S. Lavirotte
Polytech'Nice – Sophia
PeiP 1
2019-2020

TD séance n° 8 et 9
Redirections et Traitements par lots

Cette semaine, nous allons étudier une spécificité de la programmation Shell qui donne toute sa puissance aux 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 donc 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 (la donnée) à l'entrée, il en ressort de la viande hachée (la donnée transformée) ; 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 types : 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.

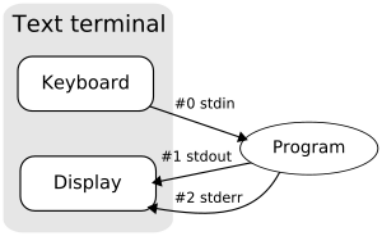
Description	Nom	Descripteur
Entrée Standard	`stdin`	`0`
Sortie Standard	`stdout`	`1`
Sortie Standard d'Erreur	`stderr`	`2`

[1] *Ce cours/TD a été réalisé en réutilisant des données issues de plusieurs sources. Nous tenons donc à remercier Eric Sanchis (IUT de Rodez) mais aussi Roberto Di Cosmo, Xavier Leroy et Damien Doligez, Nicolas George, Baptiste Mélès pour leurs supports de cours ainsi que les illustrations Wikipedia.*

S. Lavirotte
Polytech'Nice – Sophia
PeiP 1
2019-2020

TD séance n° 8 et 9
Redirections et Traitements par lots

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. Vous pouvez orienter l'entrée ou la sortie des données dans les programmes de manière différentes à l'aide des redirections.

2 Redirections

On peut rediriger séparément chacune des trois entrées/sorties/erreurs 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 une assiette (un fichier) au lieu qu'elle ne tombe simplement 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 `poêle` pour directement en réaliser la cuisson et obtenir un steak-haché cuit.

Nous allons donc voire 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
```

TD séance n° 8 et 9
Redirections et Traitements par lots

```
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 saisie 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. 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 travail. 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.

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`.

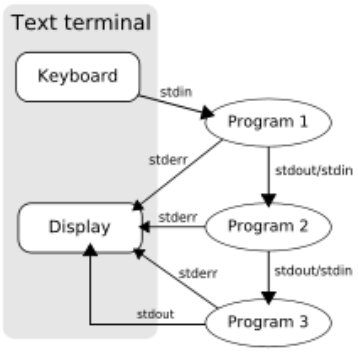
Pour reprendre l'exemple du `hachoir`, au lieu de mettre la viande hachée qui sort du hachoir dans une assiette pour ensuite prendre la viande hachée dans l'assiette pour la mettre dans le `poêle` pour la cuisson, on peut directement mettre la sortie du `hachoir` pour que la viande hachée tombe directement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

TD séance n° 8 et 9
Redirections et Traitements par lots

3.1 Des nouvelles commandes utiles

`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 (`-m`), 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

3.2 Une commande un peu plus en détail : `grep`

Pour finir cette présentation, voici une nouvelle commande qui vous sera utile par la suite : `grep`.

`grep`	Sélectionne les lignes d'un fichier ou d'un flux qui contiennent une expression

`grep` est un programme en ligne de commande de recherche de chaînes de caractères dans un flux de texte (dans un fichier par exemple). La commande s'utilise de la manière suivante :

```
grep [options] chaine_recherchee [fichier1 ...]
```

Le comportement habituel de `grep` est de recevoir en premier paramètre une expression rationnelle, de lire les données (dans un fichier ou une liste de fichier par exemple) et d'écrire les lignes qui contiennent des correspondances avec l'expression rationnelle.

Attention : `grep` recherche une chaîne de caractères et non un mot.

Les options les plus courantes sont les suivantes (mais cela ne dégage pas votre responsabilité de la lecture de la page de manuel) :

Option	Description
`-v`	(*invert*) Affichage des lignes ne contenant pas la chaîne indiquée
`-l`	(*list files only*) Affichage uniquement des noms des fichiers contenant la chaîne recherchée. Les lignes ne sont pas affichées.
`-s`	Les messages d'erreur ne seront pas affichés.

Dans la chaîne de caractère, le symbole `^` représente le caractère de début de ligne et le symbole `$` représente le caractère de fin de ligne.

Exemples d'utilisation de `grep` :

- Rechercher les lignes contenant la chaine de caractères « pat » dans le fichier « mon_fichier.txt » :

  ```
  grep "pat" mon_fichier.txt
  ```

- Rechercher les lignes qui commencent par la chaîne de caractères « pat » dans le fichier « mon_fichier.txt » :

  ```
  grep "^par" mon_fichier.txt
  ```

TD séance n° 8 et 9
Redirections et Traitements par lots

Exercices

Nous allons donc faire quelques exercices pour mettre en pratique l'utilisation des redirections et voir ainsi la puissance de ce mécanisme. Et nous finirons par un exercice montrant la puissance du Scripting Shell à savoir faire du traitement par lot sur un ensemble de fichiers (où sur les données qu'ils contiennent).

Exercice n° 1:

Pour les différentes questions de cet exercice, vous donnerez la commande et indiquerez ce que contient le fichier après l'exécution.

- Redirigez le résultat de la commande « `date` » dans le fichier `sortie.txt`
- Ajoutez au fichier `sortie.txt` créé précédemment le résultat de la commande « `ls -l` »
- Rediriger le résultat des erreurs produites par la commande « `ls *.com` » dans le fichier `erreur.txt`

Exercice n° 2:

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° 3:

A l'aide d'un enchainement de commandes avec des redirections, compter le nombre de dossiers présents dans le dossier courant. Une fois que vous aurez trouvé la bonne solution, faites-en un script `count_dir.sh`.

Modifier ce programme `count_dir.sh` pour passer en paramètre le nom du dossier dans lequel faire ce calcul (c'est une manière d'automatiser ce que l'on vous a demandé de faire au TD3 quand vous avez du compter le nombre de dossier à la racine du système de fichiers).

Exercice n° 4:

Vous allez écrire un script `tri_photos.sh` qui va prendre `2` arguments : le nom d'un dossier contenant des photos `jpg` (vous utiliserez le dossier contenant une cinquantaine de photos que vous avez récupérées dans le fichier de ressource pour ce TD) et le nom d'un dossier dans lequel il va recopier ces photos en les triant et les renommant.

Voici quelques précisions pour créer votre script. Suivez bien ces étapes pour simplifier votre développement

1. Vous commencerez par récupérer la date de numérisation d'une image. Cette information est stockée dans le fichier image lui-même. Vous pouvez récupérer cette information grâce à la commande `exif` en spécifiant la valeur de tag `0x0132`. Faites déjà un test en lançant la commande dans votre interprète de commande avant d'inclure celle-ci-ci dans un script.
2. A partir de cette extraction de l'information, commencez maintenant votre script en stockant dans des variables la date (sous la forme année-mois-jour), l'heure (sous la forme heure-minute-seconde) et l'année. Dans une première version de ce script, vous récupérerez le nom du fichier sur lequel faire le traitement en paramètre et vous afficherez le résultat sous la forme :

```
$ tri_photo.sh 00722_door_1680x1050.jpg
La photo du fichier "00722_door_1680x1050.jpg" a été prise en 2006 (précisement
le 2006-03-16) à 21-12-08
```

3. Modifiez le résultat obtenu précédemment en n'affichant plus le message (mettez-le en commentaire) mais remplacez-le par la création d'un dossier correspondant à l'année de prise de la photo. Ce dossier devra être créé dans le dossier de destination spécifié en deuxième paramètre de votre script.

```
$ tri_photo.sh 00722_door_1680x1050.jpg dest_dir
```

TD séance n° 8 et 9
Redirections et Traitements par lots

4. Ajoutez une copie du fichier dans ce nouveau dossier en la renommant avec un nom du type *année-mois-jour_heure-minute-seconde*.jpg.

5. Enfin nous allons apporter une dernière modification à notre script pour qu'il ne travaille pas sur un fichier mais sur tous les fichiers du dossier passé en premier paramètre. Pour effectuer cette opération, vous allez avoir besoin d'appliquer le traitement spécifiez jusqu'à présent sur l'ensemble des fichiers d'un dossier. Nous n'avons pas vu les structures de contrôle du Shell, mais cela consiste à faire une boucle sur l'ensemble des fichiers. Voici la syntaxe à utiliser pour réaliser cette opération :

```
for file in $(ls $1/*)
do
    ... # la variable file contiendra le nom de fichier suivant à chaque itération
done
```

Et voilà un script qui vous fera économiser des heures de manipulations à la souris et ainsi effectuer pour vous une suite d'opérations répétitives qui peut facilement être automatisées. Vous pourrez aussi vous rendre compte dans le cadre du cours Environnement Informatique 2 que la programmation Shell simplifie ce type de traitements par lots ou sur le contenu de fichier par rapport à beaucoup de langages de programmation (vous obligeant à des opérations intermédiaires pour accéder au contenu d'un dossier ou d'un fichier).

Exercice n° 5:

Créez le fichier `untexte.txt` à l'aide d'une redirection au lieu de le faire avec un éditeur de texte.

Exercice n° 6:

Ecrire 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 apparaitre 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 le comportement suivant :

```
$ ./words.sh untexte.txt
Occurrence des mots :
3 carotte
1 haricot
5 patate
2 poireau
```

Commencez par créer le fichier untexte.txt.

Afin de vous aider, voici les étapes à suivre pour décomposer le problème et arriver à le résoudre. On utilisera pour cela intensivement les pipes pour combiner le résultat de chacune des commandes qui réaliseront successivement les étapes suivantes :

1. lire le contenu de `untexte.txt`
2. sur le contenu du fichier `untexte.txt` (donc le résultat de la commande précédente), remplacer les ponctuations par des retours à la ligne
3. sur le résultat précédent, supprimer les espaces et tabulations
4. sur le résultat précédent, ne retenir que les lignes non vides
5. puis appliquer au résultat un tri sur la liste des mots obtenus
6. et enfin ne retenir qu'un seul de ces mots en cas de doublon dans la liste précédente en affichant le nombre d'occurrences.

Exercice n° 7:

TD séance n° 8 et 9
Redirections et Traitements par lots

On reprend l'exercice `words.sh` pour avoir le résultat suivant :

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

Voici une décomposition du problème :

1. On stocke le résultat de la liste complète des mots (sans supprimer les doublons) dans une variable `list`
2. On calcule et affiche le nombre total de mots dans le texte ainsi que le nombre unique de mots
3. On exécute ensuite le programme obtenu à l'exercice précédent