

TD 0 : Découverte de Linux


Exercice 1. Prise en main du terminal

(Source : BE 1 de François Cayre)

Symboles du shell utilisés — Chaînage ou pipe (`|`), redirection (`<`, `>`, `>>`, `2>`, `2>>`, `2>&1`).

Lorsqu'un processus est lancé par le système d'exploitation, celui-ci ouvre par défaut les fichiers (les flux) associés à l'entrée standard (`stdin`, en lecture seule), à la sortie standard (`stdout`, en écriture seule *tamponnée*) et à la sortie d'erreur standard (`stderr`, en écriture seule *non tamponnée*). Historiquement, `stderr` correspondait à l'imprimante, mais elle est maintenant redirigée vers l'écran, tout comme `stdout`. L'entrée standard `stdin` correspond au clavier.

Quelques rappels

Dans le terminal ( *shell*), il est possible de rediriger la sortie standard d'un processus vers un fichier. Par exemple, pour stocker la sortie de `ls` dans le fichier `catalogue.txt`, on écrira :

```
$ ls > catalogue.txt
```

Cette redirection a cependant pour effet de supprimer le contenu du fichier `catalogue.txt` s'il existe déjà. Si l'on souhaite conserver le contenu du fichier `catalogue.txt` et écrire à la fin de ce dernier, on doublera le symbole en `>>`. De la même manière, la sortie d'erreur standard peut se rediriger vers un fichier à l'aide des opérateurs `2>` et `2>>`.

```
$ ls > catalogue.txt 2> erreurs.txt
```

La sortie d'erreur standard peut être redirigée vers le même fichier que la sortie standard et dans ce cas, pour éviter d'entrer deux fois le nom du fichier, on utilisera le raccourci `2>&1`. Les deux lignes suivantes sont donc équivalentes :

```
$ ls > catalogue.txt 2> catalogue.txt
```

```
$ ls > catalogue.txt 2>&1
```


Si l'on ne souhaite pas afficher les erreurs ou le résultat d'une commande, ni à l'écran, ni dans un fichier, il existe un puits sans fond nommé `/dev/null` dans lequel on peut rediriger les sorties standard pour détruire immédiatement toute information. Faites le test :

```
$ ls trouNoir_2.0_leRetourDeLaVengeance
```


```
$ ls trouNoir_2.0_leRetourDeLaVengeance 2> /dev/null
```


Il est également possible de rediriger l'entrée standard depuis un fichier : on utilisera pour cela le symbole `<`.



Si l'on veut chaîner la sortie standard d'un processus vers l'entrée standard d'un autre processus, on utilisera le symbole `|` ( *pipe*). Par exemple, pour compter le nombre de fichiers (ou presque...) dans le répertoire courant, on utilisera :

```
$ ls -l | wc -l
```

Ici, la sortie standard de `ls` est redirigée vers l'entrée standard de `wc` (pour  *word count* qui sert, entre autres, à compter les lignes et les caractères des fichiers).

En chaînant plusieurs commandes simples et spécialisées pour réaliser des tâches complexes, on forme alors un  *pipeline*. Cette dernière remarque est très importante car elle est au cœur de la philosophie UNIX. Vous devriez donc réfléchir en décomposant mentalement la tâche que vous voulez réaliser en un *pipeline* composé de commandes plus simples. C'est ainsi que ce TP est conçu !

Un filtre est un programme qui effectue un traitement sur les données qu'il reçoit depuis son entrée standard, et en donne le résultat sur sa sortie standard. Sauf en cas d'erreur (affichée sur `stderr` !), il n'affiche rien d'autre que le résultat de son traitement.

Parmi les filtres les plus connus sous UNIX, on mentionnera :

- `cat(1)` — qui affiche le contenu d'un fichier sur sa sortie standard (on peut donc l'utiliser pour *initier un pipeline*) ;
- `grep(1)` — pour rechercher une chaîne de caractères (ou une expression régulière) dans l'entrée standard ou un fichier passé en paramètre ;
- `sort(1)` — pour trier les lignes de l'entrée standard ou d'un fichier passé en paramètre ;
- `uniq(1)` — pour ne conserver qu'un seul exemplaire des lignes de l'entrée standard ou d'un fichier passé en paramètre ;
- `head(1)` — pour n'afficher que les premières lignes de l'entrée standard ou d'un fichier passé en paramètre ;
- `tail(1)` — pour n'afficher que les dernières lignes de l'entrée standard ou d'un fichier passé en paramètre ;
- `tee(1)` — pour recopier son entrée standard sur sa sortie standard *et* dans un fichier (par exemple pour produire une sortie dans un étage d'un *pipeline*) ;
- `tr(1)` — pour faire correspondre, supprimer, *etc.* des caractères ;



- `wc(1)` — pour compter les lignes, les mots et/ou les caractères de l'entrée standard ou d'un fichier passé en paramètre ;
- `sed(1)` et `awk(1)` — pour réaliser des traitements aussi complexes que possible.

Référez-vous à leur page du manuel (`$ man commande`) pour connaître plus précisément leur fonction et leurs options !

Ces filtres travaillent par défaut sur leur entrée standard et sont donc exploitables dans des *pipelines* mais ils peuvent aussi souvent travailler sur un (ou plusieurs) fichier(s) passé(s) en paramètre(s).

Il est souvent sage, lorsque la fonctionnalité attendue du programme le suggère, de prévoir qu'un programme devra pouvoir fonctionner comme un filtre. On pourra alors le combiner avec d'autres programmes très facilement, par exemple dans un script *shell* ou dans un *pipeline*, mais on évitera surtout de coder à nouveau ce qui l'est déjà (voir Sec. 4).


Mise en pratique

Chacune des opérations suivantes peut s'exécuter en une seule ligne de commande en combinant des filtres simples ¹ :

1. Créez un fichier liste contenant la liste des dossiers et fichiers présents dans votre répertoire personnel ;
2. Ajoutez la date (`date`) à la fin du fichier liste ;
3. Dans le répertoire `FruitsEtLegumes`, listez tous les documents ayant un nom de fruit dans un fichier `fruits` ;
4. Listez à présent l'ensemble des légumes dans un fichier `legumes`. Ayez à l'esprit qu'un légume n'est pas un fruit ;
5. Dans le répertoire `Tri`, listez les documents par ordre alphabétique inverse et ne gardez que les 5 premiers résultats ;
6. Comptez le nombre d'éléments uniques présents dans le fichier `colors`.

À savoir : le résultat d'une commande peut servir d'argument à une autre commande grâce à l'utilisation de l'accent grave (```) (AltGr 7). Par exemple, pour afficher tous les fichiers dont le nom contient l'année en cours :

```
$ ls | grep `date +%Y`
```

1. On parle alors de  *one-liner*. Voir par exemple :
 — <https://onceupon.github.io/Bash-Oneliner/> — pour démarrer dans cette discipline ;
 — <http://www.bashoneliners.com/> — pour une sorte de florilège...



Le fichier `messageChiffre` du répertoire `FiltreDechiffrage` est illisible car il a été chiffré à l'aide d'une clef. Le chiffage est très simple : à chaque symbole a été ajouté la valeur de la clef (vous aurez reconnu le Chiffre de César).

La clé de déchiffrement a été cachée d'une manière un peu particulière dans le répertoire `C1e`. Pour la trouver, il faut compter le nombre de lignes contenant le terme `KEY` parmi les 3 derniers fichiers (par ordre alphabétique) dont le nom se termine par la lettre `k`.

Maintenant que vous possédez la valeur `X` de la clef, vous pouvez construire un filtre en C pour déchiffrer le message en une simple commande :

```
$ cat messageChiffre | ./dechiffrage X
```

Votre filtre lira un à un les caractères sur l'entrée standard, soustraira la valeur de la clef et écrira les caractères ainsi obtenus sur la sortie standard. Enfin, plutôt que d'entrer la valeur de la clef à la main, écrivez en une ligne la commande permettant de récupérer la clef et de déchiffrer le message. Il ne vous aura pas échappé que ce filtre vous permettra également de chiffrer un message avec une clef que vous aurez vous-même choisie.

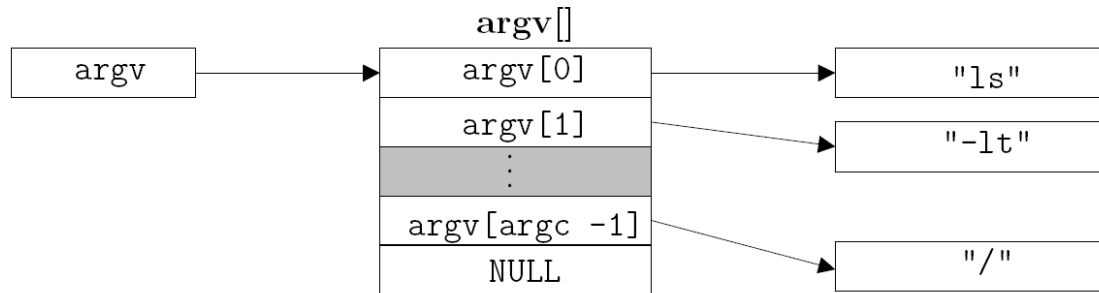
Exercice 2. Exécution d'un programme

La famille de primitive `exec` permet de créer un processus pour exécuter un programme déterminé (qui a auparavant été placé dans un fichier, sous forme binaire exécutable). On utilise en particulier `execvp` pour exécuter un programme en lui passant un tableau d'arguments. Le paramètre *filename* pointe vers le nom absolu ou relatif du fichier exécutable, *argv* vers le tableau contenant les arguments (terminé par `NULL`) qui sera passé à la fonction *main* du programme lancé. Par convention, le paramètre `argv[0]` contient le nom du fichier exécutable, les arguments suivants étant les paramètres successifs de la commande.

```
1 #include <unistd.h>
2 int execvp(char *filename, char *argv[]);
```

Noter que les primitives `exec` provoquent le « recouvrement » de la mémoire du processus appelant par le nouveau fichier exécutable. Il n'y a donc pas normalement de retour (sauf en cas d'erreur –fichier inconnu ou permission - auquel cas la primitive renvoie -1).





2.1 Que fait le programme suivant ?

```

1 #include<stdio.h>
2 #include<unistd.h>
3 #define MAX 5
4 int main {
5     char *argv[MAX];
6     argv[0] = "ls"; argv[1] = "-lt"; argv[2] = "/"; argv[3] = NULL;
7     execvp("ls", argv);
8 }
  
```

2.2 Ecrire un programme *doit* qui exécute une commande Unix qu'on lui passe en paramètre.

Exemple :

```

1 doit ls -lt /
  
```

Exercice 3 Question de réflexion

Un système temps réel avec tolérance a quatre événements périodiques, avec des périodes de 50, 100, 200 et 250 ms chacun. Supposons que les quatre événements aient besoin respectivement de 35, 20, 10 et x ms de temps processeur. Quelle est la valeur de x la plus importante pour laquelle il est possible d'ordonnancer le système ?

Exercice 4 Interblocage

Un système possède 2 processus et 3 ressources identiques. Chaque processus a besoin de 2 ressources maximum.

Un interblocage est-il possible ? Expliquez votre réponse.

En généralisant le problème : soit un ensemble de p processus ayant besoin au maximum besoin de m ressources et un total de r ressources disponibles, quelle condition faut-il satisfaire pour que le système soit exempt d'interblocage ?

