TP n°11
Introduction à la Programmation Shell Unix

Cette semaine, nous allons étudier une spécificité de la programmation Shell qui donne toute sa puissance à vos scripts : les redirections ou le fait de pouvoir utiliser le résultat d'une commande pour la commande suivante 1.

1 Entrée et Sorties

Un programme consiste à traiter des données, et à renvoyer des données transformées : il transforme des informations. On fait entrer des données dans un programme et elles ressortent sous une autre forme et dans l'intervalle elles subissent des transformations régulières réalisées par le programme. Pour illustrer nos propos, prenons comme exemple un programme hachoir : si on met un steak à l'entrée, il en ressort de la viande hachée ; si on met des carottes, on récupère des carottes râpées à la sortie. Deux concepts permettent de modéliser cette transformation d'informations : les concepts d'entrée et de sortie. L'entrée, c'est le steak ou les carottes ; la sortie, c'est la viande hachée ou les carottes râpées.

1.1 Entrée

L'entrée d'un programme est l'endroit où l'on va lire les données. Par exemple, quand vous utilisez la commande read var, par défaut, vous récupérez dans la variable var les caractères que vous avez tapé au clavier dans le terminal où vous avez lancé la commande.

```
$ read var
Ceci est un message tapé au clavier
$ echo $var
Ceci est un message tapé au clavier
```

1.2 Sortie Standard et Sortie Standard d'Erreur

Contrairement à l'entrée qui est unique, la sortie d'un programme, c'est-à-dire les messages qu'il renvoie, peut-être de deux: il y a les messages normaux relevant de la transformation d'informations (par exemple la viande hachée ou les carottes râpées), mais il y a aussi des messages d'erreur. Par exemple, en cas de problème d'utilisation du hachoir, le programme va s'arrêter et vous renvoyer un message d'erreur pour vous informer du problème. Il faut donc pouvoir distinguer la **sortie standard**, qui contient les informations traitées, de la **sortie d'erreur**.

Voici un exemple imprimant un résultat sur la sortie standard :

```
$ ls -l
total 0
-rw-r--r-- 1 lavirott None 0 21 janv. 11:08 fichier.txt
```

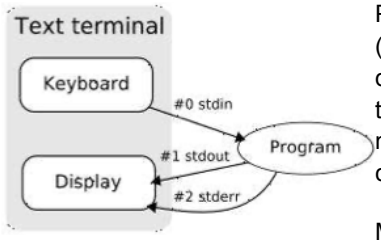
Et un deuxième qui envoie le message sur la sortie standard d'erreur :

```
$ ls -l fichier_inexistant.txt
ls: impossible d'accéder à fichier_inexistant.txt: No such file or directory
```

1.3 Synthèse

Donc un processus Unix (un programme pour faire simple) possède trois voies d'interaction avec l'extérieur

Description	Nom	Descripteur
Entrée Standard	stdin	0
Sortie Standard	stdout	1
Sortie Standard d'Erreur	stderr	2



Par défaut, l'entrée standard (numéro 0) est le clavier, la sortie standard (numéro 1) est l'écran, et la sortie d'erreur (numéro 2) est aussi l'écran. Donc quand on ne spécifie rien, un programme prend son entrée via les caractères tapés au clavier dans le terminal de lancement de la commande et les messages (les résultats de la transformation ou d'erreur) sont affichés dans ce même terminal.

Mais il ne s'agit là que du comportement par défaut, pas d'un comportement obligatoire. Vous pouvez orienter l'entrée ou la sortie des données dans les programmes de manière différente à l'aide des redirections.

2 Redirections

On peut rediriger séparément chacune des trois entrées/sorties standards d'une commande. Cela signifie qu'une commande pourra :

- lire les données à traiter à partir d'un fichier et non du clavier de l'utilisateur,
- écrire les résultats ou erreurs dans un fichier et non à l'écran.

La redirection est mise en œuvre grâce à un ou des caractères spéciaux placés entre deux commandes.

Pour illustrer ce que l'on peut faire avec les redirections, reprenons notre exemple de *hachoir*. Avec une redirection, vous pouvez envoyer la viande hachée dans le réfrigérateur (un fichier) au lieu de juste l'avoir sur le plan de travail à la sortie du hachoir (affichage à l'écran). Oui bien encore, le steak traité par le programme hachoir qui produit de la viande hachée pourrait être directement envoyée dans le programme cuisson en la mettant directement dans la poêle.

Nous allons donc voir successivement :

- comment rediriger la sortie d'une commande vers un fichier ;
- comment ajouter la sortie d'une commande à la fin d'un fichier ;
- comment utiliser un fichier comme entrée d'une commande ;
- comment utiliser la sortie d'une commande comme entrée d'une autre.

2.1 Redirection Sorties

2.1.1 Rediriger la sortie standard dans un fichier : > ou 1>

On peut rediriger la sortie standard d'une commande vers un fichier (caractère « > »). Le résultat de la commande sera placé dans le fichier au lieu de s'afficher sur l'écran.

$ ls –l > list.txt

Le résultat de ls -l ne s'affiche pas à l'écran, mais il est placé dans le fichier *list.txt*. Si le fichier *list.txt* n'existe pas, il est créé. Si le fichier existait déjà, son contenu est effacé pour y mettre le résultat produit par la commande.

On peut alors vérifier que le fichier contient bien le résultat de la commande ls –l.

$ cat list.txt

Si on veut créer ou vider le contenu d'un fichier sans l'effacer, il suffit de faire la redirection de « rien » vers ce fichier :

$ > list.txt

La commande *cat* avec un argument permet de visualiser le contenu d'un fichier. Mais si on ne lui donne pas d'argument et que l'on redirige sa sortie vers un fichier que ce passe-t-il ?

```
$ cat > fichier.txt
Bonjour
^D
$ cat fichier.txt
Bonjour
$
```

Le flux de données nécessaire à la commande *cat* est pris à partir de l'entrée standard (donc le clavier) et est envoyé vers le fichier spécifié après la redirection. Cela permet donc de saisir un contenu succinct dans un fichier. Sans remplacer un éditeur de texte, cette fonctionnalité est souvent utile pour saisir des textes très courts et les envoyer directement vers un fichier.

2.1.2 Ajouter la sortie à la fin d'un fichier : >>

On veut parfois ajouter la sortie d'un programme à un fichier, sans effacer ce qui précède. Or, si on utilise >, il écrasera le contenu du fichier si celui-ci existe. Pour éviter cela, il existe l'outil de redirection >>. Ainsi, si vous tapez plusieurs commandes à la suite redirigée vers le même fichier, les résultats seront ajoutés les uns aux autres dans le fichier :

```
$ ls –l > list.txt
$ ls –l >> list.txt
$ cat list.txt
```

Le fichier *list.txt* contiendra deux fois la liste des fichiers du répertoire (et dans la deuxième liste, le fichier *list.lst* qui a été créé par la première commande si celui-ci n'existait pas).

2.1.3 Redirection de la sortie standard d'erreur : 2>

On a parfois besoin de savoir si une commande a réussi ou non avant d'en lancer une autre. Nous avons vu dans la séance précédente l'utilisation des indicateurs && et || qui permettent respectivement de lancer une commande si (et seulement si) la précédente a réussi ou échoué.

Dans le cas où une commande échoue, en plus de la valeur de retour différente de 0, on peut aussi avoir un message qui s'affiche dans la console. Tous les messages d'erreur que l'on imprime sont en fait envoyés vers la sortie standard d'erreur et non sur la sortie standard (afin de les différencier des données). Toutefois, par défaut, la sortie standard d'erreur est renvoyée sur le terminal.

```
$ ls tmp
ls: tmp vi: Aucun fichier ou répertoire de ce type
$ ls tmp 2> err.txt
$ cat err.txt
ls: tmp vi: Aucun fichier ou répertoire de ce type
```

2.1.4 Redirection vers un autre canal : >&

Il est possible d'avoir besoin de rediriger un canal vers un autre canal, par exemple, rediriger la sortie standard d'erreur vers la sortie standard (on va alors fusionner les deux flux d'information). Pour réaliser cette opération, on utilisera la redirection >& en spécifiant quel est le numéro de canal x que l'on redirige à gauche vers le numéro y de canal à droite. Donc on obtiendra une redirection de la forme x>&y.

Ainsi pour rediriger la sortie standard d'erreur vers la sortie standard, on utilisera la redirection suivante:

```
ls fichiers.txt 2>&1
```

2.2 Redirection Entrée

On peut aussi rediriger l'entrée standard d'une commande (caractère « < »). La commande lira alors dans le fichier au lieu du clavier. Dans le cas de l'utilisation de la commande read que nous avons vu précédemment, cela aura pour conséquence de lire les données à partir du fichier au lieu de récupérer ce qui a été tapé au clavier :

read ligne < fichier.txt

2.3 Synthèse de ce type de redirection

On peut résumer toutes ces premières opérations de redirection par l'exemple suivant:

prog < fichier.txt 1> fichier.out 2> fichier.err

On retiendra que l'on peut ajouter le numéro de canal avant l'opérateur de redirection afin de spécifier le canal avec lequel on travaille. Sachez aussi que nous avons travaillé avec les canaux standards (0 : entrée, 1 : sortie standard, 2 : sortie standard d'erreur), mais on peut aussi travailler avec ces propres canaux (3, 4, 5, ...), même si cela arrive peu souvent, cela peut être utile.

Attention : les redirections étant traitées de gauche à droite, l'ordre des redirections est important.

À noter : vous pouvez maintenant comprendre pourquoi on n'utilise pas les caractères > et < pour faire des tests dans les structures de contrôle (if [$i > 0] par exemple). Ces symboles sont utilisés pour réaliser les fonctions spécifiques mais primordiales en shell qui sont les redirections.

2.4 Tube ou pipe

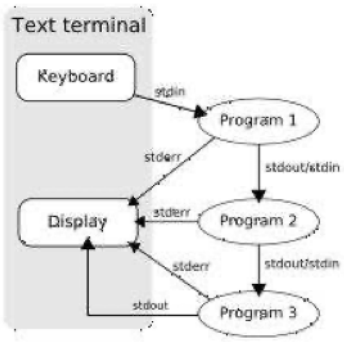
Quand on utilise les redirections vues précédemment, on se retrouve rapidement confronté à un problème : je voudrais que le résultat de la commande 1 soit utilisé comme entrée pour la commande 2.

Pour reprendre l'exemple du hachoir, au lieu de mettre la viande hachée qui sort du hachoir dans une assiette pour ensuite prendre la viande hachée dans l'assiette pour la mettre dans le poêle pour la cuisson, on peut directement mettre la sortie du hachoir pour que la viande hachée tombe directement dans la poêle. On évitera ainsi de faire la vaisselle d'une assiette qui n'a servi qu'à faire le relais.

Une solution qui n'utilise que les opérateurs de redirection précédents nécessite d'utiliser un fichier intermédiaire pour stocker le résultat de la commande 1 puis de relire ce fichier pour l'envoyer sur l'entrée de la commande 2. Mais dans ce cas, il faudra effacer le fichier temporaire utilisé pour stocker les données produites par la première commande après les avoir utilisées dans la deuxième commande (et donc devoir faire la vaisselle d'une assiette qui n'a servi qu'à stocker temporairement la viande hachée).

Voici un exemple de commandes pour compter le nombre de fichiers qui se terminent par l'extension *.txt*:

```
$ ls -l *.txt > tmp.out
$ wc -l < tmp.out
2
$ rm tmp.out
```



On peut se passer du fichier intermédiaire (*tmp.out* dans notre exemple) grâce à un pipe (caractère « | »). Un pipe connecte directement la sortie standard d'une commande sur l'entrée standard d'une autre commande. Et on peut enchaîner comme cela plusieurs commandes les unes à la suite des autres.
Pour donner un exemple, si l'on souhaite récupérer la liste des utilisateurs qui possèdent un compte sur votre machine, trié par ordre alphabétique, on utilisera la commande suivante :

cat /etc/passwd | cut -d ":" - f 1 | sort

On liste le contenu du fichier */etc/passwd* que l'on envoie dans la commande qui permet d'isoler le premier champ dont le séparateur est un : et on trie le résultat par ordre alphabétique.
Ainsi, en une suite de 3 commandes, on obtient très facilement un résultat qui aurait pu être beaucoup plus compliqué à programmer dans un autre langage de programmation.

3 Quelques commandes utiles

Ce petit résumé des commandes utiles à la création de vos scripts ne vous dispense pas de l'indispensable lecture de la page de manuel.

cat	Affiche sur la sortie standard le contenu des fichiers passés en paramètre
more	(ou less, bien plus évolué) à la fin du pipe-line, affiche le résultat page par page, pour laisser le temps de le lire.
cut	découpe et extrait des informations sur une ligne
tr	traduit ou supprime des caractères
wc	compte le nombre de caractères (-c), de mots (-w) et de lignes (-l) de son entrée
sort	lit toutes les lignes de son entrée, les trie, et les écrit dans l'ordre sur sa sortie. Par défaut l'ordre est alphabétique.
uniq	supprime les lignes consécutives identiques

TP n°11
Introduction à la Programmation Shell Unix

Exercices

Nous allons donc faire quelques exercices pour mettre en pratique l'utilisation des redirections et voir ainsi la puissance de ce mécanisme.

Exercice n°1:

Faire un programme *msg_err.sh* qui écrit un premier message « Message normal » vers la sortie standard et un deuxième message « Message d'erreur » vers la sortie standard d'erreur.

Vérifiez que votre programme fait bien ce qu'il faut.

Exercice n°2:

Ecrivez le script *nospace.sh* qui prend exactement un argument, qui vérifie que cet argument est un fichier ou un répertoire, et qui, dans le cas où le nom du fichier ou du répertoire contient au moins un espace, renomme ce fichier ou ce répertoire en remplaçant tous les espaces par le caractère souligné. Par exemple :

$ ls
Bonjour tout le monde.txt
$ ./nospace Bonjour\ tout\ le\ monde.txt
$ ls
Bonjour_tout_le_monde.txt

Quand le script est appelé sur un fichier ou répertoire dont le nom ne contient pas d'espace, il ne fait rien et si le fichier n'existe pas, votre programme affichera un message d'erreur sur la sortie standard d'erreur (*File does not exist*, par exemple).

Exercice n°3:

Écrire un script words.sh qui prend un argument (un nom de fichier) et qui affiche chaque mot du fichier, à raison d'un mot par ligne, chaque mot étant suivi par son nombre d'occurrences dans le fichier. Les mots doivent apparaître triés alphabétiquement. Par exemple, si le fichier *untexte.txt* contient les lignes suivantes :

carotte, patate... carotte, poireau !
haricot; poireau; carotte; patate…
patate+patate=patate

alors on a :

$ ./words.sh untexte.txt
Nombre de mots dans ce texte: 11
Nombre de mots différents dans ce texte: 4
carotte 3
haricot 1
patate 5
poireau 2

Afin de vous aider, voici les étapes à suivre pour décomposer le problème et arriver à le résoudre :

1. Créer le fichier untexte.txt à l'aide de cat et d'une redirection.
2. On testera déjà la manière d'obtenir la liste des mots qui sont contenus dans le fichier. On utilisera pour cela intensivement les pipes pour combiner le résultat de chacune des commandes qui réaliseront successivement les étapes suivantes :
    a. lire le contenu de untexte.txt
    b. remplacer les ponctuations par des retours à la ligne
    c. supprimer les espaces et tabulations
    d. ne retenir que les lignes non vides
    e. trier la liste des mots obtenus
    f. ne retenir qu'un seul de ces mots

3. On stockera le résultat de la liste complète des mots (sans supprimer les doublons) dans une variable list.
4. On calculera et affichera le nombre total de mots dans le texte ainsi que le nombre unique de mots
5. On fera une boucle qui parcourt cette liste de mots pour afficher et compter l'occurrence de chacun de ces mots. Pour vous aider, vous devrez stocker le mot précédent (initialisé à ""). Si celui-ci est différent du mot courant alors on afficher le résultat et on initialise la variable mot précédent et la variable occurrence aux bonnes valeurs, sinon, on augmente juste l'occurrence.

Et si on avait souhaité simplement afficher le nombre d'occurrences de chacun des mots avant le mot, comment auriez-vous procédé ? Quel programme cela aurait-il donné ? (Pensez à lire la page de manuel de la commande uniq).