Examen Pratique d'Unix, 2007/2008 Printemps

Le 14/05/2008. 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. `ls .`
2. `echo -*`
3. `A=5 + 2`
4. `rm cd rm`
5. `vi $(date)`
6. `ls -R -a /`
7. `mv /bin/ls ..`
8. `cd toto.tar.gz`
9. `man sauvegarde`
10. `cp /bin/sh /tmp`
11. `echo a ; echo b`
12. `rm .  && echo a`
13. `tar -cvf toto.tar`
14. `chmod 640 filename`
15. `read B <toto ; rm $A`
16. `find ~exco -name '\*'`
17. `cp -r .  tous mes fichiers`
18. `echo bonjour | sed -e 's/o/A/'`
19. `cat /etc/a* | grep '->' | sort`
20. `(date ; ls -R / ; date) >liste`
21. `(read C ; read C) </usr ; echo $C`
22. `D= ; while read E ; do D="$D$E" ; done`
23. `grep '^[ \t]*[{}]?[ \t]*$' <*.c | wc -l`
24. `for I in in cat titi ; do cp $I $I.old ; done`
25. `kill $(ps -e | grep truc | read A ; echo $A)`
26. `ps -e | grep truc | while read A ; do kill $A ; done`
27. `expr '(' 100 * $(gzip </bin/sh | wc -c) \) / $(wc -c </bin/sh)`

1. `ls .`
   Liste les noms des entités (fichier, répertoire, ...) non cachées directement contenues dans le répertoire `.` qui est le répertoire courant. Il est plus court de taper `ls` sans paramètre.

2. `echo -*`
   `-*` est remplacé par la liste des noms d'entités commençant par `-` qui sont contenues dans le répertoire courant `.` Il est inchangé s'il n'y a aucun nom correspondant. Normalement la commande `echo` affiche ses paramètres, mais ici elle fera certainement une erreur car les noms des fichiers seront pris comme des options car ils commencent par `-`.

   Il faudrait taper `echo ./-*` pour afficher la bonne liste. Attention `echo '-*'` fait systématiquement une erreur car `*` n'est pas une option de `echo`. **N'oubliez pas, le `-` n'est pas un caractères spécial pour le shell**, les guillemets et cotes ne servent à rien.

3. `A=5 + 2`
   `5` est stocké dans la variable `A`. Le shell essaye ensuite d'exécuter la commande `+` (qui n'existe pas) avec comme paramètre `2`.

   L'utilisateur voulait peut-être faire `A='5 + 2'` pour stocker `5 + 2` dans `A` ou bien `A=$(expr 5 + 2)` pour stocker le résultat du calcul.

   Beaucoup ont donné la commande fausse suivante : `A=expr 5 + 2` qui met `expr` dans `A` et exécute la commande `5` avec comme paramètres `+` et `2`.

   N'oubliez pas, le shell standard ne sait pas ce qu'est un nombre.

4. `rm cd rm`
   Si l'on a le droit en écriture dans le répertoire courant et que celui-ci contient des **fichiers** nommés `cd` et `rm` ceux-ci seront détruits.

5. `vi $(date)`
   `$(date)` est remplacé par ce que la commande `date` écrit (la date actuelle). L'éditeur de texte `vi` va donc éditer le ou **les** fichiers dont le nom lui est passé en paramètre, cela dépend de l'affichage de la date qui peut se faire avec ou sans espace. Pour éditer un fichier qui porte la date présente il faut taper `vi "$(date)"`

6. `ls -R -a /`
   Liste les noms des fichiers récursivement (`-R`) y compris les fichiers cachés (`-a`) à partir de la racine du système de fichier (`/`)

   Il est plus court d'écrire : `ls -Ra /`

7. `mv /bin/ls ..`
   La commande `mv` déplace le fichier (ou répertoire) `ls` du répertoire `/bin` dans le répertoire père du répertoire courant. Si `../ls` existait déjà, il est écrasé.

8. `cd toto.tar.gz`
   Le répertoire courant devient `./toto.tar.gz` si c'est possible : droits d'accès et existence du dossier nommé `toto.tar.gz`

   C'est un très mauvais nom pour un dossier car il représente une archive compressée, mais c'est un nom légal.

9. `man sauvegarde`
   Affiche la page de manuel de la commande, du fichier, de la fonction, ... nommée `sauvegarde`. Comme il y a rarement des noms en français cela va certainement provoquer une erreur. L'utilisateur voulait peut-être chercher dans la description des pages de manuels qui peut être en français : `man -k sauvegarde`

10. `cp /bin/sh /tmp`
    `/bin/sh` est l'interpréteur de commande, c'est un fichier que tout le monde peut lire. `/tmp` est un répertoire pour que les utilisateurs stockent leurs fichiers temporaire, tout le monde peut écrire dedans. Si `/tmp` ne contient pas `sh` alors le contenu de `/bin/sh` sera copié sous le nom `/tmp/sh` sans aucun problème.

11. `echo a ; echo b`
    Exécution séquentielle des 2 commandes. Cela affiche `a\nb\n`

12. `rm .  && echo a`
    `.` est le répertoire courant. `rm` fait une erreur car elle ne peut détruire de répertoires sans l'option `-r`. `echo a` ne s'exécute pas à cause de l'erreur de `rm` et du séparateur `&&` qui arrête l'exécution en cas d'erreur.

13. `tar -cvf toto.tar`
    `tar` permet de convertir un hiérarchie de fichiers en un seul fichier et inversement sans perte d'information.

    `-c` indique que l'on veut **c**réer une archive.

    `-v` indique que la commande est **v**erbeuse et affiche ce qu'elle fait.

    `-f` indique que l'argument suivant est le nom du **fi**chier archive à créer, donc on crée `toto.tar`

    Il manque le nom du répertoire à archiver, il y aura donc une erreur.

    Peut-être l'utilisateur voulait extraire l'archive : `tar -xvf toto.tar`

14. `chmod 640 filename`
    Si `./filename` existe et que son propriétaire est le même que celui du processus courant (ou on est `root`) alors le mode du fichier devient `rw-r-----`

15. `read B <toto ; rm $A`
    Met la première ligne du fichier `toto` dans la variable `B` et détruit le ou les fichiers dont le ou les noms sont dans la variable `A`.

    Si l'on sait qu'il y a un seul nom de fichier dans la variable il faut faire : `rm "$A"`

    Si l'on veut détruire un fichier nommé `$A` il faut faire : `rm '$A'`

    Cela ne fonctionnera pas si le nom du fichier commence par un tiret : `-`

16. `find ~exco -name '\*'`
    Affiche tous les fichiers dont le nom est `*` dans la hiérarchie de fichier du répertoire de connexion de l'utilisateur `exco`.

    La commande `find ~exco -name *` fait n'importe quoi car l'étoile est remplacée par la liste des noms contenues dans le répertoire courant.

    `find ~exco -name \*`, `find ~exco -name '*'` et `find ~exco -name "*"` sont identiques et recherchent les fichiers portant un nom quelconque (donc tous) car la commande `find` voit comme *pattern* la chaine de caractère `*`

    `find ~exco -name \\*` fonctionne comme celle de l'énoncé car la commande `find` cherche le *pattern* `\*` car le *shell* remplace le premier *backslash* annule la signification du deuxième **pour le shell**.

17. `cp -r . tous mes fichiers`
Copie récursive du répertoire courant, de `./tous` et de `./mes` dans `./fichiers` si ce dernier n'existe pas (il sera créé) ou si c'est un répertoire.

    Le `./tous` et le `./mes` ne servent à rien car faisant parti du répertoire courant ils seront copiés deux fois.

    L'utilisateur voulait peut-être taper : `cp -r . 'tous mes fichiers'`

    Cette dernière commande ne fonctionne pas non plus ou plutôt on ne sait pas ce qu'elle fait. En effet `tous mes fichiers` est dans le répertoire courant et doit être copié à l'intérieur de lui-même. Est-ce une boucle infinie ?

18. `echo bonjour | sed -e 's/o/A/'`
Affiche `bAnjour` sur la sortie standard.

    La commande `echo bonjour | sed -e 's/o/A/g'` écrit `bAnjAur`

19. `cat /etc/a* | grep '->' | sort`
Affiche de manière triée, sur la sortie standard les lignes contenues dans les fichiers dont le nom commence par `a` dans le répertoire `/etc` qui contiennent la chaine de caractère `->`

    Cette commande ne fonctionne pas car `grep` dit que `>` n'est pas une option valide. La bonne commande est : `grep -e '->'`

    D'autre part, `cat` affiche des erreurs pour chaque répertoire que l'on tentera d'afficher.

20. `(date ; ls -R / ; date) >liste`
La sortie standard de ce qui est entre parenthèses est redirigé vers le fichier `liste` qui est créé s'il n'existe pas. Il y aura une erreur si `liste` est un répertoire ou si l'on manque de droits.

    Le fichier `liste` contiendra la date de début d'exécution, la liste de tous les fichiers et la date de fin d'exécution.

21. `(read C ; read C) </usr ; echo $C`
Cette commande affiche la valeur courante de la variable `C` car ce qui est entre parenthèses est fait par un processus qui est fini. Donc si `C` n'avait pas été utilisé avant cela affiche une ligne vide.

    De plus cette commande fait une erreur car `/usr` est un répertoire.

    Il est préférable de mettre des guillemets autour des variables pour que les retours à la ligne et les espace multiples ne soient pas transformés en un espace unique.

    `(read C ; read C ; echo "$C") </usr/include/stdio.h` : cette commande affiche la deuxième ligne du fichier `/usr/include/stdio.h`

22. `D= ; while read E ; do D="$D$E" ; done`
Concatène dans la variable `D` tous ce qui est lu dans l'entrée standard en enlevant les retours à la ligne.

    Les espaces en début et fin de ligne sont aussi enlevés.

23. `grep '^[ \t]*[{}]?[ \t]*$' <*.c | wc -l`
L'entrée standard est un fichier, pas de multiples fichiers.
Il manque l'option `-r` à `grep` pour indiquer que l'expression régulière est étendue à cause du point d'interrogation.
La bonne commande est :
`cat *.c | grep -r '^[ \t]*[{}]?[ \t]*$' | wc -l`
Cette commande compte le nombre de lignes vides ou ne contenant qu'une seule accolade dans l'ensemble des fichiers du répertoire courant dont le nom ce termine par `.c`
Elle ne tient pas compte des espaces et tabulations.

24. `for I in in cat titi ; do cp $I $I.old ; done`
    Copie `in` dans `in.old`, `cat` dans `cat.old`, `titi` dans `titi.old` si ces différentes copies sont autorisées.

    Les guillemets ne sont pas obligatoires car la variable ne contient pas d'espaces dans cet exemple.

25. `kill $(ps -e | grep truc | read A ; echo $A)`
    `kill` est lancé avec le contenu courant de `A` il y a donc toute les chances pour que cela fasse une erreur. Pour avoir la bonne valeur dans `A` voici la ligne de commande :
    `kill $(ps -e | grep truc | (read A ; echo $A))`
    Cette commande ne marche pas car `A` contient toute la ligne et pas seulement le `PID` du processus. Il faut prendre seulement le `PID` :
    `kill $(ps -e | grep truc | (read A B ; echo $A))`
    Cette ligne détruit le premier processus listé par la commande `ps` avec `truc` dans la ligne. Le processus peut s'appeler `trucs` et sera tué.

26. `ps -e | grep truc | while read A ; do kill $A ; done`
    Il faut donner le `PID` à `kill` donc la commande doit être :
    `ps -e | grep truc | while read A B ; do kill $A ; done`
    Cette commande "arrête" tous les processus qui ont `truc` dans leur nom.

27. `expr '(' 100 * $(gzip </bin/sh | wc -c) \) / $(wc -c </bin/sh)`
    Il faut protéger l'étoile pour que le *shell* ne la remplace pas par la liste des fichiers.

    Cela affiche la taille en pourcentage du fichier `/bin/sh` après sa compression par `gzip`.