Introduction aux systèmes d'exploitation (IS1)

TP 5 – Entrées / sorties

1 Les redirections

Une commande `UNIX` est une "boîte noire" à laquelle on peut fournir, en plus de ses arguments, un fichier logique nommé *entrée standard* et qui renvoie deux types de réponses éventuelles sur des fichiers logiques appelés respectivement *sortie standard* et *sortie erreur standard*, comme illustré à la figure 1 (on pourra aussi parler de *flots* ou de *flux* d'entrée et de sortie).

FIG. 1 – commande `UNIX`

Quelques exemples :
- La commande `cat` écrit sur la sortie standard ce qu'elle reçoit sur l'entrée standard.
- La commande `date` ne prend rien sur le flot d'entrée et renvoie quelque chose sur le flot de sortie.
- La commande `cd` ne prend rien en entrée et ne renvoie rien en sortie.

Toutes ces commandes peuvent renvoyer des informations sur la sortie erreur standard pour signaler un problème quelconque (mauvaise utilisation de la commande, arguments inexistants, problèmes de droits, etc.).

Par commodité, il est souvent nécessaire de sauver les données fournies par un programme dans un fichier pour pouvoir ensuite les voir en totalité ou les traiter, ou inversement de réutiliser des données qui sont déjà dans un fichier. Par défaut, le canal d'entrée est le clavier et les canaux de sortie et sortie erreur sont l'écran. Mais il est très facile de leur substituer un fichier.

Redirection de la sortie : les symboles `>` et `>>`

Pour les exercices de cette feuille, sauf mention contraire, vous vous placerez dans un répertoire dédié au cours (par exemple un répertoire nommé `IS1` que vous créerez pour l'occasion si ce n'est pas déjà fait).

Exercice 1 – *Le symbole >*

1. Tapez `echo "toto"` dans une fenêtre de terminal. La commande `echo` affiche sur sa sortie standard le message qu'on lui passe en argument.
2. Tapez maintenant `echo "toto"> fic`. Cette fois rien ne s'affiche. Affichez le contenu du répertoire dans lequel vous vous trouvez, vous constaterez qu'il s'y trouve maintenant un fichier `fic`. Affichez son contenu.
3. Vous avez déjà utilisé le mécanisme de redirection avec la commande `echo` lors des précédents TP, pour créer des fichiers. Vous allez voir maintenant qu'il s'applique à toutes les commandes : tapez `ps > fic`. Rien ne s'affiche. Affichez à nouveau le contenu du fichier `fic`.

4. Faites maintenant `date > fic`. Regardez à nouveau le contenu du fichier `fic`.

Concluez sur le rôle de >.

Correction. Le symbole > permet d'écrire la sortie standard de la commande dans un fichier. Le fichier est créé s'il n'existait pas, le contenu initial du fichier est écrasé dans le cas contraire.

Exercice 2 – *Le symbole >>*

- Refaites l'exercice précédent en utilisant à chaque fois >> au lieu de > (par exemple, `ps >> fic`).
- Concluez sur le rôle de >>.

Correction. Le symbole >> permet d'écrire la sortie standard de la commande à la fin d'un fichier, sans effacer le contenu initial du fichier. Là encore, si le fichier n'existe pas, il est créé.

Exercice 3 – *écrire dans un autre terminal*

Sous Unix, les périphériques sont accessible sous forme de liens dans l'arborescence des répertoires, à partir de `/dev`.

1. Lancez la commande `tty`. Elle affiche un chemin fournissant un lien vers le terminal dans lequel elle est exécutée.
2. Affichez les caractéristiques (droits, etc.) du "fichier" renvoyé.
3. Dans un autre terminal, utilisez la commande `echo coucou` en redirigeant sa sortie standard vers le "fichier" renvoyé par la commande `tty`. Le message s'affichera dans le premier terminal, et non celui qui a lancé la commande.

L'entrée standard

Exercice 4 – *Commande* `cat`

Si on ne lui donne pas d'arguments, la commande `cat` prend ce qui lui arrive dans le flot d'entrée et le réécrit en sortie.

1. Lancez `cat` sans argument. La ligne reste vide, le shell attend que vous lui fournissiez des données.
2. Tapez quelques caractères puis frappez la touche entrée. Observez le résultat.

   Correction. La commande traite l'entrée ligne par ligne.

3. Tapez encore quelques lignes, puis pour indiquer au processus la fin des données à traiter, pressez la combinaison de touches *Ctrl-D* (aussi appelé EOF, ou *end of file*) au début d'une ligne vide.
4. Relancez `cat`, tapez une ligne et au lieu d'appuyer sur la touche Entrée, appuyez deux fois sur *Ctrl-D*. La ligne tapée sera recopiée sans saut de ligne à la fin
5. Relancez à nouveau `cat`, tapez une ligne et au lieu d'afficher sur la touche Entrée ou *Ctrl-D*, tapez maintenant `Ctrl-C`. Cette fois, le texte ne sera pas répété. Rappelez-vous le TP sur les processus pour expliquer la différence

   Correction. La combinaison de touches `Ctrl-C` envoie le signal SIGINT au processus qui exécute la commande `cat`, ce qui a pour effet de la tuer avant qu'elle ne copie sur la sortie ce qu'elle a reçu de son entrée.

6. Essayez l'option `-n` de `cat`, toujours sans argument pour que la commande lise son entrée au clavier.

Correction. les lignes recopiées sont numérotées

7. Cherchez dans la page `man` ce que fait l'option `-s` et testez-la.

   Correction. elle supprime les lignes vides consécutives (ça se voit mieux avec l'option -n en plus)

Exercice 5 – *Commandes `head` et `tail`*

Les commandes `head` et `tail` lisent sur leur entrée standard et ne conservent que les premières lignes (pour `head`) et dernières (pour `tail`). Le nombre de lignes conservées est 10 par défaut, ou l'entier passé en argument après l'option `-n`.

1. Lancez la commande `head -3`. Comme dans l'exercice précédent, le shell attend que vous fournissiez des données, que la commande traitera ligne par ligne. Tapez trois lignes de texte. À la troisième ligne, la commande a terminé son travail et le shell vous rend la main pour taper une autre commande.
2. Comparez avec la commande `tail -3`. Bien entendu, cette dernière doit attendre la fin du flot d'entrée pour produire le résultat.
3. Cherchez dans la page `man` à quoi sert l'option `-c` de ces deux commandes et testez-la pour les deux.

   Correction. Elle compte les caractères au lieu des lignes.

4. Pour la commande `tail`, le nombre passé avec les options `-n` ou `-c` peut être précédé d'un signe `+`, pour indiquer qu'il est compté à partir du début et non à partir de la fin. Testez `tail -n +2` et tapez cinq lignes sur le flot d'entrée, pour vérifier que vous avez bien compris.

Exercice 6 – *Commande `wc`*

La commande `wc` (pour *word count*) sert à compter un nombre de caractères, de mots et/ou de lignes qu'elle lit en entrée. On l'utilise soit en lui fournissant comme paramètre le nom d'un fichier, soit avec le flot de l'entrée standard.

1. Lancez `wc` sans arguments. Comme avec `tail`, la commande a besoin d'attendre la fin du flot d'entrée pour pouvoir produire le résultat.
2. Lancez `wc` avec l'option permettant de n'obtenir, en résultat, que le nombre de lignes que vous aurez tapé en entrée (consultez la page `man`).

Redirection de l' entrée : le symbole <

Le symbole < sert à rediriger le flot d'entrée.

Exercice 7

Lancez la commande `cat` sans argument, mais en prenant comme entrée le contenu du fichier `fic` créé précédemment. Comparez le résultat à celui de la commande `cat fic`. (Il n'y a pas de différence).

Exercice 8

Même question que précédemment mais avec la commande `wc`. Relevez la différence de résultats et tentez de l'expliquer.

Correction.

```
wc < fic
wc fic
```

Dans le deuxième cas, la commande ouvre elle-même le fichier, et affiche son nom en plus du résultat. Dans le premier cas, c'est le shell qui ouvre le fichier, par conséquent la commande ne connaît pas son nom et ne peut donc pas l'afficher.

Redirection de la sortie erreur : les symboles `2>` et `2>>`

Attention : ce symbole est valable en `bash`, qui est le shell par défaut sur les machines du Script, et sur de nombreux systèmes Unix. Mais il ne s'étend pas aux autres shells[1].

Exercice 9 – *Rediriger les messages d'erreur*

1. Tapez la commande `ls /bob` (en supposant que `/bob` est n'existe pas (ce qui est très probable). Vous obtenez un message d'erreur.
2. Refaites la même opération en redirigeant la sortie standard dans un fichier `erreur`. Le message s'affiche toujours dans le terminal, et le fichier `erreur` est vide.
3. Tapez ensuite `ls /bob 2> erreur`. Cette fois, rien ne s'affiche, mais le fichier `erreur` créé n'est plus vide. Affichez le contenu de ce fichier.
4. Tapez la commande `dater` (qui n'existe pas), et redirigez la sortie erreur vers le fichier `erreur`. Affichez le contenu du fichier.
5. Refaites les opérations précédentes en remplaçant `2>` par `2>>`.

Correction. Les symboles `2>` et `2>>` permettent d'écrire la sortie d'erreur de la commande dans un fichier en préservant ou non son contenu initial.

Exercice 10 – *Fichier `/dev/null`*

Le fichier `/dev/null` est un "puits sans fond" : on peut écrire dedans tant qu'on le veut et les données sont alors perdues. Il peut servir à jeter par exemple la sortie erreur quand on n'en a pas besoin.

Lancez la commande `ls -R /home`. Vous pouvez constater que vous n'avez pas accès à un certain nombre de répertoires et fichiers. Relancez cette commande en dirigeant la sortie erreur sur `/dev/null`.

Combiner les redirections : `>&`

On peut bien entendu faire plusieurs redirections en même temps :

```
commande > fichier_out 2> fichier_err
commande > fichier_out < fichier_in
commande < fichier_in > fichier_out
```

etc.

On peut aussi vouloir rediriger un flot sur un autre. Les exemples les plus courants étant de vouloir écrire la sortie standard et la sortie erreur dans le même fichier ou encore de vouloir écrire sur la sortie erreur. On utilise alors le symbole `>&` suivi du descripteur concerné[2] : `x>&y` signifie que x est redirigé sur y. Par exemple :

```
commande > fichier 2>&1
echo "message d'erreur">&2
```

Exercice 11 – *Toutes les sorties dans le même fichier*

On veut lancer la commande `ls fic fictoto` (en supposant que `fictoto` n'existe pas) et faire en sorte que la sortie et la sortie erreur soient écrites dans un nouveau fichier `sorties`. Comment faire ? Donnez plusieurs solutions.

Correction.

- `ls fic fictoto > sorties 2>&1`
- `ls fic fictoto 2> sorties >&2`
- `ls fic fictoto >> sorties 2>> sorties`

[1] Par exemple en `ksh` ou en `tcsh`, le symbole équivalent est `>&` qui a une toute autre signification en `bash`.

[2] L'entrée standard correspondant au descripteur `0`, la sortie standard au descripteur `1` et la sortie erreur standard au descripteur `2`.

2 Les tubes et les filtres

On peut vouloir utiliser le résultat d'une commande (sortie standard) pour le réinjecter dans une autre commande (entrée standard) comme indiqué dans la figure 2, sans passer par un fichier régulier intermédiaire.

FIG. 2 – illustration du tube : <*commande 1*> | <*commande 2*>

Pour faire cela, on utilise le symbole | nommé *tube* (*pipe* en anglais) de la façon suivante : <*commande 1*> | <*commande 2*>.

Exercice 12 – *Utilisation de tube*

On veut compter le nombre de processus attachés au terminal courant, grâce aux commandes `ps` et `wc`. Comment faire ? Et si on veut écrire le résultat dans un fichier ?

Correction. `ps | wc -l > nb`

Exercice 13 – *Défilement page par page*

On veut voir tous les processus en cours d'exécution. Si vous lancez la commande `ps -ax`, le résultat défile très vite et on rate le début. Comment faire ?

Correction. `ps -ax | more` ou `ps -ax | less`

Les programmes qui traitent des données provenant de l'entrée standard et produisent un résultat sur la sortie standard sont communément appelés *filtres*. On peut donc utiliser un filtre *avant* un tube et *après* un tube, comme illustré à la figure 3.

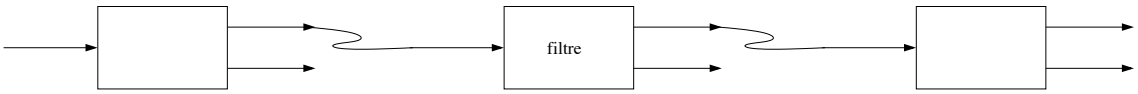

FIG. 3 – un filtre

Voici quelques exemples typiques de filtres :
- `cat` : recopie sur la sortie ce qu'il reçoit sur l'entrée (notamment pour utiliser certaines options de `cat`)
- `head` : recopie les premières lignes
- `tail` : recopie les dernières lignes
- `sort` : trie les données par ordre alphabétique
- `less` : affiche sur la sortie ce qu'il reçoit sur l'entrée, page par page
- `grep` : recherche un motif dans les lignes d'un texte
- `tee` : comme `cat`, mais copie également sur un fichier (le nom de la commande indique un tube qui a la forme de la lettre T)

Les filtres sont particulièrement utiles lorsqu'on les combine à l'aide de tubes. Par exemple, pour trier par ordre alphabétique les lignes d'un fichier `liste.txt` puis les afficher page par page, on utilisera :

```
cat liste.txt | sort | less
```

Exercice 14 – *head et tail*

1. Créez au moyen de la commande `cat` un fichier contenant plus de dix lignes.
2. Affichez la première ligne du fichier.

   Correction. `head -n 1 fichier`

3. Faites précéder la commande de la question précédente par la commande `cat` avec la bonne option pour que la ligne affichée soit numérotée, pour vérifier que vous ne vous êtes pas trompé.

   Correction. `cat -n fichier | head -n 1`

4. Affichez la dernière ligne du fichier, précédée de son numéro de ligne.

   Correction. `cat -n fichier | tail -n 1`

5. Faites de même pour afficher la troisième ligne du fichier, numérotée.

   Correction. `cat -n fichier | head -n 3 | tail -n 1`

Exercice 15 – *Tri*

La commande `sort` affiche ce qu'elle reçoit sur l'entrée standard après avoir trié les lignes.

1. Affichez le contenu du fichier précédent, mais en triant les lignes par ordre alphabétique.

   Correction. `sort fichier | cat`

2. Triez le contenu de votre répertoire personnel par ordre alphabétique.

   Correction. `ls ~ | sort`

3 Tubes nommés.

Les tubes créés par l'opérateur | sont ce que l'on appelle des tubes anonymes. Ce sont des sortes de fichiers "tampons" créés directement par le shell pour une commande donnée. Il est possible de créer manuellement des tubes en leur donnant un nom. Cela permet ensuite de les utiliser dans plusieurs commandes. Ces tubes sont ce que l'on appelle des tubes nommés.

La commande `mkfifo` permet de créer un tube nommé. La syntaxe est la suivante :

```
mkfifo nom
```

où `nom` est le nom du tube que vous voulez créer.

Exercice 16 – *Création de tubes.*

1. Ouvrez un deuxième terminal, dans lequel vous vous placerez dans le même répertoire que dans le premier. On voudrait utiliser ce terminal pour écrire dans un fichier, et le premier terminal pour lire le contenu du fichier au fur et à mesure qu'on écrit dedans.

   Dans le deuxième terminal, vous commencerez à créer, au moyen de la commande `cat` et d'une redirection de sa sortie standard, un fichier `fic` dont le contenu sera ce que vous tapez au clavier, mais sans terminer encore la saisie par *Ctrl-D*. Dans le premier terminal, afficher le contenu du fichier `fic`.
2. Rajoutez deux lignes de saisie dans le deuxième terminal, bien entendu, ces deux lignes ne s'affichent pas dans le premier. Pour les voir, on doit affiche à nouveau tout le contenu (essayez). Terminez la saisie par *Ctrl-D*dans le deuxième terminal.

3. Créez un fichier nommé `tube1` avec la commande `mkfifo`. Regardez ses caractéristiques (droits, taille, type de fichier, etc.), que constatez-vous ?
4. Essayez d'ouvrir ce fichier avec un éditeur, que constatez-vous ? Dans la suite, toutes les lectures et écritures dans les différents fichiers se feront à l'aide de la commande `cat`.
5. Écrivez dans ce fichier sans terminer la commande.
6. Ouvrez un deuxième terminal et lisez le fichier `tube1`. Que constatez-vous ? Essayez d'écrire autre chose dans la commande de la question précédente. Que se passe-t-il ?
7. Que constatez-vous lorsque vous terminez la commande qui écrit dans le tube ?
   Que pouvez-vous en déduire pour la question 1 ?
8. On appelle généralement `écrivain` le programme chargé d'écrire dans le tube et `lecteur` celui qui lit son contenu. Peut-on avoir plusieurs écrivains pour un lecteur ? Essayez avec au moins deux écrivains.
9. Que se passe-t-il si on a plusieurs lecteurs ? Essayez avec un seul écrivain et au moins deux lecteurs. Qu'en déduisez-vous sur le fonctionnement des tubes nommés ?

Correction.

–La lecture et l'écriture dans un fichier classique ne peuvent être fait en même temps.

–Les tubes nommés sont caractérisés par la lettre p devant les droits :

```
prw-r--r-- 1 login groupe 0 19 oct. 11:35 tube
```

–Les tubes ne sont pas fait pour être lus par des programmes tels que `emacs` ou `kwrite`.

–Les tubes nommés permettent à des processus écrivains de communiquer avec des processus lecteurs. Chaque écrivain peut ajouter des messages dans le tube mais un message disparaît dès qu'il est lu. Un message n'est donc lu que par un seul lecteur. De plus les processus lecteurs et écrivains s'attendent mutuellement.

Exercice 17 – *Messagerie instantanée*

Le but de cet exercice est de reproduire à l'aide de tubes nommés un système de messagerie instantanée comme celui fourni par la commande `talk`.

1. Ouvrez deux terminaux. Faites en sorte que ce que vous écrivez dans le premier soit recopié dans le deuxième (vous l'avez déjà fait à un exercice précédent).
2. Recommencez en lançant la lecture du tube en arrière-plan (rappelez-vous le TP sur les processus).
3. Faites en sorte que la communication fonctionne dans les deux sens. (Si vous faites plusieurs essais, pensez à tuer les processus inutiles.)
4. Utilisez la commande `tee` pour enregistrer la conversation dans un fichier `historique`.

Correction.

–Il est nécessaire d'utiliser deux tubes correspondant aux deux sens de la conversation :

`mkfifo f1 f2`

Terminal 1 : `cat < f1 &` puis `cat > f2`

Terminal 2 : `cat < f2 &` puis `cat > f1`

–Pour enregistrer la conversation :

Terminal 1 : `cat < f1 | tee -a historique &` puis `cat | tee -a historique > f2`

Terminal 2 : `cat < f2 &` puis `cat > f1`

Note : il est bien sûr possible aussi d'enregistrer la conversation depuis le terminal 2.

Exercice 18 – *Lancer des commandes dans un autre terminal.*

Le but de cet exercice est d'exécuter des commandes dans un autre terminal. Pour que l'exercice soit plus amusant, vous pouvez vous logger dans un terminal en mode texte.

1. Lancez la commande `ps -l`. La commande elle-même est exécutée dans un processus, qui apparaît dans la liste. Identifiez son processus parent. La commande qui est à l'origine de ce processus parent est le shell dans lequel vous tapez les commandes. Cette commande s'appelle `bash` (sur d'autres systèmes, elle pourrait s'appeler `sh`, `ksh`, `tcsh`, etc.)
2. Ouvrez un autre terminal et vérifiez qu'un nouveau processus `bash` a été lancé.
3. Lancez la commande `bash` depuis un terminal et vérifiez qu'elle a généré un nouveau processus, dont le père est le précédent.
4. Le shell lit les commandes que vous tapez sur son entrée standard. Il est donc possible de rediriger cette entrée. Créez un fichier de nom `commande`, contenant uniquement la ligne suivante :
   `ps -l`
   et lancez la commande `bash` en redirigeant son entrée pour qu'elle lise le fichier `commande`.
   Sur le résultat produit, vous pourrez vérifier l'identité du processus ainsi exécuté, et l'identité de son père. Vérifiez que le processus (fils) n'existe plus.
5. À partir d'un terminal, essayez d'exécuter des commandes dans un autre terminal en utilisant ce que vous avez vu dans l'exercice précédent.
6. Essayez de voir le résultat des commandes dans le premier terminal. Faites en sorte de pouvoir voir aussi les messages d'erreur.

Correction. Pour les deux dernières questions, en supposant que des tubes nommés tube, f1 et f2 ont déjà été créés :

–Premier terminal : `cat > tube`
–Second terminal : `bash < tube`

Pour que le résultat apparaisse dans le premier terminal :

–Premier terminal : `cat < f1 &` puis `cat > f2`
–Second terminal : `bash < f2 > f1`

Pour récupérer aussi les messages d'erreur :

–Premier terminal : `cat < f1 &` puis `cat > f2`
–Second terminal : `bash < f2 > f1 2>&1`