Tubes de communication

Un tube (**pipe** en anglais) permet de faire communiquer deux processus. Le tube est représenté par une barre verticale (touches [Alt Gr] **6** sur un clavier AZERTY) située entre deux commandes Unix. Le résultat de la commande de gauche va partir dans le tube, tandis que la commande de droite va en extraire les données afin de les traiter.

Les figures 12 et 13 représentent le mécanisme interne associé au tube de communication.

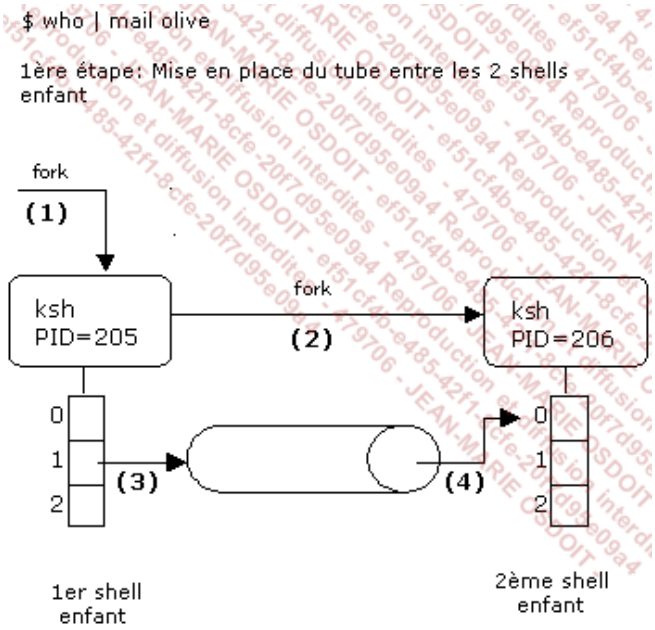

Figure 12 : Mécanisme interne du tube de communication - Première étape

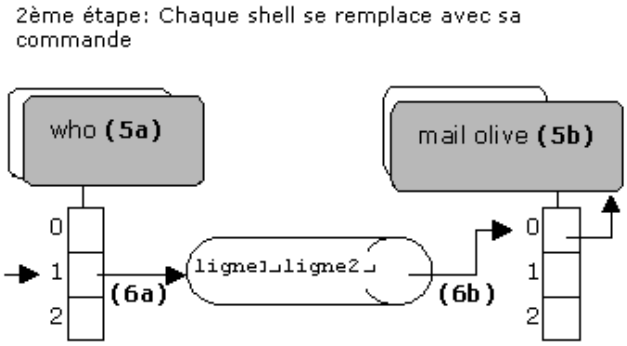

Figure 13 : Mécanisme interne du tube de communication - Deuxième étape

Quelles que soient les commandes présentes de chaque côté du tube, le shell de travail détecte le caractère | sur la ligne de commande et va engendrer un shell enfant **(1)** qui, à son tour, fait de même **(2)**. Le premier shell enfant (PID=205) dissocie sa sortie standard du terminal et la connecte sur l'entrée du tube **(3)**. Le deuxième shell enfant (PID=206) dissocie son entrée standard du terminal et la connecte sur la sortie du tube **(4)**.

Les shells enfants n'ont rien d'autre à faire et chacun va se remplacer avec sa commande **(5a** et **5b)**. Chaque commande commence alors à s'exécuter. Lorsque la commande **`who`** écrit sur sa sortie standard, les messages partent dans le tube **(6a)**. Parallèlement, la commande **`mail`** lit son entrée standard **(6b)**, ce qui provoque l'extraction des données contenues dans le tube.

Quelques remarques importantes

- La sortie d'erreur standard de la commande de gauche ne part pas dans le tube.
- Pour que l'utilisation d'un tube ait un sens, il faut que la commande placée à gauche du tube envoie des données sur sa sortie standard et que la commande placée à droite lise son entrée standard.

1. Commandes ne lisant pas leur entrée standard

Un certain nombre de commandes Unix n'ont aucun intérêt à être placées derrière un tube, car elles n'exploitent pas leur entrée standard. C'est le cas par exemple des commandes suivantes : ls, who, find, chmod, cp, mv, rm, ln, mkdir, rmdir, date, kill, file, type, echo...

2. Commandes lisant leur entrée standard

Les commandes qui lisent leur entrée standard sont facilement identifiables car elles demandent une saisie au clavier.

a. Exemples triviaux

```
$ mail olive
saisie<$I[]mail> clavier
saisie clavier
^d
$
$ write olive
saisie<$I[]write> clavier
saisie clavier
^d
$
```

Ces deux commandes peuvent donc être placées derrière un tube :

```
$ who | mail olive
$ echo "RV pour dejeuner a 13 heures" | write olive
```

b. Cas des filtres

Sous Unix, un certain nombre de commandes sont regroupées sous le nom de filtres. Les plus communes sont : grep, cat, sort, cut, wc, lp, sed, awk... Ces commandes peuvent fonctionner de deux manières :

Première manière

La commande reçoit au moins un nom de fichier en argument.

Exemple

```
$ wc -l /etc/passwd
   46 /etc/passwd
```

Ici, cette commande ne lit pas son entrée standard. Elle n'a aucun besoin de le faire puisqu'elle a reçu un nom de fichier en argument, c'est donc le contenu de ce dernier qu'elle va traiter. Placer cette commande derrière un tube n'est pas impossible mais est inutile car la commande n'extraira jamais les données contenues dans celui-ci.

Exemple

```
$ who | wc -l /etc/passwd
   46 /etc/passwd
$
```

*La commande **who** écrit dans le tube. Son résultat n'apparaîtra donc pas à l'écran. Par contre la commande **wc** se moque éperdument de son entrée standard (puisqu'elle reçoit un fichier en paramètre), donc par extension du contenu du tube. Elle ouvre le fichier **/etc/passwd** en lecture, compte le nombre de lignes du fichier et affiche le résultat sur sa sortie standard.*

Deuxième manière

La commande ne reçoit aucun nom de fichier en argument. Dans ce cas, la commande va traiter les données qui arrivent sur son entrée standard.

Exemple

*La commande **wc** compte le nombre de lignes qui arrivent sur son entrée standard et affiche le résultat sur la sortie standard :*

```
$ wc -l
saisie clavier        (Entrée standard)
saisie clavier        (Entrée standard)
saisie clavier        (Entrée standard)
^d                    (Entrée standard)
   3                  (Sortie standard)
$
```

Il est donc possible de placer cette commande derrière un tube :

```
$ who | wc -l
   4
$
```

Comment savoir si une commande lit son entrée standard ?

Voici deux méthodes qui permettent de savoir si une commande lit son entrée standard :

- L'information est contenue dans le manuel de la commande.

Exemple

*Voici un extrait de la page de manuel de la commande **wc**. On constate que l'argument **[file ...]** est facultatif, ce qui est une première indication.*

```
$ man wc
...
SYNOPSIS
wc [ -c | -m | -C ] [ -lw ] [ file ... ]
...
```

*Un peu plus, loin, se trouve l'explication de l'argument **file** ; si le nom de fichier est omis, la commande lit son entrée standard.*

```
...
file  A path name of an input file. If no file operands
      are specified, the standard input will be used.
...
```

- Une autre possibilité consiste à tester la commande sans donner de nom de fichier en argument :

Premier exemple

Voici une commande qui traite un fichier. Elle ne déclenche pas de lecture de l'entrée standard :

```
$ cut -d':' -f1,3 /etc/passwd
root:0              (Sortie standard)
bin:1               (Sortie standard)
daemon:2            (Sortie standard)
adm:3               (Sortie standard)
...
$
```

Il est donc inutile de placer cette commande derrière un tube :

```
$ echo "1:2:3:4" | cut -d':' -f1,3 /etc/passwd
root:0                      (Sortie standard)
bin:1                       (Sortie standard)
daemon:2                    (Sortie standard)
adm:3                       (Sortie standard)
...
```

Voici la même commande sans le nom du fichier. La commande attend une saisie au clavier :

```
$ cut -d':' -f1,3
1:2:3:4                   (Entrée standard)
1:3                       (Sortie standard)
10:20:30:40               (Entrée standard)
10:30                     (Sortie standard)
100:200:300:400           (Entrée standard)
100:300                   (Sortie standard)
^d                        (Entrée standard)
$
```

Cette commande peut donc être placée derrière un tube :

```
$ echo "1:2:3:4" | cut -d':' -f1,3
1:3
$
```

Deuxième exemple

Voici une autre commande qui traite un fichier. Elle ne déclenche pas de lecture de l'entrée standard :

```
$ file /etc/passwd
/etc/passwd: ASCII text
```

La même commande sans le nom du fichier génère un message d'erreur. Le nom du fichier est donc obligatoire :

```
$ file
Usage: file [-bciknvzL] [-f namefile] [-m magicfiles] file...
Usage: file -C [-m magic]
```

Placée derrière un tube, elle affiche le même message d'erreur. Il est donc inutile d'insister, cette commande ne lit pas son entrée standard :

```
$ cat /etc/passwd | file
Usage: file [-bciknvzL] [-f namefile] [-m magicfiles] file...
Usage: file -C [-m magic]
$
```

Cas particulier de certaines commandes

La majorité des commandes ne se soucient pas de savoir si elles sont placées derrière un tube ou non. Pour une commande donnée, l'action sera toujours la même :

Exemple

***wc -l** lit son entrée standard dans les deux cas :*

```
$ wc -l
$ who | wc -l
```

Quelques commandes font exception à la règle. Elles testent si leur entrée standard est connectée sur la sortie d'un tube ou sur un terminal. C'est le cas de la commande **more** :

Exemples

*La commande **more** reçoit un nom de fichier en argument et pagine son contenu à l'écran. Elle ne lit pas son entrée standard :*

```
$ more /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
....
--Encore--(43%)
```

*Sans le nom du fichier, la commande affiche un message d'erreur (l'argument [filename] est pourtant facultatif !). Ici, **more** ne lit pas son entrée standard :*

```
$ more
Usage: more [-cdflrsuw] [-lines] [+linenumber] [+/pattern] [filename ...].
```

*Le nom du fichier peut être omis lorsque **more** est placée à droite d'un tube. Dans ce cas, elle lit son entrée standard et pagine les lignes qu'elle y extrait :*

```
$ cat /etc/passwd | more
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
....
--Encore--(43%)
$
```

3. Compléments

a. Enchaîner des tubes

Il est possible d'enchaîner plusieurs tubes sur une ligne de commande.

Exemple

Afficher le nombre de connexions de l'utilisateur christie :

```
$ who | grep christie | wc -l
   3
```

b. Dupliquer les sorties

La commande **tee** permet de visualiser un résultat à l'écran et de le conserver également dans un fichier.

Exemples

*La commande **tee** affiche sur sa sortie standard les lignes extraites du tube et les écrit également dans le fichier **listefic**. Si **listefic** existe déjà, il est écrasé :*

```
$ ls | tee listefic
Desktop
```

```
FIC
fichier
$ cat listefic
Desktop
FIC
fichier
$
```

*Le résultat de la commande **date** est affiché à l'écran et concaténé (ajout) au fichier **listefic** existant :*

```
$ date | tee -a listefic
lun jan 27 17:54:21 CET 2003
$ cat listefic
Desktop
FIC
fichier
lun jan 27 17:54:21 CET 2003
$
```

c. Envoyer la sortie standard et la sortie d'erreur standard dans le tube

Exemples

La commande suivante affiche un message d'erreur et une ligne de résultat :

```
$ ls -l f* Z*
ls: Z*: Aucun fichier ou répertoire de ce type
-rw-rw-r--  1 christie cours     110 jan 22 14:21 fichier
$
```

Seule la sortie standard passe dans le tube :

```
$ ls -l f* Z* | tee listefic
ls: Z*: Aucun fichier ou répertoire de ce type (affiché par ls)
-rw-rw-r-- 1 christie cours 110 jan 22 14:21 fichier (affiché par tee)
$ cat listefic
-rw-rw-r--  1 christie cours     110 jan 22 14:21 fichier
$
```

En utilisant la duplication de descripteur (cf. Redirections - Redirections avancées), la sortie 2 est redirigée sur la sortie 1 (terminal) :

```
$ ls -l f* Z* 2>&1 | tee listefic
ls: Z*: Aucun fichier ou répertoire de ce type (affiché par tee)
-rw-rw-r-- 1 christie cours 110 jan 22 14:21 fichier (affiché par tee)

$ cat listefic
ls: Z*: Aucun fichier ou répertoire de ce type
-rw-rw-r--  1 christie cours     110 jan 22 14:21 fichier
$
```