.
4. Ecrivez un commande qui compile tous les fichiers java vous appartenant (on se fiera à l'extension du fichier pour connaître son format).
   Faites-en sorte que cette commande n'agisse que sur les fichiers que vous avez le droit de lire.

Exercice 4 : Carnet d'adresses (7 points)

Dans cet exercice, on se propose de développer quelques exécutables `shell` pour un petit répertoire électronique. On suppose que l'on dispose d'un fichier `.phonebook` à la racine de notre répertoire personnel dont les lignes sont de la forme `nom=prenom=numero=email`. Tous les exécutables se trouvent dans le répertoire `~/bin/phonebook`.

1. Quelles sont les commandes pour faire en sorte que tous nos scripts :
   (a) soient exécutables ?
   (b) puissent être lancés sans spécifier leur emplacement depuis n'importe quel répertoire ?
2. Écrire un exécutable `affiche.sh` qui prend en argument une ligne au même format que celles du fichier `.phonebook` et l'affiche. Par exemple, si l'argument est

   ```
   machin=bidule=0123456789=machin.bidule@p7.fr
   ```

   on veut que ce programme affiche

   ```
   surnom : bid
   nom complet : bidule machin
   telephone : 0123456789
   email : machin.bidule@p7.fr
   ```

   (le surnom est constitué des trois premières lettres du prénom).

3. (a) Écrire un exécutable `cherche.sh` qui prend en argument une chaîne de caractères et renvoie toutes les lignes du fichier `.phonebook` qui contiennent cette chaîne.

   (b) Comment faut-il modifier cet exécutable pour que la recherche du motif ne se fasse que dans le nom et pas dans le prénom ? Et si on voulait uniquement les noms commençant par cette chaîne ? Par exemple, on voudrait que la commande `cherche.sh m` renvoie la ligne

   ```
   machin=bidule=0123456789=machin.bidule@p7.fr
   ```

   mais aucune des deux lignes

   ```
   truc=muche=0246813579=trucmuche@p7.fr
   tam=tam=9876543210=tamtam@tomtom.be
   ```

4. On suppose que notre répertoire ne contient des renseignements que sur des amis français ou belges. On sait que les adresses électroniques des français terminent par `.fr` et que celles des belges terminent par `.be`.

   En utilisant les commandes `grep` et `wc -l`, écrire une ligne de commande qui compte les adresses françaises. Écrire une ligne de commande dont la valeur de retour est `0` si et seulement s'il y a autant de belges que de français dans mon fichier `.phonebook`.

5. On suppose que les numéros de téléphone apparaissant dans le fichier `.phonebook` sont précédés des indicatifs de pays (l'indicatif français est le `0033` et l'indicatif belge est le `0032`). Le fichier contient donc des lignes du type

   ```
   truc=muche=00330246813579=trucmuche@p7.fr
   tam=tam=00329876543210=tamtam@tomtom.be
   ```

   Décrire ligne par ligne ce que fait l'exécutable suivant.

   ```
   for ligne in $(cat ~/.phonebook) do
     identifiant=$(echo $ligne | cut -d= -f3 | cut -c1-4)
     pays=$(ligne:${#ligne}-3}
     case $identifiant in
       0033) if test $pays = .de ; then echo erreur $ligne ; fi ;;
       0032) if test $pays = .fr ; then echo erreur $ligne ; fi ;;
       *) echo pas d\'identifiant ;;
     esac
   done
   ```


Annexe : Manuel de la commande `find` (extraits)

`SYNOPSIS`

```
find [chemin...] [expression]
```

`DESCRIPTION`

- `find` parcourt les arborescences de répertoires commençant en chacun des *chemins* mentionnés, en évaluant les *expressions* fournies pour chaque fichier rencontré.
- Le premier argument commencant par '-', '(', ')', ',', ou '!' est considéré comme le début de l'expression, tous les arguments précédents sont considérés comme des chemins à parcourir.
- Tous les arguments ultérieurs sont considérés comme le reste de l'expression régulière.
- Si aucun chemin n'est mentionné, le repertoire en cours sert de point de depart.
- Si aucune expression n'est fournie, find utilise l'expression '-print' par défaut.
- `find` renvoie une valeur de retour égale à 0 si tous les fichiers ont pu etre examinés correctement, et supérieure a 0 si une erreur s'est produite.

`EXPRESSIONS`

- L'expression est constituée d'options, de tests et d'actions, tous ces éléments étant séparés des opérateurs. Quand un opérateur est manquant, l'opération par défaut `-and` est appliquée.

  `OPTIONS`

  - Les options ont toujours un effet global plutôt que de s'appliquer uniquement à leur emplacement dans l'expression. Néanmoins, pour améliorer la lisibilité, il est préférable de les placer au début de l'expression.

    `-daystart`
    Mesurer les temps (avec `-amin`, `-atime`, `-cmin`, `-ctime`, `-mmin`, et `-mtime`) depuis le debut de la journée plutôt que depuis 24 heures.
    `-depth`
    Traiter d'abord les sous-répertoires avant le répertoire lui-même.
    `-help, --help`
    Afficher un message d'aide sur la sortie standard et terminer normalement.
    `-maxdepth n`
    Ne pas appppliquer les tests à des niveaux de profondeur supérieure à n.
    `-mindepth n`
    Ne pas appppliquer les tests à des niveaux de profondeur inférieure à n.

  `ARGUMENTS NUMERIQUES`
  Les arguments numériques peuvent être indiqués comme suit :
  `+n` supérieur à `n`,
  `-n` inférieur à `n`,
  `n` égal à `n`.

  `Opérateurs`
  - `( expr )`
    Force la précédence.
  - `! expr`
    Vrai si `expr` est fausse.
  - `-not expr`
    Comme `! expr`.
  - `expr1 expr2`
    ET (implicite) ; `expr2` n'est pas évaluée si `expr1` est fausse.

- `expr1 -a expr2`
  Comme `expr1 expr2`.
- `expr1 -o expr2`
  OU ; `expr2` n'est pas évaluée si `expr1` est vraie.

`TESTS`

Voici la liste des tests que vous pouvez effectuer sur les fichiers :

`-amin n`
Dernier accès au fichier il y a `n` minutes.

`-anewer file`
Dernier accès au fichier plus récent que la dernière modification du fichier `file`.

`-cmin n`
Dernière modification du statut du fichier il y a `n` minutes.

`-cnewer file`
Dernière modification du statut du fichier plus récente que la dernière modification du fichier `file`.

`-gid n`
Fichier de GID numérique égal à `n`.

`-inum n`
Fichier de numéro d'i-noeud égal à `n`.

`-links n`
Fichier possédant `n` liens.

`-mmin n`
Fichier dont les données ont été modifiées il y a `n` minutes.

`-name motif`
Fichier dont le nom de base (sans les répertoires du chemin d'accès), correspond à l'expression shell `motif`.

`-newer file`
Fichier modifié plus récemment que le fichier `file`.

`-path motif`
Fichier dont le nom complet correspond à l'expression shell `motif`.

`-perm mode`
Fichier dont les droits d'accès correspondent au mode indiqué.

`-regex motif`
Fichier dont le nom complet correspond à l'expression régulière `motif`.

`-size n`
Fichier utilisant `n` blocs de 512 octets sur le disque.

`-type x`
Selon le caractère `x` :
- `b` fichier spécial en mode bloc (avec buffer)
- `c` fichier spécial en mode caractère (sans buffer)
- `d` répertoire
- `p` tube nommé (FIFO)
- `f` fichier régulier
- `l` lien symbolique
- `s` socket

`-uid n`
Fichier dont le propriétaire possède l'UID `n`.

`-used n`

Fichier dont le dernier accès date de `n` jours après la dernière modification de son statut.

`-user toto`
Fichier appartenant à l'utilisateur `toto`.

`ACTIONS`

`-exec cmd \;`
Exécute la commande `cmd`. La chaîne '{}' est remplacée par le nom du fichier en cours de traitement.

`-fprint file`
Ecrit le nom complet du fichier en cours de traitement dans le fichier `file`. Si fichier n'existe pas au démarrage de find, il est créé. S'il existe, il est écrasé. Les noms de fichiers `/dev/stdout` et `/dev/stderr` sont traités de manière spécifique, ils correspondent respectivement aux sorties standard et erreur.

`-ok cmd \;`
Comme -exec mais interroge d'abord l'utilisateur (en utilisant l'entrée standard). Si la réponse ne commence pas par 'y' ou 'Y', la commande n'est pas exécutée, et le test devient faux.

`-print`
Affiche le nom complet du fichier sur la sortie standard, suivi d'un saut de ligne.