Examen Pratique d'Unix, 2008/2009 Printemps

Le 29/04/2009. Durée : 1 heure 30.

Tout document autorisé. Tout dispositif électronique interdit.

Écrivez bien s'il vous plait pour que la lecture ne soit pas ambiguë.
On veut la réponse la plus courte.
Si vous n'avez qu'une partie de la réponse, indiquez-la et expliquez ce qui manque.

1. Donnez une ligne de commande détruisant tous les fichiers dont le nom se termine par `.o` dans le répertoire courant.

2. Donnez une ligne de commande détruisant tous les fichiers dont le nom se termine par `.o` dans le répertoire courant et tout ce qui est au dessous.

3. Donnez une ligne de commande affichant les lignes contenant une affectation dans la variable `toto`. On cherchera dans les fichiers dont le nom contient `_tst_`

4. Donnez 3 lignes de commande complètement différentes détruisant tous les fichiers dont les noms sont indiqués dans le fichier `detruire`. Indiquez les avantages et inconvénients des 3 méthodes.

5. Donnez une ligne de commande affichant `coucou` dans 3600 secondes. Cette ligne de commande doit "rendre la main" et afficher l'invite de commande immédiatement pour que l'on puisse continuer à travailler.

6. Donnez une ligne de commande affichant le nombre de lignes contenant la chaine de caractère `/*XXX*/` dans tous les fichiers du répertoire courant. Cette commande ne doit afficher que le nombre total.

7. Donnez une ligne de commande remplaçant dans tous les fichiers dont le nom se termine par `.c` le mot `t` par le mot `temps`.

8. Donnez une ligne de commande affichant le contenu d'une fichier de la dernière à la première ligne.

9. Donnez une ligne commande affichant un par ligne, sans s'arrêter, les entiers pairs successifs :

   ```
   0
   2
   4
   6
   ...
   ```

10. Donnez une ligne de commande listant tous les couples de fichiers qui ont un contenu identique dans le répertoire courant. Vous ne devez pas lister le fait que les fichiers sont identiques à eux-mêmes.

Une correction

1. Donnez une ligne de commande détruisant tous les fichiers dont le nom se termine par `.o` dans le répertoire courant.

   `rm *.o` : Attention cela essaye aussi de détruire sans succès les fichiers du répertoire courant. Si vous mettez des guillemets ou cotes autour du `*.o` ou de l'étoile cela essayera de détruire UN fichier s'appelant `*.o`

2. Donnez une ligne de commande détruisant tous les fichiers dont le nom se termine par `.o` dans le répertoire courant et tout ce qui est au dessous.

   Diverses possibilités, de la meilleur à la moins bonne :

   - `find . -type f -name '*.o' -print0 | xargs -0 rm`
   - `find . -type f -name '*.o' -exec rm {} \;`
   - `rm $(find . -type f -name '*.o')`

   La commande `rm -r *.o` ne traite que les fichiers `.o` qui sont dans le répertoire courant et pas en dessous.

3. Donnez une ligne de commande affichant les lignes contenant une affectation dans la variable `toto`. On cherchera dans les fichiers dont le nom contient `_tst_`

   Réponse presque ultime : `grep "\btoto[\t ]*=[^=]" *_tst_*`

   Explications :

   - `\b` indique une fin de mot, c'est-à-dire un séparateur ou un début de ligne.
   - `[\t ]*=[^=]` indique que le symbole d'affectation peut être précédé de 0 ou plusieurs tabulations et espaces et il n'est pas suivi d'un autre égal pour que cela ne soit pas un test d'égalité au lieu d'une affectation.

4. Donnez 3 lignes de commande complètement différentes détruisant tous les fichiers dont les noms sont indiqués dans le fichier `detruire`. Indiquez les avantages et inconvénients des 3 méthodes.

   De la meilleur à la moins bonne.

   - xargs -0 rm – <detruire
     L'option `-0` implique que les noms de fichier soient séparés par un code NUL (0) dans le fichier `detruire` au lieu d'un passage à la ligne.
   - `while read I ; do rm -- "$I" ; done <detruire`
     Ne supporte pas les noms de fichier contenant un retour à la ligne.
   - `for I in $(cat detruire) ; do rm -- "$I" ; done`
     Ne supporte pas les noms de fichier contenant un retour à la ligne ou un espace.
   - `rm -- $(cat detruire)`
     Ne supporte pas les noms de fichier contenant un retour à la ligne ou un espace. Ne fonctionne pas s'il y a trop de noms de fichiers.

   L'option – indique que les arguments suivants ne sont pas des options et donc peuvent commencer par un tiret. Cette option n'était pas dans le cours.

5. Donnez une ligne de commande affichant `coucou` dans 3600 secondes. Cette ligne de commande doit "rendre la main" et afficher l'invite de commande immédiatement pour que l'on puisse continuer à travailler.

   `(sleep 3600 ; echo coucou) &`
   la suivante fonctionne car le `&&` est plus prioritaire que le `&`
   `sleep 3600 && echo coucou &`
   La commande suivante **ne fonctionne pas** car le ; est moins prioritaire que le `&` :
   `sleep 3600 ; echo coucou &`

6. Donnez une ligne de commande affichant le nombre de lignes contenant la chaine de caractère `/*XXX*/` dans tous les fichiers du répertoire courant. Cette commande ne doit afficher que le nombre total.

   ```
   grep '/\*XXX\*/' * | wc -l
   ```

   Si l'on ne veut pas voir les erreurs sur les répertoires on ajoute `2>/dev/null` pour les cacher.

   Une solution plus propre (mais longue à taper) est :

   ```
   for F in * ; do if [ -f "$F" -a -r "$F" ] ; then grep '/\*XXX\*/' ; fi ; done
   | wc -l
   ```

   ou (pas dans le cours) :

   ```
   find .  -type f -maxdepth 1 -exec grep '/\*XXX\*/' {} \; | wc -l
   ```

7. Donnez une ligne de commande remplaçant dans tous les fichiers dont le nom se termine par `.c` le mot `t` par le mot `temps`.

   Une solution qui fonctionne dans tous les cas :

   ```
   for I in *.c ; do sed 's/\bt\b/temps/g' <$I >xxx.tmp && mv xxx.tmp $I ; done
   ```

   Attention `<$I >$I` ne fonctionne pas car le fichier est vidé avant d'être lu.

   Si l'on ne connaît pas `\b` on peut faire :

   ```
   sed -r 's/([^a-zA-Z0-9_]|^)t([^a-zA-Z0-9_]|$)/\1temps\2/g'
   ```

8. Donnez une ligne de commande affichant le contenu d'une fichier de la dernière à la première ligne.

   Version simple, mais avec une grosse variable :

   ```
   FICHIER='' ; while read LIGNE ; do FICHIER="$LIGNE
   $FICHIER" ; done ; echo "$FICHIER"
   ```

   Version lente :

   ```
   I=$(wc -l <fichier) ; while [ $I != 0 ] ; do head -$I fichier | tail -1 ; I=$(expr
   $I - 1) ; done
   ```

   Version astucieuse utilisant un fichier intermédiaire numérotant les lignes :

   ```
   I=0 ; while read LIGNE ; do echo "$I $LIGNE" ; I=$(expr $I + 1) ; done | sort
   -r -n | sed 's/^[^ ] //'
   ```

   Il existe plein d'autres possibilités mais elles utilisent des options non vues en cours.

9. Donnez une ligne commande affichant un par ligne, sans s'arrêter, les entiers pairs successifs :

   ```
   0
   2
   4
   6
   ...
   ```

   ```
   I=0 ; while [ 1 = 1 ] ; do echo $I ; I=$(expr $I + 2) ; done
   ```

10. Donnez une ligne de commande listant tous les couples de fichiers qui ont un contenu identique dans le répertoire courant. Vous ne devez pas lister le fait que les fichiers sont identiques à eux-mêmes.

La commande suivante a un bug et imprime les paires en double :

```
for I in *
   do for J in *
      do if [ "$I" != "$J" ]
            then if cmp "$I" "$J" >/dev/null 2>&1
                    then echo "$I" "$J"
                 fi
         fi
      done
   done
```

Pour éliminer le problème simplement (mais pas efficacement), il suffit de comparer les noms de fichiers

```
for I in *
   do for J in *
      do if [ $(expr "$I" '<' "$J") = 1 ]
            then if cmp "$I" "$J" >/dev/null 2>&1
                    then echo "$I" "$J"
                 fi
         fi
      done
   done
```

On peut aussi éviter le test sur les répertoires :

```
for I in *
   do if [ -f "$I" ]
         then for J in *
                 do if [ -f "$J" -a $(expr "$I" '<' "$J") = 1 ]
                       then if cmp "$I" "$J" >/dev/null
                               then echo "$I" "$J"
                            fi
                    fi
                 done
      fi
   done
```

Vous remarquerez que l'on a plus besoin de rediriger la sortie d'erreur.

Voici la version optimale $O(n\ log\ n)$ au lieu de $O(n^2)$ qui n'affiche pas seulement des paires mais des n-uplet de fichiers identiques. Elle utilise des clef de hachage :

```
X=''
md5sum * | sort |
while read A B
    do if [ "$A" = "$AA" ]
          then X="$X $B"
          else echo $X ; AA=$A ; X="$B"
          fi
    done | grep ' '
```