Examen Pratique d’Unix, 2010/2011 Automne

Le 9/11/2010. 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. `man cd`
2. `rm D >E`
3. `cat F*G`
4. `cp A B C`
5. `date >\>`
6. `mkdir ’/’`
7. `mv ../I .`
8. `xargs ls <H`
9. `wc [a][b][c]`
10. `grep \\\* tata`
11. `expr length $J`
12. `M=expr "$M" + 1`
13. `echo ’"$V’ # bonjour`
14. `echo ’[$W]’ "[$W]" [$W]`
15. `cat *.[jpg] | grep "^ *$"`
16. `date | read K L ; echo $K`
17. `ls Q >/dev/null 2>&1 && ./R`
18. `sed -e ’s/ *//’ <tutu | sort`
19. `cd ../../../../../../../../../../..`
20. `sort sort | ( read read ; cat ) | head`
21. `find ..  -name *.c -mtime -1 | grep toto`
22. `if test "T" = "U" ; then echo coucou ; fi`
23. `for Z in $(cat titi) ; do cat "$Z" ; done`
24. `for S in */.  ; do cp -r "$S" "$S".bak & done`
25. `echo $(read N ; read N ; read N ; echo $N) <P`
26. `sed --regexp-extended -e ’s/[0-9]*([0-9]*).*/\1/’`
27. `case "$W" in [oOyY]*) echo O ; 0|1) echo X ; *) echo N ; esac`
28. `X=’’;while read Y;do if [ ’’ = "$Y" ];then echo "$X";X=’’;else X="$X $Y";fi`

Une correction

1. `man cd`
   Il n'y a pas de commande unix `cd`, l'affichage de la page de manuel ne se fera pas. `cd` est un *bulitin*, il faut donc faire `help cd` ou bien `cd --help`

2. `rm D >E`
   Détruit le fichier `D` (erreur si c'est un répertoire car il manque l'option `-r`). Et crée ou vide le fichier `E`. La commande `rm` n'affichant rien sur sa sortie standard le fichier `E` reste vide.

3. `cat F*G`
   `F*G` est remplacé par la liste des noms du répertoire courant qui commencent par `F` et finissent par `G`, s'il y en a aucun, `F*G` reste inchangé. La commande `cat` va afficher le contenu de tous ces fichiers si c'est possible et faire une erreur pour les répertoires.

4. `cp A B C`
   Elle fait la copie seulement si `A` et `B` sont des fichiers et que `C` est un répertoire. `C/A` et `C/B` sont créés. Si `A` ou `B` est un répertoire il faut ajouter l'option `-r` pour faire la copie.

5. `date >\>`
   Le contenu du fichier \ devient la date d'exécution de la commande.

6. `mkdir '/'`
   Les cotes sont inutiles. La commande fait une erreur car le répertoire racine existe déjà.

7. `mv ../I .`
   Déplace le fichier/répertoire nommée `I` dans le père dans le répertoire courant. Si un fichier `I` était déjà présent il est écrasé. Si un directory `I` était déjà présent il y a une erreur.

8. `xargs ls <H`
   La commande `xargs` prend les paramètres à passer à la commande `ls` en les lisant dans le fichier `H`. S'il y a trop d'arguments à passer, la commande `xargs` lancera la commande `ls` plusieurs fois.

9. `wc [a][b][c]`
   Compte le nombre de caractères, mots et lignes du fichier nommé `abc` s'il existe.

10. `grep \\\* tata`
    Le shell lance `grep` avec comme paramètres `\*` et `tata`. `grep` affiche toutes les lignes du fichier `tata` qui contiennent une étoile.

11. `expr length $J`
    Affiche la longueur de la chaine de caractères contenu dans la variable `J`. Pour que cette commande fonctionne quand la variable `J` contient un espace, il faut mettre les guillemets : `expr length "$J"`

12. `M=expr "$M" + 1`
    Affecte la valeur `expr` dans la variable `M`. Puis exécute a commande contenu dans `M` avec comme arguments `+` et `1`. L'utilisateur voulait certainement faire : `M=$(expr "$M" + 1)` pour ajouter 1 à la valeur de l'entier contenu dans `M`.

13. `echo '"$V' # bonjour`
    Affiche `"$V` sur la sortie standard. La fin de ligne est un commentaire.

14. `echo '[$W]' "[$W]" [$W]`
    Cela affiche `[$W] [contenu de W protégés]` suivi de la liste des fichiers dont le nom contient une lettre qui est indiquée dans la variable `W`.

La réponse indiquant que cela affiche les noms de fichier d'une lettre parmi `$` et `W` a été acceptée car elle est tout à fait logique.

Il est donc conseillé de ne pas mettre de dollar entre crochets...

15. `cat *.[jpg] | grep "^ *$"`
    Affiche les lignes des fichiers dont le nom se termine par `.j` ou `.p` ou `.g` qui ne contiennent que des espaces ou qui sont vides.

16. `date | read K L ; echo $K`
    Le premier mot (pas ligne) de la date courante est stocké dans `K` et le reste dans `L`. Quand le processus se termine les variables disparaissent. La valeur affiché est la valeur non modifiée de `K`. Il fallait donc faire `date | (read K L ; echo $K)`

17. `ls Q >/dev/null 2>&1 && ./R`
    La sortie standard PUIS la sortie d'erreur de `ls` sont redirigés vers la poubelle. rien n'apparait sur l'écran alors que `ls` a écrit `Q` si c'est un fichier ou son contenu si c'est un répertoire.

    Si `Q` est un fichier, ou bien si c'est un répertoire sur lequel vous avez les droits d'accès alors la commande `ls` ne fait pas d'erreur et donc la commande `R` du répertoire courant s'exécute.

    ATTENTION, ce n'est pas parce que vous avez mis les messages d'erreurs à la poubelle que la commande ne retourne pas d'erreur !

18. `sed -e 's/ *//' <tutu | sort`
    La commande `sed` lit le contenu de `tutu` et le réécrit dans le pipe en remplaçant le premier (il n'y a pas `/g` ce qui correspond à ` *` par rien du tout, la commande `sort` trie ce qui a été écrit dans le pipe.

    La pattern correspond à une suite d'espace pouvant être vide. Donc si la ligne commence par des espaces alors ils sont enlevés, et sinon elle n'est pas modifiée car ` *` correspond à la chaine vide qui est avant le premier caractère et qui est remplacée par rien, donc la ligne est inchangée.

19. `cd ../../../../../../../../../../..`
    Le répertoire courante monte de 11 niveau vers la racine. S'il l'atteint avant alors il y reste sans faire d'erreur car la père de la racine c'est la racine.

20. `sort sort | ( read read ; cat ) | head`
    Écrit dans le pipe le fichier nommé `sort` trié, met la première ligne dans la variable `read` du sous-shell, affiche les lignes suivantes (la première a été lue) dans le deuxième pipe, affiche les lignes triée sans la première.

21. `find ..  -name *.c -mtime -1 | grep toto`
    Pour simplifier, on va supposer que le `*.c` est protégé sinon le fonctionnemant de la commande dépend du contenu du répertoire courant.

    Le `find` affiche les **noms** des fichiers contenus dans la hiérarchie dont la racine est le père du répertoire courant dont le nom se termine par `.c` ET qui ont été modifiés il y a moins de 1 jour. Puis la commande `grep` n'affiche que les noms qui contiennent `toto`. Cette commande ne fonctionne pas si des noms de fichier contiennent un retour à la ligne. Il aurait été plus simple (et moins buggé) d'écrire :
    `find ..  -name '*toto*.c' -mtime -1`

22. `if test "T" = "U" ; then echo coucou ; fi`
    Les deux chaines sont différentes donc cela n'affiche rien.

23. `for Z in $(cat titi) ; do cat "$Z" ; done`
    Affiche le contenu de tous les fichiers dont le nom est contenu dans le fichier `titi`. Pour que cela fonctionne les noms ne doivent pas contenir d'espace, de retour à la ligne et ne doivent pas commencer par un tiret.

24. `for S in */.  ; do cp -r "$S" "$S".bak & done`
    Pour chaque **répertoire** contenu dans le répertoire courant lance une copie récursive du répertoire en nommant la copie avec un `.bak` à la fin. Si la copie existe et est un répertoire, la nouvelle copie copie va être stockée dans l'ancienne.

25. `echo $(read N ; read N ; read N ; echo $N) <P`
    La redirection n'agissant pas sur les paramètres, le fichier `P` n'est pas lu. Cela affiche donc la troisième ligne saisie sur l'entrée standard courante, la clavier si aucune redirection n'a été faite.

    La réponse : cela affiche la troisième ligne du fichier `P` a été acceptée car bien que fausse elle est logique.

26. `sed --regexp-extended -e 's/[0-9]*([0-9]*).*/\1/'`
    La première expression `[0-9]*` représentante la plus longue suite de chiffres à partir du début de la ligne. Cette première expression est potentiellement vide. Ce qui est entre parenthèse ne peut être que vide car tous les chiffres ont été pris par le début de l'expression. Le `.*` représente la suite de la ligne. donc l'expression régulière représente systématiquement une ligne complète. Celle ci est remplacée par `\1` qui représente le contenu du premier groupe (ce qui est entre parenthèse). Donc cette commande lit les lignes sur son entrée standard et les réécrit vide sur sa sortie standard.

27. `case "$W" in [oOyY]*) echo O ; 0|1) echo X ; *) echo N ; esac`
    Il faut remplacer les `;` par des `;;`
    Cela affiche `O` si le contenu de la variable `W` commence par une des lettres `o`, `O`, `y`, `Y`. Cela affiche `X` si le contenu est exactement `0` ou `1`. Dans les autres cas cela affiche `N`

28. `X='';while read Y;do if [ '' = "$Y" ];then echo "$X";X='';else X="$X $Y";fi`
    Il manque `; done` à la fin. Cette commande concatène les lignes lues sur l'entrée standard pour afficher chaque paragraphe sur une seule ligne.

    Un paragraphe est terminé par une ligne vide.