Entrées / sorties

1. Les redirections

Une commande UNIX est une « boîte noire » à laquelle on fournit des arguments sur 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.

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 évidemment renvoyer quelque chose 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 il est très facile de leur substituer un fichier.

1.1. Redirection de la sortie : > et >>

Exercice 1 (redirection >)

1. Taper la commande `ps` dans un terminal. Consulter le manuel de la commande pour décrire ce qu'elle fait.

2. Taper maintenant `ps > mon_fichier`. Un fichier `mon_fichier` est créé dans le répertoire courant. Que contient-il ?

3. Taper ensuite `date > mon_fichier` et regarder à nouveau le contenu du fichier.

4. Conclure sur le rôle du symbole >, en particulier dans le cas où le fichier qui figure après existe déjà.

Exercice 2 (redirection >>)

1. Taper la commande `who` dans un terminal. Consulter le manuel de la commande pour décrire ce qu'elle fait.

2. Taper maintenant `who >> mon_fichier`, où le fichier en question est celui provenant de l'exercice précédent. Regarder à nouveau le contenu du fichier.

3. Conclure sur le rôle du symbole >>.

1.2. Redirection de l'entrée : <

Le symbole < sert à rediriger le flot d'entrée, ou autrement dit à remplacer une saisie qui serait faite au clavier.

Exercice 3 (commande `cat`)

La commande `cat` prend ce qui lui arrive dans le flux d'entrée et le réécrit en sortie.

1. Lancer `cat` sans argument. L'invite change de forme pour vous permettre de fournir des données. Taper quelques caractères puis appuyer sur la touche Entrée. Observer le résultat.

2. Taper encore quelques lignes, puis pour indiquer au processus la fin des données à traiter, appuyer sur la combinaison de touches Ctrl-D (aussi appelé EOF, ou *end of file*). Que se passe-t-il ?

3. Que se passe-t-il si à la fin d'une ligne contenant des caractères, on presse Ctrl-C au lieu de Ctrl-D ? Expliquer.

4. Lancer maintenant `cat < mon_fichier`. Comment expliquer ce qui se passe ?

Exercice 4 (commande `wc`)

La commande `wc` sert à compter un nombre de caractères, de mots et/ou de lignes. On l'utilise soit en lui fournissant comme paramètre le nom d'un fichier, soit avec le flux de l'entrée standard.

1. Appliquer cette commande au fichier `mon_fichier` précédemment créé en utilisant les deux méthodes.

2. Décrire en détail l'affichage obtenu. On pourra se référer au manuel de la commande `wc`.

1.3. Redirection de la sortie erreur : `2>` et `2>>`

Exercice 5 (rediriger les messages d'erreur)
1. Taper dans un terminal la commande `dater`, qui n'existe pas. On obtient un message d'erreur. Rediriger la sortie standard de cette commande vers un fichier. Regarder le contenu de ce fichier. Que constate-t-on ?

2. En fait, les messages d'erreur ne sortent pas sur le même canal que le résultat des commandes. Recommencer l'opération précédente en redirigeant cette fois avec le symbole `2>`. Que constate-t-on ?

3. Proposer des tests pour comprendre et décrire la signification du symbole `2>>`.

Exercice 6 (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 en n'a pas besoin.

1. Lancer la commande `ls -R /home`. On pourra constater qu'on n'a pas accès à un certain nombre de répertoires et fichiers.

2. Relancer la commande en dirigeant la sortie erreur sur `/dev/null`. Quel est l'intérêt ?

1.4. Combiner les redirections : `>&`

On peut bien entendu faire plusieurs redirections en même temps :

- `commande > fichier_out 2> fichier_err`
- `commande > fichier_out < fichier_in`
- `commande < fichier_in > fichier_out`
- etc.

On peut aussi vouloir rediriger un flux 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 sur la sortie erreur. On utilise alors le symbole `>&` suivi du descripteur concerné. Par exemple :

- `commande > fichier 2>&1`
- `echo "message d'erreur" >>\&2`
- `echo "message d'erreur" >&2`

Exercice 7 (sorties dans un même fichier)
On veut lancer la commande `ls toto tutu` en supposant que le fichier `tutu` n'existe pas, et faire en sorte que la sortie et la sortie erreur soient écrites dans un nouveau fichier. Comment faire ? Proposer au moins deux solutions.

2. Les tubes

On peut vouloir utiliser le résultat d'une commande (sortie standard) pour le réinjecter dans une autre commande (entrée standard), sans passer par un fichier régulier intermédiaire.

Pour faire cela, on utilise le symbole `|` nommé tube (*pipe* en anglais) de la façon suivante :
`commande 1 | commande 2`.

Exercice 8 (compter les fichiers d'un répertoire)
On veut compter le nombre de fichiers ou dossiers d'un répertoire donné grâce aux commandes `ls` et `wc`. Comment faire ? Et si l'on veut écrire le résultat obtenu dans un fichier ?

Exercice 9 (défilement page par page)
On veut lister toutes les commandes qui figurent dans le répertoire `/usr/bin`. Si l'on utilise la commande `ls`, le résultat défile très vite et on rate le début. Comment faire ?