Système de fichiers (*file system*)
Unix - GNU/Linux

Thierry Vaira

BTS SN La Salle Avignon

v1.0 - 7 décembre 2020

Sommaire

Sommaire

Système de fichiers I

- Un système de fichiers (*file system*) définit l'organisation d'un volume physique ou logique.
- C'est une **structure de données** permettant de stocker les informations et de les organiser dans des **fichiers** sur ce que l'on appelle des **mémoires secondaires** (physiquemement des mémoires de masse comme disque dur, disquette, CD-ROM, clé USB, disques SSD, etc.).
- Un système de fichiers offre à l'utilisateur une **vue abstraite** sur ses données (une **arborescence** de fichiers et de répertoires) et permet de les localiser à partir d'un **chemin d'accès**.

Système de fichiers II

- Le **fichier** est la plus petite entité logique de stockage. Un nom lui est associé pour accéder à son contenu. Les données du fichier sont stockées dans des suites de **blocs** (la plus petite unité du périphérique de stockage).
- Le **formatage** (action de formater c'est-à-dire créer un système de fichiers) prépare un support de données de stockage en y inscrivant un système de fichiers, de façon à ce qu'il soit reconnu par un système d'exploitation.
- Il existe de nombreux systèmes de fichiers différents : FAT, NTFS, HFS, ext, UFS, ISO 9660, ReiserFS, APFS, NFS, etc.

☞ Il existe d'autres façons d'organiser les données, par exemple les bases de données.

Besoins

- la quantité de données à traiter est trop grande et ne peut pas être stockée dans la mémoire centrale (RAM)
- un stockage persistant à long terme permettant de sauvegarder les données traitées ou à traiter pour une utilisation future

➠ Il est donc nécessaire de stocker ces données dans des mémoires secondaires sous forme de fichiers.

Système de gestion des fichiers

Le système de gestion des fichiers (SGF) assure plusieurs fonctions :

- Manipulation des fichiers : des opérations sont définies pour permettre la manipulation des fichiers par les programmes d'application.
- Allocation de la place sur mémoires secondaires : les fichiers étant de taille différente et cette taille pouvant être dynamique, le SGF alloue à chaque fichier un nombre variable de blocs de taille fixe.
- Localisation des fichiers : il est nécessaire de pouvoir identifier et retrouver les données ; pour cela, chaque fichier possède un ensemble d'informations descriptives regroupées dans un descripteur de fichier.
- Sécurité et contrôle des fichiers : le SGF permet le partage des fichiers par différents programmes d'applications tout en assurant la sécurité et la confidentialité des données. Le SGF se doit aussi de garantir la conservation des fichiers en cas de panne du matériel ou du logiciel.

Fonctionnalités supplémentaires

Les systèmes de fichiers peuvent inclure :

- la compression
- le chiffrement automatique des données
- une gestion plus ou moins fine des droits d'accès aux fichiers (ACL)
- une journalisation des écritures (robustesse en cas de défaillance du système)
- une distribution ou répartition réseau
- une virtualisation
- des fonctionnalités pour le temps réel
- des restrictions de nommage pour les fichiers

Arborescence

Un système de fichiers est vu comme une **arborescence** : les fichiers sont regroupés dans des **répertoires** (*directory*) ou **dossiers** (concept utilisé par la plupart des systèmes d'exploitation).

- Ces répertoires contiennent soit des fichiers, soit d'autres répertoires.
- Une telle organisation génère une hiérarchie de répertoires et de fichiers organisés en **arbre**.
- Il y a donc un **répertoire racine** et des **sous-répertoires**.
- Sous Unix/Linux, les utilisateurs voient une arborescence de fichiers unique (/). Cet arbre est en fait l'unification de plusieurs systèmes de fichiers (cf. opérations de montage/démontage).
- Dans un système Windows, les périphériques de stockage de données et les partitions sont affichés comme des lecteurs indépendants (`C:`, `D:`, ...) possédant leur propre arborescence (\).

Notion de fichier

- Un fichier est une suite d'octets portant un nom et conservé dans une mémoire.
- Le nom d'un fichier est une chaîne de caractères (généralement en Unicode) souvent de taille limitée. Certains caractères spécifiques peuvent être interdits ou déconseillés.
- Le contenu du fichier peut représenter n'importe quelle donnée binaire : un programme, une image, un texte, etc.
- Les fichiers sont classés dans des groupes appelés **répertoires**, chaque répertoire peut contenir d'autres répertoires formant ainsi une organisation arborescente.
- Les fichiers sont la plupart du temps conservés (stockés) sur des mémoires de masse tels que les disques durs.
- Dans un système d'exploitation multi-utilisateurs, les programmes qui manipulent le système de fichier effectuent des contrôles d'accès (notion de droits).

Extension

- Le nom d'un fichier peut possèder un suffixe (une extension) séparé par un point qui est fonction du contenu du fichier : `.txt` pour un fichier texte par exemple.
- Sous Windows, cette extension permettra de choisir les applications qui prendront en charge ce format de fichier. Historiquement, l'extension était codée sur 3 caractères dans le système de fichiers FAT utilisé par Microsoft.
- Sous GNU/Linux, l'extension fait simplement partie du nom de fichier. Le format du fichier est détecté par le type **MIME** inscrit dans l'en-tête des fichiers.

Le fichier dans un système de fichiers

Chaque fichier est vu par le système de fichiers de plusieurs façons :

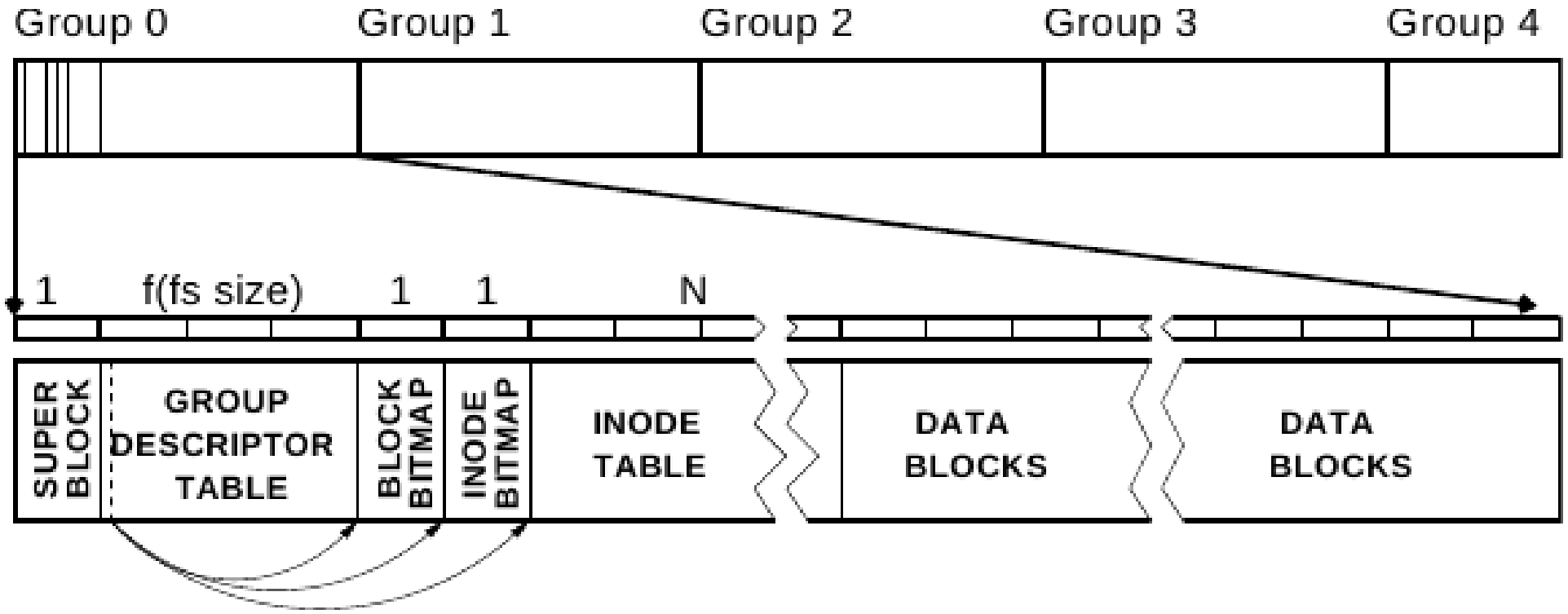
- un **descripteur de fichier** (souvent un entier unique) permettant de l'identifier
- une **entrée dans un répertoire** permettant de le situer et de le nommer
- des **métadonnées** sur le fichier permettant de le définir et de le décrire
- **un ou plusieurs blocs** (selon sa taille) permettant d'accéder aux données du fichier (son contenu)

Sommaire

Le système de fichiers *extfs*

- `ext` ou `extfs` (*extended file system*) est le système de fichiers historique de GNU/Linux. Il a été créé à l'origine par **Rémy Card**, un développeur français. Il a connu plusieurs versions successives : `ext2`, `ext3` (ajout de la journalisation), `ext4`.
- L'unité d'allocation d'`ext2` est le **bloc**. Un bloc représente n x 512 octets (valeur de n fixée à la création du FS). Par défaut, un bloc a une taille de **4096 octets** (4 KiO).
- La taille maximale d'une partition `ext2`/`ext3` dépend de la taille d'un bloc (16 Tio pour un bloc de 4 Kio).
- Le système de fichier `ext2` découpe la partition en **groupe de blocs**.
- Chaque groupe de blocs est géré indépendamment.

ext2


Group 0
Group 1
Group 2
Group 3
Group 4
1
f(fs size)
1
1
N
SUPER BLOCK
GROUP DESCRIPTOR TABLE
BLOCK BITMAP
INODE BITMAP
INODE TABLE
DATA BLOCKS
DATA BLOCKS

Le système de fichiers *ext* I

- Le super block contient les informations de contrôle sur le FS. Il est dupliqué dans plusieurs groupes de blocs.
- La GDT (*Group Descriptor Table*) contient un ou plusieurs blocs qui contiennent des descripteurs de groupe. Un descripteur de groupe contient l'emplacement du *block bitmap*, *inode bitmap*, et le début de la table des inodes pour son groupe de bloc. Il contient également le nombre de blocs libres, d'inodes libres, et des répertoires attribués pour son groupe.
- Il y a un descripteur de groupe pour chaque groupe dans le système de fichiers, de sorte que le nombre de blocs qui compose le descripteur de groupe dépend du nombre de groupes dans le système de fichiers, ce qui à son tour, dépend de la taille du système de fichiers.

Le système de fichiers *ext* II

- La table d'allocation des blocs (*block bitmap*) est une table de bits où chaque bit indique si le bloc correspondant est alloué (bit à 1) ou disponible (à 0).
- La table d'allocation des inodes (*inode bitmap*) est une table de bits où chaque bit indique si l'inode correspondant est alloué (bit à 1) ou disponible (à 0).
- La table des inodes (*inode table*) contient la table des descripteurs de fichiers. Chaque fichier est identifié par un numéro unique. La table des inodes a une taille fixée statiquement à la création du système de fichiers qui détermine donc le nombre maximal de fichiers qu'elle pourra contenir. Cette table contient plusieurs entrées réservées (le premier inode disponible est réellement le 11).

Le système de fichiers *ext* III

☞ Parce que la table de descripteurs de groupe est une donnée critique du système de fichiers, des copies de sauvegarde de la table des descripteurs de groupe sont placés dans les mêmes groupes que la sauvegarde des superblocs. Il existe des copies de sauvegarde du superbloc stockées dans les groupes de bloc avec des nombres entiers qui sont des puissances de 3, 5 et 7 (soit 1, 3, 5, 7, 9, 25, 27, 49, ...).

➠ `https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout`

inode

- Le terme **inode** (contraction de « index » et « node », en français : nœud d'index) désigne le descripteur d'un fichier sous UNIX.
- L'inode d'un fichier est identifié par un **numéro** (*i-number*) dans le système de fichiers et contient les **métadonnées** du fichier.
- La déclaration de cette structure en langage C se trouve dans ce fichier : `https://elixir.bootlin.com/linux/v2.6.25/source/include/linux/ext4_fs.h`
- Le contenu du fichier est écrit dans un ou plusieurs blocs (de taille fixe) du support de stockage selon la taille du fichier.
- La commande `stat` permet d'afficher l'intégralité du contenu de l'inode.

✍ Un inode avait une taille de 128 octets en `ext2`.

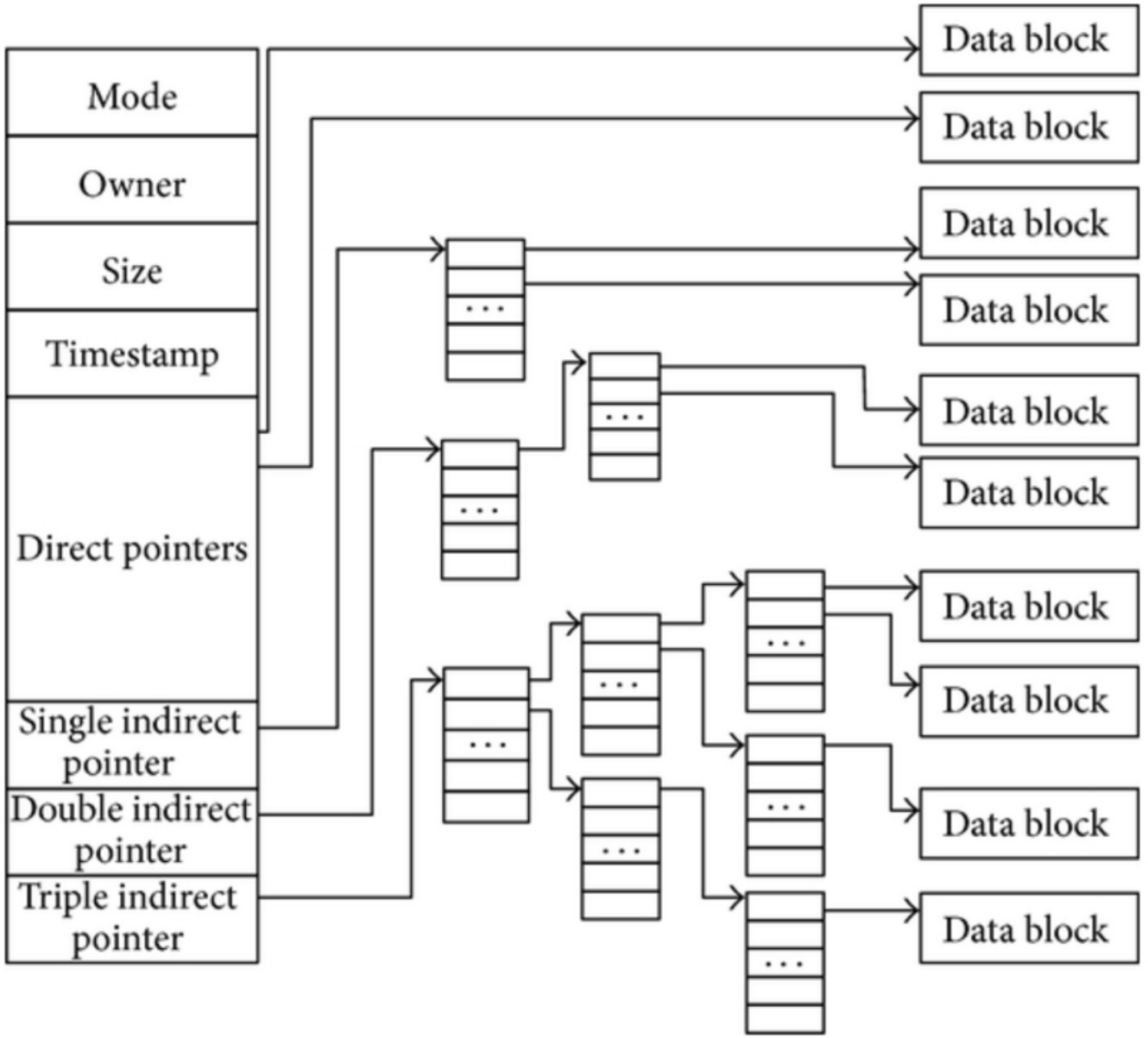
Métadonnées

Les métadonnées les plus courantes sous UNIX sont :

- type de fichier : fichier simple, lien symbolique, répertoire, périphérique, etc.
- droits d'accès en lecture, écriture et exécution selon l'utilisateur, le groupe, ou les autres
- propriétaire et groupe propriétaire du fichier
- horodatages de dernier accès, de modification des métadonnées (inode), de modification des données (bloc)
- nombre de noms de fichiers (liens) pointant vers l'inode
- nombre et numéro de blocs utilisés par le fichier
- taille du fichier

☞ **Les inodes des fichiers ne contiennent pas les noms des fichiers.**

Principe d'un inode



Structure d'un inode ext4

```
/*
 * Structure of an inode on the disk
 */
struct ext4_inode {
  __le16 i_mode;  /* File mode */
  __le16 i_uid;   /* Low 16 bits of Owner Uid */
  __le32 i_size_lo; /* Size in bytes */
  __le32 i_atime; /* Access time */
  __le32 i_ctime; /* Inode Change time */
  __le32 i_mtime; /* Modification time */
  __le32 i_dtime; /* Deletion Time */
  __le16 i_gid;   /* Low 16 bits of Group Id */
  __le16 i_links_count; /* Links count */
  __le32 i_blocks_lo; /* Blocks count */
  __le32 i_flags; /* File flags */
  ...
  __le32 i_ctime_extra; /* extra Change time   (nsec << 2 | epoch) */
  __le32 i_mtime_extra; /* extra Modification time(nsec << 2 | epoch) */
  __le32 i_atime_extra; /* extra Access time   (nsec << 2 | epoch) */
  __le32 i_crtime;      /* File Creation time */
  __le32 i_crtime_extra; /* extra FileCreationtime (nsec << 2 | epoch) */
  __le32 i_version_hi; /* high 32 bits for 64-bit version */
};
```

Exemple

```
$ ls -il Makefile
13107382 -rw-r--r-- 1 tvaira tvaira 22074 mai 22 18:06 Makefile
^
Numéro d'inode

$ stat Makefile
  Fichier : « Makefile »
   Taille : 22074     Blocs : 48        Blocs d'E/S : 4096 fichier
Périphérique : 805h/2053d Inoeud : 13107382 Liens : 1
Accès : (0644/-rw-r--r--) UID : ( 1029/ tvaira) GID : ( 1000/ tvaira)
 Accès : 2017-05-09 08:18:56.725487886 +0200
Modif. : 2017-05-22 18:06:04.921242241 +0200
Changt : 2017-05-22 18:06:04.921242241 +0200
  Créé : -

// Attention :
$ stat Makefile --printf="Nb blocs = %b (avec taille bloc : %B octets)\n"
Nb blocs = 48 (avec taille bloc : 512 octets)
```

Entrées de répertoire

- Un répertoire est un fichier de type particulier, composés de blocs de données structurés en une suite d'entrées.
- Les « entrées » d'un répertoire sont donc des fichiers et des sous-répertoires.
- Les explications sur la structure d'une « entrée » de répertoire : `https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout`
- La déclaration de cette structure en langage C se trouve dans ce fichier : `https://elixir.bootlin.com/linux/v2.6.25/source/include/linux/ext4_fs.h`

Structure d'une entrée de répertoire

```
#define EXT4_NAME_LEN 255

struct ext4_dir_entry {
  __le32 inode;     /* Inode number */
  __le16 rec_len;   /* Directory entry length */
  __le16 name_len;  /* Name length */
  char  name[EXT4_NAME_LEN]; /* File name */
};

// The new version of the directory entry. ...
struct ext4_dir_entry_2 {
  __le32 inode;     /* Inode number */
  __le16 rec_len;   /* Directory entry length */
  __u8  name_len;  /* Name length */
  __u8  file_type;
  char  name[EXT4_NAME_LEN]; /* File name */
};
```

Type de fichiers

```
/*
 * Ext4 directory file types. Only the low 3 bits are used. The
 * other bits are reserved for now.
 */
#define EXT4_FT_UNKNOWN   0
#define EXT4_FT_REG_FILE  1
#define EXT4_FT_DIR       2
#define EXT4_FT_CHRDEV    3
#define EXT4_FT_BLKDEV    4
#define EXT4_FT_FIFO      5
#define EXT4_FT_SOCK      6
#define EXT4_FT_SYMLINK   7

#define EXT4_FT_MAX       8
```

Remarques

- Depuis `ext4`, les données sont stockées dans l'ordre des octets Intel, c'est-à-dire en *little endian*.
- Pour assurer une optimisation, la taille des « entrées » de répertoire sera alignée sur une architecture 32 bits (soit un multiple de 4 octets). Il y aura donc un bourrage avec des valeurs 0x00 pour assurer cet alignement.
- Une structure factice `ext4_dir_entry` est placée à la fin de chaque bloc pour contenir une somme de contrôle (*checksum*). Cette entrée de répertoire a une taille de 12 octets. Les champs `inode` et `name_len` sont mis à zéro pour permettre une compatibilité avec les anciennes version d'`ext`. La somme de contrôle est stockée à l'emplacement où se trouve normalement le nom.

Exemple de décodage

Exemple de décodage de la première « entrée » de répertoire soit les 12 octets du bloc :

```
9c 00 84 01 0c 00 01 02 2e 00 00 00
```

Champ	Taille	Valeur	Description
Inode number	4 octets	9c 00 84 01	0184009c = 25428124
Directory entry length	2 octets	0c 00	12 octets
Name length	1 octet	01	1 caractère ou 1 octet
Type	1 octet	02	EXT4_FT_DIR = un répertoire
File name	255 octets max	2e	.

☞ Ici, il y a 3 octets (0x00) qui ont été ajoutés pour assurer un alignement 32 bits.

Administration d'un système de fichiers

- La création d'une partition sur un disque existant ou un nouveau disque peut se faire avec `parted`, `fdisk`, `cfdisk`, `sfdisk` ou l'outil **GParted**. Voir aussi : `testdisk` pour la récupération/réparation.
- La création d'un système de fichier (formatage) sur une partition peut se faire avec la commande `mkfs`.
- L'administration d'un système de fichier peut se faire avec les commandes `fsck`, `tune2fs`, `dumpe2fs` et `debugfs`.
- On peut réaliser des copies physiques avec la commande `dd`.
- Un système de fichiers doit être monter sur l'arborescence racine avec la commande `mount`. Et le démontage avec la commande `umount`.
- Les systèmes de fichiers montés automatiquement au démarrage du système sont listés dans `/etc/fstab`. Le fichier `/etc/mtab` contient la liste des systèmes de fichiers actuellement montés sur le système.

L'arborescence

Le système de fichiers sous Linux se compose d'une hiérarchie de répertoires, sous-répertoires et fichiers. Le répertoire le plus élevé dans l'arborescence est nommé la racine (*root*) symbolisé par `/`. L'arborescence est unique. Quelques répertoires importants :

- `/etc` : contient les fichiers de configuration du système et des applications
- `/dev` : contient les fichiers spéciaux de périphériques qui représente les points d'accès au matériel
- `/bin` : contient les commandes de base du système
- `/sbin` : contient les outils systèmes pour l'administration
- `/usr` : contient les commandes et applications pour les utilisateurs, dont les environnements graphiques
- `/home` : contient les répertoires personnels des utilisateurs
- `/var` : contient les fichiers dont le contenu varie en fonction de l'utilisation du système (bases de données, fichiers de logs, ...)
- `/proc` : représente le point d'accès aux informations (variables, tables, liste, ...) du noyau et des processus

Type de fichiers

- Sous Unix, un fichier n'est pas structuré : c'est une suite d'octets. On distingue en général deux types de fichiers : **texte** et **binaire**.
- Les **fichiers textes** ont un contenu pouvant être interprété comme du texte (des caractères la plupart du temps codés en ASCII). On utilise habituellement un éditeur de texte pour manipuler ce type de fichiers. Exemples de fichiers textes : code source d'un programme, scripts, fichiers de configuration, etc .
- Les **fichiers binaires** sont tous les fichiers autres que des fichiers textes. Le contenu d'un fichier binaire correspond souvent à un format précis lié à l'usage d'un logiciel applicatif spécifique. Exemples de formats binaires usuels : fichiers exécutable (code machine), fichiers de base de données, fichiers multimédias : images, sons, vidéos, documents textes (traitement de texte), etc.

Autres fichiers

- Un fichier (texte ou binaire) qui a subi une transformation par un algorithme en vue de diminuer sa taille est appelé **fichier compressé**. Le fichier transformé est un fichier binaire.
- Une **archive** est un fichier dans lequel se trouve regroupé des fichiers ou tout le contenu d'une arborescence. Le but principal d'une archive est de tout contenir (données + descriptions) en un seul fichier. Le fichier archive est un fichier binaire. Les archives sont souvent compressées.

Philosophie

Sous UNIX, **TOUT EST FICHIER !**

Ce principe permet au système de fournir une API générique basée sur 4 appels : `open`, `read`, `write` et `close` (complétée par des appels comme `ioctl`, `fnctl`, ...).

Le système d'exploitation distinguera ensuite les types de fichiers suivants :

- les fichiers « normaux » (*regular*) : texte ou binaire (`-`)
- les fichiers « répertoires » (*directory*) (`d`)
- les fichiers « périphériques » (*device*) (`c` ou (`b`))
- les fichiers « liens symboliques » (`l`)
- les fichiers « *socket* » (`s`)
- les fichiers « tubes nommés » (*pipe*) (`p`)

☞ Le type de fichier est indiqué dans l'inode (commande `stat`) ou visualisé par la commande `ls -l`.

Notions de chemin

Chemins relatifs :

- `.` : répertoire courant ("ici")
- `..` : répertoire parent ("au dessus")
- `~` ou `$HOME` : répertoire personnel ("chez moi")

Chemin absolu :

- `/` : la racine (`root`)

Manipuler les fichiers

- `ls` : lister le contenu d'un répertoire
- `stat`, `file`, `getfacl`, `lsattr` : lister les informations sur un fichier
- `cat`, `more`, `less`, `strings`, ... : afficher le contenu texte d'un fichier
- `touch` : créer un fichier vide, modifier l'horodatage d'un fichier
- `cp`, `mv`, `rm` : copier, déplacer, renommer, supprimer
- `find`, `which`, `whereis`, `locate` : rechercher et localiser des fichiers
- `ed`, `vi`, `vim` : éditer un fichier
- `chmod`, `setfacl`, `chattr`, `umask` : modifier les droits, permissions d'accès, attributs, masque de création
- `chown`, `chgrp` : modifier le propriétaire
- `du`, `df` : évaluer l'espace disque
- `fuser`, `lsof` : identifier les processus qui utilisent des fichiers et lister les fichiers ouverts.

Contrôler l'accès aux fichiers

Sous UNIX, il existe deux types de sécurité pour les fichiers et répertoires : les droits et permissions UNIX et les ACL (*Access Control List*).
Il y a trois types de permissions :

- `r` : accès en lecture
- `w` : accès en écriture
- `x` : exécution (fichier), traversée (répertoire)

Chacune de ces permissions peuvent être attribuée à :

- `u` : l'utilisateur (propriétaire)
- `g` : le groupe (propriétaire)
- `o` : les autres
- `a` : tout le monde

Droits et permissions supplémentaires UNIX

- SETUID : exécution avec l'UID du propriétaire du fichier, pas d'effet sur les répertoires
- SETGID : exécution avec l'GID propriétaire du fichier, création d'un répertoire avec le GID propriétaire du répertoire parent
- BIT STICKY : suppression restreinte pour le répertoire, conserver l'image d'un programme en mémoire

Attributs des fichiers

Les nouveaux attributs des fichiers :

- ajout uniquement (`a` : *append*)
- pas de mise à jour de la date d'accès (`A` : *no atime updates*)
- compressé (`c` : *compressed*)
- pas de copie sur écriture (`C` : no copy on write)
- pas de dump (`d` : *no dump*)
- mises à jour synchrones des répertoires (`D` : *synchronous directory updates*)
- format étendu (`e` : *extent format*)
- immuable (`i` : *immutable*)
- journalisation des données (`j` : *data journalling*)
- suppression sécurisée (`s` : *secure deletion*)
- mises à jour synchrones (S : *synchronous updates*)
- pas de fusion des fins de fichiers (`t` : *no tail-merging*)
- répertoire racine (`T` : *top of directory hierarchy*)
- non supprimable (`u` : *undeletable*)

✍ Tous les attributs ne sont pas pris en charge ou utilisés par l'ensemble des systèmes de fichiers.

ACL (*Access Control List*) I

Les ACL permettent d'ajouter une gestion avancée des droits :

- Ainsi, il devient possible d'autoriser un utilisateur à effectuer des opérations sur un fichier (ou répertoire) sans autoriser tout un groupe ou tout le reste du monde.
- Au moyen des ACL, on peut donc étendre le nombre d'utilisateurs et de groupes ayant des droits sur un même fichier (ou répertoire).
- Un fichier dont les ACL auront été spécifiés verra s'ajouter un + à la fin de la liste des droits (`ls -l`)
- Il faut installer le *package* `acl` : `[sudo] apt-get install acl`

Les commandes :

- l'attribution des droits : commande `setfacl`
- la lecture des droits : commande `getfacl`

ACL (*Access Control List*) II

Remarques :

- droits par défaut : ajout d'un attribut *default* (`d:`) aux répertoires seulement et qui se transmet à tous les fichiers créés dans le répertoire.
- masque : son intérêt est de pouvoir limiter toutes les permissions d'un fichier sauf celles du propriétaire

Si le système de fichiers de destination le permet :

- La commande `mv` préserve toujours les droits.
- La commande `cp` peut préserver les droits avec l'option `-a`.

Montage d'un système de fichiers

- Le montage d'un système de fichiers consiste à l'attacher à un répertoire (point de montage) d'un système de fichiers déjà actif (l'arbre racine `/`).
- Cette opération est réalisée par la commande `mount`.
- Dans les systèmes Unix, le point de montage externe par défaut est `/mnt` ou `/media`. Le point de montage par défaut des périphériques au démarrage du système est spécifié dans le fichier de configuration `/etc/fstab`.

☞ On peut également monter des fichiers qui constituent un système de fichiers à eux-seuls (*loopback*) grâce à l'option `-loop` sous GNU/Linux. Ceci est particulièrement utile dans le cas d'images (`.iso`) représentant des disquettes, CDROMs, DVDs. Les commandes `dd` et `mkisofs` peuvent aider à fabriquer de tels fichiers.

Commande mount

Pour monter un périphérique ou une partition avec la commande `mount`, il faut indiquer :

- le type du système de fichiers par l'option `-t`
- le fichier spécial représentant le périphérique ou la partition (généralement `/dev/xxx`)
- le répertoire de montage (généralement `/mnt/yyy`)

✍ Exemple :

```
$ mount -t iso9660 /dev/cdrom /media/cdrom
```

Lorsque le montage a réussi, une mise à jour est effectuée dans un fichier système recensant les montages en cours : le fichier `/etc/mtab` sous GNU/Linux.

Démontage d'un système de fichiers

- Pour démonter une partition ou un périphérique, il faut utiliser la commande `umount`.
- Une fois démonté, le système de fichier n'est plus accessible.
- Le démontage ne marche que si la partition n'est pas utilisée, à savoir :
  - aucun fichier n'est en train d'être lu ou écrit sur la partition ;
  - aucun processus n'a son répertoire de travail sur la partition.

Commande umount

✍ Exemple :

```
$ umount /media/cdrom
```

Lorsque le démontage a eu lieu, le fichier `/etc/mtab` est mis à jour.

Si le démontage est refusé, on peut utiliser la commande `fuser` pour savoir quels processus l'utilisent : `$ fuser -v -m /media/cdrom`

- Pour envoyer à chaque processus, un signal SIGTERM (qui priera le processus de se terminer proprement) :
  `$ fuser -k -TERM -v -m /media/cdrom`
- ou en envoyant à chaque processus un signal SIGKILL (qui le tuera d'autorité) :
  `$ fuser -k -v -m /media/cdrom`
- Voir aussi : la commande `lsof`

Systèmes de fichiers en mémoire : *ramfs* et *tmpfs*

Les systèmes de fichiers en mémoire sont parfois nommés pseudo systèmes de fichiers. Ils ne résident pas sur un disque physique mais en mémoire. Il en existe deux sortes :

- ceux qui permettent de créer un disque en mémoire : **ramfs**.
- ceux qui permettent d'accéder à des informations du système par l'intermédiaire de fichiers : **tmpfs** (il utilise la mémoire partagée du système).

✍ Les répertoires `/tmp`, `/var/log`, `/var/tmp` et `/var/run` sont de « bons candidats » pour `tmpfs`. C'est déjà le cas pour `/var/run` (cf. `/run`). Il suffit d'ajouter une ligne dans `/etc/fstab` :

```
tmpfs /tmp tmpfs defaults,noatime,nosuid,size=100m 0 0
```

ramdisk

- Un *ramdisk* est une zone de mémoire utilisée comme partition.
- Une fois montée, cette partition est utilisable comme n'importe quelle partie du système de fichiers.
- Pour créer un *ramdisk*, rien de plus simple : il suffit de le monter avec la commande `mount` et l'option `-t ramfs` :

```
$ sudo mkdir /ramdisque
$ sudo mount -t ramfs -o maxsize=10M none /ramdisque
```

- On peut aussi faire en sorte qu'il soit créé au démarrage en rajoutant une ligne au fichier `/etc/fstab`.

Qu'est ce que le RAID ?

- Le RAID (*Redudant Array Of Independent Disks*) permet de combiner plusieurs disques en une seule partition dans le but d'augmenter soit la performance, soit la redondance des informations, soit les deux.
- Les différentes méthodes RAID les plus courantes sont nommées par **niveau** (*level*) :
  - le *disk striping* (volume segmenté) ou RAID level 0
  - le *mirroring* (volume en miroir) ou RAID level 1
  - le *disk striping with parity* (volume segmenté avec parité) ou RAID 5
- Il existe deux sortes de RAID :
  - Le RAID matériel (les contrôleurs de disques gèrent eux-mêmes le RAID)
  - Le RAID logiciel (il existe plusieurs solutions pour Linux, dont le pilote **MD**)

Principe

Le principe de base est la distribution des données sur plusieurs disques d'un même ensemble (*disk array*), les disques étant regroupés en un seul volume logique. Les données sont découpées en blocs de taille fixe (*chunks*), puis ces blocs sont distribués sur les différents disques du volume logique suivant un algorithme déterminé par le niveau RAID.

- Level 0 : est utilisé pour améliorer la **performance** en distribuant la charge de lecture/écriture de façon équitable sur tous les disques de l'ensemble.
- Level 1 : est utilisé pour améliorer la **sécurité** en augmentant la redondance, les données étant écrites en plusieurs exemplaires sur plusieurs disques en même temps.
- Level 4 : est destiné à améliorer la sécurité en utilisant un disque pour le stockage de la parité. Ce niveau est très peu utilisé.
- Level 5 : offre un maximum de sécurité avec une bonne performance. Il utilise le principe du striping du Level 0 et la technique de la parité du Level 4, sauf que la parité est répartie sur l'ensemble des disques, éliminant le problème de performance du Level 4. C'est le niveau le plus utilisé.

Configuration

- La configuration d'un ensemble RAID (pilote **MD**) se fait à l'aide de la commande `mdadm`
- Les informations concernant les ensembles RAID en cours de fonctionnement sont accessibles avec le fichier `/proc/mdstat`
- Pour que les ensembles soient activés au démarrage il faut créer un fichier de configuration `/etc/mdadm/mdadm.conf`
- Pour que l'ensemble soit monté automatiquement au démarrage, il faut ajouter une ligne au fichier `/etc/fstab`

LVM

- LVM (*Logical Volume Management*) est à la fois une méthode et un logiciel de gestion de l'utilisation des espaces de stockage d'un ordinateur.
- La gestion par volumes logiques permet de gérer, sécuriser et optimiser de manière souple les espaces de stockage en ligne dans les systèmes d'exploitation de type UNIX.
- `https://fr.wikipedia.org/wiki/Gestion_par_volumes_logiques`

Sommaire

Disques, partitions et systèmes de fichiers

Identifier les disques, les partitions et les systèmes de fichiers montés sur un système :

- les commandes

```
$ lsblk
$ blkid

$ df -hT
$ mount
$ findmnt
```

- les fichiers

```
$ cat /etc/fstab
$ cat /etc/mtab

$ cat /proc/partitions
$ cat /proc/mounts
```

Administration d'un système de fichiers

```
// Création d'un système de fichiers FAT32 sur la partition n°1 du disque sdb
$ sudo mkfs.vfat -F 32 /dev/sdb1

// Création d'un système de fichiers ext2 sur la partition n°2 du disque sdb
$ sudo mkfs.ext2 /dev/sdb2

// Vérification et réparation d'un système de fichiers
$ sudo fsck -a /dev/sdb2

// Affichage des informations sur le superbloc et les groupes de blocs
$ sudo tune2fs -l /dev/sdb2
$ sudo dumpe2fs /dev/sdb2

// Copie du MBR dans un fichier
$ sudo dd if=/dev/sdb of=./MBR.image bs=512 count=1
```

Création et informations sur un fichier

```
$ cd /tmp

// Créer un fichier vide
$ touch toto.txt

// Lister les informations sur le fichier
$ ls -ali toto.txt
7608079 -rw-r--r-- 1 tv tv 0 déc. 6 10:35 toto.txt

$ stat toto.txt
  Fichier : toto.txt
   Taille : 0         Blocs : 0         Blocs d'E/S : 4096 fichier vide
Périphérique : 801h/2049d Inoeud : 7608079 Liens : 1
Accès : (0644/-rw-r--r--) UID : ( 1026/   tv)   GID : (65536/    tv)
Accès : 2020-12-06 10:35:16.015502168 +0100
Modif. : 2020-12-06 10:35:16.015502168 +0100
Changt : 2020-12-06 10:35:16.015502168 +0100
  Créé : -
```

Informations complémentaires sur un fichier

```
// Type de fichier
$ file toto.txt
toto.txt: empty

// Lister les attributs du fichier
$ lsattr toto.txt
--------------e--- toto.txt

// Lister les droits avancés du fichier
$ getfacl toto.txt
# file: toto.txt
# owner: tv
# group: tv
user::rw-
group::r--
other::r--
```

Édition d'un fichier

```
// Modifier le contenu du fichier
$ vim toto.txt
Hello world!

// Lister les informations sur le fichier
$ ls -ali toto.txt
7608079 -rw-r--r-- 1 tv tv 13 déc. 6 11:17 toto.txt

$ file toto.txt
toto.txt: ASCII text

// Le point de montage
$ stat --printf="%m\n" toto.txt
/

// Identifier le disque à partir du moint de montage
$ df -hT | grep -E "/$"
/dev/sda1                       ext4      440G   262G  155G  63% /
```

Informations sur un inode

```
// Lister les informations sur le fichier
$ ls -ali toto.txt
7608079 -rw-r--r-- 1 tv tv 13 déc. 6 11:17 toto.txt

$ echo "stat <7608079>" | sudo debugfs /dev/sda1
Inode: 7608079 Type: regular  Mode: 0644  Flags: 0x80000
Generation: 1447292454 Version: 0x00000000:00000001
User: 1026  Group: 65536 Project:    0  Size: 13
File ACL: 0
Links: 1  Blockcount: 8
Fragment: Address: 0  Number: 0  Size: 0
 ctime: 0x5fccafbb:068f6c90 -- Sun Dec 6 11:17:31 2020
 atime: 0x5fccafbf:232017f8 -- Sun Dec 6 11:17:35 2020
 mtime: 0x5fccafbb:04a72a50 -- Sun Dec 6 11:17:31 2020
crtime: 0x5fcca5d4:03b22d60 -- Sun Dec 6 10:35:16 2020
Size of extra inode fields: 32
Inode checksum: 0xc5e521d5
EXTENTS:
(0):106263584
```

Affichage des données brutes contenues dans un bloc

```
// Affichage d'un bloc
$ sudo dd if=/dev/sda1 bs=4096 skip=106263584 count=1 | hexdump -C
4096 bytes (4,1 kB, 4,0 KiB) copied, 0,000297062 s, 13,8 MB/s
00000000 48 65 6c 6c 6f 20 77 6f 72 6c 64 21 0a 00 00 00 |Hello world!....|
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
00001000

// Affichage du texte
$ strings toto.txt
Hello world!

// Affichage en hexédécimal
$ hexdump -C toto.txt
00000000 48 65 6c 6c 6f 20 77 6f 72 6c 64 21 0a       |Hello world!.|
0000000d
```

Informations sur un répertoire

```
// Lister les informations sur le répertoire
$ ls -alid /tmp
7602177 drwxrwxrwt 42 root root 28672 déc. 6 12:09 /tmp

$ echo "stat <7602177>" | sudo debugfs /dev/sda1
Inode: 7602177 Type: directory  Mode: 01777  Flags: 0x81000
Generation: 3115245653 Version: 0x00000000:00354efc
User:    0   Group:    0   Project:    0   Size: 28672
File ACL: 0
Links: 42  Blockcount: 64
Fragment: Address: 0   Number: 0   Size: 0
 ctime: 0x5fccbc07:e03272d8 -- Sun Dec 6 12:09:59 2020
 atime: 0x5fccbbef:1c055c0c -- Sun Dec 6 12:09:35 2020
 mtime: 0x5fccbc07:e03272d8 -- Sun Dec 6 12:09:59 2020
crtime: 0x5b4dd045:2cb70a24 -- Tue Jul 17 13:17:25 2018
Size of extra inode fields: 32
Inode checksum: 0xd5729a6b
EXTENTS:
(ETB0):30418557, (0):30416928, (1):30417543, (2):30417296, (3):30417285, (4)
     :30418553, (5-6):30426425-30426426
```

Affichage des données brutes contenues dans un bloc

```
// Affichage d'un bloc associé à un répertoire
$ sudo dd if=/dev/sda1 bs=4096 skip=30418553 count=1 | hexdump -C
4096 bytes (4,1 kB, 4,0 KiB) copied, 0,000297062 s, 13,8 MB/s
...
00000200 2d 64 58 53 30 4c 39 2e 6c 79 78 68 0f 17 74 00 |-dXS0L9.lyxh..t.|
00000210 14 00 08 01 74 6f 74 6f 2e 74 78 74 2e 78 6b 6d |....toto.txt.xkm|
00000220 36 1c 74 00 14 00 0a 01 6e 53 37 39 32 30 2e 61 |6.t.....nS7920.a|
...

740001

// Affichage du texte
$ strings toto.txt
Hello world!

// Affichage en hexédécimal
$ hexdump -C toto.txt
00000000 48 65 6c 6c 6f 20 77 6f 72 6c 64 21 0a       |Hello world!.|
0000000d
```