Examen Pratique d'Unix, 2006/2007 semestre 2

Le 25/04/2007. 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 '$PATH'`
2. `rm \$X`
3. `ls "$Y"`
4. `grep * /etc/passwd`
5. `grep "a*" </etc/passwd`
6. `cat /etc/* | grep "a\*"`
7. `cp A/* B/*`
8. `read PS1 <toto`
9. `echo Tapez return ; read $REPONSE`
10. `mkdir C && cd C && rm *`
11. `man cd`
12. `(echo A ; cat titi ; echo B) >tutu`
13. `expr $A + 1`
14. `A="$A$A"`
15. `for I in "*.c" ; do rm $I ; done`
16. `for I in $(find .  -print) ; do ls -l "$I" ; done`
17. `find / -name *.c | while read A ; do cat "$A" ; done`
18. `A=5 ; ( A=5$A ; echo $A ) ; ( A=6$A ; echo $A )`
19. `rm toto ; echo ls >toto ; ./toto`
20. `sort D >D`
21. `sed "s/\*\**/*/" >a <b`
22. `sed --regexp-extended "s/(.)(.*)/\2\1/" # --regexp-extended ≡ -r`
23. `for I in *.c ; do rm $(echo $I | sed "s/..$/.o/") ; done`
24. `read read ; cat $read`
25. `I=0 ; while read A ; do echo "$I $A" ; I=$(expr $I + 1) ; done |`
    `sort --numeric-sort --reverse | sed "s/[0-9]* //"`
26. `mv "*.c" "*.C"`
27. `find / -size +100k -exec wc --lines {} | sort --numeric-sort |`
    `tail -1 | ( read A B ; echo $B )`

1. `echo '$PATH'`

   Écrit sur la sortie standard (l'écran par défaut) le mot `$PATH` les apostrophes (*quotes* en anglais) empêchent le remplacement de la variable par sa valeur.

2. `rm \$X`

   Détruit le fichier portant le nom `$X`. Erreur si on a pas le droit, s'il n'existe pas ou si c'est un répertoire.

3. `ls "$Y"`

   Si la variable `Y` contient un nom de fichier, il est affiché, si elle contient un nom de répertoire son contenu est affiché. Sinon il y a une erreur.

4. `grep * /etc/passwd`

   Le shell remplace l'étoile par la liste des noms de fichiers/répertoires du répertoire courant. Les lignes (de `/etc/passwd` et des fichiers (sauf le premier) du répertoire courant) contenant le premier nom seront affichées. Si le premier nom de fichier commence par un tiret, il sera interprété comme des options pour la commande `grep`. Ce n'est certainement pas ce que la personne voulait faire...

5. `grep "a*" </etc/passwd`

   Le shell ne fait pas de remplacement car ce que l'on recherche est entre guillemets. On recherche une suite de 0, 1, ... de 'a' n'importe ou dans la ligne. Cela va donc afficher toutes les lignes car elle contiennent toutes une chaine vide. D'autre part, comme on utilise une redirection de l'entrée standard le nom du fichier n'est pas affiché.

6. `cat /etc/* | grep "a\*"`

   `cat` écrit le contenu de tous les fichiers contenus dans `/etc` (sauf les fichiers cachés) dans le pipeline. `grep` affiche les lignes contenant `a*`. Le \ sert à dire que l'étoile est un vrai caractère * et n'indique pas une répétition de la lettre `a`. La commande `cat` affiche des erreurs sur la sortie d'erreur pour les répertoires.

7. `cp A/* B/*`

   Ne jamais faire ce genre de commande ! Le shell fait les remplacements par des noms de fichier ou laisse tel quel s'il n'y en a pas. Remarque : `cp` ne copie pas les répertoires par défaut. Renarque : Si le premier nom de fichier commence par tiret (-) alors ce qui se passe dépend du nom du fichier. Plusieurs cas possibles :

   - Les remplacements donnent 2 paramètres, alors la copie est faite si le premier est un fichier qui existe.
   - Les remplacements donnent plus de 2 paramètres. Si le dernier de la liste est un fichier rien n'est fait, sinon tous les fichiers précédents venant de A ou B sont copiés dans ce répertoire.

8. `read PS1 <toto`

   La commande `read` (*builtin* du shell), lit la première ligne du fichier `toto` du répertoire courant et la stocke (sans le *linefeed*) dans la variable nommée `PS1` (le *prompt* de premier niveau).

9. `echo Tapez return ; read $REPONSE`

   Affiche `Tapez return` sur la sortie standard puis (même si une erreur a eu lieu) lit une ligne de l'entrée standard (le clavier par défaut) et la stocke non pas dans la variable `REPONSE` mais dans la variable dont le nom est dans la variable `REPONSE`

10. `mkdir C && cd C && rm *`

    Si une commande échoue la suite ne s'exécute pas. Le répertoire C est créé, il devient le répertoire courant, comme il est vide le * n'est pas remplacé et donc la commande `rm` fait une erreur en disant que le fichier * n'existe pas.

11. `man cd`

    Il n'y a pas de page de manuel sur commande `cd` car c'est une commande du shell (*builtin*). Il faut faire *help cd* pour que le shell lui-même affiche l'aide.

12. `(echo A ; cat titi ; echo B) >tutu`

    Le fichier `tutu` est créé ou vidé si c'est possible. Il prend comme contenu une ligne 'A' le contenu de `titi` et une ligne 'B'. Si `titi` n'existe pas, il contient seulement les 2 lignes.

13. `expr $A + 1`

    Si la variable `A` contient quelque chose qui n'est pas un nombre, le résultat dépend du contenu (si par exemple `A` contient `1 + 1` le résultat sera 3). Sinon le contenu de `A` auquel sera ajouté 1 sera affiché sur la sortie standard.

14. `A="$A$A"`

    Stocke la concaténation du contenu de la variable `A` avec elle même dans la variable `A`.

15. `for I in "*.c" ; do rm $I ; done`

    Détruit le fichier unique nommé `*.c` du répertoire courant. Pour détruire tous les fichiers se terminant par `.c` il faut enlever les guillemets autour du `*.c` et mettre des guillemets autour du `$I` pour éviter les problèmes avec les noms de fichier contenant un espace par exemple.

16. `for I in $(find . -print) ; do ls -l "$I" ; done`

    La commande `find` génère la liste de tout ce que l'on trouve dans la hiérarchie de fichier à partir du répertoire courant, en effet il n'y a aucun critère de sélection. La commande `for` parcourt les mots de la liste fournie et fait un affichage avec beaucoup d'information sur le fichier ou bien le contenu du répertoire si s'en est un. Il manque l'option `-d` à la commande `ls` pour ne pas lister le contenu des répertoires. D'autre par **cette ligne de commande fait n'importe quoi si les noms de fichier contiennent des espaces**, en effet l'espace sépare les mots.

17. `find / -name *.c | while read A ; do cat "$A" ; done`

    S'il n'y a pas de fichier `.c` dans le répertoire courant, alors cela affiche le contenu de tous les fichiers à partir de la racine qui ont comme extension `.c`. Il y aura une erreur si un fichier n'est pas lisible (répertoire ou droit d'accès) ou bien si le nom contient un retour à la ligne. **Il faut obligatoirement mettre les guillemets autour de** `*.c`, et il est préférable d'ajouter `-type f` aux options de `find`.

18. `A=5 ; ( A=5$A ; echo $A ) ; ( A=6$A ; echo $A )`

    Cela affiche 55 puis 65 car les modifications faites entre les parenthèses sont perdues car elles sont faites dans un autre processus que celui du shell. Les guillemets ne sont pas obligatoire **dans cet exemple** car le contenu de la variable ne contient pas de caractères spéciaux.

19. `rm toto ; echo ls >toto ; ./toto`

    Si possible, un fichier nommé `toto` et contenant `ls` est créé. Son exécution (qui devrai lancer la commande `ls`) ne se fait normalement pas car il n'est pas exécutable.

20. `sort D >D`

    Le fichier nommé `D` est créé ou vidé. La commande `sort` va donc trier ligne par ligne un fichier vide. On obtiendra donc un fichier vide.

21. `sed "s/\*\**/*/" >a <b`

    Copie le contenu du fichier `b` dans le fichier `a` en remplaçant la **première occurrence** (car pas de /g) d'une suite d'étoile dans chacune des lignes par une seule étoile. Il n'y a pas besoin de mettre un \ avant l'étoile de l'expression de droite car ce n'est pas une expression régulière. L'expression commence par deux étoiles pour assurer que même si l'on trouve 0 étoile la chaine de caractère ne soit pas vide, sinon cela insérerait une étoile en premier caractère de ligne (car c'est le premier emplacement vide).

22. `sed --regexp-extended "s/(.)(.*)/\2\1/"` # `--regexp-extended` ≡ `-r`

    Écrit l'entrée standard sur la sortie standard en déplaçant le premier caractère de chaque ligne à la fin.

23. `for I in *.c ; do rm $(echo $I | sed "s/..$/.o/") ; done`

    Dans le répertoire courant détruit les fichiers avec l'extension `.o` qui correspondent à un fichier `.c`. Il manque des guillemets autour du `$I` et du `$()`. On pouvait écrire plus court pour la substitution : `s/.$/o/`

24. `read read ; cat $read` Stocke la ligne lue dans l'entrée standard dans une variable nommée `read`. Puis affiche le contenu du fichier (si c'est possible) dont le nom est dans la variable `read`. Il manque des guillemets autour du `$read`

25. `I=0 ; while read A ; do echo "$I $A" ; I=$(expr $I + 1) ; done |`
    `sort --numeric-sort --reverse | sed "s/[0-9]* //"`

    La boucle affiche les lignes lues sur l'entrée standard en les préfixant des entiers successifs. L'ordre des lignes est inversé par `sort` et `sed` supprime les nombres en début de ligne (seulement celui-ci car il n'y a pas l'option `g` de `sed`). Au final les lignes lues sur l'entrée standard sont affichées dans l'ordre inverse de leur saisie.

26. `mv "*.c" "*.C"`

    Renomme le fichier `*.c` en `*.C` car la commande `mv` comme toutes les commandes sauf de rares exceptions (`tar` et `find` par exemple) ne connaissent pas les *patterns*. Si l'on ne met pas les guillemets le fonctionnement va dépendre des noms des fichiers et répertoire présents dans le répertoire courant.

27. `find / -size +100k -exec wc --lines {} | sort --numeric-sort |`
    `tail -1 | ( read A B ; echo $B )`

    Il manque \; avant le premier pipe. Sinon cela affiche pour tous les fichiers de plus de 100Ko le nombre de lignes du fichier suivi du nom du fichier. Le tri est donc fait sur le nombre de lignes, la commande `tail` prend que la dernière ligne. Le `read` lit le nombre de ligne et le nom du fichier visible sur cette dernière ligne et le `echo` affiche donc le nom du fichier de plus de 100Ko contenant le plus de ligne. Cette commande ne fonctionne pas s'il y a un fichier dont le nom contient un retour à la ligne.