

Corrigé du TP 5 Système

C. Pain-Barre

INFO - IUT Aix-en-Provence

version du 27/11/2012

1 Découpage, recherche, compression et archivage de fichiers

Corrigé de l'exercice 1

Utiliser **split** pour générer des fichiers en morcelant de l'information

1. `pc:~$ ssh -X toto@allegro`
`toto@allegro's password: ...`
`Linux allegro...`
`toto@allegro:~$`
2. `mkdir split ; cd split`
3. Découpage de `../des_lignes.txt` par blocs de lignes :
 - (a) `split -l 3 ../des_lignes.txt`
 - (b) `ls`
 - (c) `cat * > des_lignes.txt`
 - (d) `cat des_lignes.txt`
4. Découpage de `~cpb/public/Firefox_wallpaper.png` par la taille :
 - (a) ... *pas besoin de corrigé pour cette question...*
 - (b) `split -b 100k ~cpb/public/Firefox_wallpaper.png fond`
 - (c) `ls`
 - (d) `cat fond* > lefond.png`
 - (e) `eog lefond.png`
 - (f) `rm *`
5. Découpage de l'entrée standard par blocs de lignes :
 - (a) `ps alx | split -l 40 - procs_`
↳ le - devant `procs_` demande à **split** de traiter l'entrée standard
 - (b) `ls`
 - (c) `cd ..`
 - (d) `rm -rf split`

Corrigé de l'exercice 2

Utiliser **find** pour rechercher et manipuler des fichiers vérifiant certaines conditions

1. `find ~ -type f -name cigale.txt -print`

⇒ l'option **-print** n'est pas nécessaire mais on la rajoutera quand même systématiquement

2. `find ~ -type d -name tpunix -print`

3. `find / -type f -user etxxxx -print`

⇒ où *etxxxx* est votre nom d'utilisateur

4. `find / -type f -user etxxxx -print 2> /dev/null | less`

5. `find ~ -type f -user etxxxx -mtime -7 -mtime +1 -print`

6. `find ~ -type f -user etxxxx -mtime -7 -mtime +1 -print > ~/tp/tpunix/hebdo.txt`

7. `find ~ -type f -user etxxxx -mtime -7 -mtime +1 | tee ~/tp/tpunix/hebdo.txt | wc -l`

⇒ **-print** a été omis car non nécessaire, et pour que la commande tienne sur la ligne...

8. `find ~cpb -type f \(-name 'ts_*' -o -name 'algo*' \) -name '*.ad[bs]' -exec ls -ld {} \;` ↵

⇒ la commande doit être tapée sur une seule ligne (le symbole ↵ indique que par manque de place, la commande se poursuit sur la ligne suivante)

9. `find ~cpb -type f \(-name 'ts_*' -o -name 'algo*' \) -name '*.ad[bs]' -exec cp -v {} ~/tp/tpunix \;` ↵

10. `find ~ -type f -name *.txt -exec chmod -v go= {} \;`

11. `$ ~cpb/public/unix/alim`

⇒ 1 fichier est créé dans le répertoire d'accueil, 1 autre dans *tp* et 2 autres dans *tpunix*

12. `find ~ -type f \(-name '*~' -o -name '*.bak' \) -exec rm -iv {} \;`

13. `find /dev/pts -type c \(-perm -002 -o -user etxxxx \) -print`

⇒ les terminaux sont des fichiers spéciaux en mode caractère (type **c**) et on ne veut que ceux qui ont au moins la permission d'écriture pour les autres.

14. `$ man find`

`$ find /dev/pts -type c -writable -print`

Corrigé de l'exercice 3

Utilisation de gzip/gunzip et de zcat

1. `ls -l cigale.txt` ou `wc -c cigale.txt` pour connaître sa taille

2. `gzip -v cigale.txt` puis `ls -l cigale.txt.gz`

3. `gunzip cigale.txt.gz`

4. à partir de *tpunix* : `gzip -rv .`

5. ... *pas besoin de corrigé pour cette question...*

6. `zcat cigale.txt.gz`

7. `gzip -dc cigale.txt.gz` (il ne faut pas oublier l'option **-d** pour décompresser !)

8. toujours à partir de *tpunix* : `gzip -drv .` ou `gunzip -rv .`

⇒ L'option **-v** est activée pour être informé sur le déroulement des opérations.

Corrigé de l'exercice 4

Utilisation de **tar**

1. `$ tar tf ~cpb/public/unix/images.tar`
images/
images/homer.gif
images/soupirs.gif
images/bart.gif
images/garfield.gif
images/simpsons.gif
2. `$ tar xvf ~cpb/public/unix/images.tar`
images/
images/homer.gif
images/soupirs.gif
images/bart.gif
images/garfield.gif
images/simpsons.gif
3. `$ tar tzf ~cpb/public/unix/files.tgz`
files/
files/a_decouper.txt
files/decale.txt
files/amphigouri.txt
files/gouri.txt
...
4. `$ gzip -dc ~cpb/public/unix/files.tgz | tar tf -`
files/
files/a_decouper.txt
files/decale.txt
files/amphigouri.txt
files/gouri.txt
...

⇒ **gzip** écrit l'archive décompressée sur sa sortie. Elle est récupérée via le tube en entrée de **tar**. L'option "**f -**" de **tar** demande de lire l'archive depuis l'entrée standard.

5. Depuis `tpunix` :
`$ rm -rf images images.tar files.tgz`
6. `$ cd ..`
7. `$ tar czvf tpunix.tgz tpunix`
tpunix/
tpunix/cigale.txt
...

⇒ Pour compresser l'archive, on a utilisé l'option **z** de **tar**. On obtient le même résultat avec un tube en tapant :

```
tar cvf - tpunix | gzip -c > tpunix.tgz
```

8. `$ tar tzf tpunix.tgz`
tpunix/
tpunix/cigale.txt
...
9. `$ rm tpunix.tgz`

2 Liens et bits set-uid, set-gid, sticky-bit

Corrigé de l'exercice 5


Création de liens

1. `cd ~/tp/tpunix`
2. `ln cigale.txt phys`
3. `ln -s cigale.txt symb`
4. `ls -l`
5. `$ ls -l symb`
`lrwxrwxrwx 1 toto toto 10 20 nov. 07:20 symb -> cigale.txt`
6. `$ ls -ll symb`
`-rw-r--r-- 1 toto toto 677 30 août 10:53 symb`
7. `vi cigale.txt`
8. `cat phys` puis `cat symb`
9. `rm cigale.txt`
10. `cat symb` échoue mais pas `cat phys`
11. `mv phys cigale.txt` puis `cat symb`

Corrigé de l'exercice 6

Bits set-uid et set-gid (sur les fichiers)

 Pour le corrigé, on suppose que l'utilisateur est toto appartenant au groupe toto.

1. `$ ls -l ~cpb/public/unix/infos*`
`-rwxr-xr-x 1 cpb prof 14810 sep 23 2004 /users/prof/cpb/public/unix/infos0`
`-rwsr-xr-x 1 cpb prof 14810 jan 13 2004 /users/prof/cpb/public/unix/infos1`
`-rwxr-sr-x 1 cpb prof 14810 jan 13 2004 /users/prof/cpb/public/unix/infos2`
`-rwsr-sr-x 1 cpb prof 14810 jan 13 2004 /users/prof/cpb/public/unix/infos3`
 on observe que `infos0` est un fichier exécutable normal, que `infos1` est *set-uid*, que `infos2` est *set-gid*, et que `infos3` est à la fois *set-uid* et *set-gid*.
2. ... pas besoin de corrigé pour cette question...
3. ... pas besoin de corrigé pour cette question...
4. `$ cp ~cpb/public/unix/infos3 ~/public`
`$ cd ~/public`
`$ ls -l infos3`
`-rwxr-xr-x 1 toto toto 14810 nov 9 15:09 infos3`
5. `$ chmod 701 infos3`
6. ... pas besoin de corrigé pour cette question...

7. `$ chmod g+s infos3`
`$ ls -l infos3`
`-rwxr-Sr-x 1 toto toto 14810 nov 9 15:09 infos3`
8. ... *pas besoin de corrigé pour cette question...*
9. `$ chmod g+x infos3`
`$ ls -l infos3`
`-rwxr-sr-x 1 toto toto 14810 nov 9 15:09 infos3`
10. ... *pas besoin de corrigé pour cette question...*
11. `$ chmod u+s,u-x infos3`
`$ ls -l infos3`
`-rwSr-sr-x 1 toto toto 14810 nov 9 15:09 infos3`
12. ... *pas besoin de corrigé pour cette question...*
13. `$ rm infos3`

Corrigé de l'exercice 7

Quelques utilitaires set-uid et/ou set-gid

1. `$ type su chfn chsh passwd gpasswd write`
`su est /bin/su`
`chfn est /usr/bin/chfn`
`chsh est /usr/bin/chsh`
`passwd est /usr/bin/passwd`
`gpasswd est /usr/bin/gpasswd`
`write est /usr/bin/write`
`$ ls -lL /bin/su /usr/bin/{chfn,chsh,passwd,gpasswd,write}`
`-rwsr-xr-x 1 root root 29152 15 févr. 2011 /bin/su`
`-rwsr-xr-x 1 root root 36372 15 févr. 2011 /usr/bin/chfn`
`-rwsr-xr-x 1 root root 27956 15 févr. 2011 /usr/bin/chsh`
`-rwsr-xr-x 1 root root 50388 15 févr. 2011 /usr/bin/gpasswd`
`-rwsr-xr-x 1 root root 34740 15 févr. 2011 /usr/bin/passwd`
`-rwxr-sr-x 1 root tty 7784 17 juin 2010 /usr/bin/write`

2. ... *pas besoin de corrigé pour cette question...*

3. nouveauterm\$ `mesg y`

4. nouveauterm\$ `tty`
`/dev/pts/4`

⇨ on suppose que le terminal est `/dev/pts/4`

nouveauterm\$ `ls -l /dev/pts/4`
`crw--w---- 1 toto tty 136, 4 20 nov. 12:42 /dev/pts/4`

⇨ le groupe `tty` a le droit d'écriture sur ce terminal

5. voisin\$ `write toto`
`blablabla`
`patati`
`patata`

CTRL-D

⇨ en se lançant, **write** peut indiquer au voisin que son terminal n'est pas ouvert en écriture (et qu'il ne pourra recevoir de réponse sur ce terminal)

Sur le terminal /dev/pts/4 de toto, apparaît le message du voisin :

```
nouveauterm$
Message from voisin@allegro on pts/6 at 12:45 ...
blablabla
patati
patata
EOF
```

6. ... *pas besoin de corrigé pour cette question...*

Corrigé de l'exercice 8

Observation de l'utilité du sticky-bit et du bit set-gid sur les répertoires

1. \$ **ls -ld /tmp**

```
drwxrwxrwt 369 root root 425984 nov  9 16:34 /tmp
```

⇒ le nombre de liens, la taille ou la date varient.

2. ... *pas besoin de corrigé pour cette question...*

3. \$ **ls -ld /tmp/depot**

```
drwxrwxrwx 2 cpb cpb 4096 nov  9 16:37 /tmp/depot
```

⇒ le nombre de liens, la taille ou la date varient.

4. \$ **echo 'hello' > /tmp/toto**

```
$ echo 'salut' > /tmp/depot/toto
```

⇒ on suppose encore que l'utilisateur en session est toto

5. voisin\$ **rm /tmp/toto**

```
rm : supprimer fichier (protégé en écriture) « /tmp/toto » ? y
```

```
rm: impossible de supprimer « /tmp/toto »: Opération non permise
```

⇒ le sticky-bit de /tmp l'empêche de supprimer /tmp/toto qui ne lui appartient pas

```
voisin$ rm /tmp/depot/toto
```

```
rm : supprimer fichier (protégé en écriture) « /tmp/depot/toto » ? y
```

```
voisin$
```

⇒ mais rien ne s'oppose à la suppression de /tmp/depot/toto

6. \$ **rm /tmp/toto**

7. \$ **mkdir /tmp/toto**

```
$ chmod a+rwxt /tmp/toto ou chmod 1777 /tmp/toto
```

```
$ ls -ld /tmp/toto
```

```
drwxrwxrwt 2 toto toto 4096 nov  9 16:54 /tmp/toto
```

8. \$ **cp ~/tp/tpunix/amphigouri.txt /tmp/toto**

9. voisin\$ **echo bablablabla.. > /tmp/toto/fic1**

```
voisin$ echo patati-patata > /tmp/toto/fic2
```

10. \$ **ls -l /tmp/toto**

```
total 12
```

```
-rw-r--r-- 1 toto  toto  1168 nov  9 17:42 amphigouri.txt
```

```
-rw-r--r-- 1 voisin voisin  14 nov  9 17:31 fic1
```

```
-rw-r--r-- 1 voisin voisin  14 nov  9 17:32 fic2
```

⇨ on suppose que le voisin est l'utilisateur voisin du groupe voisin.

```
11. $ chmod g+s /tmp/toto
    $ ls -ld /tmp/toto
    drwxrwsrwt 2 toto toto 4096 nov  9 16:54 /tmp/toto

12. voisin$ echo çacommenceàbienfaire > /tmp/toto/fic3
    voisin$ mkdir /tmp/toto/rep

13. $ ls -l /tmp/toto
    total 20
    -rw-r--r-- 1 toto  toto  1168 nov  9 17:42 amphigouri.txt
    -rw-r--r-- 1 voisin voisin  14 nov  9 17:36 fic1
    -rw-r--r-- 1 voisin voisin  14 nov  9 17:36 fic2
    -rw-r--r-- 1 voisin toto   23 nov  9 17:38 fic3
    drwxr-sr-x 2 voisin toto  4096 nov  9 17:38 rep

14. voisin$ rm /tmp/toto/amphigouri.txt
    rm : supprimer fichier (protégé en écriture) « /tmp/toto/amphigouri.txt » ? y
    rm: impossible de supprimer « /tmp/toto/amphigouri.txt »: Opération non permise
    voisin$
```

⇨ échoue car le fichier ne lui appartient pas, ni le répertoire /tmp/toto

```
15. voisin$ rm /tmp/toto/fic1
    voisin$
```

⇨ réussit sans problème

```
16. $ rm -f /tmp/toto/fic2
    $
```

⇨ réussit sans problème

```
17. $ rm -rf /tmp/toto
    $
```

3 Variables et tableaux

Corrigé de l'exercice 9

Manipulation de variables

```
1. var1=3
2. echo $var1
3. var2="une___suite__de___caractères"
4. echo "$var2"
5. ((var3 = var1 * 4))  ou  let 'var3 = var1 * 4'
6. echo $var3
7. ((var3 &= 7))  ou  let var3\&=7
8. echo $var3
9. unset var3
```

10. `echo $var3`

Corrigé de l'exercice 10

Manipulation de quelques variables prédéfinies

1. `echo $USER`
2. `echo $UID`
3. `echo $HOME`
4. `cd /etc ; cd ~/tp/tpunix`
5. `echo $PWD`
6. `echo $PWD`

Les manipulations qui suivent ont un intérêt pratique très limité ! On les effectue ici pour montrer que bash ne fait pas que mettre à jour certaines variables. Il s'en sert aussi :

7. `toto@allegro:~/tp/tpunix$ HOME=/etc`
`toto@allegro:/home/toto/tp/tpunix$`

➡ en gras est mise en évidence la portion du prompt qui a changé

8. `toto@allegro:/home/toto/tp/tpunix$ cd ~/init.d`
`toto@allegro:~/init.d$`

9. `toto@allegro:~/init.d$ pwd`
`/etc/init.d`

10. `toto@allegro:~/init.d$ cd`
`toto@allegro:~$`

➡ Sans argument, `cd` utilise le contenu de `HOME` pour savoir où se placer.

11. `toto@allegro:~$ pwd`
`/etc`

12. `toto@allegro:~$ HOME=/home/toto`
`toto@allegro:/etc$`

➡ pour l'utilisateur toto. On voit que le prompt change à nouveau

13. `toto@allegro:/etc$ OLDPWD=~/tp/tpunix`

14. `toto@allegro:/etc$ cd -`
`toto@allegro:~/tp/tpunix$`

15. `toto@allegro:~/tp/tpunix$ pwd`
`/home/toto/tp/tpunix`

➡ pour l'utilisateur toto

16. `toto@allegro:~/tp/tpunix$ echo $OLDPWD`
`/etc`

➡ puisque'on a utilisé `cd`, la variable `OLDPWD` a été mise à jour par bash

Corrigé de l'exercice 11

Utilisation des substitutions étendues de variables

1. `echo ${#var2}`
2. `echo ${var2:6:5}`
3. `reffic=/home/tp/tpunix/cigale.txt`
4. `cat $reffic`
5. (a) `echo ${reffic%/*}`
 ⇨ on ignore le dernier slash et tout ce qui suit
- (b) `echo ${reffic##*/}`
 ⇨ on ignore le dernier slash et tout ce qui précède
- (c) `echo ${reffic##*.}`
 ⇨ on ignore le dernier point et tout ce qui précède
- (d) `echo ${reffic%.*}`
 ⇨ on ignore le dernier point et tout ce qui suit
6. `cp $reffic ${reffic%.*}.old`
7. `cp $reffic ${reffic/cigale/fff}`

Les substitutions qui suivent sont surtout utilisées dans des programmes bash, lorsqu'on doit tester et utiliser la valeur d'une variable ou d'un paramètre :

8. `echo ${var3:-inexistante}`
9. `echo ${var3:=inexistante}`
10. `echo $var3`