Examen Pratique d'Unix, 2008/2009 Automne

Le 29/10/2008. Durée : 1 heure 30.

Tout document autorisé. Tout dispositif électronique interdit.

Pour chacune des lignes de commandes indiquées sur ce sujet, expliquez en moins de 5 lignes ce que fait la ligne de commande. Sur votre copie, **laissez les questions dans l'ordre**.

Quelques commandes contiennent des pièges : elles font une erreur à l'exécution, ne font pas ce à quoi l'utilisateur s'attend ou ne sont pas les plus courtes. Vous aurez d'autant plus de points que vous déjouez de pièges.

1. `/`
2. `a-5`
3. `PATH=`
4. `ls ls`
5. `rm '*'`
6. `cp x .`
7. `./ls -l`
8. `pwd <x`
9. `echo $X`
10. `touch = 5`
11. `cd /../etc`
12. `grep \\\\1`
13. `(read A ; cat) <x`
14. `TMP=A ; A=B ; B=TMP`
15. `mv [0-9][0-9][0-9] X`
16. `rm $(read A;echo $A)`
17. `pwd | sed 's/\//\\/g'`
18. `[ !  -d Y ] && mkdir Y`
19. `diff a - <a 2>/dev/null`
20. `ps | grep '-[1-9][0-9]*'`
21. `if [ -d X ] ; then mv *.o X ; fi`
22. `find .  -name "* *" -print | args rm`
23. `for I in *.ch ; do grep "$I" "$I" ; done`
24. `find .  -type f -name *.tar -exec gzip {}`
25. `while read A B <nom_prenom ; echo "$B" "$A" ; done`
26. `for I in *' '* ; do mv "$I" $(echo "$I" | sed 's/ /-/') ; done`

Une correction

1. `/`
   Le shell tente de lancer l'exécution de la racine du système de fichier. Un répertoire n'étant pas un programme que l'on peut exécuter, le shell va donc afficher un message d'erreur en conséquence.

2. `a-5`
   Si le shell trouve une fichier exécutable nommée `a-5` dans les répertoires indiqués dans la variable `PATH` alors le programme est exécuté sinon il y a une erreur. L'utilisateur voulait peut-être faire :
   `expr $a - 5`
   ou
   `a=$(expr $a - 5)`

3. `PATH=`
   La variable `PATH` est vidée. Après l'exécution de cette ligne les seules commandes possibles seront les commandes *builtin* du shell comme `cd`, `pwd`, `export`...

4. `ls ls`
   Si `ls` est un fichier dans le répertoire courant cela affiche le mot `ls` (et non le contenu du fichier), si c'est un répertoire cela liste les noms des fichiers non cachés contenus dans ce répertoire si on a les droits, si il n'existe pas on a une erreur.

5. `rm '*'`
   Détruit le fichier nommé `*` du répertoire courant s'il existe et si l'on a le droit en écriture sur le répertoire courant. Si c'est un répertoire il faut ajouter l'option `-r`. Si le fichier est protégé en écriture, une confirmation sera demandée à l'utilisateur, on peut l'éviter en donnant l'option `-f`.

6. `cp x .`
   Copie du fichier nommé `x` du répertoire courant sous le même nom au même endroit. En réfléchissant un peu on s'aperçoit que cette manipulation est dangereuse car quand la commande va commencer à écrire dans le fichier cela va le vider... La commande `cp` refuse de faire la copie.

7. `./ls -l`
   Exécute la commande `ls` se trouvant dans le répertoire courant avec `-l` comme argument.

8. `pwd <x`
   Affiche le nom du répertoire courant sur la sortie standard si un fichier nommé `x` existe. En effet s'il n'existe pas il y a une erreur car on ne peut pas ouvrir le fichier en lecture pour l'affecter à l'entrée standard de `pwd` et la commande `pwd` n'est pas lancée. La commande `pwd >x` aurait écrit le nom du répertoire courant dans le fichier nommé `x`.

9. `echo $X`
   Affiche le contenu de la variable `X` sur la sortie standard. Pour respecter les espaces il faudrait mettre des guillemets : `echo "$X"`
   Cette commande peut provoquer des erreurs si `X` commence par un tiret. Si `X` n'existe pas c'est comme si c'était une chaine de caractères vide.

10. `touch = 5`
    Cela crée les fichiers nommés `=` et `5`. Pour mettre `5` dans la variable `touch` il ne faut pas mettre les espaces.

11. `cd /../etc`
    Le répertoire courant devient `/etc` en effet `/..` est la même chose que `/`

12. `grep \\\\1`
    Affiche les lignes venant de l'entrée standard (car on n'a pas donné de nom de fichier) contenant la chaine de caractère `\1`
    On aurait aussi put écrire `grep '\\1'` ou `grep '[\]1'`
    `grep '\1'` cherche la chaine `1` car le *backslash* annule la signification du caractère suivant.

13. `(read A ; cat) <x`
    Affiche sur la sortie standard le contenu du fichier `x` sans sa première ligne car elle a été stockée dans la variable `A`

14. `TMP=A ; A=B ; B=TMP`
    On affecte les chaînes de caractères `A`, `B` et `TMP` aux variables `TMP`, `A` et `B`
    Pour inverser les contenus des variables `A` et `B` il fallait taper `TMP="$A" ; A="$B" ; B="$TMP"`

15. `mv [0-9][0-9][0-9] X`
    Fonctionnement le plus probable : déplace tous les fichiers dans le nom est formé de 3 chiffres dans le répertoire `X` sans changer leur nom.
    Si `X` n'est pas un répertoire il n'y aura pas d'erreur s'il n'y a qu'un seul fichier à copier.
    Dans tous les autres cas il y aura une erreur.

16. `rm $(read A;echo $A)`
    Une fois la commande lancée interactivement, l'utilisateur doit saisir les noms des fichiers à détruire au clavier sur une seule ligne en les séparant par un espace.
    L'utilisateur ne doit pas donner un premier nom de fichier commençant par un tiret.

17. `pwd | sed 's/\//\\/g'`
    Affiche le nom du répertoire courant en remplaçant les `/` par des \

18. `[ ! -d Y ] && mkdir Y`
    Création d'un répertoire nommé `Y` s'il n'existe pas. Si un fichier nommé `Y` existe il y a une erreur.
    crochet ouvrant c'est la commande `test` qui se passe bien si le résultat de l'expression est vrai.

19. `diff a - <a 2>/dev/null`
    Affiche les différences entre le fichier `a` et l'entrée standard de la commande qui est aussi `a`
    Il n'y a donc aucune différence.
    Si jamais la commande `diff` proteste parce qu'on lui demande quelque chose d'idiot on ne le verra pas car on a dirigé la sortie d'erreur vers la poubelle.

20. `ps | grep '-[1-9][0-9]*'`
    Filtre seulement les lignes contenant un nombre négatif dans la sortie de la commande `ps`
    La présence de `-0 ou -010` ne déclenche pas l'affichage de la ligne.
    Ceci ne fonctionne pas à cause du tiret, il faut faire : `ps | grep -e '-[1-9][0-9]*'`

21. `if [ -d X ] ; then mv *.o X ; fi`
    Si `X` est un répertoire, déplace tous les fichiers dont le nom se termine par `.o` dans `X`

22. `find . -name "* *" -print | args rm`
    C'est la commande `xargs` et non `args`. La commande `find` trouve **à partir** du répertoire courant les fichiers dont le nom contient un espace. La commande `xargs` prend ces noms et les passe à `rm`, malheureusement comme il y a des espaces cela ne fonctionne pas. De plus il y aura un erreur pour les répertoires. La bonne commande est donc :
    `find . -name "* *" -type f -print0 | xargs -0 rm`

23. `for I in *.ch ; do grep "$I" "$I" ; done`
    Affiches les lignes des fichiers dont le nom se termine par `.ch` (et non `.c` ou `.h` car il n'y a pas les crochets) qui contiennent leur propre nom.

24. `find . -type f -name *.tar -exec gzip {}`
    Comprime tous les fichiers de la hiérarchie courante dont le nom se termine par `.tar` avec la commande `gzip`. Ceci ne fonctionne pas car il manque le `\;` terminant le `-exec`. Ceci ne fonctionne pas s'il y a un fichier `.tar` dans le répertoire courant car il manque les guillemets autour de `*.tar` pour empêcher sa substitution par le shell. La bonne commande est :
    `find . -type f -name '*.tar' -exec gzip {} \;`

25. `while read A B <nom_prenom ; echo "$B" "$A" ; done`
    Si l'on ajoute le `do` qui manque, cela fait une boucle infini affichant à chaque fois la première ligne du fichier `nom_prenom` en déplaçant le premier mot de la ligne à la fin de la ligne.
    Si l'on veut faire ce traitement pour toutes les lignes du fichier successivement on doit faire la redirection à l'extérieur de la boucle :
    `while read A B; do echo "$B" "$A" ; done <nom_prenom`

26. `for I in *' '* ; do mv "$I" $(echo "$I" | sed 's/ /-/') ; done`
    Renomme toutes les entités du répertoire courant contenant un espace en remplaçant le premier espace par un tiret. Il manque l'option `g` à `sed` pour remplacer tous les espaces par des tirets.
    Pour résoudre le problème des tirets pris comme option par `mv` on peut écrire :
    `for I in ./*' '* ; do mv "$I" $(echo "$I" | sed 's/ /-/g') ; done`
    Attention, cette commande peut écraser des fichiers existants, par exemple si l'on a les fichiers `a b` et `a-b` le premier va écraser le second.