Les flux de redirection

Vous devriez maintenant avoir l'habitude d'un certain nombre de commandes que propose la console de Linux. Le fonctionnement est toujours le même :

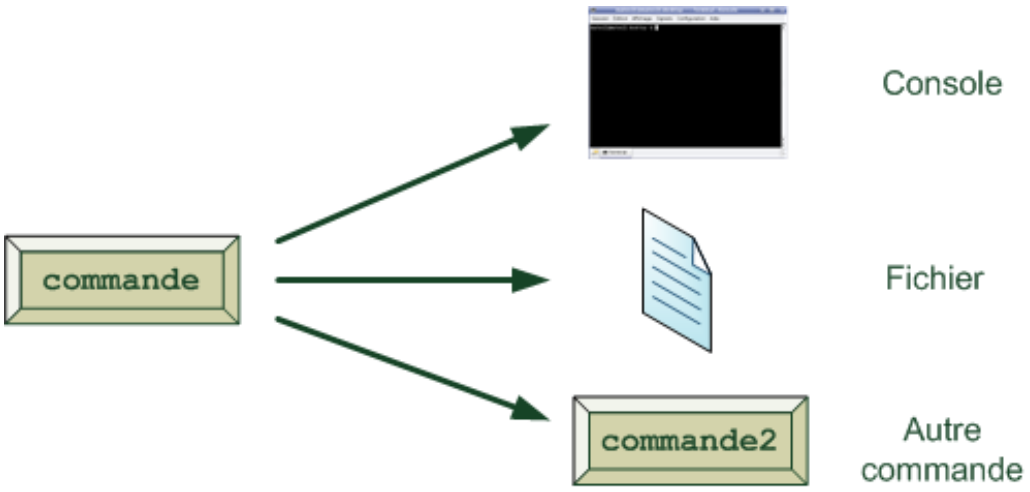
1. vous tapez la commande (par exemple `ls`) ;
2. le résultat s'affiche dans la console.

Ce que vous ne savez pas encore, c'est qu'**il est possible de rediriger ce résultat**. Au lieu que celui-ci s'affiche dans la console, vous allez pouvoir l'envoyer ailleurs : dans un fichier ou en entrée d'une autre commande pour effectuer des « chaînes de commandes ».

Grâce à ce chapitre sur les flux de redirection, vous allez beaucoup gagner en maîtrise de la ligne de commandes !

Dans ce chapitre, nous allons découvrir qu'il est possible de rediriger le résultat d'une commande ailleurs que dans la console.
Où ? Dans un fichier, ou en entrée d'une autre commande pour « chaîner des commandes ». Ainsi, le résultat d'une commande peut en déclencher une autre !
Comment ? À l'aide de petits symboles spéciaux, appelés *flux de redirection*, que vous allez découvrir dans ce chapitre.

Le principe peut être résumé dans le schéma de la figure suivante.



Jusqu'ici, nous n'avons donc exploité que la première possibilité (celle par défaut) : afficher le résultat dans la console. Il nous reste donc bien d'autres techniques à découvrir !

Les flux de redirection constituent une composante essentielle de la console sous

Linux et ce, depuis l'époque d'Unix. Ils vont très certainement changer votre façon de « voir » comment la console fonctionne et démultiplier votre contrôle sur les commandes que vous lancez. C'est dire si ce chapitre est important !

Je vais donc d'abord vous demander d'être encore plus attentifs que d'habitude. Non pas que le chapitre soit réellement « compliqué », mais il **doit** bien être compris pour que vous puissiez suivre le reste du livre convenablement.
Au pire des cas, vous pourrez toujours revenir lire ce chapitre si vous avez un trou de mémoire sur les notions que vous y avez apprises. ;-)

Préparatifs

La manipulation la plus simple que nous allons voir va nous permettre d'**écrire le résultat d'une commande dans un fichier**, au lieu de l'afficher bêtement dans la console.

Prenons une commande au hasard. Vous vous souvenez de `cut`, que nous avons appris dans le chapitre précédent ?

Nous avions travaillé sur un petit fichier de type « CSV » que les tableurs peuvent générer.
Ce sont les notes des élèves d'une classe à un contrôle :

```
Fabrice,18 / 20,Excellent travail
Mathieu,3 / 20,Nul comme d'hab'
Sophie,14 / 20,En nette progression
Mélanie,9 / 20,Allez presque la moyenne !
Corentin,11 / 20,Pas mal mais peut mieux faire
Albert,20 / 20,Toujours parfait
Benoît,5 / 20,En grave chute
```

Si vous ne l'aviez pas déjà fait dans le chapitre précédent, je vous recommande d'enregistrer ce fichier dans un éditeur de texte (comme Nano) en récupérant le contenu ci-dessus à l'aide du code web. Enregistrez le tout sous le nom `notes.csv`.

La commande `cut` nous avait permis de « couper » une partie du fichier et d'afficher le résultat dans la console. Par exemple, nous avions demandé à `cut` de prendre tout ce qui se trouvait avant la première virgule afin d'avoir la liste des noms de tous les élèves présents à ce contrôle :

```
$ cut -d , -f 1 notes.csv

Fabrice
Vincent
Sophie
Mélanie
Corentin
Albert
Benoît
```

Ce résultat s'est affiché dans la console. C'est ce que font toutes les commandes par défaut… à moins que l'on utilise un flux de redirection !

> : rediriger dans un nouveau fichier

Supposons que nous souhaitions écrire la liste des prénoms dans un fichier, afin de garder sous le coude la liste des élèves présents au contrôle.
C'est là qu'intervient le petit symbole magique **>** (appelé **chevron**) que je vous laisse trouver sur votre clavier (ceux qui font du HTML le connaissent bien. ;-).

Ce symbole permet de rediriger le résultat de la commande dans le fichier de votre choix. Essayez par exemple de taper ceci :

```
cut -d , -f 1 notes.csv > eleves.txt
```

Regardez la fin de la commande. J'y ai rajouté la petite flèche > qui redirige la sortie de la commande dans un fichier.
Normalement, si vous exécutez cette commande, **rien ne s'affichera dans la console**. Tout aura été redirigé dans un fichier appelé `eleves.txt` qui vient d'être créé pour l'occasion dans le dossier dans lequel vous vous trouviez.

Je le rappelle au cas où : sous Linux, on se moque pas mal de l'extension des fichiers. J'aurais très bien pu créer un fichier sans extension appelé `eleves`. Ici j'ai rajouté un « `.txt` » pour ne pas dérouter ceux qui viennent de Windows, mais il faudra vous habituer à travailler avec des noms de fichiers parfois sans extension.

Faites un petit `ls` (ou `ls -l`, comme vous préférez) pour voir si le fichier est bien présent dans le dossier :

```
$ ls -l
total 20
-rw-r--r-- 1 mateo21 mateo21  91 2008-04-19 19:36 doublons.txt
-rw-r--r-- 1 mateo21 mateo21  56 2008-09-26 12:01 eleves.txt
-rw-r--r-- 1 mateo21 mateo21  35 2008-04-
19 17:06 fichier_trie.txt
-rw-r--r-- 1 mateo21 mateo21  20 2008-04-19 19:03 nombres.txt
-rw-r--r-- 1 mateo21 mateo21 253 2008-09-26 12:01 notes.csv
```

Comme vous pouvez le voir, un fichier vient bien d'être créé !
Vous pouvez l'ouvrir avec Nano ou encore l'afficher dans la console avec la commande `cat` (pour afficher tout d'un coup s'il est court) ou `less` (pour afficher

page par page s'il est long).

Attention : si le fichier existait déjà il sera écrasé sans demande de confirmation !

Parfois, vous ne voulez ni voir le résultat d'une commande ni le stocker dans un fichier. Dans ce cas, l'astuce consiste à rediriger le résultat dans `/dev/null`. C'est un peu le « trou noir » de Linux : tout ce qui va là-dedans disparaît immédiatement.
Par exemple : `commande_bavarde > /dev/null`

>> : rediriger à la fin d'un fichier

Le double chevron >> sert lui aussi à rediriger le résultat dans un fichier, mais cette fois à la fin de ce fichier.

Avantage : vous ne risquez pas d'écraser le fichier s'il existe déjà. Si le fichier n'existe pas, il sera créé automatiquement.

Normalement, vous devriez avoir créé un fichier `eleves.txt` lors des manipulations précédentes. Si vous faites :

```
cut -d , -f 1 notes.csv >> eleves.txt
```

… les noms seront ajoutés à la fin du fichier, sans écraser le résultat précédent.

Bon, du coup, on a des noms en double maintenant :

```
$ cat eleves.txt
Fabrice
Mathieu
Sophie
Mélanie
Corentin
Albert
Benoît
Fabrice
Mathieu
Sophie
Mélanie
Corentin
Albert
Benoît
```

Heureusement, vous connaissez les commandes `sort` et `uniq` qui peuvent vous permettre de faire un peu de ménage là-dedans. Je vous laisse supprimer les doublons.
N'oubliez pas qu'il faut que le fichier soit trié pour que la commande `uniq` fonctionne !

Quand utilise-t-on le double chevron pour mettre le résultat à la fin d'un fichier ?

Personnellement, j'ai des commandes qui s'exécutent automatiquement à certaines heures (on verra comment faire ça plus tard). Comme je ne suis pas devant mon ordinateur lorsque ces commandes s'exécutent, j'enregistre un log de ce qui s'est passé dans un fichier :

```
macommande >> resultats.log
```

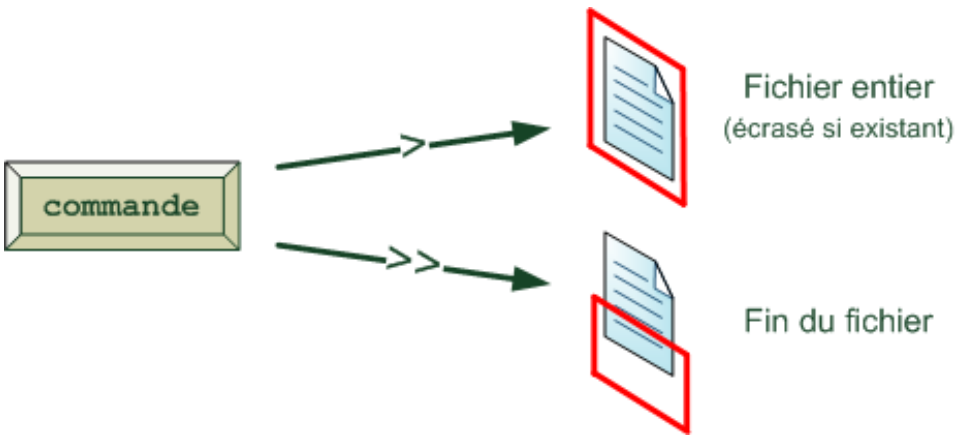
Grâce à ça, si j'ai un doute sur ce qui a pu se passer lors de l'exécution d'une commande, je n'ai qu'à consulter le fichier `resultats.log`.

Résumé

Nous venons de découvrir deux flux de redirection dans des fichiers :

- > : redirige dans un fichier et l'écrase s'il existe déjà ;
- >> : redirige à la fin d'un fichier et le crée s'il n'existe pas.

Le schéma de la figure suivante récapitule ce que nous venons de voir.



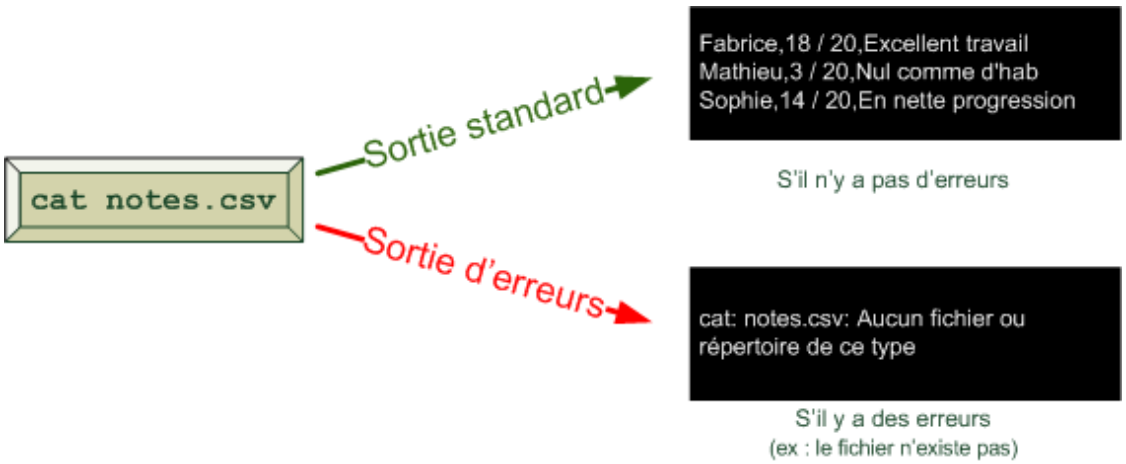
2>, 2>> et 2>&1 : rediriger les erreurs

Allons un peu plus loin. Il faut savoir que toutes les commandes produisent deux flux de données différents, comme le montre la figure suivante :

- **la sortie standard** : pour tous les messages (sauf les erreurs) ;
- **la sortie d'erreurs** : pour toutes les erreurs.

Prenons un exemple concret pour voir comment ça se passe.
Supposons que vous fassiez un `cat` du fichier `notes.csv` pour afficher son contenu. Il y a deux possibilités :

- **si tout va bien**, le résultat (le contenu du fichier) s'affiche sur la sortie standard ;
- **s'il y a une erreur**, celle-ci s'affiche dans la sortie d'erreurs.



Par défaut, tout s'affiche dans la console : la sortie standard comme la sortie d'erreurs. Cela explique pourquoi vous ne faisiez pas la différence entre ces deux sorties jusqu'ici : elles avaient l'air identiques.

Tout à l'heure, nous avons vu comment rediriger la sortie standard dans un fichier. Toutefois, les erreurs continuent d'être affichées dans la console. Faites le test :

```
cut -d , -f 1 fichier_inexistant.csv > eleves.txt
cut: fichier_inexistant.csv: Aucun fichier ou répertoire de ce type
```

Le fichier `fichier_inexistant.csv` n'existe pas (comme son nom l'indique). L'erreur s'est affichée dans la console au lieu d'avoir été envoyée dans `eleves.txt`.

Rediriger les erreurs dans un fichier à part

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>`. Vous avez bien lu : c'est le chiffre 2 collé au chevron que nous avons utilisé tout à l'heure.

Faisons une seconde redirection à la fin de cette commande `cut` :

```
cut -d , -f 1 fichier_inexistant.csv > eleves.txt 2> erreurs.log
```

Il y a deux redirections ici :

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

Vous pouvez vérifier : si `fichier_inexistant.csv` n'a pas été trouvé, l'erreur aura été inscrite dans le fichier `erreurs.log` au lieu d'être affichée dans la console.

Notez qu'il est aussi possible d'utiliser `2>>`

pour ajouter les erreurs à la fin du fichier.

Fusionner les sorties

Parfois, on n'a pas envie de séparer les informations dans deux fichiers différents. Heureusement, il est possible de fusionner les sorties dans un seul et même fichier. Comment ?

Il faut utiliser le code suivant : `2>&1`.
Cela a pour effet de rediriger toute la sortie d'erreurs dans la sortie standard. Traduction pour l'ordinateur : « envoie les erreurs au même endroit que le reste ».

Essayez donc ceci :

```
cut -d , -f 1 fichier_inexistant.csv > eleves.txt 2>&1
```

Tout ira désormais dans `eleves.txt` : le résultat (si cela a fonctionné), de même que les erreurs (s'il y a eu un problème).

Petite subtilité : je vous ai dit tout à l'heure qu'il était possible de faire `2>>` pour rediriger les erreurs à la fin d'un fichier d'erreurs.
Toutefois, il n'est pas possible d'écrire `2>>&1`. Essayez, ça ne marchera pas.

En fait, le symbole `2>&1` va envoyer les erreurs dans le même fichier et **de la même façon** que la sortie standard. Donc, si vous écrivez :

```
cut -d , -f 1 fichier_inexistant.csv >> eleves.txt 2>&1
```

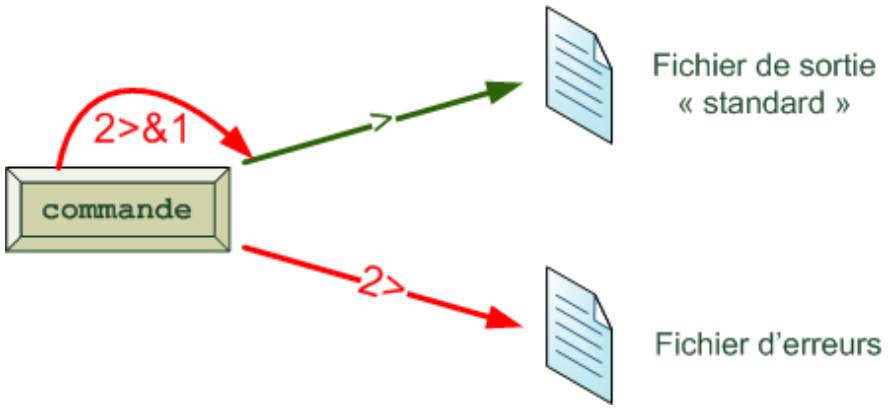
… les erreurs seront **ajoutées** à la fin du fichier `eleves.txt` comme le reste des messages.

Résumé

Nous avons découvert trois symboles :

- `2>` : redirige les erreurs dans un fichier (s'il existe déjà, il sera écrasé) ;
- `2>>` : redirige les erreurs à la fin d'un fichier (s'il n'existe pas, il sera créé) ;
- `2>&1` : redirige les erreurs au même endroit et de la même façon que la sortie standard.

Le tout est illustré sur la figure suivante.



Comprenez-vous bien ce schéma ?

On peut choisir de rediriger les erreurs dans un fichier à part (avec `2>`) ou bien de les rediriger au même endroit que la sortie standard (avec `2>&1`).

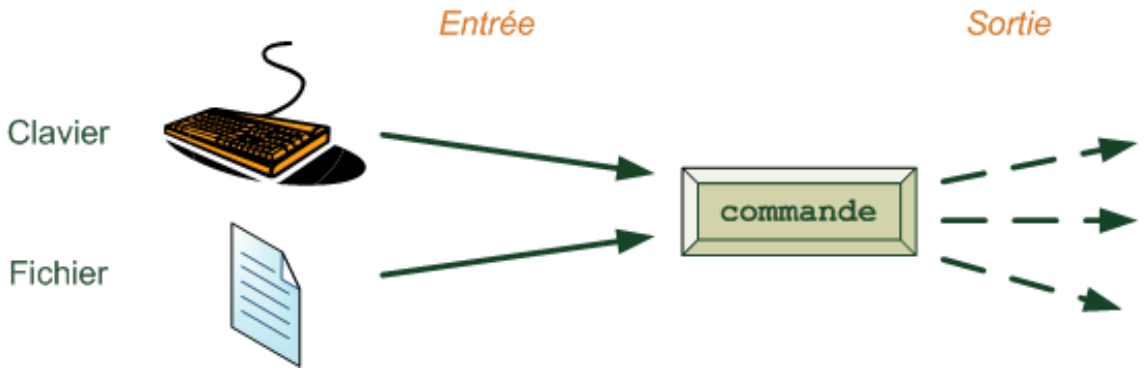
J'ai volontairement omis de parler sur ce schéma de `>>`

et de `2>>` afin de ne pas le surcharger, mais le principe est le même sauf qu'on ajoute à la fin d'un fichier au lieu de l'écraser.

< et << : lire depuis un fichier ou le clavier

Pour le moment, nous avons redirigé uniquement la **sortie** des commandes. Nous avons décidé où envoyer les messages issus de ces commandes.

Maintenant, je vous propose de faire un peu l'inverse, c'est-à-dire de décider d'où vient l'**entrée** d'une commande. Jusqu'alors, l'entrée venait des paramètres de la commande… mais on peut faire en sorte qu'elle vienne d'un fichier ou d'une saisie au clavier ! Regardez l'illustration de la figure suivante.



< : lire depuis un fichier

Le chevron ouvrant < (à ne pas confondre avec le chevron fermant que nous avons utilisé tout à l'heure) permet d'indiquer d'où vient l'entrée qu'on envoie à la commande.

On va prendre un exemple tout bête : la commande `cat`.

```
cat < notes.csv
```

Cela aura pour effet d'afficher le contenu du fichier envoyé en entrée :

```
$ cat < notes.csv
Fabrice,18 / 20,Excellent travail
Mathieu,3 / 20,Nul comme d'hab'
Sophie,14 / 20,En nette progression
Mélanie,9 / 20,Allez presque la moyenne !
Corentin,11 / 20,Pas mal mais peut mieux faire
Albert,20 / 20,Toujours parfait
Benoît,5 / 20,En grave chute
```

Il n'y a rien d'extraordinaire.
On ne faisait pas pareil avant en écrivant juste `cat notes.csv` *par hasard ?*

Si. Écrire `cat < notes.csv` est strictement identique au fait d'écrire `cat notes.csv`… du moins en apparence. Le résultat produit est le même, mais ce qui se passe derrière est très différent.

- Si vous écrivez `cat notes.csv`, la commande `cat` reçoit en entrée le nom du fichier `notes.csv` qu'elle doit ensuite se charger d'ouvrir pour afficher son contenu.

- Si vous écrivez `cat < notes.csv`, la commande `cat` reçoit **le contenu** de `notes.csv` qu'elle se contente simplement d'afficher dans la console. C'est le shell (le programme qui gère la console) qui se charge d'envoyer le contenu de `notes.csv` à la commande `cat`.

Bref, ce sont deux façons de faire la même chose mais de manière très différente.

Pour le moment, je n'ai pas d'exemple plus intéressant à vous proposer à ce sujet, mais retenez cette possibilité car vous finirez par en avoir besoin, faites-moi confiance. ;-)

<< : lire depuis le clavier progressivement

Le double chevron ouvrant << fait quelque chose d'assez différent : il vous permet d'envoyer un contenu à une commande avec votre clavier.
Cela peut s'avérer très utile. Je vous propose un exemple concret pour bien voir ce que ça permet de faire en pratique.

Essayez de taper ceci :

```
sort -n << FIN
```

La console vous propose alors de taper du texte.

```
$ sort -n << FIN
>
```

Comme `sort -n` sert à trier des nombres, on va justement écrire des nombres, un par ligne (en appuyant sur la touche `Entrée` à chaque fois).

```
$ sort -n << FIN
> 13
```

```
> 132
> 10
> 131
```

Continuez ainsi jusqu'à ce que vous ayez terminé.

Lorsque vous avez fini, tapez `FIN` pour arrêter la saisie.
Tout le texte que vous avez écrit est alors envoyé à la commande (ici `sort`) qui traite cela en entrée. Et, comme vous pouvez vous en douter, la commande `sort` nous trie nos nombres !

```
$ sort -n << FIN
> 13
> 132
> 10
> 131
> 34
> 87
> 66
> 68
> 65
> FIN
10
13
34
65
66
68
87
131
132
```

Sympa, non ?
Cela vous évite d'avoir à créer un fichier si vous n'en avez pas besoin.

Vous pouvez faire la même chose avec une autre commande, comme par exemple `wc` pour compter le nombre de mots ou de caractères.

```
$ wc -m << FIN
> Combien de caractères dans cette phrase ?
```

```
> FIN
42
```

Une question : ce mot `FIN` est-il obligatoire ?

Non, vous pouvez le remplacer par ce que vous voulez.
Lorsque vous tapez la commande, vous pouvez utiliser le mot que vous voulez. Par exemple :

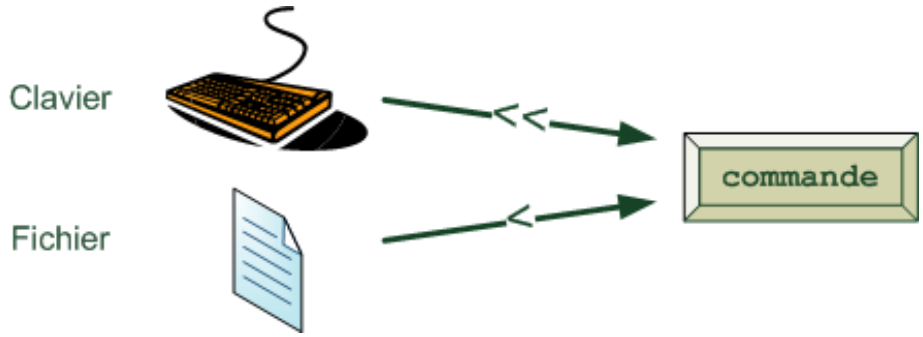
```
$ wc -m << STOP
> Combien de caractères dans cette phrase ?
> STOP
42
```

Ce qui compte, c'est que vous définissiez un mot-clé qui servira à indiquer la fin de la saisie.
Notez par ailleurs que rien ne vous oblige à écrire ce mot en majuscules.

Résumé

Nous pouvons donc « alimenter » des commandes de deux manières différentes, comme le montre la figure suivante :

- < : envoie le contenu d'un fichier à une commande ;
- << : passe la console en mode saisie au clavier, ligne par ligne. Toutes ces lignes seront envoyées à la commande lorsque le mot-clé de fin aura été écrit.



Vous pouvez tout à fait combiner ces symboles avec ceux qu'on a vus précédemment. Par exemple :

```
$ sort -n << FIN > nombres_tries.txt 2>&1
> 18
> 27
```

```
> 1
> FIN
```

Les nombres saisis au clavier seront envoyés à `nombres_tries.txt`, de même que les erreurs éventuelles.

Hé, mine de rien, on commence à rédiger là des commandes assez complexes !
Mais vous allez voir, on peut faire encore mieux.

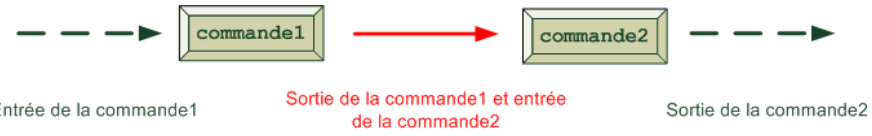
La théorie

Passons maintenant au symbole le plus intéressant que vous utiliserez le plus souvent : le **pipe** | (prononcez « païpe », comme un bon Anglais). Son but ? Chaîner des commandes.

Le pipe | n'est pas un symbole qu'on a l'habitude d'écrire. Pourtant, il y en a forcément un sur votre clavier (parfois représenté sous la forme d'une ligne verticale en pointillés).
Sur un clavier AZERTY français par exemple, vous pouvez l'écrire en combinant les touches `Alt Gr + 6` et sur un clavier belge, `Alt Gr + 1`. Sur un clavier Mac, c'est `Alt + Shift + L`

.

« Chaîner des commandes » ? Cela signifie connecter la sortie d'une commande à l'entrée d'une autre commande (comme le montre la figure suivante).



En gros, **tout ce qui sort de la `commande1` est immédiatement envoyé à la `commande2`**. Et vous pouvez chaîner des commandes comme cela indéfiniment !

Cette fonctionnalité est vraiment une des plus importantes et décuple littéralement les possibilités offertes par la console.
Souvenez-vous : dans le chapitre précédent, **je vous disais que chaque commande Unix avait un et un seul rôle, mais qu'elle le remplissait bien**. Parfois, l'utilité de certaines commandes seules peut paraître limitée, mais celles-ci prennent en général tout leur sens lorsqu'on les combine à d'autres commandes.

La pratique

Voyons quelques cas concrets (on pourrait trouver une infinité d'exemples).

Trier les élèves par nom

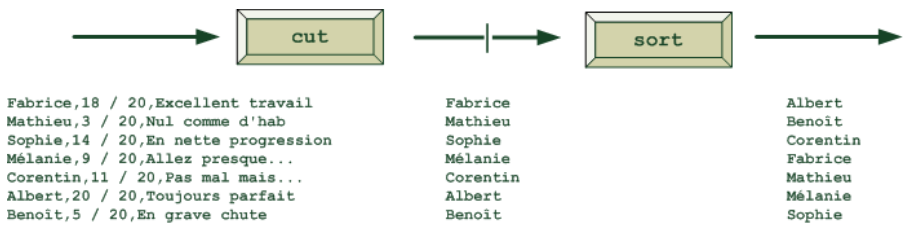
Si vous vous souvenez bien, nous avons toujours un fichier `notes.csv` qui contient la liste des élèves et leurs notes :

```
Fabrice,18 / 20,Excellent travail
Mathieu,3 / 20,Nul comme d'hab'
Sophie,14 / 20,En nette progression
Mélanie,9 / 20,Allez presque la moyenne !
Corentin,11 / 20,Pas mal mais peut mieux faire
Albert,20 / 20,Toujours parfait
Benoît,5 / 20,En grave chute
```

Avec `cut`, on peut récupérer les noms. Avec `sort`, on peut les trier par ordre alphabétique. Pourquoi ne pas connecter `cut` à `sort` pour avoir la liste des noms triés ?

```
$ cut -d , -f 1 notes.csv | sort
Albert
Benoît
Corentin
Fabrice
Mathieu
Mélanie
Sophie
```

Le pipe effectue la connexion entre la sortie de `cut` (des noms dans le désordre) et l'entrée de `sort`, comme l'illustre la figure suivante.



On peut même aller plus loin et écrire cette liste triée dans un fichier :

```
cut -d , -f 1 notes.csv | sort > noms_tries.txt
```

Trier les répertoires par taille

La commande `du` permet d'obtenir la taille de chacun des sous-répertoires du répertoire courant (je vous conseille de vous placer dans votre home en tapant d'abord `cd`) :

```
$ du
4       ./.gnome2_private
40       ./.local/share/Trash/files
4       ./.local/share/Trash/info
12      ./.local/share/Trash
160      ./.local/share
20      ./.local
...
```

Deux problèmes : cette liste est parfois très longue et n'est pas triée.

Un problème à la fois. Tout d'abord, on aimerait par exemple avoir cette même liste dans l'ordre décroissant de taille des répertoires pour repérer plus facilement les plus gros d'entre eux qui prennent de la place sur notre disque.

Pour avoir cette liste du plus grand au plus petit, il nous suffit d'écrire :

```
du | sort -nr
```

On envoie tout le contenu de `du` à `sort` qui se charge de trier les nombres au début de chacune des lignes.

```
$ du | sort -nr

...

4       ./.evolution/memos/config
4       ./.evolution/calendar/config
4       ./.evolution/cache
4       ./bin
```

Problème : comme les plus gros répertoires ont été affichés en premier, et que j'ai beaucoup de sous-répertoires, je dois remonter très haut dans la console pour retrouver les plus gros d'entre eux.

Que diriez-vous de connecter cette sortie à `head` ? Cette commande permet de filtrer uniquement les premières lignes qu'elle reçoit, nous l'avons déjà étudiée dans un chapitre précédent.

```
$ du | sort -nr | head
120920  .
59868   ./.ies4linux
43108   ./.ies4linux/ie6
41360   ./.ies4linux/ie6/drive_c
41248   ./.ies4linux/ie6/drive_c/windows
40140   ./Desktop
34592   ./.ies4linux/ie6/drive_c/windows/system32
16728   ./.ies4linux/downloads
13128   ./.mozilla
13124   ./.mozilla/firefox
```

Vous pouvez paramétrer le nombre de résultats affichés avec l'option `-n` de `head`. Si vous avez oublié comment l'utiliser, retournez lire le cours sur `head`, ou consultez le manuel.

Si vous voulez naviguer à travers tous les résultats, vous pouvez connecter la sortie à `less`. Cette commande permet d'afficher des résultats page par page ; ça nous est justement utile dans le cas présent où nous avons beaucoup de résultats !

```
du | sort -nr | less
```

Essayez !
Vous allez vous retrouver avec un affichage de `less`, page par page.

```
120920  .
59868   ./.ies4linux
43108   ./.ies4linux/ie6
41360   ./.ies4linux/ie6/drive_c
41248   ./.ies4linux/ie6/drive_c/windows
40140   ./Desktop
34592   ./.ies4linux/ie6/drive_c/windows/system32
16728   ./.ies4linux/downloads
13128   ./.mozilla
13124   ./.mozilla/firefox
13112   ./.mozilla/firefox/v5p4a55d.default
12604   ./.ies4linux/downloads/ie6
11808   ./.ies4linux/downloads/ie6/FR
5848    ./.mozilla/firefox/v5p4a55d.default/Cache
3656    ./.ies4linux/ie6/drive_c/windows/profiles
3616    ./.ies4linux/ie6/drive_c/windows/profiles/mateo21
3496    ./.ies4linux/ie6/drive_c/windows/profiles/mateo21/Local Settings
3416    ./.ies4linux/ie6/drive_c/windows/profiles/mateo21/Local Settings/Temporary Internet Files
3408    ./.ies4linux/ie6/drive_c/windows/profiles/mateo21/Local Settings/Temporary Internet Files/Conte

2220    ./.ies4linux/ie6/drive_c/windows/fonts
2012    ./ies4linux-2.99.0.1
:
```

Vous pouvez maintenant voir les premiers fichiers (les plus gros) et descendre progressivement vers les fichiers plus petits, page par page avec la touche `Espace` ou ligne par ligne, avec la touche `Entrée` (ou les flèches du clavier).

Exercice : peut-être avez-vous toujours trop de répertoires sous les yeux et que vous vous intéressez seulement à certains d'entre eux. Pourquoi ne pas filtrer les résultats avec `grep`, pour afficher uniquement la taille des répertoires liés à… `Firefox` par exemple ?

Lister les fichiers contenant un mot

Allez, un dernier exercice tordu pour finir en beauté.

Avec `grep`, on peut connaître la liste des fichiers contenant un mot dans tout un répertoire (option `-r`). Le problème est que cette sortie est un peu trop **verbeuse** (il y a trop de texte) : il y a non seulement le nom du fichier mais aussi la ligne dans laquelle le mot a été trouvé.

```
/var/log/installer/syslog:Apr  6 15:14:43 ubuntu NetworkManager: <debug> [1207494883.004888]
/var/log/installer/syslog:Apr  6 15:23:27 ubuntu python: log-output
```

Heureusement, le nom du fichier et le contenu de la ligne sont séparés par un deux-points. On connaît `cut`, qui permet de récupérer uniquement une partie de la ligne. Il nous permettrait de conserver uniquement le nom du fichier.
Problème : si le même mot a été trouvé plusieurs fois dans un fichier, le fichier apparaîtra en double ! Pour supprimer les doublons, on peut utiliser `uniq`, à condition d'avoir bien trié les lignes avec `sort` auparavant.

Alors, vous avez une petite idée de la ligne qu'il va falloir écrire ?
Je vous propose de rechercher les fichiers qui contiennent le mot « log » dans le dossier `/var/log`. Notez qu'il faudra passer root avec `sudo` pour avoir accès à tout le contenu de ce répertoire.

Voici la commande que je vous propose d'utiliser :

```
sudo grep log -Ir /var/log  | cut -d : -f 1  | sort | uniq
```

Que fait cette commande ?

1. Elle liste tous les fichiers contenant le mot « log » dans `/var/log` (`-I` permettant d'exclure les fichiers binaires).
2. Elle extrait de ce résultat uniquement les noms des fichiers.
3. Elle trie ces noms de fichiers.
4. Elle supprime les doublons.

Et voilà le résultat !

```
$ sudo grep log -Ir /var/log  | cut -d : -f 1  | sort | uniq
```

```
/var/log/acpid
/var/log/auth.log
/var/log/boot
/var/log/bootstrap.log
/var/log/dist-upgrade/apt-term.log
/var/log/dmesg
/var/log/dmesg.0
/var/log/gdm/
/var/log/installer/partman
/var/log/installer/syslog
/var/log/kern.log.0
/var/log/messages
/var/log/messages.0
/var/log/syslog
/var/log/syslog.0
/var/log/udev
/var/log/Xorg.0.log
/var/log/Xorg.0.log.old
/var/log/Xorg.20.log
/var/log/Xorg.20.log.old
/var/log/Xorg.21.log
```

Résumé

Le résumé est simple, et c'est dans sa simplicité qu'il tire toute sa beauté et sa puissance (non, je ne suis pas fou !), comme l'illustre la figure suivante.

S'il y avait un schéma à retenir, ce serait celui-là. Ça tombe bien, c'est le plus simple.

Je vous laisse vous entraîner avec le pipe, nous le réutiliserons très certainement dans les prochains chapitres. Essayez d'inventer des combinaisons ! ;-)

Les espaces avant et après le pipe ne sont en général pas obligatoires, mais je préfère les mettre ici pour une meilleure lisibilité.

En résumé

- Au lieu d'afficher le résultat d'une commande dans une console, il est possible de l'enregistrer dans un fichier. Il suffit d'ajouter le symbole `>` suivi du nom du fichier à la fin de la commande. Par exemple `ls > liste_fichiers.txt` enregistre la liste des fichiers dans un fichier plutôt que de l'afficher en console.
- Le symbole`>>` enregistre à la fin du fichier au lieu de l'écraser s'il existe déjà.
- Les symboles `2>` et `2>>` permettent de rediriger seulement les erreurs dans un fichier. Quant à `2>&1` il redirige les erreurs dans le même fichier que les messages normaux.
- `<` permet de lire des données depuis un fichier et de les envoyer à une commande, tandis que `<<` lit les données depuis le clavier.
- Le symbole `|` combine des commandes : les données de la commande à sa gauche sont envoyées à la commande à sa droite. Ainsi, `du | sort -nr` récupère la liste des fichiers avec leur taille et l'envoie à `sort` pour qu'il la trie.