Redirections

1 - Canaux de communication

Une commande est généralement lancée à partir d'un terminal.

Un terminal c'est :

- Un clavier : un canal d'entrée de données.
- Un écran : un canal de sortie de données.

Un canal d'entrée

Il n'y a qu'un seul canal d'entrée, généralement le clavier. Il s'appelle STDIN (STanDard INput).

La lecture du clavier se fait, par exemple en C, avec scanf(), gets(), getc()... Ces fonctions attendent que l'utilisateur saisisse des choses au clavier, généralement jusqu'à l'appui sur ENTREE.

Deux canaux de sortie

Il existe 2 canaux de sortie. Ils s'appellent STDOUT (STanDard OUTput) et STDERR (STanDard ERRor output). Ils s'agit généralement de l'écran.

En fonctionnement normal, comme ces 2 canaux ont une même destination, l'écran, on ne distingue pas toujours ce qui arrive sur l'un ou l'autre des canaux. Tout finit indifféremment à l'affichage.

L'écriture sur ces canaux se fait avec printf(), puts(), putc()...

Pourquoi distinguer 2 canaux de sortie ?

Pour séparer la sortie des données utiles (les données traitées) de celle des données informatives (erreurs et explications éventuelles).

Quand tout se passe bien, la sortie des erreurs est vide.

Exemple de code C

```
void main()
{
  char nom[40];

  printf("Qui es-tu ? ");          // Sortie : STDOUT
  scanf("%s", nom);                // Entrée : STDIN
  printf("Yo %s !\n", nom);        // Sortie : STDOUT
}
```

Rappels :

- printf() écrit sur STDOUT.
- fprintf(stdout, ...) autre moyen d'écrire sur STDOUT.
- fprintf(stderr, ...) écrit sur STDERR.

2 - Redirection de l'entrée

STDIN - L'entrée standard

Saisir des données au clavier c'est gérable pour de très petites quantités d'informations, mais quand le volume des données grossit, ça n'est plus possible.

Ca requiert la présence de l'utilisateur devant le clavier.

Comment peut-on automatiser l'entrée des données ?
Peut-on simuler la saisie au clavier ?

L'astuce s'appelle la redirection.

On va demander au shell de ne plus relier le clavier au canal d'entrée de la commande, mais d'utiliser le contenu d'un fichier pour alimenter ce canal, comme si on simulait la frappe au clavier d'un utilisateur.

Le contenu du fichier est envoyé directement dans le canal d'entrée, dans STDIN.

Voici comment procéder : commande < fichier

Exemple : wc -l < liste

wc (word count) compte les mots, les lignes de texte et les caractères qui arrivent sur son canal d'entrée. Ici on choisit d'afficher le nombre de lignes (-l).

Comme on alimente son canal d'entrée par le fichier liste, elle affiche donc le nombre de lignes du fichier liste.

Ca fonctionne avec toutes les commandes qui lisent le clavier. La commande ne fait aucune différence, elle ne sait pas si les données proviennent du clavier ou d'un fichier.

La commande continue de faire des scanf() ou des gets() par exemple.

Elle ne sait pas qu'il y a une redirection entrante, c'est le shell qui gère ça pour elle.

3 - Redirection des sorties

STDOUT - La sortie standard

Ce qu'une commande affiche peut représenter beaucoup de lignes et on peut aussi vouloir garder une trace de ce qui s'affiche pour le consulter plus tard ou plus aisément dans un éditeur de texte par exemple.

Une commande peut aussi être lancée automatiquement la nuit par exemple, quand personne n'est devant l'écran pour voir ce qui s'affiche.

On appelle ça des batchs, des traitements automatisés et autonomes.

Comment peut-on conserver la trace, la mémoire de ces affichages ?

L'astuce s'appelle, ici aussi, la redirection.

Mais cette fois-ci, ça se passe dans l'autre sens.

On va demander au shell de ne plus relier le canal de sortie de la commande à l'écran, mais d'écrire dans un fichier ce qui aurait dû s'afficher sur l'écran.

Le contenu du canal de sortie, de STDOUT, est envoyé directement dans un fichier.

Voici comment procéder : commande > fichier

Exemple : ls -l > mes_fichiers

Ca fonctionne avec toutes les commandes qui écrivent à l'écran. La commande ne fait aucune différence, elle ne sait pas si les données sont écrites sur l'écran ou dans un fichier.

La commande continue de faire de printf() par exemple.

Elle ne sait pas qu'il y a une redirection sortante, c'est le shell qui gère ça pour elle.

Attention, > écrase le fichier s'il existe déjà.

Même si la commande ne peut pas s'exécuter (erreur de paramètres ou même commande inexistante), le fichier est quand même créé car le shell commence par ça, par créer/écraser le fichier et ensuite il lance la commande.

Il est possible d'ajouter le résultat de l'affichage du commande à la fin d'un fichier s'il existe déjà.

Il suffit de faire : commande >> fichier

Exemple :

```
date > liste
echo '-------' >> liste
ls -l src/ >> liste
```

STDERR - La sortie des erreurs

La sortie des erreurs est un canal de sortie supplémentaire mis à disposition des commandes pour afficher des informations autres que les données résultant du traitement de la commande.

En général, comme son nom le laisse entendre, il s'agit d'erreurs ou d'avertissements.

Il est possible que ce canal soit aussi utilisé pour afficher, par exemple, des informations sur la progression du traitement pour des commandes qui prennent beaucoup du temps.

Par défaut, les 2 canaux STDOUT et STDERR sont reliés à l'écran. On ne fait alors pas la différence entre les 2, tout s'affiche, au fur et à mesure, sur le même écran.

Mais on peut utiliser une redirection spécifique pour STDERR, et ainsi séparer les 2 canaux. STDERR est toujours le canal de sortie n°2, il faut donc faire :

commande 2> fichier

ou, pour ajouter en fin de fichier :

commande 2>> fichier

Il est possible de rediriger les 2 canaux :

- soit séparément :
  commande > resultat 2> erreurs
- soit dans le même fichier :
  commande 2>&1 > fichier

La syntaxe 2>&1 indique que le canal 2 (STDERR) doit être fusionné avec le canal 1 (STDOUT), et > fichier envoie le canal 1 (+ la fusion du 2) dans le fichier indiqué.

Rappel de syntaxe : commande 2>&1 > fichier

La syntaxe 2>&1 se lit “Envoie aussi les erreurs là où tu envoies déjà le résultat”.

Bash propose aussi une version raccourcie de cette syntaxe : &>. Exemple : commande &> fichier

NB : Ce & n'a rien à voir avec le & de la mise en arrière plan !

A savoir

Lancer une commande qui utilise un fichier fic comme canal d'entrée et qui écrit le résultat dans le même fichier fic, ne peut pas s'écrire :

commande < fic > fic

Le fichier fic sera écrasé avant même d'avoir été consommé par la commande.

De même, avec la redirection <, le shell commence d'abord par ouvrir le fichier. Si le fichier n'existe pas, la commande n'est même pas lancée.

Pour ne pas confondre < et >. Il faut les lire comme si ça dessinait des flèches.

- commande < fic : le contenu du fichier fic “rentre” dans la commande.
- commande > fic : ce qui est produit par commande “rentre” dans le fichier fic.

Exécution en arrière plan

Avec la redirection entrante, un processus s'exécutant en arrière plan ne sera plus bloqué quand il va lire des données car il les lira depuis un fichier et non plus le clavier.

Les redirections sortantes sont aussi bien pratiques pour les processus s'exécutant en arrière plan, ils ne viennent pas polluer l'affichage du shell qui exécute peut-être d'autres commandes pendant ce temps.

Il est donc très fréquent d'utiliser des redirections entrantes et sortantes pour des processus lancés en arrière plan.

4 - Tubes

Sur le principe de base des redirections (entrantes et sortantes), un tube est simplement un moyen de connecter le canal de sortie (STDOUT uniquement) d'une commande avec le canal d'entrée d'une autre commande.

Ceci permet d'éviter la création de fichiers temporaires.

Supposons qu'on veuille compter le nombre d'objets dans le répertoire courant.

On sait déjà :

- Afficher la liste des objets avec ls -l et envoyer le résultat dans un fichier :
  ls -l > liste
- Compter le nombre de lignes dans un fichier en utilisant une redirection :
  wc -l < liste

On voit bien que le point commun de ces 2 commandes, c'est le fichier liste.

La 1ère commande écrit dedans et la seconde lit dedans.

C'est comme si on avait un robinet qui remplit un seau et qu'on vide ensuite le seau dans une cuve.

Ne peut-on pas brancher le robinet sur la cuve ? Bien sûr que oui, on utilise un tuyau et on n'a plus besoin du seau.

En shell, un tel “tuyau” s'appelle un tube (pipe).
Et on écrit ça ainsi :

ls -l | wc -l

Au fur et à mesure que ls -l s'exécute et produit des lignes, cette “production” vient se déverser dans notre commande wc -l qui les compte, et affiche le total quand plus rien n'arrive depuis le tube.

5 - Combinaisons

On peut combiner les redirections et les tubes, sans limite.

cmd1 < fic | cmd2 | cmd3 | cmd4 > res

- Une redirection entrante peut se faire uniquement sur la 1ère commande.
- Une redirection sortante peut se faire uniquement sur la dernière commande.

ATTENTION : lire (cmd1 < fic). Ce n'est pas fic la commande.

Quelques exemples classiques

Affichage paginé d'une longue liste de fichiers :

```
ls -lR | less
```

Idem avec des processus :

```
ps -edf | less
```

Numérotation globale des lignes de plusieurs fichiers :

```
cat promo16 promo17 promo18 | cat -n > promos_3ans
```

Comptage du nombre d'occurrences d'une chaîne dans une série de fichiers :

```
cat *.txt | grep Dupont | wc -l
```

Tri alphabétique des noms du groupe I2 et affichage du premier nom :

```
sort groupe_hideux | head -1
```

Idem avec les 3 derniers du groupe F1 :

```
sort groupe_f1 | tail -3
```

https://commons.wikimedia.org/wiki/File:Thats_all_folks.svg