# Examen Pratique d'Unix, 2007/2008 semestre 1

Le 21/11/2007. 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. `cd ;`
2. `ls toto`
3. `cat $I`
4. `rm "*"`
5. `date >>log`
6. `mv *.[ch] TOTO`
7. `read $A`
8. `B="$(date)"`
9. `cp /etc/p* tutu`
10. `man -k remove`
11. `cd $X`
12. `mkdir "A B"`
13. `sort "*.c" >../.`
14. `grep "toto[ \t]*=[^ =]" */*.c`
15. `sed 's/:.*//' </etc/passwd`
16. `A=5 ; (export A=6) ; echo $A`
17. `echo rm >tata ; ls $(cat tata)`
18. `rm titi && rm titi && rm titi && echo titi`
19. `cut -delimiter ':'  -fields=1 <passwd`
20. `if [ -f $A ] ; then echo ?  ; fi`
21. `case '$B' in $) echo a ;; *) echo b ;; esac`
22. `for I in *.c ; do cp "$I" "$I".bak ; done`
23. `find .  -name "a*" -name "b*" -print`
24. `find .  -print | while read A ; do wc "$A" ; done`
25. `I=0 ; while [ $I != 10 ] ; do echo >$I ; I=$(expr $I * 2) ; done`
26. `for I in *.c ; do sed 's/int/long/' <$I >$I ; done`

1. `cd ;`
   Le ';' est un séparateur de commande, comme il n'y a rien à sa droite, seule la commande `cd` sans paramètre est exécutée. Le répertoire courant devient alors le répertoire de connexion. Le ';' est en trop ou bien l'utilisateur voulait réellement aller dans un répertoire nommé ';' et dans ce cas il aurait du écrire `cd ';'` ou `cd ";"` ou `cd \;`

2. `ls toto`
   Si `toto` n'existe pas alors cela affiche un message d'erreur. Si `toto` est un fichier alors cela affiche `toto`. Si `toto` est un répertoire et que l'on a le droit de lister son contenu (mode *r*) alors cela liste les noms des entités qu'il contient, sinon cela fait une erreur d'accès.

3. `cat $I`
   Suivant la valeur de la variable `I` :

   - Vide : la commande recopie l'entrée standard sur la sortie standard.
   - Commence par un '-', alors le comportement dépend de la suite de la valeur de `I`.
   - Contient une liste : Si un élément de la liste est un nom de fichier existant et accessible en lecture alors son contenu est écrit sur la sortie standard. Sinon une erreur contextuelle s'affiche.
   - Si elle contient un nom de fichier avec des espaces, alors la commande ne fonctionnera pas correctement car la bonne syntaxe serait : `cat "$I"`

4. `rm "*"`
   Si un fichier s'appelle '*' dans le répertoire courant et que l'on a les droits pour le détruire, il sera détruit. S'il n'existe pas, si l'on a pas les droits ou si c'est un répertoire un message d'erreur s'affichera. Pour le détruire si c'est un répertoire il faut ajouter l'option `-r`, pour le détruire s'il est sans poser de question s'il est protégé par nous-même il faut l'option `-f`. Si l'on voulais détruire tous les fichiers (mais pas les répertoires) il fallait taper `rm *`

5. `date >>log`
   La sortie standard est redirigée vers la fin du fichier `log` qui doit être accessible en écriture, le fichier n'est pas vidé. Si le fichier n'existe pas il est créé (si c'est possible). La date au moment ou la commande s'exécute est donc ajoutée à la fin du fichier.

6. `mv *.[ch] TOTO`
   Si `TOTO` est un répertoire, alors tous les fichiers/répertoires du répertoire courant se terminant par `.c` ou `.h` sont déplacés dans `TOTO` (si on a les droits en écriture). Si aucun n'existe, et que le fichier `*.[ch]` n'existe pas alors il y aura une erreur.

   Si `TOTO` est un fichier ou n'existe pas, et que `*.[ch]` représente plusieurs fichiers alors il y aura une erreur. Si `*.[ch]` représente un fichier existant, il sera renommé en `TOTO`.

   On suppose ne pas avoir de fichier dont le nom commence par un tiret qui pourrait être pris comme une option de `mv`.

7. `read $A`
   Lit un **ligne** de l'entrée standard et la stocke dans la variable dont le nom est dans la variable `A`. Il faut bien sur que le contenu de la variable `A` soit un nom légal de variable.

8. `B="$(date)"`
   `$(date)` est remplacé par ce qui a été écrit sur la sortie standard par la commande `date`. Donc on affecte à la variable `B` la date courante. Les guillemets ne sont pas nécessaires, mais ils sont conseillés.

9. `cp /etc/p* tutu`
   Copie tous les fichiers (par répertoires car il manque l'option `-r`) contenus dans le répertoire `/etc` dont le nom commence par `p` dans le répertoire `tutu` en leur laissant leur nom. Si `tutu` n'est pas un répertoire cela fait une erreur.

10. `man -k remove`
    Liste les noms des commandes, la section et la description des commandes pour lesquelles on trouve le mot `remove` dans la description de la commande qui est indiquée dans la page du manuel.

11. `cd $X`
    Si la variable `X` contient des espaces alors cela ne fera pas ce qui est attendu car il manque des guillemets. La commande `cd` se retrouvera avec plusieurs arguments alors que ce n'est pas prévu (sauf cas très particulier).

    Si la variable `X` commence par un tiret `-` alors il y a de fortes chance que cela ne fonctionne pas. Il n'y a pas de moyen simple pour que cela fonctionne.

    Si la variable `X` n'existe pas ou est vide, alors on est envoyé dans le répertoire de connexion.

    Si la variable `X` contient un nom de répertoire ne contenant pas d'espace et dans lequel on peut aller, alors il devient le répertoire courant.

    Dans les autres cas, une erreur est affichée.

12. `mkdir "A B"`
    Si on a le droit et s'il n'existe pas, le répertoire nommé `A B` est crée avec un espace dans le nom. Sans les guillemets, 2 répertoires auraient été créées. On aurait aussi pu écrire `mkdir 'A B'` ou `mkdir A\ B`

13. `sort "*.c" >../.`
    Trie les lignes contenues dans LE fichier nommé `*.c` (à cause des guillemets). Le résultat est stocké dans le *fichier* nommé `.` dans le répertoire père en le vidant. Mais comme ce fichier `.` est le répertoire père lui-même et qu'il existe forcément et que l'on ne peut rediriger la sortie standard vers un répertoire, la commande ne se lance même pas.

14. `grep "toto[ \t]*=[^ =]" */*.c`
    Affiche les lignes contenant l'expression régulière trouvées dans tous les fichiers dont le nom se termine par `.c` dans tous les sous-répertoires directs du répertoire courant.

    L'expression régulière recherche `toto` suivi d'un certain nombre d'espaces et de tabulations suivi du signe = et suivi d'autre chose que = et espace.

    Cette commande va donc trouver toutes les affectation faites dans des variables dont le nom se termine par `toto`. Elle ne se fera pas piégé par `toto == 5` par contre elle ne trouvera pas `toto =5` car on a indiqué (par erreur certainement) que l'on ne voulais pas d'espace après le signe =.

15. `sed 's/:.*//' </etc/passwd`
    Lit le contenu de l'entrée standard (le fichier `/etc/passwd`) et le réécrit sur la sortie sur la sortie standard en remplaçant dans chacune des lignes le premier ':' et ce qui suit (`.*`) par rien.

    L'option `g` est inutile car UNE SEULE substitution de ce type est possible car on prend la plus longue.

    Le shell n'interprète pas l'étoile car le paramètre est entre cotes.

    On obtient au final la liste des noms d'utilisateur.

16. `A=5 ; (export A=6) ; echo $A`
    On met 5 dans `A`. Dans un nouveau processus, on met 6 dans `A` en l'exportant à ses descendants (il n'en a pas). Le nouveau processus meure et le shell exécute la troisième commande qui affiche 5 car pour lui la variable `A` n'a pas été modifiée.

17. `echo rm >tata ; ls $(cat tata)`
    Stocke le mot `rm` dans le fichier `tata` (si possible) puis lance `ls` avec comme paramètre le contenu du fichier `tata` qui devrait être `rm`. Cela revient donc à faire `ls rm`

18. `rm titi && rm titi && rm titi && echo titi`
    Le && arrête l'exécution à la première erreur. À moins qu'un processus recrée constamment `titi` le message final ne sera pas affiché.

19. `cut -delimiter ':' -fields=1 <passwd`
    Affiche sur la sortie standard la première colonne (délimité par :) du fichier `passwd` situé dans le répertoire courant.

    Si l'on est dans `/etc` cela affiche donc les noms des utilisateurs.

    Il faut 2 tirets devant les options, la syntaxe indiquée provoque une erreur.

    Il est plus logique de mettre 2 fois le signe = ou bien de n'en mettre aucun.

20. `if [ -f $A ] ; then echo ? ; fi`
    Si le contenu de la variable `A` est le nom d'un fichier texte alors on affiche un point d'interrogation.

    Ceci ne fonctionne pas dans le cas ou `A` contient des espaces car il manques les guillemets autour de `$A`.

    Ceci ne fonctionne pas si le répertoire courant contient des noms de fichiers sur une seule lettre. Dans ce cas, ces noms de fichiers sont affichés à la place du point d'interrogation.

21. `case '$B' in $) echo a ;; *) echo b ;; esac`
    Affiche `b` car le texte `$B` ne correspond à `$` (il aurait pu rentrer dans le moule `$*`) mais il *match* `*`.

    L'utilisateur doit mettre des guillemets à la place des cotes. Dans ce cas `a` s'affiche si `B` contient `$` et `b` s'affiche dans les autres cas.

22. `for I in *.c ; do cp "$I" "$I".bak ; done`
    Copie chaque fichier du répertoire courant dont le nom se termine par `.c` sous le nom même nom mais en ajoutant `.bak`.

    Pour que cette commande fonctionne même si le nom du fichier commence par tiret, il faut écrire `./*.c` On ne le fait pas en interactif, mais il est conseillé de le faire dans les scripts.

23. `find . -name "a*" -name "b*" -print`
    Affiche les chemins à partir du répertoire courant qui correspondent à un nom d'entité commençant par `a` ET par `b`. Cela ne va donc rien afficher car c'est impossible. Il faut écrire `-name "a*"` **-o** `-name "b*"`

24. `find . -print | while read A ; do wc "$A" ; done`
    Indique le nombre de caractères, mots et lignes pour chacun des fichiers (erreur pour répertoires) trouvé à partir du répertoire courant.

    Cette commande ne fonctionne pas si des fichiers ont un retour à la ligne dans leur nom. Ou bien si le nom commence par un tiret.

25. `I=0 ; while [ $I != 10 ] ; do echo >$I ; I=$(expr $I * 2) ; done`
    Si l'on protège l'étoile de la multiplication (sinon ça ne marche pas) alors c'est une boucle infini écrivant constamment un retour à la ligne dans le fichier nommé 0 (zéro).

26. `for I in *.c ; do sed 's/int/long/' <$I >$I ; done`
    Remplacerai la première occurrence de `int` par `long` dans toutes les lignes de tous les fichiers se terminant par `.c` dans le répertoire courant.

    Cette commande ne fonctionne pas car le fichier est vidé avant d'être lu. On se retrouve donc avec tous les fichiers `.c` VIDES.