Examen Pratique d'Unix, 2006/2007

Le 13/11/2006. 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 ou ne font pas ce à quoi l'utilisateur s'attend. Vous aurez d'autant plus de points que vous déjouez de pièges.

1. `echo ../*`
2. `cd toto/..`
3. `date >>toto`
4. `rm /dev/null`
5. `PATH='.'  ; ls`
6. `echo 'toto' | wc`
7. `chmod 750 '*.exe'`
8. `cp -r toto/* titi`
9. `rm * ; expr 2 * 5`
10. `tail /usr/include`
11. `echo ls >rm ; rm`
12. `sort <toto 2>titi`
13. `cat toto titi | sort`
14. `rm -r toto & echo ok`
15. `sort -n <toto >toto`
16. `echo $(echo toto ; echo titi)`
17. `for I in *.c ; do cat $I ; done`
18. `expr $(expr 4 + 5) / $(expr 1 + 2)`
19. `find .  -name [A-Z]* -exec mv {} a{} \;`
20. `while read A ; do echo '$A' ; done <toto`
21. `case $(date) in Dim) echo 'Dormir' ;; esac`
22. `PATH=$(echo "$PATH" | sed -e 's/:\/sbin:/:/')`
23. `ps | ( read A ; cat ) | ( cd /tmp ; cat >toto )`
24. `I=1 ; while [ 1 == 1 ] ; do echo $I ; I=$(expr $I \* 2) ; done`
25. `while read A ; do echo "$(expr length "$A") $A";done|sort -n|sed 's/[^ ]* //'`

Une correction

1. `echo ../*` : Affiche la liste des fichiers/répertoires (non cachés pour utilisateur normal) du répertoire père (on considère que la racine est son propre père).

2. `cd toto/..` : Si `toto` est un répertoire, le répertoire courant n'est pas modifié car l'on fait un aller retour. Sinon, un message d'erreur est affiché.

3. `date >>toto` : Si `toto` n'existe pas il est créé. Si `toto` est un fichier accessible en écriture alors on ajoute la sortie standard de la commande `date` à la fin. Sinon on a une erreur et la commande `date` n'est pas lancée.

4. `rm /dev/null` : Bien que tous les utilisateurs puissent écrire dans ce fichier seul le super utilisateur peut le détruire. Le fait de détruire ce fichier ne détruit pas son contenu car il ne contient rien (ce n'est pas un répertoire). Si jamais il est détruit toutes les commandes l'utilisant ne fonctionneront plus sauf si elles sont exécutées par le super utilisateur qui a le droit d'écrire dans `/dev`, par contre un fichier normal sera créé et tout ce qui sera écrit dans `/dev/null` sera écrit sur le disque...

5. `PATH='.' ; ls` : La valeur `.` est stockée dans la variable `PATH` puis le shell va exécuter une commande `ls` se trouvant dans le répertoire courant. S'il n'en existe pas, il affichera par exemple `ls: command not found`

6. `echo 'toto' | wc` : `wc` lit `toto` sur son entrée standard. Il affiche donc `5 1 1` car il y a 5 caractères (fin de ligne), 1 mot et une ligne.

7. `chmod 750 '*.exe'` : Si **le** fichier `*.exe` existe et appartient à l'utilisateur alors son mode devient `rwxr-x---` sinon il y a une erreur. Les apostrophes autour de `*.exe` étaient certainement une erreur de l'utilisateur.

8. `cp -r toto/* titi` : Copie récursive du contenu de `toto` (sans les fichiers cachés du niveau supérieur) dans `titi`. Il y a des erreurs si `toto` n'est pas accessible en lecture ou si `titi` n'est pas un répertoire accessible en écriture.

9. `rm * ; expr 2 * 5` : Efface tous les fichiers du répertoire courant (pas les répertoires) si l'on a les droits d'écriture sur le répertoire courant. Si après cela **le répertoire est vide** alors `expr` affiche 10 car l'étoile reste une étoile. Sinon, suivant les noms des fichiers restant la commande `expr` fera une erreur ou calculera n'importe quoi. Par exemple s'il reste un unique répertoire nommé `-` (moins) elle affichera -3. Il aurait fallu mettre des cotes (par exemple) autour de l'étoile.

10. `tail /usr/include` : `tail` affiche les dernières lignes de fichiers. Comme c'est un répertoire elle fait une erreur.

11. `echo ls >rm ; rm` : Le fichier `rm` du répertoire courant prend (si c'est possible) comme contenu `ls`. Ensuite, si '.' est avant '/bin' dans `PATH` alors le shell fait une erreur car le fichier `rm` du répertoire courant n'est pas exécutable. Sinon, la vrai commande `rm` fait une erreur car elle n'a pas de paramètres.

12. sort <toto 2>titi : Le contenu trié de `toto` s'affiche sur la sortie standard. `toto` n'est pas modifié. Les erreurs de tris sont stockées dans `titi` en effaçant son contenu.

13. `cat toto titi | sort` : La commande `cat` lit le contenu des deux fichiers et l'écrit dans le *pipe*, la commande `sort` fait le tri des lignes lues dans le pipe donc du contenu des deux fichiers.

14. `rm -r toto & echo ok` : Lance la destruction récursive de `toto` en tache de fond et affiche donc immédiatement sans attendre la fin le mot `ok`

15. sort -n <toto >toto : Le shell ouvre `toto` en lecture puis en écriture, ceci vide le fichier `toto`. Quand la commande `sort` essaye de faire le tri numérique elle tri une fichier vide. Le résultat est donc un fichier vide.

16. `echo $(echo toto ; echo titi)` : Les commandes entre parenthèses sont exécutées, elles affichent `toto\ntiti\n`. Ensuite le shell interprète cette chaine comme deux arguments car il n'y a pas de guillemets. Le premier `echo` affiche donc les deux arguments sur **la même ligne**.

17. `for I in *.c ; do cat $I ; done` : Affiche le contenu de tous les fichiers avec l'extension `.c` qui sont dans le répertoire courant. Sauf ceux dont les noms contiennent des caractères spéciaux comme l'espace. S'il n'y a aucun nom `cat` fera une erreur sur le fichier nommé `*.c`

18. `expr $(expr 4 + 5) / $(expr 1 + 2)` : Calcul 9/3 ce qui donne 3. Pas besoin de guillemets autour du / car ce n'est pas un caractère spécial.

19. `find . -name [A-Z]* -exec mv {} a{} \;` : Cherche dans la hiérarchie courante les fichiers/répertoires dont le nom contient une majuscule (si le répertoire courant n'en contient pas car il manque des guillemets ou cotes)autour du `[A-Z]*`. Pour chaque nom trouvé renomme en ajoutant un 'a' devant. Ceci va faire des erreurs car './Toto' ne peut généralement pas être renommé en 'a./Toto' car le répertoire 'a,' n'existe pas.

20. `while read A ; do echo '$A' ; done <toto` : Pour chaque ligne du fichier `toto` on affiche le texte `$A` sur l'écran. Si l'on avait voulu afficher le contenu du fichier il aurai fallu mettre des guillemets au lieu des cotes.

21. `case $(date) in Dim) echo 'Dormir' ;; esac` : Si la date courante est `Dim` cela affiche dormir. Cela n'affichera jamais rien car la date peut contenir `Dim` mais n'est pas `Dim`. La syntaxe correcte était `*Dim*` pour indiquer qu'il y avait n'importe quoi avant et après.

22. `PATH=$(echo "$PATH" | sed -e 's/:\/sbin:/:/')` : Remplace le premier `:/sbin:` par `:` dans la variable `PATH`. Attention ce n'est pas équivalent au remplacement `:/sbin` par rien car il pourrait y avoir un problème si le répertoire `/sbin/truc` existait. Le retour à la ligne final ne pose pas de problème car le `$()` l'enlève automatiquement (je ne suis pas certain que cela soit portable).

23. `ps | ( read A ; cat ) | ( cd /tmp ; cat >toto )` : Le command `ps` liste les processus lancés à partir du terminal. Le bloc suivant permet d'enlever la première ligne (l'entête de la liste) car la première ligne est lue dans `A` mais n'est pas affichée. Le dernier bloc permet de stocker la liste des processus dans le fichier `/tmp/toto`. Le répertoire courant n'a pas changé après l'exécution de cette commande.

24. `I=1 ; while [ 1 == 1 ] ; do echo $I ; I=$(expr $I \* 2) ; done` : Affiche les puissances de 2. `\*` fait la même chose que `'*'` ou `"*"`

25. `while read A ; do echo "$(expr length "$A") $A";done|sort -n|sed 's/[^ ]* //'` : Pour chaque ligne lu ajoute sa longueur devant, fait un tri numérique puis enlève le premier mot de la ligne (la longueur). On obtient les lignes triées par longueur. C'est cette ligne qui a permis de trier les lignes de cet énoncé par longueur.