Introduction aux systèmes d'exploitation (IS1)

TP 5 – Entrées / sorties

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).

FIGURE 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 le mécanisme de *redirection* permet de leur substituer un fichier.

Redirection de la sortie : les symboles > et >>

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

Concluez sur le rôle de >.

Exercice 2 – *le symbole* >>

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

Exercice 3 – *écrire dans un autre terminal*

Sous Unix, les périphériques sont considérés comme des fichiers spéciaux, accessibles par des liens situés dans la sous-arborescence `/dev` du système de fichiers.

1. La commande « `tty` » retourne la référence absolue du fichier spécial correspondant au terminal dans lequel elle est exécutée. Affichez les caractéristiques (droits, etc.) de ce fichier.
2. Dans un autre terminal, utilisez la commande `echo coucou` en redirigeant sa sortie standard vers le fichier spécial correspondant au premier terminal. Que se passe-t-il ?

Quelques commandes manipulant l'entrée standard

Exercice 4 – *la 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.
3. Tapez encore quelques lignes, puis pour indiquer au processus la fin des données à traiter, pressez la combinaison de touches (aussi appelée 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 (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 , tapez maintenant (cette fois, le texte ne sera pas répété). Expliquez cette différence de comportement.
6. Essayez l'option « `-n` » de `cat`, toujours sans argument pour que la commande lise son entrée au clavier.
7. Cherchez dans la page `man` ce que fait l'option « `-s` » et testez-la.

Exercice 5 – *les 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 -n 3`

Exercice 6 – *la commande wc*

La commande « wc » (pour *word count*) sert à compter les nombres de caractères, de mots et/ou de lignes qu'elle lit en entrée.

1. Lancez « wc » sans arguments ; comme « 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 ne compter que les lignes.

Redirection de l'entrée : le symbole <

Exercice 7 – *le symbole* <

1. Lancez la commande « cat » sans argument, mais en redirigeant l'entrée vers le fichier fic créé précédemment. Comparez le résultat à celui de la commande cat fic.
2. Même question que précédemment mais avec la commande « wc ». Tentez d'expliquer la différence éventuelle de résultats.

On peut naturellement combiner ces redirections. Nous reviendrons là-dessus à la fin du TP.

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.

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

Pour faire cela, on utilise le symbole | nommé *tube* (*pipe* en anglais) de la façon suivante : *commande1* | *commande2*.

Exercice 8 – *première utilisation d'un 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 ?

Exercice 9 – *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 ?

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.

FIGURE 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 de ses options)
- « `head` » : recopie seulement les premières lignes
- « `tail` » : recopie seulement les dernières lignes
- « `sort` » : trie les données
- « `less` » : affiche 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)
- « `tr` » : recopie l'entrée en remplaçant certains caractères par d'autres
- « `cut` » : sélectionne certaines portions de chaque ligne de l'entrée

Les filtres sont particulièrement utiles lorsqu'on les combine entre eux 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 10 – *retour sur « `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, numérotée à l'aide de « `cat` » pour pouvoir vérifier que vous ne vous êtes pas trompé.
3. De même, affichez la dernière ligne du fichier, précédée de son numéro de ligne.
4. Faites de même pour afficher la troisième ligne du fichier, numérotée.

Exercice 11 – *« `grep` » et la recherche de motifs*

La commande « `grep` » permet de trouver les lignes contenant une chaîne de caractères donnée.

1. À l'aide de « `ps` » et de « `grep` », trouver tous les processus qui exécutent une commande se trouvant dans `/usr/sbin`.

Les caractères `^` et `$` servent à représenter respectivement le début et la fin de ligne.

2. À l'aide de « `ls` » et de « `grep` », lister (uniquement) tous les sous-répertoires de votre répertoire personnel.
3. Listez tous les fichiers de votre *sous-arborescence personnelle* ayant l'extension `.java`.
4. Cherchez quelle option de « `grep` » permet de compter les occurrences du motif cherché. Utilisez-la pour compter les fichiers ordinaires (c'est-à-dire pas les sous-répertoires) de votre sous-arborescence personnelle.
5. Cherchez quelle option de « `grep` » permet d'inverser la sélection des lignes. Utilisez-la pour éliminer les lignes contenant des `/` dans la sortie de `ls -R`.

Le caractère `.` permet de représenter un caractère quelconque.

6. Listez tous les fichiers de la sous-arborescence `/usr/lib` dont le nom contient deux `c` séparés par exactement un caractère.

Le caractère `*` permet de représenter une chaîne de longueur quelconque formée uniquement du caractère qui le précède ; ainsi `a*` représente un mot de longueur quelconque formé uniquement de `a`, tandis que `.*` représente un mot absolument quelconque.

7. Listez tous les fichiers de la sous-arborescence `/usr/lib` dont le nom contient *au moins* deux `c`.
8. Listez tous les fichiers de la sous-arborescence `/usr/lib` dont le nom contient *exactement* deux `c`.
9. Listez tous les fichiers de la sous-arborescence `/usr/lib` dont le nom est formé uniquement de lettres majuscules.

Exercice 12 – *la commande « `tr` »*

La commande « `tr` » permet de modifier le texte reçu en entrée en remplaçant certains caractères par d'autres. Par exemple, `tr 'abc' 'ABC'` met en majuscule les lettres `a`, `b` et `c` (et uniquement celles-là).

Utiliser « `tr` » pour crypter un texte selon la méthode dite *de César*, qui utilise un décalage circulaire des lettres. Par exemple, avec un décalage d'une lettre, chaque lettre est remplacée par celle qui la suit dans l'ordre alphabétique, sauf `z` qui est remplacé par `a`.

Exercice 13 – *la commande « `cut` »*

La commande « `cut` » permet de sélectionner des portions de ligne. Utilisez-la pour améliorer votre réponse à la question 2 de l'exercice 11.

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 14 – *rediriger les messages d'erreur*

1. Tapez la commande `ls /bob` (en supposant que `/bob` n'existe pas, ce qui est très probable). Vous devez obtenir 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. Refaites les opérations précédentes en remplaçant 2> par 2>>.

Exercice 15 – *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`.

Exercice 16 – *Sortie erreur d'un processus*

Tapez la commande `truc 2> erreur` (la commande « `truc` » n'existe pas). Affichez le contenu du fichier. La commande n'existe pas, pourtant le message a été redirigé. Tentez d'expliquer cela.

Combiner les redirections : >&

On peut bien entendu faire plusieurs redirections en même temps, par exemple :

commande > *sortie* 2> *erreur*
commande > *sortie* < *entree*
commande < *entree* > *sortie*

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 un message 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 17 – *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.

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.