Numérique et sciences informatiques

Lycée Hoche

année scolaire 2025-2026

Contents

1 un peu d'histoire

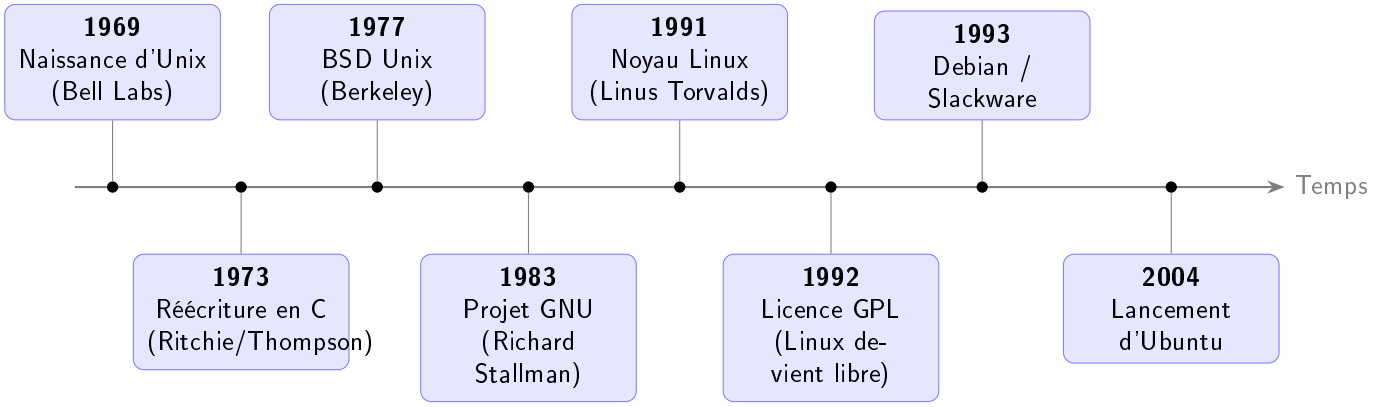


Figure 1: Chronologie simplifiée de l'évolution d'Unix et Linux

2 Système d'expoitation

L'utilisateur d'un ordinateur (ou d'un téléphone portable , console de jeux ,etc) est en général conscient de l'existence de deux types d'entités sur son équipement:

- Des **fichiers** comme des photos,des fichiers texte , des vidéos etc , souvent modifiables grâces à des logiciels (traitement de texte , logiciel de retouche d'images..), mais pas toujours.
- Des **processus** qui sont des **applications** en train de tourner , (ce qu'on appelle des données dynamiques): leur état est modifié d'un instant à l'autre , souvent sans intervention de l'utilisateur.

2.1 Rôle d'un système d'exploitation

- Pour simplifier , le système d'exploitation est une interface entre le matériel et les programmes de l'utilisateur.Par exemple:
  - Communication de l'utilisateur vers le matériel : on appuie sur une touche du clavier et le caractère concerné s'affiche à l'écran.
  - Communication du matériel vers l'utilisateur: Si un fichier qu'on tente d'ouvrir n'existe pas , le système affiche un message d'erreur.
- Isoler le code utilisateur du matériel .Le système permet par exemple:
  - de changer le matériel de façon transparente (ou presque) pour l'utilisateur .
  - d'assurer l'intégrité des données : s'assurer par exemple que l'utilisateur a les droits d'accès à un fichier lors d'une tentative de lecture ou d'écriture.Il assure également qu'aucun utilisateur ne puisse détruire la cohérence de l'ensemble des données .
  - faire en sorte que tous les programmes puissent s'exécuter de façon équitable (utilisation des ressources en mémoire , délai d'accès raisonnable à une ressource).

Dans ce cours , on s'intéressera en particulier aux systèmes d'exploitation de type UNIX .
Même si les systèmes actuels sont plus compliqués , les questions posées ici sont toujours d'actualité.

On peut schématiser le fonctionnement d'un ordinateur à l'aide de couches qui communiquent entre elles de la façon suivante :

utilisateur——————————appels systèmes
signaux

noyau

matériel——————————interface matérielle
interruptions

Si un processus (une application par exemple) tente d'accéder au matériel , alors qu'il n'en n'a pas l'autorisation , le noyau tue ce processus (c'est le célèbre ***segmentation fault*** que rencontre souvent les programmeurs en C)

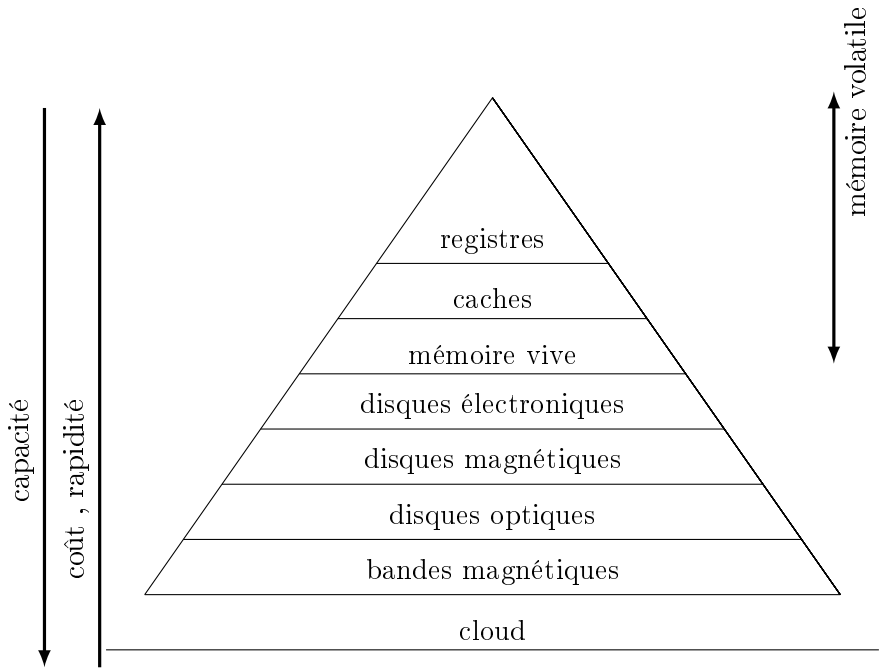
2.2 Ressources et mémoire

Pour bien comprendre le fonctionnement d'un système d'exploitation , il faut connaître les types de mémoire présents dans un ordinateur:

Registres : Emplacement mémoire interne au processeur rapide d'accès mais limité par la taille (quelques dizaines d'octets)

Caches: Emplacement mémoire interne au processeur qui sert à stocker temporairement des instructions ou des données et qui permet d'accélérer les traitements en limitant les accès à la mémoire vive.

Mémoire vive: mémoire volatile assez rapide (de quelques MO à quelques GO , jusqu'à 16 GO ou plus).



Le noyau a intérêt à avoir des données disponibles dans la mémoire la plus rapide d'accès.Plus précisément , quand le noyau a besoin d'une ressource (un fichier ou un programme), il la rapatrie dans la mémoire volatile.

Il travaille sur cette ressource et , si besoin, met à jour la **mémoire permanente** à la fin du traitement , (disques durs , clés usb etc)

C'est ce qui explique que si un ordinateur s'éteind d'un coup et qu'on n'a pas sauvegardé , on ne retrouve pas tout ce qu'on a écrit (dans un fichier texte par exemple).

Mais la mémoire rapide a un coup élevé:
Il est donc impossible de mettre les contenus de tous les fichiers dans la mémoire volatile;

Une partie du travail du système d'exploitation est de **gérer le contenu de la mémoire volatile** , mais aussi de faire en sorte que les applications ignorent où se trouvent les données qu'elles traitent (problème de sécurité).

Ce qui fait la vitesse d'un ordinateur , ce n'est pas tant que la rapidité du CPU que la taille du cache: le CPU peut être aussi rapide qu'on veut, s'il n'y a pas de cache la machine sera lente.

La question se pose alors de savoir pourquoi ne pas installer des mémoires cache plus grandes?

En fait ceci est préjudiciable au niveau rapidité puisque plus une ressource est grande plus c'est compliqué de trouver des données à l'intérieur.

3 Le système UNIX

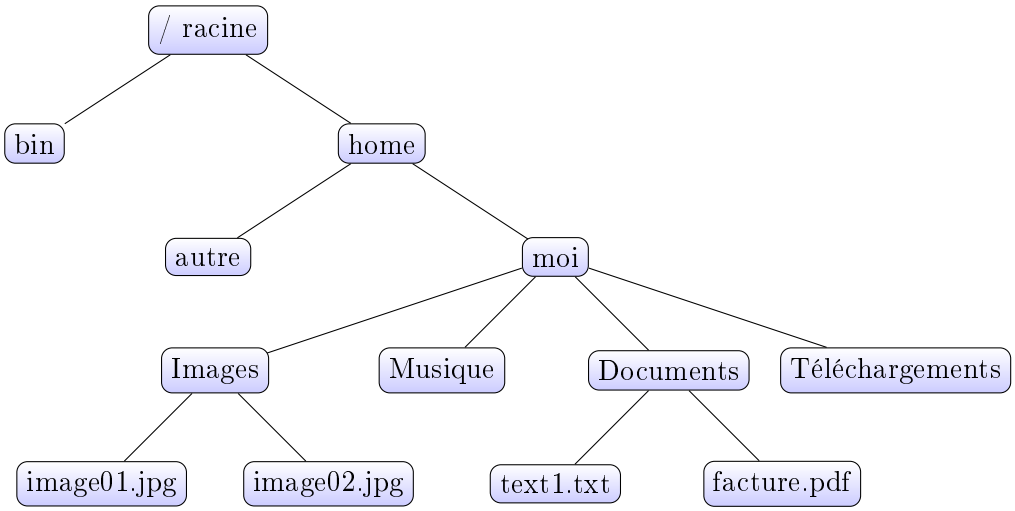
Nous nous intéresserons exclusivement à la **représentation logique** des données (fichiers et processus).

3.1 Fichiers

Intéressons-nous en premier à une abstraction intellectuelle permettant de représenter l'organisation logique des données dans un système UNIX.Dans un tel système, les données sont rangées dans des fichiers (réguliers) , eux-mêmes organisés de façon hiéarchique , grâce à des répertoires.

Cette hiéarchie peut être représentée à l'aide d'une arborescence , ou un **arbre** au sens informatique du terme, dont les **noeuds** internes seraient les répertoires et les **feuilles** les fichiers réguliers.

Le répertoire qui se trouve en haut de cette hiéarchie s'appelle le répertoire **racine** (ou racine) et l'unique répertoire contenant un autre répertoire s'appelle le répertoire **parent** (ou père), exception faite de la racine qui est son propre parent par convention.



Dans l'arborescence ci-dessus , le répertoire *moi* a pour répertoire parent *home* .
Les feuilles de l'arbre , *image01.jpg et images02.jpg* sont des fichiers (réguliers).

Sur un système de type UNIX , le caractère / a deux significations:

- Il désigne la racine du SGF (système de gestion des fichiers) .
- il joue le rôle de séparateur dans un chemin: il permet de séparer les noms de deux répertoires lorsqu'on décrit un chemin dans l'arbre.

Le caractère ~ désigne le répertoire de *login* de l'utilisateur , c'est-à-dire le répertoire où se trouve l'utilisateur au moment où il se connecte à son compte.
Les utilisateurs et les processus ont souvent besoin de désigner des fichiers. Considérons le fichier images01.jpg . Pour l'atteindre il existe deux façons:

- **chemin absolu**: On décrit comment arriver jusqu'au fichier en partant d'un point fixe du système , généralement la racine:

  /home/moi/Images/images01.jpg

- **chemin relatif**: On décrit comment arriver jusqu'au fichier en partant du répertoire courant (le répertoire dans lequel on se trouve) :

  Images/images01.jpg

si l'on se trouve dans le répertoire *moi*.

Les qualificatifs absolu et relatif sont à prendre dans le sens suivant:

Un chemin absolu désigne de façon non ambiguë un fichier, quel que soit le répertoire courant , tandis qu'un chemin relatif le désigne relativement au répertoire courant.

Par la suite ,

On désignera par ***nom*** d'un fichier tout chemin absolu ou relatif permettant d'atteindre ce fichier.

Un **répertoire** est un fichier spécial : le système n'utilise pas les mêmes algorithmes que pour les fichiers réguliers pour le créer et le manipuler (par exemple un répertoire contient à tout moment un lien vers lui-même noté **.** et un lien vers son père **..**).

Pour créer ou modifier un fichier , on utilise souvent un logiciel ou une commande , qui font eux-mêmes appel au système , à travers un appel système , pour toutes les opérations critiques (par exemple celles susceptibles de corrompre l'intégrité du système).

Le système vérifie que l'opération demandée est acceptable avant de l'exécuter.
Par exemple , tout répertoire doit contenir un lien vers lui-même et un lien vers son père: à la création du répertoire , le système crée ses liens; quand on manipule le répertoire , le système s'assure que ces liens persistent.Le système vérifie également que l'utilisateur a les bons droits sur un fichier pour y accéder.

3.2 Déplacements dans l'arbre des fichiers

Pour se déplacer dans l'arbre des fichiers , on peut utiliser les commandes **pwd**(*print working directory*) et **cd** (*change directory*)

La commande pwd est utilisée pour connaître le répertoire courant:

- `$ pwd`

  ```
  utilisateur25@debian9:~$ pwd
  ```

- La commande **cd** *reference* permet de changer le répertoire de travail:Le répertoire donné dans *reference* devient le répertoire de travail.

  ```
  utilisateur25@debian9:~$ cd /home/moi/Documents
  utilisateur25@debian9:~/Documents$ ls
  ```

  ```
  utilisateur25@debian9:~/Documents$ cd ../Musique
  utilisateur25@debian9:~/Musique$
  ```

  Sans indication de référence , la commande *cd* ramène au répertoire privé de l'utilisateur:

  ```
  utilisateur25@debian9:~/Musique$ cd
  utilisateur25@debian9:~$
  ```

- La commande **ls** affiche le contenu du répertoire de travail (répertoire courant).

```
utilisateur25@debian9:~/Documents$ ls
total 2
text1.txt
facture.pdf
```

On peut ajouter des options dans la commande *ls* :

-s : indique la taille des fichiers.

-l : donne des renseignements sur les fichiers (type de fichier, taille du fichier ,date de dernière modification,propriétaire,nombre de liens, droits d'accès)

```
utilisateur25@debian9:~/Documents$ ls -l
total 2
text1.txt
facture.pdf
```

- La commande **cp** permet de recopier un fichier *source* en un fichier *destination*:

  cp *fichier source fichier destination*

  Il y a copie physique de fichier source : Sur le disque , il existe donc deux fichiers distincts dont les contenus sont identiques (Dans le système DOS la commande équivalente est *copy*)

- La commande **ln** sert à donner un *nom supplémentaire* au fichier source : il n'y a donc pas de recopie physique.

3.3 Autres opérations sur le SGF

- *less* : pour voir le contenu d'un fichier.

```
bob@c118-401~$ less nom_fichier
```

- *mv* : Pour déplacer un fichier (ou modifier son nom ce qui est semblable)

  Par exemple pour déplacer le fichier-01 (on peut ou non changer le nom).

```
bob@c118-401~$ mv chemin_vers_fichier-01  destination
```

- *rm*: pour supprimer un fichier régulier:

```
rm chemin_vers_fichier-01
```

- *mkdir* : pour créer un répertoire.
- *rmdir* : pour supprimer un répertoire vide (et dans ce cas , il contient une référence à lui-même et à son père).

3.4 I-noeuds

Pour comprendre le fonctionnement de la commande *ln* , examinons comment est structuré le SGF sous UNIX:
A tout fichier est associé un certain nombre d'informations, écrits sur le disque .Ce bloc d'informations est appelé un **i-noeud**. Il contient les informations suivantes sur le fichier:

- la taille du fichier.
- date de dernière modification.

- Les adresses des blocs du disque où est stocké le fichier.
- L'identification du propriétaire et les droits d'accès au fichier pour les différents utilisateurs .
- Le type du fichier (répertoire, fichier ordinaire ou fichier spécial).
- Le nombre de liens du fichier.

D'autres informations sont susceptibles d'être stockées..
Dans la liste ci-dessus ne figure pas le nom du fichier!
Où est stocké alors le nom du fichier?Dans les répertoires (ou catalogues).Voyons ceci dans le détail:

i-noeud

En réalité , **un répertoire est un fichier** qui contient une liste de couples (*numéro,nom*) où *numéro* désigne un numéro de i-noeud et *nom* est un nom de fichier.

A un fichier physique est associé un numéro unique appelé son index. Mais on peut lui associer plusieurs couples (*numéro,nom*).

Chaque couple associé à un fichier est appelé **lien** de ce fichier.

Dans le i-noeud , on stocke également le nombre de liens.

Lorsque le nombre de liens devient nul , l'espace disque occupé par le fichier et par son i-noeud est récupéré par le système.

La suppression d'un fichier est en fait la suppression d'une paire dans la liste du i-noeud , c'est donc la modification du contenu d'un répertoire.
Déplacer un fichier d'un répertoire à un autre revient à modifier les listes associées aux deux répertoires: supprimer une paire dans le répertoire d'origine et en ajouter une autre dans le répertoire destination.

3.5 Retour sur la commande *cp* - La commande ln

La commande :

```
cp moi toi
```

et plus généralement

```
cp dossier-01/dossier-02/.../moi   dossier-03/dossier-04/.../toi
```

a les effets suivant

- Elle entraîne la création sur le disque d'un nouvel i-noeud associé au nouveau fichier *toi*.
- Création d'un nouveau couple (indexe-de-toi,*to*i)
- Le contenu du fichier *moi* est recopié sur le disque à une nouvelle adresse conservée dans le i-noeud de *toi*.

La commande:

```
ln fichier-source  fichier-destination
```

a pour effet d'attribuer un nom supplémentaire (appelé **lien**) à fichier-source;cette opération ne génère pas une recopie physique du fichier *moi*.
La commande ln n'a pas d'équivalent sous windows.

Ainsi la commande *ln* ne crée pas de nouvel *i-noeud*:

- Elle incrémente de 1 le nombre de liens dans le *i-noeud* de *moi*.
- Il y a création d'un nouveau couple (indexe-de-moi,*toi*).

4 Les droits

Sous UNIX, chaque fichier (et répertoire) est la propriété d'un utilisateur particulier: par défaut l'utilisateur qui a créé le fichier.

Les utilisateurs sont réunis en groupe (un utilisateur pouvant faire parti de plusieurs groupes , pour chaque fichier spécifié le groupe propriétaire, c'est-à-dire en tant que membre ..).
Les droits sur les fichiers sont alors définis en fonction de la "position" du demandeur par rapport au propriétaire du fichier:

- propriétaire
- faisant partie du groupe propriétaire
- autre utilisateur

Seul le super-utilisateur **root** peut modifier le propriétaire d'un fichier, en revanche chacun peut modifier le groupe propriétaire d'un des ses fichiers , à condition d'appartenir au nouveau groupe propriétaire.

Il est particulièrement intéressant d'appartenir à plusieurs groupes car cela permet de définir des ensembles d'actions qui sont autorisées pour certains et pas pour d'autres, sans que ces deux groupes n'aient de liens logiques entre eux.On peut ainsi créer un groupe restreint d'utilisateurs qui auraient accès à certains fichiers.

droits

Il y a trois types de droits:

- Lecture: identifié par la lettre **r** ou la valeur **4**.
- Ecriture: identifié par la lettre **w** ou la valeur 2 .
- Exécution: identifié par la lettre **x** ou la valeur 1.

La signification est résumée dans le tableau ci-dessous.

	fichier régulier	répertoire
lecture r 4	lire le contenu	lister le contenu
écriture w 2	modifier le contenu	ajouter ou supprimer un élément
exécution x 1	exécuter	passer à travers

On peut noter que le droit de supprimer un fichier n'est pas lié à un droit sur le fichier, mais au droit sur le répertoire dans lequel se trouve le fichier .
On modifie les droits d'un fichier (ou d'un répertoire) avec la commande **chmod** avec deux arguments: le premier correspond au nouveaux droits et le deuxième au nom du fichier.

Seuls le propriétaire d'un fichier et le super utilisateur peuvent modifier les droits d'accès à un fichier.
Le changement de droits se fait de deux façons différentes:

- **changement absolu** : il s'agit de spécifier de nouveaux droits, indépendamment des droits existants; on calcule pour chaque type d'utilisateur un chiffre correspondant à la somme des droits à lui attribuer et on écrit côte à côte les droits en spécifiant d'abord ceux du propriétaire du fichier, puis ceux du groupe propriétaire et enfin ceux des utilisateurs.

  Par exemple pour un répertoire *Mathematiques* , si on veut que tout le monde ait les droits en exécution , que seuls le propriétaire et le groupe propriétaire aient le droit en lecture et que seul le propriétaire ait les droits en écriture, il suffira d'utiliser la ligne de commande:

```
chmod 751 Mathematiques
```

Ainsi les utilisateurs appartenant au même groupe que le propriétaire pourront lister (ls) le contenu du répertoire , tandis que les autres ne pourront consulter un fichier du répertoire que s'ils connaissent son nom et ont le droit de lecture sur ce fichier.

- **changement relatif**: il s'agit de spécifier les droits à ajouter ou à supprimer relativement aux droits existants , en précisant:

  1. les utilsateurs concernés: propriétaire du fichier (**u**) , groupe propriétaire sans le propriétaire (**g**), autres utilisateurs (**o**), tous les utilisateurs (**a**).
     En l'absence de spécification , c'est l'ensemble des utilisateurs qui est concerné.
  2. s'il s'agit d'un ajout (+) ou d'un retrait (-)
  3. droits concernés (r,w ou x).

  par exemple , pour enlever au fichier *Mathematiques/dm1* les droits en lecture pour le groupe propriétaire et les autres utilisateurs on utilisera la commande:

```
chmod go-r  Mathematiques/dm1
```

  L'option -l de la commande ls permet de connaître les droits sur un fichier.

On pourra consulter avec profit le site :
http://juliend.github.io/linux-cheatsheet/

5 Processus

Le système UNIX est multi-tâches: Il donne l'impression d'exécuter plusieurs traitement simultanément.(On peut par exemple à la fois consulter ses mails , exécuter un programme python sous *Pyzo* tout en écoutant de la musique sur le web).
Le processeur de la machine ne peut pourtant exécuter qu'une tâche (un *processus*) à la fois :on parle de *processus actif*.

Définition 1 *Un processus est l'objet dynamique associé à un programme:*
Le programme est une suite d'instructions à exécuter associées à des données, le processus est une instance en mémoire de ce programme en train de s'exécuter.Il peut y avoir plusieurs processus associés à un même programme.(on peut par exemple lancer simultanément plusieurs interpréteurs python)

L'ensemble de la mémoire associée à un processus est appelé son *contexte*.
Dans le processeur , l'ordonnanceur change régulièrement de processus actif en changeant le contexte:Le nouveau processus n'a pas conscience qu'il a été interrompu et continue là où il s'était arrêté !

On peut voir la liste des processus qui tournent sur le système avec la commande ps:

```
rene@debian9:~$ ps
PID TTY          TIME CMD
3827 pts/0    00:00:00 bash
3832 pts/0    00:00:00 gedit
3842 pts/0    00:00:00 ps
```

Le cycle de vie d'un processus est compliqué.En particulier,quand il utilise le processeur,un processus peut être en mode noyau ou en mode utilisateur:
Le noyau correspond aux moments où le processus délègue au noyau du système son action quand celle-ci est critique:Par exemple quand on demande l'ouverture d'un fichier en lecture , le système doit vérifier les droits, c'est donc à lui de faire cette ouverture.
Quand le processus fait des actions non critiques (par exemple une opération arithmétique)il reste en mode utilisateur.
Un processus en mode noyau ne peut être interrompu , ceci afin de s'assurer que toutes les opérations visant à garantir l'intégrité des données du système ont bien été appliquées.

5.1 Systèmes propriétaires/Systèmes libres

Il existe deux familles de systèmes d'exploitation:Les systèmes propriétaires et les systèmes libres.
On peut modifier un système libre et en faire l'usage que l'on souhaite (il est cependant sage de lire la licence sous laquelle le logiciel a été distribué : CGU ou Creative common etc).
A l'opposé , on ne peut pas modifier un système propriétaire sans l'accord explicite du propriétaire.
Un système propriétaire est maintenu par l'entreprise qui l'a créée alors qu'un système libre est maintenu par la *communauté*(développeurs bénévoles voire des entreprises de manière bénévole).
Plus précisément à l'intérieur du système d'exploitation , on distingue entre les *appels système* et un ensemble d'outils mis à la disposition de l'utilisateur.
L'outil le plus important étant *un interpréteur de langage de commande* appelé **Shell**.Les commandes standard utilisées au niveau de l'interpréteur permettent de manipuler les fichiers : copier , créer , supprimer , déplacer etc.
L'utilisateur peut également créer de nouvelles commandes appelées **scripts shell** , un script regroupant plusieurs commandes.

Annexe

- La commande `less` permet d'afficher page par page le contenu d'un ou plusieurs fichiers dont les noms sont passés en argument. Contrairement aux commandes précédentes qui font leur travail et se terminent, celle-ci est interactive et ne se termine pas, car elle attend des instructions de l'utilisateur. Celui-ci tape les instructions au clavier, voici les principales instructions :

  h: affiche une page d'aide sur les commandes connues par less
  Espace : passe à la page suivante
  Entrée : passe à la ligne suivante
  /**toto** : cherche le mot «toto» dans la suite du texte
  n : passe à l'occurrence suivante du dernier mot cherché (next)
  q : quitte la commande `less`

- La commande **mv** (move) permet de déplacer un fichier :

  d'un répertoire à l'autre (si dans le même répertoire, cela revient à renommer le fichier) ; en changeant son nom ; en écrasant la destination si elle existe (sans prévenir).

- La commande **sort** permet de lire des fichiers et de les afficher en triant l'ensemble des lignes dans l'ordre alphabétique.