Introduction aux systèmes d'exploitation (IS1)
TP nº 8 : les tubes et les filtres

Exercice 1 – *révisions : processus, signaux, redirections*

1. Télécharger depuis moodle le fichier `toto.sh`. Après lui avoir donner les droits en exécution, l'exécuter *en avant-plan*. Quel est le pid du processus correspondant ? Taper *ctrl-C* dans le terminal depuis lequel il a été lancé. Quel message s'affiche ?

2. Envoyer ensuite le signal `SIGTERM` au processus exécutant `toto.sh`. Quel message s'affiche ? Tuer tout de même le processus exécutant `toto.sh`.

3. Relancer `toto.sh` en avant-plan, cette fois en redirigeant sa sortie vers un deuxième terminal. Vérifier que la saisie de *ctrl-C* dans le premier terminal provoque bien un affichage dans le deuxième.

4. Passer le processus exécutant `toto.sh` en arrière-plan, *sans le tuer*.

5. Fermer le premier terminal en cliquant sur la croix. ✍ Quel message s'affiche dans le deuxième terminal ? Le processus exécutant `toto.sh` est-il encore vivant ? Pourquoi ?

6. Envoyer le signal `SIGINT` au processus `toto.sh`. Sa sortie est-elle encore redirigée vers le deuxième terminal ?

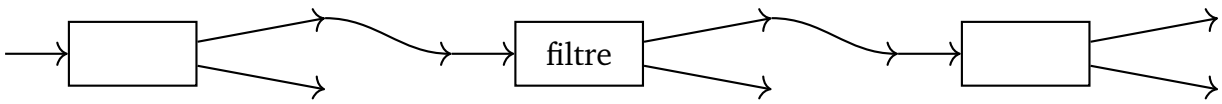
7. Tuer le processus exécutant `toto.sh`, et vérifier qu'il est bien mort avec « `ps` ».

Lors du TP nº 6, nous avons étudié les redirections des sorties et de l'entrée standard vers des fichiers ordinaires. En `UNIX` existe également une redirection qui, étant donné deux commandes « `cmd1` » et « `cmd2` », réinjecte la sortie standard de « `cmd1` » sur l'entrée standard de « `cmd2` » sans passer par l'intermédiaire d'un fichier ordinaire. La syntaxe de cette redirection est

« `cmd1 | cmd2` »

où « | » est un connecteur marquant l'utilisation d'un fichier spécial pour la redirection appelé *tube (anonyme)*. Les flots de « `cmd1` » et « `cmd2` » sont liés par un tube comme indiqué ci-dessous :

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* et *après* un tube, et les enchaîner comme illustré ci-dessous :



Quelques exemples de filtres :

- « `cat` » : recopie sur la sortie ce qu'il reçoit sur l'entrée (utile essentiellement pour certaines de ses options)
- « `head` » : recopie seulement les premières lignes
- « `tail` » : recopie seulement les dernières lignes
- « `less` » : affiche l'entrée page par page
- « `tee` » (« T » en anglais) : dédouble la sortie – sur la sortie standard et sur un (ou plusieurs) fichier(s)
- « `tr` » : recopie l'entrée en remplaçant certains caractères par d'autres
- « `wc` » : compte les lignes, mots et caractères
- « `yes` » : écrit une ligne donnée à l'infini.

Exercice 2 – *première utilisation d'un tube*

1. L'option « `ax` » de « `ps` » affiche la liste de tous les processus en cours d'exécution, mais le résultat défile très vite. ✍ Comment obtenir un défilement page par page en utilisant la commande « `less` » (ne pas hésiter à utiliser « `man` ») ?

> « `wc` »(***word count***) affiche sur la sortie standard des informations numériques à propos d'un ou plusieurs fichiers passés en paramètre (ou l'entrée standard lorsqu'aucun paramètre n'est spécifié).
> Sans option, « `wc` » affiche, dans l'ordre, les nombres de lignes, de mots et d'octets. Avec l'option « `-l` », elle affiche seulement le nombre de lignes, avec l'option « `-w` », le nombre de mots, etc.

2. ✍ Depuis un terminal, lancer quelques tâches en arrière-plan (« `sleep 100 &` », « `xeyes &` », « `xclock &` »...). Comment compter les processus rattachés à ce premier terminal depuis un *autre* terminal ? *(« `tail` » et « `wc` » peuvent être utiles)*

3. ✍ À l'aide des commandes « `ls` » et « `wc` », écrire une ligne de commande qui permet de compter tous les fichiers du répertoire courant dont l'extension est *.java*. Tester la commande dans un répertoire avec de tels fichiers et vérifier que ça fonctionne.

> « `grep` » **(*global regular expression print*)** sans option, permet de sélectionner les lignes d'un fichier qui contiennent un motif[1] passé en paramètre. Le fichier lu par défaut est l'entrée standard.

4. ✍ Compter tous les fichiers de votre *arborescence personnelle* ayant l'extension `.java`. À l'aide de l'option `-e` de « `grep` », compter ceux ayant l'extension `.java` ou `.sh`.

1. Le motif passé en paramètre à « `grep` » est une chaîne de caractères pouvant contenir des *jokers* avec un pouvoir d'expression plus grand que ceux du shell. Nous reviendrons en détail là-dessus la semaine prochaine et n'utiliserons pour le moment que des motifs fixes.

5. ✍ À l'aide d'une option de « `grep` » et de la commande « `xargs` », afficher les fichiers d'extension `.java` du répertoire courant contenant le mot `TODO`.

> « `tr` » **(*translate*)** avec en paramètres deux chaînes de caractères de même longueur, recopie sur la sortie standard le contenu de l'entrée standard tout en remplaçant chaque caractère de la première chaîne par le caractère correspondant de la deuxième. Par exemple « `echo ambassade | tr 'abc' 'ABC'` » écrit sur la sortie standard « `AmBAssAde` ».

Exercice 3 – *la commande « `tr` »*

La méthode de chiffrement dite *de César* consiste à remplacer chaque lettre d'un texte par une autre selon 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`.

1. Déterminer les paramètres adéquats pour que le filtre « `tr` *param1 param2* » chiffre son entrée standard selon la méthode de César avec un décalage de 1.

2. ✍ Créer un fichier `chiffre.sh` contenant le filtre de la question précédente. Après avoir rendu le fichier `chiffre.sh` exécutable, vérifier son bon fonctionnement.

3. ✍ Créer de même un filtre `dechiffre.sh` permettant de déchiffrer l'entrée standard.

4. ✍ Écrire une ligne de commande permettant de confirmer que `dechiffre.sh` effectue bien la modification inverse de `chiffre.sh`.

5. ✍ À l'aide de la commande « `tee` », modifier la ligne précédente pour qu'elle stocke le résultat du chiffrement dans un fichier `message_secret`, en affichant toujours le message déchiffré sur la sortie standard.

Exercice 4 – *créer un gros fichier (un peu répétitif)*

1. Le fichier `/dev/zero` est un fichier spécial qui renvoie un flot de caractères nuls `'\0'` (ASCII NUL, 0x00) lors d'une lecture (à ne pas confondre avec le caractère `'0'` ASCII 0x30). ✍ Comment l'utiliser avec la commande head pour créer un fichier `nul` contenant 10000 caractères (nuls) ?

2. ✍ En utilisant de plus la commande « `tr` », donner un enchaînement de commandes pour créer un fichier `grosfichier` contenant 200 caractères `'a'`.

3. ✍ À l'aide de la commande « `yes` », concaténer 1000 fois au fichier `grosfichier` la ligne « `Répète encore une fois : aaaah...` ».
Et rajouter encore une dernière salve de 200 caractères `'a'` pour finir.

Exercice 5 – *retour sur « `head` » et « `tail` »*

1. ✍ Afficher (uniquement) les 3 dernières lignes du fichier `grosfichier`, numérotées à l'aide de « `cat` » pour pouvoir vérifier que vous ne vous êtes pas trompé(e).
2. ✍ Afficher la septième ligne du fichier, toujours précédée de son numéro de ligne.
3. ✍ Afficher le 579e mot de `grosfichier`. Pour cela, « `tr` » pourra être utile...

Exercice 6 – *un peu de hasard*

Le fichier `/dev/random` est un fichier (très) spécial correspondant à un générateur aléatoire de bits.

1. À l'aide de ce fichier, créer un fichier `alea` contenant un octet aléatoire.

En général, un octet ne représente pas un caractère *imprimable*. La commande « `cat` » est donc inappropriée pour consulter le contenu d'`alea`. Nous allons plutôt utiliser la commande « `od` », qui, comme « `cat` », recopie son entrée sur la sortie, mais en permettant de spécifier le format d'affichage.

2. Déterminer l'option de « `od` » permettant d'afficher la valeur décimale de l'octet contenu dans `alea`.
3. ✍ À l'aide de filtres déjà utilisés durant la séance, écrire une ligne de commande permettant de tirer un entier aléatoire entre 0 et 256 (et de l'afficher en décimal, tout seul sur la sortie standard).

Exercice 7 – *utiliser des filtres sur la sortie erreur*

Il est possible de rediriger un flot de sortie sur l'autre, par exemple pour écrire les deux sorties dans le même fichier, ou écrire un message sur la sortie erreur. On utilise alors le symbole `>&` suivi du descripteur concerné : `x>&y` signifie que `x` est redirigé sur `y`.

Au cours de leur exécution, les processus sollicitent abondamment les fonctions du système d'exploitation en réalisant ce qu'on appelle des *appels système*. La commande « `strace` » permet d'afficher les appels système réalisés par la commande passée en paramètre.

1. Expérimenter « `strace` » sur la commande « `xclock` ». ✍ Déterminer sur quel flot de sortie « `strace` » réalise ses affichages.
2. ✍ Écrire une ligne de commande permettant de compter les appels système effectués par la commande « `touch grosminet` ».
3. ✍ Filtrer les appels système utilisant le mot `titi` lors de l'exécution de « `rm titi` ». Comparer le résultat obtenu selon que le nom passé en paramètre à « `rm` » correspond au nom d'un fichier existant ou non.