Chevron >

- Objet : commandes de redirection
- Niveau requis :
  débutant, avisé
- Commentaires : *Par redirection, on entend la possibilité de rediriger l'affichage de l'écran vers un fichier ou tout autre périphérique...*
- Débutant, à savoir : Utiliser GNU/Linux en ligne de commande, tout commence là !.
- Suivi :
  - Création par smolski le 16/08/2011
  - Testé sur squeeze par **smolski** le 16/10/2012
- Commentaires sur le forum : Lien vers le forum concernant ce tuto [1)]

Il existe une commande toute simple pour créer un fichier vide ou bien pour vider un fichier existant de son contenu.

On peut se placer

tout d'abord dans le dossier désiré (pour ce faire voir la commande cd), puis on tape la commande suivante :

```
> nomdufichier
```

Ou, de façon plus générale

on tape :

```
> /chemin/du/repertoire/nomdufichier
```

C'est-à-dire que l'on crée le “nomdufichier” depuis son répertoire personnel, sans s'être déplacé au niveau du répertoire dans lequel on veut placer “nomdufichier”.

Si le fichier existe :

```
 son contenu est effacé.
```

Si le fichier n'existe pas :

```
 il est créé vide.
```

Complément

Pour créer un fichier, les commandes > et commande touch sont équivalentes.

La touch sert avant tout à mettre à jour la date et l'heure d'accès pour la dernière modification d'un ou plusieurs fichiers selon la date actuelle.

Dans le cas d'un seul fichier, s'il n'existe pas, il sera alors créé.

La subtilité de la commande > c'est qu'elle permet de vider rapidement un fichier.

Avec la commande `touch` tu feras :

```
rm fichier | touch /fichier
```

Avec la commande >, tu feras simplement :

```
> /fichier
```

Ce sont bien sûr des détails, mais cela nous donne une façon différente de procéder.

Libre à chacun de choisir la commande qu'il désire (ou de ne pas utiliser de commande et passer par les applications graphiques, ce que feront beaucoup de débutants).

Merci Martin pour cette explication ! 😀

Corollaire rigolo

Prenons un fichier d'environs 100Mo.

```
dd if=/dev/zero of=fichier bs=1 count=1 seek=100M
```

retour de la commande

```
1+0 enregistrements lus
1+0 enregistrements écrits
1 octet (1 B) copié, 4,4267e-05 s, 22,6 kB/s
```

```
ls -lhi
```

retour de la commande

```
>total 4,0K
1163 -rw-r--r-- 1 user user 101M oct.  16 20:34 fichier
```

Vider le fichier se fait aussi simplement que ça :

```
> fichier
```

```
ls -lhi
```

retour de la commande

```
total 0
1163 -rw-r--r-- 1 fgivors fgivors 0 oct.  16 20:33 fichier
```

L'intérêt de vider un fichier de cette manière, par rapport à un rm et un touch, c'est que l'on conserve le numéro d'inode (le numéro marqué à gauche).

Autrement dit : il n'y a pas **création ni suppression** de fichier.

Merci captnfab pour ce corollaire ! 😀

Redirection

Les redirections sont l'une des plus importantes possibilités offerte par le shell.

Par redirection, on entend la possibilité de rediriger l'affichage de l'écran vers un fichier, une imprimante ou tout autre périphérique, les messages d'erreurs vers un autre fichier, remplacer la saisie clavier par le contenu d'un fichier.

Unix utilise des canaux d'entrées/sorties pour lire et écrire ses données.
Par défaut le canal d'entrée est le clavier, et le canal de sortie, l'écran.
Un troisième canal, le canal d'erreur, est aussi redirigé vers l'écran.

Il est donc possible de rediriger ces canaux vers des fichiers, ou du flux texte de manière transparente pour les commandes Unix.

Il est aussi important de savoir dans quel sens le shell interprète les redirections.
Les redirections étant en principe en fin de commande, le shell recherche d'abord les caractères « <, >, » » en fin de ligne.

Préparation

Créer[2)] les éléments nécessaires pour réaliser les exemples de ce tuto :

```
mkdir dossier1 dossier2
```

```
vim dossier1/test1.txt
```

Écrivez :

```
fichier de test1
```

```
vim dossier2/test2.txt
```

Écrivez :

```
fichier de test2
```

Entrée

Les commandes qui attendent des données ou des paramètres depuis le clavier peuvent aussi en recevoir depuis un fichier, à l'aide du caractère inverse :

```
 <
```

Un exemple avec la commande wc (word count) qui permet de compter le nombre de lignes, de mots et de caractères d'un fichier.

```
wc dossier1/test1.txt
```

retour de la commande

```
 1  3 17 dossier1/test1.txt
```

Ces trois nombres signifie :

1. “1” : est le nombre de lignes.
2. “3” : est le nombre de mots.
3. “17” : est le nombre d'octets.

options intéressantes :

- -l : compte le nombre le lignes uniquement ;
- -w : compte le nombre de mots uniquement ;
- -c : compte le nombre d'octets uniquement ;
- -m : compte le nombre de caractères uniquement.

Utilisation avec “ < ” :

```
< ~/dossier1/test1.txt wc    # retour :  1  3 17
```

ou

```
wc < ~/dossier1/test1.txt   # retour :  1  3 17
```

Pour la commande cat voir : cat

Sortie

Sortie simple

Première commande dans le terminal :

```
ls -l dossier1
```

retour de la commande

```
total 4
-rw-r--r-- 1 user user 17 oct.  16 18:31 test1.txt
```

On se sert du caractère « > » pour rediriger la sortie standard (celle qui va normalement sur écran). On indique ensuite le nom du fichier où seront placés les résultats de sortie.

```
ls -l dossier1 > resultat.txt
```

```
cat resultat.txt
```

retour de la commande

```
total 4
-rw-r--r-- 1 user user 17 oct.  16 18:31 test1.txt
```

1. Si le fichier n'existe pas, il sera créé.
2. S'il existe, son contenu sera écrasé, même si la commande tapée est incorrecte.

Le shell commence d'abord par créer le fichier puis exécute ensuite la commande.

Seconde commande :

```
ls -l dossier2 > resultat.txt
```

```
cat resultat.txt
```

retour de la commande

```
total 4
-rw-r--r-- 1 user user 17 oct.  16 18:31 test2.txt
```

On voit que l'écriture précédente a été remplacée.

Continuons avec une troisième commande :

```
ls -l dossier3 > resultat.txt
```

```
cat resultat.txt
```

Le contenu du répertoire a été remplacé par rien, puisque la commande ne donnait aucune écriture à transcrire, il n'y avait pas de dossier3 !

Sorties ajoutées

Pour ne rien effacer et/ou ajouter des données à la suite du fichier resultat.txt on utilise la double redirection :

```
 >>
```

Le résultat est ajouté à la fin du fichier. Exemple :

```
ls -l dossier1 > resultat.txt
```

```
ls -l dossier2 >> resultat.txt
```

```
cat resultat.txt
```

retour de la commande

```
total 4
-rw-r--r-- 1 user user 17 oct.  16 18:53 test1.txt
total 4
-rw-r--r-- 1 user user 17 oct.  16 18:50 test2.txt
```

Les canaux standards

On peut considérer un canal comme un fichier, qui possède son propre descripteur par défaut, et dans lequel on peut ou lire ou écrire.

1. Le canal d' entrée standard se nomme « stdin » et porte le descripteur 0.
2. Le canal de sortie standard se nomme « stdout » et porte le descripteur 1.
3. La canal d' erreur standard se nomme « stderr » et porte le descripteur 2.

On peut ainsi rediriger les canaux de sortie 1 et 2 vers un autre fichier.

```
rmdir dossier3
```

retour de la commande

```
rmdir: échec de suppression de « dossier3 »: Aucun fichier ou dossier
de ce type
```

```
rmdir dossier3 2>error.log
```

```
cat error.log
```

retour de la commande

```
rmdir: échec de suppression de « dossier3 »: Aucun fichier ou dossier
de ce type
```

Sorties dirigées

On pourrait souhaiter enregistrer les erreurs dans un fichier à part pour ne pas les oublier et pour pouvoir les analyser ensuite. Pour cela, on utilise l'opérateur :

```
 2>
```

Sorties doubles

Faisons une seconde redirection avec une erreur en utilisant le dossier3 inexistant à la fin d'une commande, par exemple :

```
ls -l dossier3 > resultat.txt 2> error.log
```

Il y a deux redirections ici :

1. > resultat.txt : redirige le résultat de la commande (sauf les erreurs) dans le fichier resultat.txt. C'est la sortie standard ;
2. 2> error.log : redirige les erreurs éventuelles dans le fichier error.log. C'est la sortie d'erreurs.

Résultat dans le fichier **resultat.txt** :

```
cat resultat.txt
```

Il est devenu vierge.
Et dans le fichier de redirection error.log :

```
cat error.log
```

retour de la commande

```
ls: impossible d'accéder à dossier3: Aucun fichier ou dossier de ce
type
```

L'erreur s'est bien redirigée pour s'écrire dans le fichier **error.log**.

Sorties doubles sauvegardées

Pour ajouter une éventuelle redirection dans le fichier error.log en conservant l'intégrité des écritures dans nos 2 fichiers de redirection :

```
 resultat.txt
 error.log
```

Nous devons utiliser la double redirection en sortie pour chacun d'eux.

Réécrivons et vérifions le fichier resultat.txt :

```
ls -l dossier1 > resultat.txt
```

```
cat resultat.txt
```

retour de la commande

```
total 4
-rw-r--r-- 1 user user 17 oct.  16 18:53 test1.txt
```

Utilisons maintenant la double direction en sortie des 2 fichiers :

```
ls -l dossier3 >> resultat.txt 2>> error.log
```

Vérifions chacun des fichiers :

```
cat resultat.txt
```

retour de la commande

```
total 4
-rw-r--r-- 1 user user 17 oct.  16 18:53 test1.txt
```

```
cat error.log
```

retour de la commande

```
ls: impossible d'accéder à dossier3: Aucun fichier ou dossier de ce
type
```

Les erreurs se sont bien ajoutées dans le fichier **error.log** et le fichier **resultat.txt** n'a pas subi de modification. 😀

Sorties fusionnées

Il est possible de fusionner les sorties dans un seul et même fichier.Il faut utiliser le code suivant :

```
 2>&1.
```

Cela a pour effet de rediriger toute la sortie d'erreurs dans la sortie standard[3].

```
ls -l dossier4 > resultat.txt 2>&1
```

```
cat resultat.txt
```

retour de la commande

```
ls: impossible d'accéder à dossier4: Aucun fichier ou dossier de ce
type
```

Impec nous avons l'erreur redirigée en écriture dans notre répertoire resultat.txt

Sorties liées

Pour préserver le répertoire de ses précédentes écritures il suffit de doubler la sortie ainsi :

```
ls -l dossier5 >> resultat.txt 2>&1
```

```
cat resultat.txt
```

retour de la commande

```
ls: impossible d'accéder à dossier4: Aucun fichier ou dossier de ce
type
ls: impossible d'accéder à dossier5: Aucun fichier ou dossier de ce
type
```

Entrée - Sortie

On peut aussi utiliser à la fois les deux types de redirection.

```
wc < resultat.txt > comptes.txt
```

```
cat comptes.txt
  2  24 170
```

On peut aussi se reporter à la commande « tee ».

Filtres

Un filtre (ou une commande filtre) est un programme sachant écrire et lire des données par les canaux standards d'entrée et de sortie. Il en modifie ou traite éventuellement le contenu. wc est un filtre.

Nous nous attarderons sur quelques filtres plus tard, mais en voici quelques uns :

1. more (affiche les données page par page)
2. sort (tri des données)
3. grep (critères de recherche)
4. pipe ou tube (|)

Lien et remerciement

Le site du zero :

- Les flux de redirection du site zero

Merci aussi à **artheface** *et* **melodie** *de nous avoir offert le tuto en pdf éclairant celui-ci !* 😃

À suivre...

1)
N'hésitez pas à y faire part de vos remarques, succès, améliorations ou échecs !
2)
vim
3)
Traduction pour l'ordinateur : « envoie les erreurs au même endroit que le reste »