Examen Pratique d'Unix, semestre 2, 2005/2006

Le 19/04/2006. Tout document autorisé. Durée : 1 heure et demie.

Pour chacune des lignes de commandes indiquées sur ce sujet, expliquez en moins de 5 lignes ce que fait la ligne de commande.

Quelques commandes contiennent des pièges (elles sont fausses). Vous aurez d'autant plus de points que vous déjouez de pièges.

1. ls -l /home/p0123456/../../p9876453
2. echo toto >$A
3. cd HOME
4. rm $(pwd)/toto
5. chmod o-w toto
6. man -k remove
7. A=$(date)
8. echo *.[ch]
9. mv *.'[ch]' XXX
10. cp *.c *.c.old
11. find . -name *.c -size +10k
12. touch toto & rm toto
13. rm toto && rm toto && ps
14. ps | grep init
15. echo toto tutu titi | read A B ; echo $A
16. while [-f toto] ; do echo 'ici' ; done
17. (echo toto >&2 ; echo titi) >/dev/null
18. sed 's/:.*//g' </etc/passwd | sort
19. I=a ; while ["$I" != "aaaa"] ; do echo "$I" ; I="a$I" ; done
20. grep '[a-zA-Z][a-z0-9]*[\t]*=' *.c
21. cut --delimiter=: --fields=1 /etc/passwd | while read A;do ls /home/$A;done

Une correction :

1. On suppose que `/home` et `/home/p0123456` sont des répertoires existants et traversables (sinon il y a un erreur). La commande est alors équivalente à `ls -l /p9876453` Cette commande liste plein d'information sur `/p9876453` si c'est un fichier, sinon sur son contenu. S'il n'existe pas, un message d'erreur s'affiche.

2. Stocke la chaine de caractère `toto\n` dans le fichier dont le nom est dans la variable nommée `A`. Le fichier est créé s'il n'existe pas ou vidé s'il existe. Si bien sùr le processus a le droit.

   `echo toto >A` crée un fichier nommé `A`

   `echo toto >'$A'` crée un fichier nommé `$A`

3. Si le répertoire `./HOME` existe et accessible alors le répertoire courant devient ce répertoire. `cd $HOME` nous aurais envoyé dans le répertoire indiqué par la variable `HOME` (normalement le répertoire de connexion)

4. `$(pwd)` est remplacé par le nom absolu du répertoire courant. Donc cela détruit le fichier `toto` du répertoire courant. Si `toto` est un répertoire, il faut ajouter l'option `-r` à la commande `rm`

   Attention, cette commande ne fonctionne pas si le répertoire courant contient un nom de fichier avec des espaces dans le nom. La commande correcte est `rm "$(pwd)/toto"`

5. En enlève le droit d'écriture aux utilisateurs différents du propriétaire et n'appartenant pas au groupe propriétaire du fichier `toto`. Les autres ne pourront donc plus écrire dans le fichier, ou si c'est un répertoire, il ne pourront pas ajouter ou enlever des fichiers.

6. Liste les commandes ayant le mot `remove` dans leur description.

7. `$(date)` est remplacé par le résultat de l'exécution de la commande `date`. Donc cela affecte la date courante à la variable `A`. Pour des raisons de lisibilité et cohérence je conseille plutôt d'écrire `A="$(date)"`

8. Affiche sur la sortie standard les noms des fichiers du répertoires courant dont le nom se termine par `.c` ou bien `.h`

   Pour un utilisateur normal, cela n'affiche généralement pas les fichiers dont le nom commence par `.` (fichiers cachés).

9. S'il n'y a aucun fichier se terminant pas `.[ch]`, la commande `mv` proteste en disant que le fichier `*.[ch]` n'existe pas.

   Si `XXX` est un répertoire : tous les fichiers du répertoire courant dont le nom se termine par `.[ch]` sont déplacés dans le répertoire en gardant leur nom.

   Si `XXX` n'existe pas ou est un fichier. S'il n'y a qu'un seul fichier, il est renommé. S'il y en a plusieurs la commande `mv` proteste en indiquant que `XXX` doit être un répertoire.

10. Quand plusieurs fichiers sont à copier, ont doit les copier dans un répertoire indiqué en dernier dans la liste.

    Quand `cp` trouve un répertoire dans la liste des fichiers à copier, il n'est pas copié si l'option `-r` n'est pas indiquée.

    La commande `cp` ne sait pas ce qu'est un *pattern*.

    Le shell remplace `*.c` et `*.c.old` par l'ensemble des fichiers qui correspondent. S'il n'y a pas correspondance le *pattern* est laissé tel quel.

La commande `cp` voit donc une liste d'argument qui peut avoir de très nombreuses formes, en voici quelques unes :

La liste des paramètres **après** substitution contient 2 paramètres : `*.c *.c.old` ou `x.c *.c.old` ou `*.c x.c.old` Il y a 9 cas possibles suivant l'existence et le type de fichiers/répertoires correspondant au 2 paramètres.

La liste des paramètres **après** substitution contient plus de 2 paramètres : `*.c a.c.old b.c.old` ou `x.c a.c.old b.c.old` ou `a.c b.c *.c.old` ou `a.c b.c x.c.old` Il y a une erreur si le dernier paramètre ne correspond pas à une nom de répertoire. Si c'est un répertoire tous les **fichiers** existant seront copiés dans le répertoire sans que leur nom soit changé y compris les `.old`.

Les seuls cas ou cette commande fait une copie en ajoutant `.old` au nom du fichier c'est quand il y a un unique fichier correspondant à chacun des *pattern* `*.c` et `*.c.old` avec le même nom pour la partie gauche ou bien s'il y a un fichier nommé `*.c` et aucun fichier correspondant au *pattern* `*.c.old`

Ce que l'utilisateur voulait faire était peut-être : `for I in *.c ; do cp $I $I.old ; done`

11. Si après que le shell est substitué les *patterns* le `*.c` est resté `*.c` alors la commande `find` recherche tous les fichiers avec l'extension `.c` qui font plus de 10 kilo-octets dans la hiérarchie à partir du répertoire courant.

    Si `*.c` a été remplacé par **un** nom de fichier alors c'est **ce** nom de fichier qui est recherché dans toute la hiérarchie.

    S'il est remplacé par plusieurs noms de fichiers, la commande `find` fait une erreur (sauf si les noms des fichiers correspondent à des options de `find` et que la syntaxe est valide !)

12. La commande `touch` qui modifie la date du fichier (ou le crée) s'exécute en arrière plan. La commande `rm` qui détruit le fichier s'exécute en parallèle. Le résultat de cette ligne de commande est imprévisible car il dépend de la commande qui va manipuler le fichier en dernier (création ou destruction ?).

13. Si la destruction de `toto` ne fonctionne pas, on exécute pas les autres commandes.

    La deuxième destruction de `toto` échoue (sauf s'il a été recréé entre temps par un autre processus). Dans la commande `ps` listant les processus ne devrait pas s'exécuter.

14. Affiche les lignes affichées pas `ps` contenant le mot `init`. Pour trouver le processus `init` il faudra ajouter une option `-e` par exemple pour voir tous les processus et non seulement ceux lancé à partir du terminal dans lequel vous tapez la commande.

15. `toto` va dans la variable `A`, `tutu titi` va dans la variable `B`, le processus `read` meurt donc toutes ses variables sont perdues. La valeur de la variable `A` du shell courant est affichée mais elle est inconnue....

    La bonne syntaxe est `echo toto tutu titi | (read A B ; echo "$A")` car le `read` et le `echo` s'exécutent dans le même processus.

16. Tant que le répertoire courant contient un fichier texte standard le mot `ici` s'affiche en boucle.

    Un autre processus peut arrêter la boucle en détruisant le fichier.

17. La sortie standard du groupe de commande va dans la poubelle `/dev/null` La sortie standard du premier `echo` est redirigée vers la sortie d'erreur, donc `toto` s'affiche à l'écran. `titi` s'affichant sur la sortie standard, il disparaît dans `/dev/null`.

18. `sed` lit le fichier `/etc/passwd` et l'écrit sur la sortie standard en enlevant ce qui est à droite du premier : (Le `g` est inutile car c'est la plus longue chaine de caractères qui est prise. `sort` lit son entrée standard (la liste des noms d'utilisateur) et l'écrit triée sur sa sortie standard (l'écran)

19. Affiche `a` puis `aa` puis `aaa`.

20. Affiche les lignes des fichiers dont le nom se termine par `.c` dans le répertoire courant qui contienne quelque part la séquence suivante :

    - Une majuscule ou minuscule.
    - Une suite de minuscules et de chiffres (pouvant être vide)
    - Une suite d'espace et de tabulation.
    - Le signe `=`

21. Pour chaque nom d'utilisateur indiqué dans `/etc/passwd` (première colonne) affiche le contenu du répertoire `/home/`*`nomutilisateur`*. Pour afficher le nom du répertoire de connexion, il aurai fallu prendre la bonne colonne du fichier `/etc/passwd`.