Linux -- Cours 2 -- Bases des systèmes d'exploitation

Halim Djerroud

LISV
Laboratoire d'ingénierie
des systèmes de Versailles

UVSQ
université PARIS-SACLAY

révision : 0.1

Plan de la séance

1. Notion de fichier
2. Permissions Unix : `ls -l`, `chmod`, `chown`
3. Groupes, utilisateurs et sécurité
4. Redirections et pipes : `>`, `»`, `|`, `tee`

Notion de fichier

Notion de fichier sous Unix/Linux

Objectifs du chapitre :

- Comprendre que sous Unix/Linux, **tout est fichier**.
- Identifier les différents types de fichiers (ordinaires, répertoires, liens, périphériques, etc.).
- Utiliser des commandes de base pour obtenir des informations sur un fichier (`file`, `stat`).

Tout est fichier

Principe fondamental : Sous Unix/Linux, **tout est représenté comme un fichier** :

- Fichiers ordinaires (textes, binaires, images. . .)
- Répertoires
- Périphériques (clavier, disque, carte son. . .)
- Processus, sockets, tubes nommés

Exemples concrets :

- Périphériques : `/dev/sda`, `/dev/tty`, `/dev/null`
- Processus dans `/proc` (`/proc/cpuinfo`, `/proc/meminfo`)
- Répertoire = fichier spécial contenant des entrées

Conséquence

Les mêmes **outils de manipulation de fichiers** (`ls`, `cat`, `less`, etc.) s'appliquent à la plupart des objets du système.

Commande `file` et `stat`

Déterminer le type d'un fichier :

```
$ file /etc/passwd
/etc/passwd: ASCII text

$ file /bin/ls
/bin/ls: ELF 64-bit LSB executable
```

Obtenir des informations détaillées sur un fichier :

```
$ stat /etc/passwd
  File: /etc/passwd
  Size: 2700       Blocks: 8   IO Block: 4096 regular file
Device: 802h/2050d Inode: 393217 Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)  Gid: (0/root)
```

Types de fichiers Unix/Linux

Premier caractère de `ls -l` :

- `-` : fichier ordinaire
- `d` : répertoire
- `l` : lien symbolique
- `b` : périphérique bloc
- `c` : périphérique caractère
- `p` : tube nommé (FIFO)
- `s` : socket

Exemple `ls -l`

```
$ls -l
drwxr-xr-x  2 user group 4096 ...  Documents/
-rw-r--r--  1 user group  512 ...  notes.txt
lrwxrwxrwx  1 user group   10 ...  lien -> notes.txt
brw-rw----  1 root disk  8, 0 ...  /dev/sda
crw--w----  1 root tty   4, 1 ...  /dev/tty1
prw-r--r--  1 user group      ...  /tmp/mafifo
srw-rw-rw-  1 user group      ...  /tmp/mysock
```

Inode : définition

Qu'est-ce qu'un inode ?

- Abréviation de **index node**.
- Structure de données utilisée par les systèmes de fichiers Unix/Linux.
- Contient les **métadonnées** d'un fichier (mais pas son nom !).

Informations stockées dans un inode :

- Type de fichier (`-`, `d`, `l`, ...)
- Taille, droits d'accès, propriétaire, groupe
- Dates (création, accès, modification)
- Nombre de liens (hard links)
- Pointeurs vers les blocs de données sur le disque

À retenir

Un fichier est identifié par son **inode** ; plusieurs noms (hard links) peuvent pointer vers le même inode.

Inode

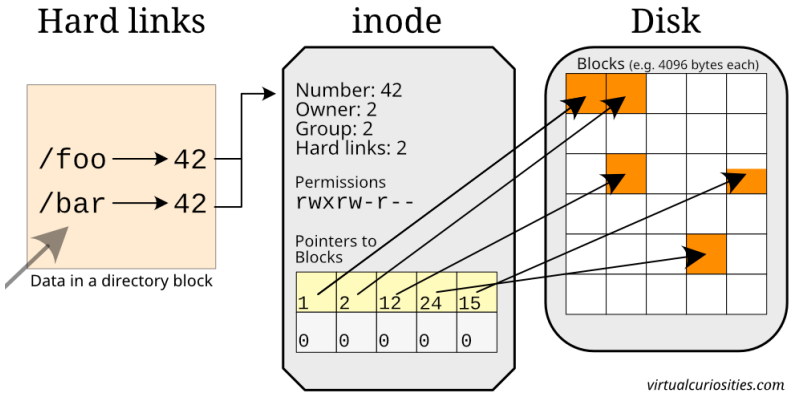


Figure – Source : www.virtualcuriosities.com

Liens : physique (hard) vs symbolique (soft)

Lien physique (hard link) :

- Même **inode** que l'original
- Identiques : contenu et droits
- Ne traverse pas les systèmes de fichiers
- Valide même si fichier original supprimé

```
ln original copie_hard
```

Lien symbolique (soft link) :

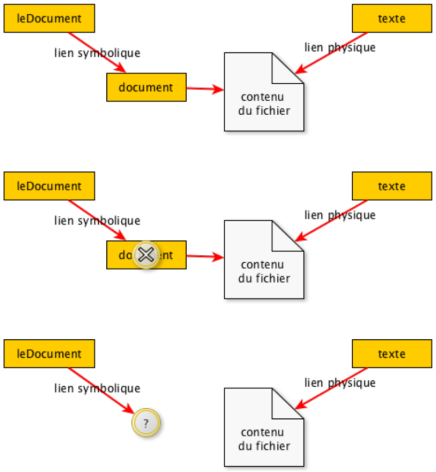
- Contient un chemin vers la cible
- Peut pointer vers fichier ou répertoire
- Traverse les systèmes de fichiers
- Si cible supprimée → lien cassé

```
ln -s cible lien
```

Exemple :

```
$ ln notes.txt notes.hard
$ ln -s notes.txt notes.lnk
$ ls -li notes*
12345 -rw-r--r-- 2 user group ... notes.txt
12345 -rw-r--r-- 2 user group ... notes.hard
67890 lrwxrwxrwx 1 user group ... notes.lnk -> notes.txt
```

Liens : physique (hard) vs symbolique (soft)



Périphériques, FIFO et sockets

Types spéciaux de fichiers

- **Périphérique bloc** (`b`) : accès par blocs (ex. disque : `/dev/sda`)
- **Périphérique caractère** (`c`) : flux de caractères (ex. terminal : `/dev/tty1`, générateur aléatoire : `/dev/random`)
- **FIFO** (`p`) : tube nommé pour communication entre processus (création : `mkfifo /tmp/mafifo`)
- **Socket** (`s`) : communication locale entre processus (IPC) (ex. sockets dans `/run/`, `/var/run/`)

Vérifier le type avec `file` :

```
$ file /dev/sda      # "block special file"
$ file /dev/tty1     # "character special file"
$ file /tmp/mafifo   # "fifo (named pipe)"
```

Permissions Unix

Gestion des permissions sous Unix/Linux

Objectifs du chapitre :

- Comprendre la structure des permissions affichées par `ls -l`.
- Modifier les droits d'accès avec `chmod`.
- Gérer propriétaire et groupe avec `chown` et `chgrp`.
- Relier permissions et sécurité des systèmes multi-utilisateurs.

Structure des permissions

Affichage `ls -l` :

- Exemple : `-rw-r-r- 1 user group 512 notes.txt`
- Champ des permissions : `-rw-r-r-`

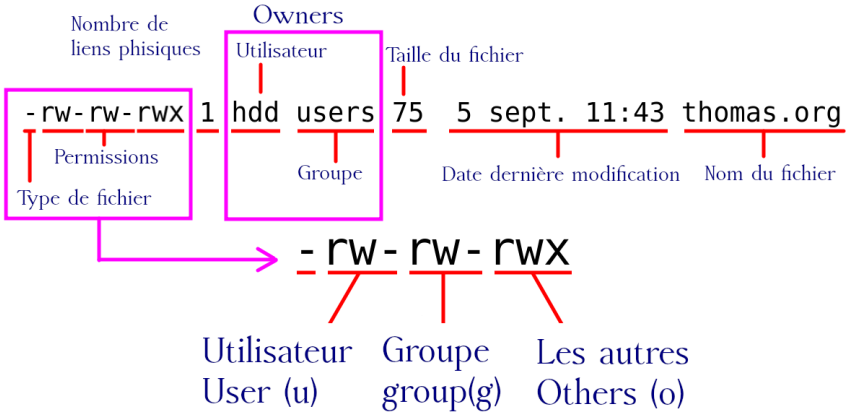
Décomposition :

- 1er caractère : type de fichier (`-`, `d`, `l`, ...)
- Suivent 3 groupes de 3 caractères :
  - **Propriétaire (user)** : lecture (`r`), écriture (`w`), exécution (`x`)
  - **Groupe (group)**
  - **Autres (others)**

Exemple

`-rw-r-r-` ⇒ fichier ordinaire, lecture/écriture pour le propriétaire, lecture seule pour groupe et autres.

Structure des permissions



Commande chmod

But : modifier les permissions d'un fichier ou d'un répertoire.

Deux modes d'utilisation :

- **Symbolique** : ajouter/retirer des droits

```
$ chmod u+x script.sh    # ajoute exécution au propriétaire
$ chmod g-w notes.txt    # enlève écriture au groupe
$ chmod o+r notes.txt    # ajoute lecture aux autres
```

- **Octal** : définir les permissions directement

```
$ chmod 644 notes.txt    # -rw-r--r--
$ chmod 755 script.sh    # -rwxr-xr-x
```

Masques et calcul binaire dans `chmod`

Comprendre les masques et le calcul binaire

Représentation binaire des droits :

- `r` = lecture = 4
- `w` = écriture = 2
- `x` = exécution = 1

Binaire	Droit	Valeur
100	r--	4
010	-w-	2
001	--x	1

Exemple :

- `chmod 754 fichier`
- Utilisateur : 7 = 111 = rwx
- Groupe : 5 = 101 = r-x
- Autres : 4 = 100 = r--

Valeur	Droits
7	rwx
5	r-x
4	r--

Commande `chown` et `chgrp`

But : changer propriétaire et groupe d'un fichier.

- `chown user fichier`

```
$ sudo chown alice notes.txt
```

- `chgrp group fichier`

```
$ sudo chgrp etudiants notes.txt
```

- `chown user:group fichier`

```
$ sudo chown alice:etudiants notes.txt
```

Attention

Seul l'utilisateur **root** ou le propriétaire peut modifier ces attributs.

Résumé des permissions

- Trois catégories : **user**, **group**, **others**.
- Trois droits : **read (r)**, **write (w)**, **execute (x)**.
- Affichés par `ls -l` sous la forme : `-rwxr-xr-`.
- Modifiés avec :
  - `chmod` (droits d'accès)
  - `chown`, `chgrp` (propriétaire et groupe)

À retenir

Les permissions sont un mécanisme essentiel pour la **sécurité et le partage** des fichiers dans un système multi-utilisateurs.

Quelques commandes à retenir (permissions)

- `ls -l`
- `stat`
- `umask`
- `namei -l`
- `getfacl`
- `setfacl`
- `id`
- `whoami`

- `chmod u+x`
- `chmod g-w`
- `chmod o+r`
- `chmod 644`
- `chmod 755`
- `chmod -R`
- `find -perm`
- `lsattr`

- `chown user fichier`
- `chgrp group fichier`
- `chown user:group`
- `sudo chown -R`
- `sudo chgrp -R`
- `groups`
- `chattr`
- `sudo`

Groupes, utilisateurs et sécurité

Gestion des utilisateurs et groupes sous Unix/Linux

Objectifs du chapitre :

- Comprendre la notion d'utilisateur et de groupe.
- Connaître les fichiers systèmes liés : `/etc/passwd`, `/etc/group`.
- Utiliser les commandes de gestion des comptes et groupes.
- Relier ces notions à la **sécurité du système**.

Notion d'utilisateur et groupe

- Chaque utilisateur est identifié par :
  - un **nom** (login)
  - un **UID** (User ID)
  - un **GID principal** (Group ID)
  - un répertoire personnel (`/home/user`)
  - un shell par défaut (souvent `/bin/bash`)
- Les **groupes** permettent de partager des ressources :
  - Chaque groupe a un **nom** et un **GID**.
  - Un utilisateur peut appartenir à plusieurs groupes.

Fichiers systèmes

Deux fichiers essentiels :

- `/etc/passwd` : infos utilisateurs
- `/etc/group` : infos groupes

Exemple :

```
/etc/passwd :
alice:x:1001:1001:Alice:/home/alice:/bin/bash
bob:x:1002:1002:Bob:/home/bob:/bin/zsh

/etc/group :
etudiants:x:1001:alice,bob
admin:x:27:root,alice
```

Commandes principales

Utilisateurs :

```
$ whoami          # affiche l'utilisateur courant
$ id              # affiche UID, GID et groupes
$ adduser alice   # ajoute un utilisateur (Debian/Ubuntu)
$ useradd bob     # ajoute un utilisateur (RedHat)
$ passwd alice    # change le mot de passe
```

Groupes :

```
$ groups alice        # affiche les groupes d'un utilisateur
$ groupadd profs      # ajoute un groupe
$ usermod -aG profs alice    # ajoute alice au groupe profs
```

Sécurité et bonnes pratiques

- Principe du **moindre privilège** : utiliser un compte normal pour les tâches courantes.
- Utiliser `sudo` plutôt que se connecter directement en `root`.
- Contrôler les accès aux fichiers via :
  - permissions (chmod)
  - propriétaires/groupes (chown, chgrp)
- Importance des groupes pour organiser les droits partagés :
  - ex. groupe `etudiants`, groupe `profs`

À retenir

La gestion fine des utilisateurs et groupes est un **pilier de la sécurité** sous Linux.

Quelques commandes à retenir (utilisateurs et groupes)

- whoami
- id
- groups
- users
- who
- w
- finger
- passwd

- adduser
- useradd
- deluser
- userdel
- usermod
- chsh
- chfn

- groupadd
- addgroup
- groupdel
- groupmod
- gpasswd
- newgrp
- sg

- sudo
- visudo
- login
- logout
- last
- lastlog
- who -u
- su

Redirections et pipes

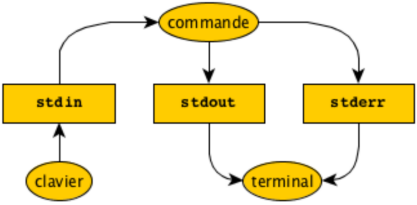
Redirections et communication entre processus

Objectifs du chapitre :

- Comprendre la notion de redirection standard sous Unix/Linux.
- Utiliser les opérateurs >, », <.
- Chaîner des commandes avec le | (pipe).
- Dupliquer la sortie avec `tee`.

Flux standards

Sous Unix/Linux, chaque processus utilise trois flux par défaut :

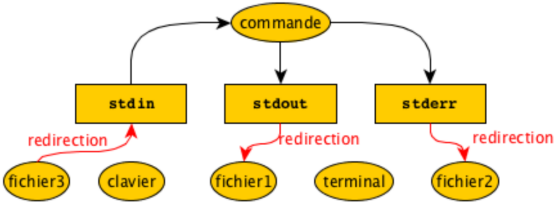


- **Entrée standard (stdin)** : clavier, fichier d'entrée (`fd 0`)
- **Sortie standard (stdout)** : écran, fichier de sortie (`fd 1`)
- **Erreur standard (stderr)** : messages d'erreur (`fd 2`)

Exemple :

```
$ ls fichier.txt       # affichage sur stdout
$ ls fichier_inexistant.txt    # message sur stderr
```

Redirections de base



Rediriger la sortie standard :

```
$ ls > liste.txt      # écrit dans liste.txt (écrase)
$ ls >> liste.txt     # ajoute à la fin de liste.txt
```

Rediriger l'entrée standard :

```
$ wc -l < liste.txt   # compte les lignes du fichier
```

Rediriger l'erreur standard :

```
$ ls /root 2> erreurs.txt   # envoie stderr dans erreurs.txt
```

Pipes |

Principe : connecter la sortie d'une commande à l'entrée d'une autre.

Exemples :

```
$ ls | grep ".txt"          # affiche seulement les .txt
$ ps aux | grep firefox     # cherche un processus Firefox
$ dmesg | less              # lire les logs système page par page
```

À retenir

Le pipe permet de **chaîner des commandes simples** pour construire des traitements complexes.

Commande tee

But : écrire en même temps sur la sortie standard et dans un fichier.

Exemples :

```
$ ls | tee liste.txt
# affiche à l'écran ET écrit dans liste.txt

$ dmesg | tee logs.txt | grep "usb"
# enregistre les logs et filtre sur "usb"
```

Utilité

Très pratique pour **garder une trace** d'une commande tout en continuant à exploiter la sortie.

Résumé : redirections et pipes

- `>` : redirige stdout (écrase le fichier).
- `»` : redirige stdout (ajoute à la fin).
- `<` : redirige stdin.
- `2>` : redirige stderr.
- `|` : envoie la sortie d'une commande vers une autre.
- `tee` : double la sortie (écran + fichier).

À retenir

Les redirections et pipes permettent d'**enchaîner les commandes** et d'automatiser des traitements complexes avec des outils simples.