Examen Pratique d'Unix, 2009/2010 Automne

Le 1/12/2009. 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. ls *
2. cd..
3. ../a
4. rm = *
5. X/Y Z
6. cat >C
7. mv */* .
8. mkdir '/'
9. date >A
10. rm -r X &
11. mkdir '$D'
12. sort AB
13. read B <ls
14. ls -lR | more
15. echo *'*'*'*'*
16. echo <bonjour>
17. echo # coucou #
18. test -d X && mv a X
19. cp 200[89]-0[1-3]-*.log JFM
20. for I in * ; cp $I $I.bak ; done
21. if [5 = +5] then echo egal ; fi
22. ps | (read A ; read A B ; kill $A)
23. sed 's/.*://' </etc/passwd | sort -u
24. while read A ; do echo " "$A" " ; done
25. for I in cat fichier ; do echo $I ; done
26. expr $(wc --bytes </etc/passwd) '/' $(wc --lines </etc/passwd)
27. while read F ; do echo $(expr length $F) "$F" ; done </etc/passwd
28. A="0 1 2 3 4 5 6 7 8 9" ; for I in $A ; do for J in $A ; do echo "$I$J" ; done

Une correction :

1. `ls *`
   L'étoile est remplacée par le shell par la liste de toutes les entités du répertoire courant qui ne sont pas cachées. `ls` va donc afficher le nom des fichiers et le **contenu des répertoires** contenu dans le répertoire courant.

2. `cd..`
   Execute la commande nommée `cd..` sans argument. L'utilisateur a certainement oublié l'espace qui indiquerait qu'il faut que le répertoire courant devienne le père.

3. `../a`
   Le shell lance l'exécution de la commande `a` contenu dans le répertoire père du répertoire courant. Si elle existe elle est lancée sans paramètres, sinon il y a une erreur.

4. `rm = *`
   La commande détruit le fichier nommé '=' puis tous les autres fichiers du répertoire. Elle affiche une erreur si '=' n'existe pas ou s'il y a des répertoires (il n'y a pas l'option `-r`).

5. `X/Y Z`
   Exécute la commande `Y` contenue dans le répertoire `X` du répertoire courant. Elle aura `Z` comme paramètre. Évidemment si la commande n'existe pas il y aura une erreur.

6. `cat` >`C`
   La commande `cat` sans argument lit son entrée standard et l'écrit sur sa sortie standard. Le fichier `C` est donc vidé/créé puis remplit avec tout ce qui est lu sur l'entrée standard. Si c'est le clavier cela sera remplit avec ce que l'utilisateur tape jusqu'à un `ctrl-D`.

7. `mv */* .`
   Déplace tous les fichiers et répertoires contenu dans les répertoires contenus dans le répertoire courant dans le répertoire courant. Ce cela écrase les fichiers qui porte le même nom. Les fichier cachés ne sont pas déplacés.

8. `mkdir '/'`
   / n'est pas un caractère spéciale, donc les cotes ne servent à rien. / est le répertoire racine (c'est un chemin absolu, il ne représente pas quelque chose dans le répertoire courant). La commande ne peut créer le répertoire, il existe déjà. D'autre part, / ne peut apparaître dans un nom de fichier, car ce fichier serait inaccessible A/B représente forcément B dans le fichier A.

9. `date` >`A`
   Le fichier `A` est créé/vidé (erreur si impossible de le faire) la sortie standard de la commande `date` est dirigé vers le fichier et la commande s'éxécute. La date est donc écrite dans le fichier.

10. `rm -r X &`
    Détruit récursivement le fichier/répertoire `X` en arrière plan.

11. `mkdir '$D'`
    Création d'un répertoire nommé `$D` dans le répertoire courant. Ce n'est pas un bonne idée car il faudra systématiquement mettre des cotes pour accéder à ce fichier.

12. `sort $A$B`
    Le shell concatène le contenu de la variable `A` et de la variable `B`. La commande `sort` va trier les lignes du fichier du fichier indiqué (ou des fichiers s'il y a des espaces. Il manque certainement des guillemets.

13. `read B <ls`
    Stocke la première ligne du fichier nommé `ls` dans la variable nommée `B`

14. `ls -lR | more`
    Affiche page par page la hiérarchie contenue dans le répertoire courant avec les détails sur les fichiers.

15. `echo *'*'*'*'*`
    Affiche la liste des entités du répertoire courant qui ne sont pas cachées et dont le nom contient 2 étoiles. Si aucun fichier de ce type n'existe cela affiche `*****`

16. `echo <bonjour>`
    L'entrée standard devient le fichier `bonjour` comme il n'y a pas de nom après la redirection de la sortie standard il y a une erreur et la commande `toto` n'est pas exécutée. Si la commande avait été `echo <bonjour>xxx` Le fichier `xxx` aurait été créé/vidé puis **une retour à la ligne** aurait été stocké dedans, le fichier `bonjour` n'aurai pas été lu car la commande `echo` ne lit pas son entrée standard.

17. `echo # coucou #`
    Cela affiche une ligne sur la sortie standard car `#` est le caractère indiquant que ce qui suit est un commentaire.

18. `test -d X && mv a X`
    Si `X` est un répertoire alors `a` est déplacé dedans sans changer de nom et en écrasant éventuellement un fichier. Si `a` est un répertoire, alors le `mv` reussira seulement si `X/a` n'existe pas. La commande `mv` n'a pas besoin d'option récursive.

19. `cp 200[89]-0[1-3]-*.log JFM`
    Copie les fichiers dont le nom se termine par `.log` et qui commence par 2008-01-, 2008-02-, 2008-03-, 2009-01-, 2009-02-, 2009-03- dans le répertoire `JFM`. S'il n'existe pas il y a une erreur.

20. `for I in * ; cp $I $I.bak ; done`
    Fait une erreur car il manque le `do`. Si on l'ajoute cela copie tous les fichiers du répertoire courant sous un nom identique + `.bak` à la fin. Il manque des guillemets pour que cela fonctionne aussi si le nom des fichiers contient un espace. Et on peut écrire `./*` au lieu de `*` pour que cela fonctionne aussi si le nom du fichier commence par un tiret.

21. `if [ 5 = +5 ] then echo egal ; fi`
    Fait une erreur car il manque un point virgule. Le message ne s'affiche pas car c'est une comparaison de chaines de caractères qui a été faite. Il aurait fallu utiliser `-eq` au lieu de `=` pour que le message `egal` s'affiche.

22. `ps | (read A ; read A B ; kill $A)`
    `ps` liste les processus lancés à partir du terminal. Le premier `read` lit la ligne d'entête. Le second lit les informations concernant le premier processus listé. Le `PID` va dans `A` et le reste de la ligne va dans `B`. Le premier processus listé par `ps` est donc tué. C'est **généralement** un shell dans il ne sera pas tué car il manque l'option `-1`

23. `sed 's/.*://' </etc/passwd | sort -u`
    `sed` lit son entrée standard (le contenu de `/etc/passwd`) et remplace la plus longue chaine de caractères se terminant par deux point, par rien. Il écrit donc ce qu'il y a après le dernier deux point (l'inperpréteur shell utilisé) dans le *pipe*. La commande `sort` va donc afficher la liste des `shell` triée dans l'ordre alphabétique en enlevant les doublons (`-u`).

24. `while read A ; do echo " "$A" " ; done`
    Recopie l'entrée standard au ajoutant un espace au début et à la fin de chaque ligne. **Attention** le `$A` n'est pas entre guillemet, donc si la variable contient de multiples espaces ils seront remplacé par un seul sur la sortie. On aurait du faire `echo " $A " ;`

25. `for I in cat fichier ; do echo $I ; done`
    Affiche sur la sortie standard `cat` puis `fichier`. L'utilisateur voulait peut-être faire :
    `for I in $(cat fichier) ; do echo $I ; done`
    Ici la variable `I` prend comme valeur chacun des mots du fichier.

26. `expr $(wc --bytes </etc/passwd) '/' $(wc --lines </etc/passwd)`
    Calcule le nombre d'**octets** (pas de bits) par ligne du fichier `/etc/passwd`. Les cotes autour du `/` ne sont pas nécessaires.

27. `while read F ; do echo $(expr length $F) "$F" ; done </etc/passwd`
    Affiche le contenu du fichier `/etc/passwd` en précédent chaque ligne de sa longueur. Ceci ne fonctionne pas si il y a des espaces car `expr length` exige un seul paramètre. La bonne syntaxe est : `while read F ; do echo $(expr length "$F") "$F" ; done </etc/passwd`

28. `A="0 1 2 3 4 5 6 7 8 9" ; for I in $A ; do for J in $A ; do echo "$I$J" ; done`
    En ajoutant `; done` à la fin cela afficherai les nombre 00, 01, 02... 98, 99 un par ligne. Les guillemets autour `"$I$J"` sont inutiles mais néanmoins recommandés.