# Examen Pratique d'Unix, 2009/2010 Printemps

Le 20/4/2010. 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. `$N`
2. `rm \\`
3. `E=E+F`
4. `man cd`
5. `mv B C D`
6. `cd ../..`
7. `ls /etc/*`
8. `cp '*.c' W`
9. `echo 2**32`
10. `mkdir date`
11. `expr 2 * 3`
12. `rm toto .doc`
13. `find / >toto &`
14. `echo *.[!j][!p][!g]`
15. `(read A ; cat) <tutu`
16. `cat [il][fait][beau]`
17. `cat *.c | sort | tail`
18. `echo *.c | xargs echo`
19. `echo titi >tata && cat tata`
20. `find /etc/a* -name *.h -size 1k`
21. `if [ -f K ] ; then rm '$K' ; fi`
22. `for I in $H ; do mv I I.bak ; done`
23. `while read A B ; echo $B $A ; done`
24. `P="5" ; R="+ 2" ; P=$(expr "$P" "$R")`
25. `case $L in [abc]) echo A ;; \*) echo B ;; esac`
26. `cd ; ln -s include truc ; cd /usr ; ls ~/truc`
27. `L=' ' ; M = " " ; if test $L = $M ; then echo = ; fi`
28. `ls | ( read X ; cat ) | ( read Y ; cat ) | read Z ; echo $Z`
29. `for T in * ; do echo $(wc --line <$T) $T ; done | sort --numeric-sort`

Une correction :

1. `$N`
   Ceci exécute la commande shell contenu dans la variable `N`

2. `rm \\`
   Détruit le fichier texte nommé \

3. `E=E+F`
   La chaine de 3 caractères `E+F` est stockée dans la variable `E`. Pour faire l'addition c'est `E=$(expr $E + $F)`

4. `man cd`
   La commande `cd` est une *builtin* il va donc probablement y avoir une erreur, il faut faire `help cd`

5. `mv B C D`
   Déplaces les entités `B` et `C` dans le répertoire `D`. Donc `D/B` et `D/C`. Si ce n'est pas possible il y aura une erreur.

6. `cd ../..`
   Le père du père du répertoire courant devient le nouveau répertoire courant.

7. `ls /etc/*`
   Affiche les noms des fichiers et répertoires contenus dans `/etc` et de ses sous-répertoire directes (à cause de l'étoile). La fichiers cachés ne sont pas affichés.

8. `cp '*.c' W`
   Copie **LE** fichier nommé `*.c` dans `W`. Si `W` était un répertoire cela le copie dedans (`W/*.c`) sinon cela crée ou remplace le fichier `W` du répertoire courant.

9. `echo 2**32`
   Affiche les noms des fichiers du répertoire courant dont le nom commence par `2` et fini par `32`. C'est équivalent à `echo 2*32` (pour le shell standard)
   Si aucun fichier ne correspond alors cela affiche `2**32`

10. `mkdir date`
    Crée un répertoire nommé `date` dans le répertoire courant. Pour créer un repertoire avec la date actuelle il faudrait faire `mkdir "$(date)"`

11. `expr 2 * 3`
    Le calcul sera fait après le remplacement de l'étoile par la liste des fichiers du répertoire courant. Il y a donc toutes les chances d'avoir une erreur de syntaxe. L'utilisateur voulait certainement faire : `expr 2 '*' 3` pour afficher le résultat de la multiplication.

12. `rm toto .doc`
    Supprime les fichiers nommés `toto` et `.doc` du répertoire courant. Si l'espace faisait parti du nom de fichier, il aurait fallu faire `rm 'toto .doc'`

13. `find / >toto &`
    Écrit la liste de tous les noms de fichiers et répertoires à partir de la racine dans le fichier `toto` du répertoire courant. La commande est lancée en arrière plan. Les messages d'erreurs seront affichés sur l'écran car la sortie d'erreur n'a pas été changée.

14. `echo *.[!j][!p][!g]`
    Affiche les noms des fichiers du répertoire courant ne commençant pas par un point et ayant une extension de 3 lettres dont la première n'est pas 'j' la deuxième pas 'p' et la troisième pas 'g'. Attention, `toto.ppm` ne sera pas affiché et `toto.pgj` sera affiché.

15. `(read A ; cat) <tutu`
    Les deux commandes vont lire lire après l'autre dans le fichier `tutu`. La première ligne du fichier est stockée dans la variable `A` (qui disparaîtra à la fin de l'exécution du groupe) et la commande `cat` lit et affiche la suite du fichier car comme elle n'a pas d'argument c'est un filtre. Cette commande affiche donc le fichier `tutu` sans sa première ligne.

16. `cat [il][fait][beau]`
    Affiche le contenu des fichiers qui ont un nom de trois lettres dont la première est 'i' ou 'l', la deuxième est 'f', 'a', 'i' ou 't' et la troisième est 'b', 'e', 'a' ou 'u'.

17. `cat *.c | sort | tail`
    Trie par code ASCII les lignes contenues dans tous les fichiers dont le nom se termine par `.c` dans le répertoire courant et n'affiche que les dernières.

18. `echo *.c | xargs echo`
    `xargs` prend la liste de fichiers écrite par `echo` et lance la commande `echo` qui les affichera. Attention : cela ne fonctionnera pas avec les noms de fichier contenant un retour à la ligne (avec des espaces c'est ok). Et le résultat pourra être sur plusieurs lignes si la liste est trop longue car `xargs` lancera la commande `echo` plusieurs fois.

19. `echo titi >tata && cat tata`
    Affiche `titi` sur la sortie standard si l'on a pu créer et modifier le fichier `tata` du répertoire courant.

20. `find /etc/a* -name *.h -size 1k`
    Affiche les noms des fichiers include de 1ko qui se trouvent dans les hiérarchies qui sont dans `etc` et dont le nom commence par `a`. Cette commande ne fonctionne que s'il n'y a pas de fichier `.h` dans le répertoire courant car `*.h` n'a pas été protégé.

21. `if [ -f K ] ; then rm '$K' ; fi`
    Si le fichier `K` existe alors on détruit le fichier nommé `$K`. L'utilisateur voulait certainement faire une des commandes suivantes :
    `if [ -f '$K' ] ; then rm '$K' ; fi`
    `if [ -f K ] ; then rm K ; fi`
    `if [ -f "$K" ] ; then rm "$K" ; fi`

22. `for I in $H ; do mv I I.bak ; done`
    Essaye de renommer `I` en `I.bak` autant de fois qu'il y a de mot dans la variable `H`. L'utilisateur à oublié les $ et les guillemets afin de renommmer les noms de fichier contenu dans `H` en leur ajoutant un `.bak` à la fin de leur nom.
    `for I in $H ; do mv "$I" "$I.bak" ; done`
    Il ne faut surtout pas de guillemets autour du `$H`

23. `while read A B ; echo $B $A ; done`
    Il manque le `do`. Réécrit l'entrée standard en passant le premier mot à la fin et en remplaçant toutes suite d'espace/tabulation par un seul espace. Pour respecter les espaces il aurait fallu faire : `while read A B ; do echo "$B" $A ; done`
    Mais ceci ne fonctionne pas si la ligne contient un seul mot...

24. `P="5" ; R="+ 2" ; P=$(expr "$P" "$R")`
    Cela fait une erreur car la commande `expr` a comme arguments `5` et `+ 2` qui n'est pas un nombre ni un opérateur. L'utilisateur voulait certainement faire l'addition :
    `P="5" ; R="+ 2" ; P=$(expr "$P" $R)`

25. `case $L in [abc]) echo A ;; \*) echo B ;; esac`
    Affiche `A` si `L` est strictement égale à `a` ou `b` ou `c`.
    Affiche `B` si `L` est strictement égale au caractère `*`

26. `cd ; ln -s include truc ; cd /usr ; ls ~/truc`
Un lien symbolique nommé `~/truc` pointant vers `include` est crée. Le `ls ~/truc` est équivalent à `ls ~/include`

    Attention, si `truc` était un fichier cela fait une erreur et si c'était un répertoire existant, cela crée un lien symbolique nommé `~/truc/include` pointant vers `include`.

27. `L=' ' ; M = " " ; if test $L = $M ; then echo = ; fi`
L'affectation à `M` échoue car il y a des espaces autour du égal. Le `test` échoue car il a un seul paramètre qui est égal et qui n'a pas d'argument (en fait il réussi car cette chaine est non vide, mais cela n'a pas été vu en cours). La *bonne* ligne est donc :
`L=' ' ; M=" " ; if test "$L" = "$M" ; then echo = ; fi`
qui affiche `=`

28. `ls | ( read X ; cat ) | ( read Y ; cat ) | read Z ; echo $Z`
La commande `ls` liste le contenu du répertoire courant.
Le `read` du premier groupe lit la première ligne écrite par `ls` et la commande `cat` lit la suite et l'écrit sur la sortie standard. Le premier groupe affiche donc la sortie de `ls` sans sa première ligne (qui est dans `X`. Le deuxième groupe fait la même chose en stockant dans `Y` la deuxième ligne. Finalement, la troisième ligne est stockée dans `Z`. Malheureusement, dans un pipeline les commandes sont exécutées dans des processus, la valeur de `Z` est immédiatement perdue. La bonne syntaxe est :
`ls | ( read X ; cat ) | ( read Y ; cat ) | (read Z ; echo $Z)`
Ou plus court :
`ls | ( read X ; read Y ; read Z ; echo $Z)`
Pour information : la commande `ls` affiche un nom par ligne quand ce n'est pas affiché sur l'écran. Donc cela affiche le troisième nom de fichier dans l'ordre alphabétique.

29. `for T in * ; do echo $(wc --line <$T) $T ; done | sort --numeric-sort`
Affiche le nombre de lignes et le nom des fichiers du répertoire courant en les triant par nombre de lignes et en affichant des erreurs pour les dossiers et les fichiers dont le nom contient un espace (il manques des guillemets autour du `$T`