

TP3-UTC502

Gestion des volumes et sauvegarde

Groupe 1 :

BIGAUD Emilie

DOUÇOT Kévin

LE COZ Julien

PARIS Loic

SARAHANABHAVAN Saranya

Manipulation 1 : Système de fichier présent

Nous remarquons que les types de système de fichiers présents sur nos VM sont les suivants :

- **tmpfs** : Temporary File System, est un système de fichier temporaire propre à unix. Celui-ci disparaît lors de l'arrêt de la machine.
- **ext4** : C'est le système de fichier propre à Linux. C'est l'équivalent sous windows au FS fat ou NTFS.
- **devtmpfs** : Correspond au système de fichier de périphérique. Tout pilote ou périphérique voulant se connecter passera par ce système. Cela nous servira pour le `/dev`.

Pour les FS tmpfs et devtmpfs, les répertoires de montages sont rattachés à des répertoires dit temporaire tel que `/dev` ou `/run`, c'est-à-dire qu'à chaque redémarrage les répertoires sont remis à 0.

Pour le FS ext4, le répertoire est la racine avec le `/` ou `/home`.

1. Utilisation de la commande *mount*

La commande *mount* sans option permet d'afficher tous les FS présent ainsi que leur répertoire de montage.

```
user@Lab01C502:~$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=1989676k,nr_inodes=497419,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=402596k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
cgroup2 on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate,memory_recursiveprot)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
none on /sys/fs/bpf type bpf (rw,nosuid,nodev,noexec,relatime,mode=700)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=29,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=10876)
debugfs on /sys/kernel/debug type debugfs (rw,nosuid,nodev,noexec,relatime)
tracefs on /sys/kernel/tracing type tracefs (rw,nosuid,nodev,noexec,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
configfs on /sys/kernel/config type configfs (rw,nosuid,nodev,noexec,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,nosuid,nodev,noexec,relatime)
/var/lib/snapd/snaps/core18_2708.snap on /snap/core18/2708 type squashfs (ro,nodev,relatime,x-gdu.hide)
/var/lib/snapd/snaps/core_14784.snap on /snap/core/14784 type squashfs (ro,nodev,relatime,x-gdu.hide)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,nosuid,nodev,noexec,relatime)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=402592k,nr_inodes=100648,mode=700,uid=1000,gid=1000)
/var/lib/snapd/snaps/core18_2721.snap on /snap/core18/2721 type squashfs (ro,nodev,relatime,x-gdu.hide)
/var/lib/snapd/snaps/core_14946.snap on /snap/core/14946 type squashfs (ro,nodev,relatime,x-gdu.hide)
```

2. Utilisation de la commande *df*

La commande *df* permet d'afficher les FS présent qui ne sont pas lié à l'OS ou SE (`/sys`), avec les options *-T* pour afficher le type de FS et *-h* pour une vision dite humaine, plus lisible la taille des fichiers sera en Go, Mo ou Ko de façon automatique :

```
$ df -Th
```

Sys. de fichiers	Type	Taille	Utilisé	Dispo	Uti%	Monté sur
udev	devtmpfs	947M	0	947M	0%	/dev
tmpfs	tmpfs	198M	944K	197M	1%	/run
/dev/sda1	ext4	24G	13G	11G	55%	/
tmpfs	tmpfs	987M	0	987M	0%	/dev/shm
tmpfs	tmpfs	5,0M	0	5,0M	0%	/run/lock
tmpfs	tmpfs	198M	76K	198M	1%	/run/user/1000

Synthèse

Le commande *mount* permet d'afficher tous les FS présent ainsi que les droits associé (rw, noexec, etc...). Cependant c'est difficilement lisible. Quant à la commande *df*, c'est moins complet concernant les droits mais plus lisible, avec notamment une information supplémentaire sur les tailles des FS ainsi que leur % d'occupation. L'option *-h* permet de traduire la taille en vison dite humain, en Go, Mo ou Ko ; sans cette option cela s'affiche par bloc de 1Ko ou 1024 octets.

Manipulation 2 : Calcul de bloc et réservation root

1. Bloc de 512 octets

Afin de calculer la place restante sous forme de bloc de 512 octets, nous utilisons la commande vue précédemment *df* en utilisant l'option *-B* qui permet d'indiquer le nombre d'octet voulu.

```
$ df -B 512
```

Sys. de fichiers	blocs de 512B	Utilisé	Disponible	Uti%	Monté sur
udev	1938672	0	1938672	0%	/dev
tmpfs	403960	1888	402072	1%	/run
/dev/sda1	49281088	25475328	21251704	55%	/
tmpfs	2019784	0	2019784	0%	/dev/shm
tmpfs	10240	0	10240	0%	/run/lock
tmpfs	403952	152	403800	1%	/run/user/1000

La première ligne indique que le FS calcul avec des blocs de 512B qui signifie Byte en anglais, soit octet en français. La disponibilité pour chaque FS est notée en colonne 4.

2. Réserve à root

Par défaut sous linux, la partie réservée à root sur chaque partition est de 5% de l'espace. Cette partie est modifiable avec la commande *tune2fs* avec l'option *-m* en indiquant le pourcentage voulu. Il est cependant risqué de réduire la partie réservée de root en fonction de la partition, car sur l'une d'elle il y a la partie système, pour le démarrage notamment.

Synthèse

Avec la commande *df* nous pouvons connaître le nombre de bloc restant disponible, ainsi que le pourcentage de la partition utilisé. Très pratique pour contrôler rapidement le taux d'utilisation. L'option *-h* est quand même beaucoup plus pratique pour cette visualisation car plus parlante en termes de taille.

Les systèmes Linux s'efforcent de toujours donner le maximum de contrôle et de personnalisation aux administrateurs, ce qui explique que 5% de l'espace de chaque système de fichier est réservé à root. Ce pourcentage est néanmoins modifiable. Cependant 5% est un excellent compromis : un espace réservé plus grand serait inutile et constituerait donc une perte de place, à contrario, moins de 5% pourrait compromettre les opérations administrateurs sur ce système de fichiers, comme par exemple y accéder même lorsque le système de fichiers en question est plein.

Manipulation 3 : Volume du FS ext4

Nous n'avons pas de FS en ext2 car c'est un système de fichier ancien. Nos VM étant installé récemment nous sommes en ext4. Nous pouvons utiliser la commande *df* sans option qui affiche par défaut la taille des blocs de 1Ko. Mais cela indique les bloc des fichiers, non le bloc de la partition.

Pour regarder la taille des blocs d'une partition, il faut utiliser la commande *dumpe2fs* avec l'option *-h* sur la partition à vérifier :

```
# dumpe2fs -h /dev/sda1 | grep -i block  
Block size: 4096
```

La partition a des blocks de 4096.

Synthèse

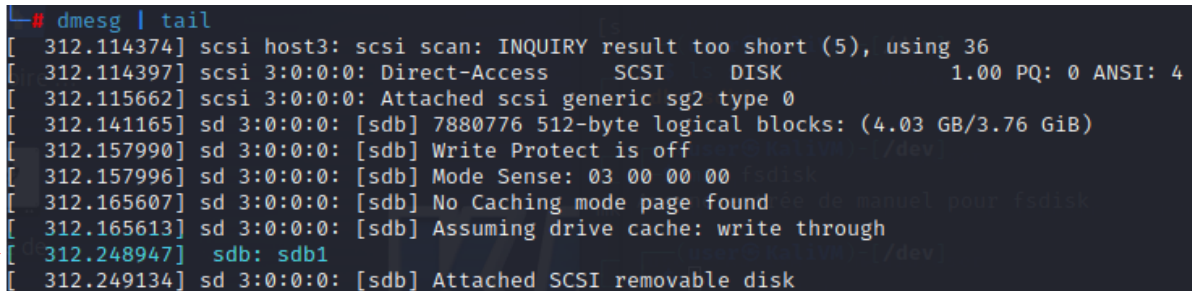
L'importance des blocs de partition est pour la capacité de la taille max d'un fichier qui peut y être stocké. Plus la taille d'un bloc de partition est petite, plus la taille du fichier doit être réduit pour être stocké, sinon un message indiquera que le fichier est trop lourd, non pas par la place restant sur le disque, mais sur la capacité à enregistrer tout le fichier.

La taille du bloc n'indique pas la taille maximale du fichier.

Manipulation 4 : Monter un périphérique

Nous ferons le test avec une clé USB. Lorsque nous branchons la clé USB, nous faisons la commande *dmesg* pour faire apparaître les derniers logs ou événement. Afin de limiter le nombre de ligne, on ajoute */tail* pour afficher les 10 dernières lignes :

```
# dmesg | tail
```



```
└─# dmesg | tail
[ 312.114374] scsi host3: scsi scan: INQUIRY result too short (5), using 36
[ 312.114397] scsi 3:0:0:0: Direct-Access    SCSI    DISK           1.00 PQ: 0 ANSI: 4
[ 312.115662] scsi 3:0:0:0: Attached scsi generic sg2 type 0
[ 312.141165] sd 3:0:0:0: [sdb] 7880776 512-byte logical blocks: (4.03 GB/3.76 GiB)
[ 312.157990] sd 3:0:0:0: [sdb] Write Protect is off
[ 312.157996] sd 3:0:0:0: [sdb] Mode Sense: 03 00 00 00
[ 312.165607] sd 3:0:0:0: [sdb] No Caching mode page found
[ 312.165613] sd 3:0:0:0: [sdb] Assuming drive cache: write through
[ 312.248947] sdb: sdb1
[ 312.249134] sd 3:0:0:0: [sdb] Attached SCSI removable disk
```

La ligne surligné en bleu indique l'emplacement par défaut de la clé USB en *sdb1*, se trouvant dans le répertoire */dev*.

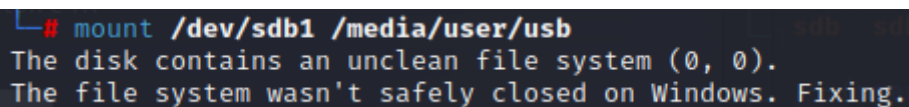
1. Commande *mount*

Pour monter la clé sur une arborescence de notre choix, nous créons un dossier spécifique. Nous utilisons l'arborescence existant */media/user* en créant un dossier que l'on nomme *usb* :

```
# mkdir /media/user/usb
```

Le dossier créer, nous utilisons la commande *mount* pour monter le périphérique :

```
# mount /dev/sdb1 /media/user/usb
```



```
└─# mount /dev/sdb1 /media/user/usb
The disk contains an unclean file system (0, 0).
The file system wasn't safely closed on Windows. Fixing.
```

Nous avons un bon exemple, le VM linux a fermé l'utilisation du périphérique sur le PC Windows pour en avoir un accès unique. En se mettant dans l'arborescence */media/user/usb*, la commande *ls* montre tous les dossier et fichier présent :

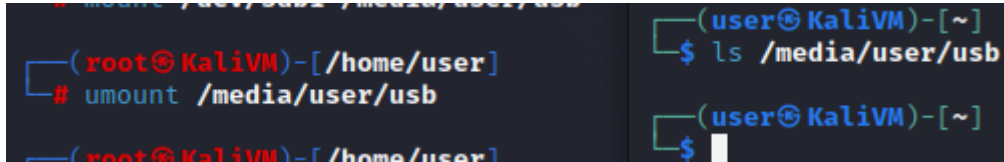


```
(user@KaliVM)-[/media/user/usb]
$ ls
adwcleaner.exe
Anglais
BCC_013400.exe
bluescreenview
bluescreenview_french1.zip
bluescreenview.zip
BMP
buildroot-2014.08-cross-compile.tgz
CCleaner.exe
Cisco
```

2. Commande *umount*/*eject*

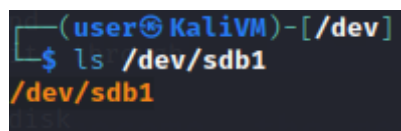
Une fois l'utilisation terminé, nous avons 2 possibilités. Soit démonter le périphérique avec la commande *umount* soit utiliser la commande *eject* pour le retirer :

```
# umount /media/user/usb
```



The screenshot shows two terminal windows. The left window, with a root prompt, shows the command `# umount /media/user/usb` being entered. The right window, with a user prompt, shows the command `$ ls /media/user/usb` being entered, and the output is empty, indicating the directory is now empty.

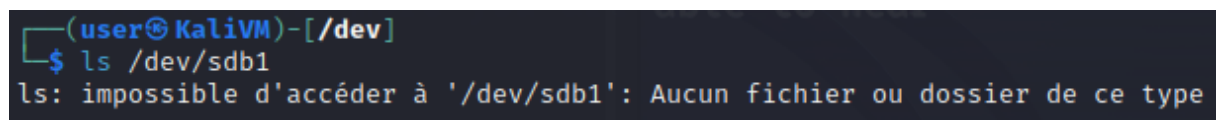
Sur la fenêtre de droite, nous remarquons que le contenu de la clé ne s'affiche plus, mais la clé est toujours présente, comme vu sur l'arborescence par défaut :



The screenshot shows a terminal window with a user prompt. The command `$ ls /dev/sdb1` is entered, and the output is `/dev/sdb1`, indicating that the directory still exists.

Si la commande *umount* est utilisé, la commande *eject* se fera au niveau du répertoire par défaut, le `/dev/sdb1`. Cela peut très bien fonctionner directement sur l'arborescence choisi au `/media/user/usb` :

```
# eject /dev/sdb1
```



The screenshot shows a terminal window with a user prompt. The command `$ ls /dev/sdb1` is entered, and the output is `ls: impossible d'accéder à '/dev/sdb1': Aucun fichier ou dossier de ce type`, indicating that the device is no longer accessible.

Nous constatons que le lecteur n'est plus joignable.

Synthèse

Lorsqu'un nouveau périphérique se connecte, celui-ci se trouve par défaut dans le répertoire `/dev`, comme vu lors de la manipulation 1 car c'est le répertoire dédié. L'utilisation de la commande *mount* permet de monter le périphérique sur une arborescence voulu, un dossier spécifique peut être créé. Pour démonter le périphérique, il est nécessaire d'utiliser la commande *umount* ou la commande *eject* pour retirer directement le périphérique.

Manipulation 5 :

Nous décidons de créer un fichier dans */usr/local* avec la commande *touch* que nous nommons *tp2.txt*. Nous nous mettons en édition de texte avec *vi* :

```
$ vi tp2.txt
```

Sur une autre fenêtre en mode root, nous utilisons la commande *umount* sur l'arborescence */usr/local* :

```
# umount /usr/local
umount: /usr/local: not mounted.
```

Nous avons un message indiquant que le chemin n'est pas monté, qu'il est donc impossible de démonter.

Nous tentons la même commande avec l'option *-f* pour forcer le démontage :

```
# umount -f /usr/local
umount: /usr/local: not mounted.
```

Nous tentons de monter le répertoire sur une arborescence */media/user/tp*, sans succès :

```
(root@kaliVM)-[/home/user]
# mount /usr/local /media/user/tp
mount: /media/user/tp: /usr/local is not a block device.
dmesg(1) may have more information after failed mount system call.
```

Synthèse

Il faut noter que le chemin */usr* est le répertoire dans lequel se trouvent les applications de l'utilisateur et */usr/local* les applications compilées, ce qui peut expliquer qu'on ne peut ni monter ni démonter cette arborescence pour assurer le bon fonctionnement du système et celui de l'utilisateur.

Pour que ce répertoire soit démontable il faut qu'il soit monté sur son propre volume. Dans ce cas-là il pourra être démonté avec la commande *umount*.

Manipulation 6 : Droit accessibilité à un périphérique monté

Nous utilisons la commande utilisée précédemment *mount* avec l'option *-o* pour ajouter des droits d'accès.

```
# mount -o umask=007,uid=1000 /dev/sdb1 /media/user/usb
```

La valeur *umask* signifie que la valeur de bit de permission ne sont pas fournis. Fonctionne dans le contraire de *chmod*.

Nous nous logons sur une fenêtre en tant que l'utilisateur *tintin* (*uid=1001*). Il voit l'emplacement USB mais il ne peut pas voir le contenu de la clé, ni s'y connecter :

```
$ ls
usb

(tintin@KaliVM)-[/media/user]
$ ls usb
ls: impossible d'ouvrir le répertoire 'usb': Permission non accordée

(tintin@KaliVM)-[/media/user]
$ cd usb
bash: cd: usb: Permission non accordée
```

Sur le compte *user* (*uid=1000*), nous voyons les droits associés, y compris à son groupe :

```
(user@KaliVM)-[/media/user]
$ ls -lah usb
total 2,2G
drwxrwx--- 1 user root 4,0K 18
drwxr-xr-x 3 root root 4,0K 24
drwxrwx--- 1 user root 4,0K 12
drwxrwx--- 1 user root  0 13
-rwxrwx--- 1 user root 24M 31
-rwxrwx--- 1 user root 573 13
-rwxrwx--- 2 user root 1,4G 18
-rwxrwx--- 2 user root 801M 29
-rwxrwx--- 2 user root 15K 23
-rwxrwx--- 2 user root 19M  5
drwxrwx--- 1 user root  0 11
-rwxrwx--- 1 user root 942K 30
drwxrwx--- 1 user root  0 11
```

Synthèse

Parmi les nombreuses options de la commande *mount*, on trouve l'option *-o* permettant de spécifier des options de montage. Ces nombreuses options sont listées dans les sections lorsqu'on nous utilise *man* pour avoir le détail. Elles permettent de limiter la consultation du système de fichiers à un seul utilisateur ou un seul groupe, de monter le système de fichier en lecture seule ou écriture seule, permettre ou interdire l'exécution de fichiers binaires, etc.

Manipulation 7 : Montage d'un périphérique automatiquement

Pour que le montage se fasse automatiquement lors d'un démarrage d'un système, il faut ajouter une ligne dans le fichier */etc/fstab*. Il y a 2 possibilités :

- En connaissant l'emplacement par défaut du périphérique :

<file system>	<mount point>	<type>	<options>	<dump>	<pass>
/dev/sdb1	/media/user/usb	auto	defaults	0	0

- Ou en renseignant l'UUID du périphérique

<file system>	<mount point>	<type>	<options>	<dump>	<pass>
UUID=EE6D-F708	/media/user/usb	auto	defaults	0	0

Pour la VM à chaud il faut faire la commande *mount -a*, cela va lire le fichier *fstab* et exécuter le montage sur les points qui ne le sont pas.

Synthèse

S'il s'agit d'un périphérique utilisé régulièrement toujours brancher, il est préférable que celui-ci se monte automatiquement lors du démarrage du système. La méthode la plus sécurisée, et plus stable, est de renseigner l'identifiant UUID du périphérique. De ce fait si son arborescence par défaut change, celui-ci sera tout de même monté. Dans les options il y a possibilité de spécifier quel type d'ouverture, accessibilité à un groupe, un utilisateur nommé etc... En fonction de l'utilisation d'un système pour garantir la sécurité informatique.

Manipulation 8 : Option de *mount*

Voici quelques exemples d'utilisation des options de la commande *mount* :

- **Monter un périphérique** : `mount /dev/sdc3 /media/stock`
- **Spécifier un format de fichier (ex : ext4)** : `mount -t ext4 /dev/sdc3 /media/stock`
- **Déplacement du point de montage** : `mount --move emplacement newemplacement`
- **Remonter une partie de la hiérarchie des fichiers ailleurs** : `mount --bind olddir newdir`
- **Remonter toute une hiérarchie ailleurs** : `mount --rbind olddir newdir`
- **Monter tous les systèmes de fichiers (d'un type donné) mentionnés dans *fstab*** : `mount -a`
- **Montage du système de fichiers en lecture seule** : `mount -ro`
- **Montage du système en lecture/écriture** : `mount -rw`

Synthèse

La commande *mount* offre de très nombreuses fonctionnalités donnant un total contrôle à l'administrateur en ce qui concerne la configuration des systèmes de fichiers. Nous développerons les options et en testerons certains au cours de ce TP.

Cette commande cherche à couvrir toutes les fonctionnalités éventuelles qu'une gestion de systèmes de fichiers peut nécessiter.

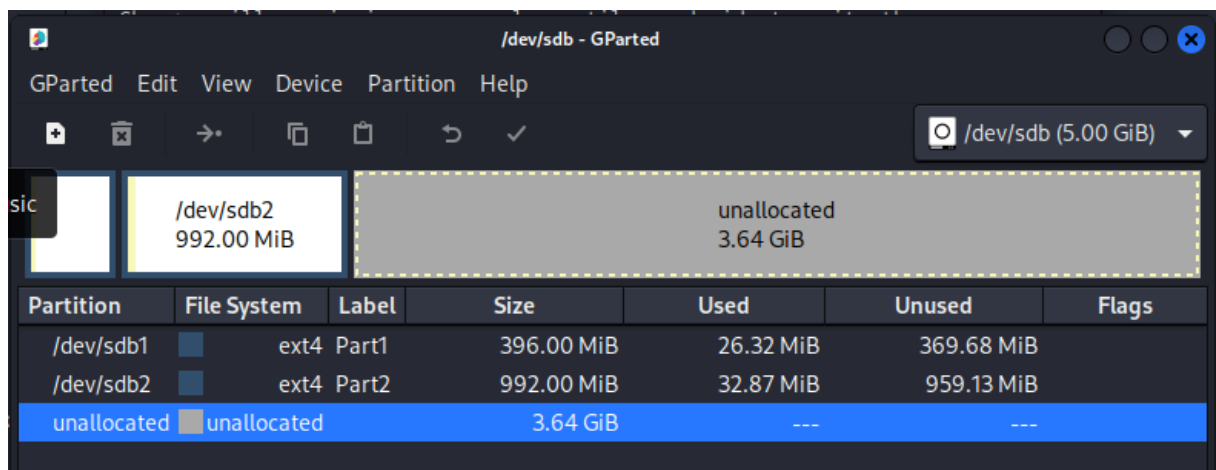
Manipulation 9 : Utilisation de LVM

Voici une représentation visuelle du concept de LVM



1. Partitionnement du disque

Pour faire la partition nous avons utilisé 2 méthodes. Une première plus simple en mode graphique en lançant la commande *gparted* :



Nous voyons distinctement les 2 partitions avec leur taille.

La 2^{ème} méthode est en mode texte avec la commande *fdisk*. Avec l'option *-l* nous listons les disques présents. Celui qui nous intéresse se trouve en */dev/sdb* :

```
Disk /dev/sdb: 465,76 GiB, 500107859968 bytes, 976773164 sectors
Disk model: HTS545050A7E
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x8c16cf43
```

Pour partitionner le disque, nous faisons la commande *fdisk /dev/sdb*. Création de la première partition :

```
Welcome to fdisk (util-linux 2.38.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): n
Partition type
  p   primary (1 primary, 0 extended, 3 free)
  e   extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2): 2
First sector (711038976-976773163, default 711038976):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (711038976-976773163, default 976773163): +396M

Created a new partition 2 of type 'Linux' and of size 396 MiB.
```

Puis de la 2^{ème} partition

```
Command (m for help): n
Partition type
  p   primary (2 primary, 0 extended, 2 free)
  e   extended (container for logical partitions)
Select (default p): p
Using default response p.
Partition number (3,4, default 3): 3
First sector (711849984-976773163, default 711849984):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (711849984-976773163, default 976773163): +992M

Created a new partition 3 of type 'Linux' and of size 992 MiB.
```

Ici nous avons renseigné la valeur *n*, pour avoir l'ensemble des commandes possible et leur action il faut rentrer la valeur *m*.

Lors de certains choix rien n'est mis car le choix par défaut proposé correspond à l'action voulu.

Une fois les 2 partitions créé, nous les vérifions de la manière suivante :

Command (m for help): p

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/sdb1		2048	711038975	711036928	339G	7	HPFS/NTFS/exFAT
/dev/sdb2		711038976	711849983	811008	396M	83	Linux
/dev/sdb3		711849984	713881599	2031616	992M	83	Linux

Nous voyons les 2 partitions créé en sdb2 et sdb3 avec la taille préalablement renseignée. Afin de valider la création des partitions, la valeur pour command est *w* :

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

2. Inclure dans un même groupe VG

Afin de créer les groupes, il faut installer lvm2 sur nos VM qui n'est pas natif. Pour la création de groupe de volume, nous utilisons la commande *vgcreate* avec l'option *-s* pour un extent :

```
# vgcreate VG1 -s 4Mo /dev/sdb1 /dev/sdb2
Physical volume "/dev/sdb1" successfully created.
Physical volume "/dev/sdb2" successfully created.
Volume group "VG1" successfully created
```

Lorsque le groupe est créé, par la suite une partition peut être intégré avec la commande *vgextend* ou bien retirer avec la commande *vgreduce* :

```
# vgextend vg1 /dev/sdb3
Physical volume "/dev/sdb3" successfully created.
Volume group "vg1" successfully extended
```

S'il y a besoin de retirer une partition d'un groupe, nous utiliseront de la commande *vgreduce* :

```
# vgreduce vg1 /dev/sdb
Removed "/dev/sdb" from volume group "vg1"
```

Nous pouvons vérifier le groupe de volume, soit avec la commande *vgs* qui affiche de façon simplifiée soit avec *vgdisplay* GV1 qui est plus détaillé :

```
# vgs
VG #PV #LV #SN Attr VSize VFree
VG1 2 0 0 wz--n- <1.35g <1.35g
```

```
# vgdisplay VG1
--- Volume group ---
VG Name                VG1
System ID
Format                 lvm2
Metadata Areas         2
Metadata Sequence No   1
VG Access              read/write
VG Status              resizable
MAX LV                 0
Cur LV                0
Open LV                0
Max PV                 0
Cur PV                2
Act PV                 2
VG Size                <1.35 GiB
PE Size                4.00 MiB
Total PE               345
Alloc PE / Size        0 / 0
```

```
Free PE / Size      345 / <1.35 GiB
VG UUID             tQzJRu-BZWq-CPob-W50F-Dj0Z-Wvoz-8jtpz1
```

Une fois le groupe créé, nous créons les volumes logiques avec la commande *lvcreate* avec l'option *-n* pour donner un nom et *-L* pour la taille :

```
# lvcreate -n volume1 -L 850M VG1
Rounding up size to full physical extent 852.00 MiB
Logical volume "volume1" created.
```

Vérification du volume :

```
# ls -l /dev/VG1/volume1
lrwxrwxrwx 1 root root 7 Apr 28 18:30 /dev/VG1/volume1 -> ../dm-0
```

Ajout d'un FS (ici ext4) pour pouvoir l'utiliser :

```
# mkfs.ext4 /dev/VG1/volume1
mke2fs 1.46.6-rc1 (12-Sep-2022)
Creating filesystem with 218112 4k blocks and 54544 inodes
Filesystem UUID: a1c0c444-f18b-4d02-a2dd-e3adc5f14fb5
Superblock backups stored on blocks:
    32768, 98304, 163840
Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

Pour le montage du FS, il est nécessaire en amont de créer le dossier sur lequel il va être monté :

```
# mkdir /utilisateur1/
# mount /dev/VG1/volume1 /utilisateur1
```

Contrôle que le FS soit bien monté :

```
# df -h | grep utilisateur1
/dev/mapper/VG1-volume1 821M 24K 763M 1% /utilisateur1
```

Possibilité de contrôler également les LV avec la commande *lvdisplay* :

```
# lvdisplay
--- Logical volume ---
LV Path                /dev/VG1/volume1
LV Name                 volume1
VG Name                 VG1
LV UUID                 W6AfVh-jePL-Cwvi-uPpu-THSx-1xf0-mws5ZK
LV Write Access         read/write
LV Creation host, time  kali, 2023-04-28 18:30:50 +0200
LV Status                available
# open                  1
LV Size                 852.00 MiB
```

```
Current LE      213
Segments       1
Allocation      inherit
Read ahead sectors  auto
- currently set to 256
Block device    254:0
```

S'il y a besoin de supprimer un groupe VG, nous utiliserons la commande *vgremove* :

```
# vgremove vg1
Volume group "vg1" successfully removed

(root@KalivM)-[/home/user]
# vgs

(root@KalivM)-[/home/user]
#
```

Synthèse

Il est important de connaître en amont si nous voulons utiliser le système LVM. Il présente de gros avantages :

- On ne s'occupe plus de connaître l'emplacement exact des données.
- Système très souple qui permet de diminuer ou d'augmenter un FS sans se préoccuper de l'emplacement sur le disque. Cela présente beaucoup moins de risque que de manipuler des partitions.

Il y a cependant un gros inconvénient :

- Si un volume physique tombe en panne, c'est tous les volumes logiques qui utilisent ce PV qui sont perdus.
- L'utilisation de disques raid permet de se protéger.

Manipulation 10 : Copie d'une USB et montage en loopback

1. Partie bootable

Nous ferons une copie d'un fichier iso entre un disque dur et la VM. Pour cela nous utiliseront la commande *dd*. Voici les options :

Options	Contenu
if	Input File, image source à copié
of	Output File, chemin destination pour coller l'image
bs	Block size, choisir la taille des blocks en octet, multiple de 2 et minimum 512octet
skip	Pour ignorer les x premiers Ko

```
(root@KaliVM)-[/home/user]
# dd if=/media/user/usb/Win_Srv2k3.iso of=/media/user/image.iso bs=4096
142521+0 enregistrements lus
142521+0 enregistrements écrits
583766016 octets (584 MB, 557 MiB) copiés, 119,065 s, 4,9 MB/s
```

Nous avons choisi comme taille de bloc 4096. Depuis un CD la valeur de bloc doit être de 2048.

Pour créer une clé bootable, nous ajouterons l'option *&& sync* à la fin :

```
(root@KaliVM)-[/home/user]
# dd if=/media/user/image.iso of=/dev/sdb1 bs=4M status=progress && sync
583766016 octets (584 MB, 557 MiB) copiés, 62 s, 9,4 MB/s
139+1 enregistrements lus
139+1 enregistrements écrits
583766016 octets (584 MB, 557 MiB) copiés, 132,245 s, 4,4 MB/s
```

L'option *status=progress* permet de voir la progression au temps réel.

Nous pouvons voir qu'il y a tous les fichiers propres à une clé bootable.

```
(user@KaliVM)-[~]
$ ls /media/user/usb
autorun.inf  docs  lisezmoi.htm  setup.exe  valueadd  win51is
bootfont.bin  i386  printers  support  win51
```

2. Partie loopback

Nous ré-utilisons la commande *mount*, en y ajoutant le commande *-o* :

```
# mount -o loop image_cd.iso /mnt
```

Le montage en "loopback" permet de créer une boucle de montage, c'est-à-dire un second montage. Monter notre image en loopback va permettre de l'utiliser plutôt que celui du CD ou de la clé USB.

Dans le cas présent notre CD est monté sur */dev/cdrom* et on a créé son image que l'on a mis sur */mnt*, si on met en place une boucle alors */mnt* va pointer sur */dev/cdrom*. Ce qui a pour conséquence qu'à chaque utilisation il y aura un switch soit on sera sur le CD soit sur l'image.

Synthèse

A la suite d'une extraction d'un fichier iso, nous pouvons créer un périphérique bootable. Par la suite sous la forme d'une boucle à l'aide de l'option *-o loop* de la commande *mount*, nous pouvons indiquer au système de lancer, lors de son démarrage, une exécution sur un périphérique, mais qui si celui-ci ne répond d'aller chercher l'image directement sur l'arborescence indiqué.

Le montage d'une image d'un CD en loopback peut-être utile de plusieurs façons :

- Si l'installation et l'utilisation d'un logiciel requiert un CD, le fait d'utiliser l'image de ce CD permet de se passer de l'objet physique. On peut ainsi monter l'image comme si le CD était réellement inséré dans la machine.
- Le montage d'une image est bien plus rapide que de graver un CD (et bien sûr ne nécessite pas la consommation d'un CD vierge).
- C'est également une façon commode de sauvegarder les données présentes sur un CD sur le disque dur. De plus, cette image peut facilement être partagée.

Manipulation 11 : Sauvegarde arborescence

La commande *tar* permet de créer une archive à partir d'un ou plusieurs fichiers. Si le fichier est un répertoire, il sera archivé avec sa sous-arborescence. Voici les principales options :

- *c* / *-x* : Construit / extrait l'archive
- *v* : mode bavard
- *f* : utilise le fichier donné en paramètre
- *z* : compression avec Gunzip
- *j* : compression avec Bzip2

1. Archivage sans compression

○ Commande *tar*

Avec la commande *tar* et les options *-cf* nous créons l'archivage :

```
(root@Emilie)-[/home/emilie/Sauvegarde]
# tar cf sauvegarde.tar /home/emilie/sauvegarde
```

En utilisant les options *-tf* :

```
(root@Emilie)-[/home/emilie/Sauvegarde]
# tar tf sauvegarde.tar
home/emilie/Sauvegarde/
home/emilie/Sauvegarde/Fichier 3.txt
home/emilie/Sauvegarde/Fichier 2.txt
home/emilie/Sauvegarde/Fichier 1.txt
```

○ Commande *cpio*

Une autre commande permet également d'archiver, il s'agit de *cpio* avec les options *-ov* :

```
(root@Emilie)-[/home/emilie/sauvegarde]
# ls
Fichier1.txt  Fichier2.txt  Fichier3.txt

(root@Emilie)-[/home/emilie/sauvegarde]
# ls | cpio -ov > /home/emilie/sauvegarde/sauvegarde.cpio
Fichier1.txt
Fichier2.txt
Fichier3.txt
sauvegarde.cpio
1 bloc

(root@Emilie)-[/home/emilie/sauvegarde]
# ls
Fichier1.txt  Fichier2.txt  Fichier3.txt  sauvegarde.cpio
```

- Commande *pax*

Nous créons l'archivage avec la commande *pax* :

```
# pax -wf archive.pax cnam-utc502-os
```

Nous comparons la taille des fichiers :

```
user@LabUTC502:~/Desktop$ ls
010editor-desktop.desktop  cnam-utc502-os  pyCharm.desktop  repository
user@LabUTC502:~/Desktop$ pax -wf archive.pax cnam-utc502-os/
user@LabUTC502:~/Desktop$ ls -lh
total 147M
-rwxr-xr-x 1 user user 145 Mar 13 11:29 010editor-desktop.desktop
-rw-r--r-- 1 user user 147M May 2 18:58 archive.pax
drwxr-xr-x 7 user user 4.0K May 2 18:58 cnam-utc502-os
-rwxr-xr-x 1 user user 201 Aug 1 2022 pyCharm.desktop
drwxr-xr-x 2 user user 4.0K May 2 18:58 repository
```

2. Archivage avec compression

- Commande *tar*

Afin de faire un archivage compressé, à la commande *tar* nous ajoutons *z* qui est le gzip (ou *gunzip*) :

```
(root@Emilie)-[/home/emilie/Sauvegarde]
# tar cvzf sauvegardecompression.tar.gz /home/emilie/sauvegarde
```

Nous comparons la taille des 2 archivages :

```
(root@Emilie)-[/home/emilie/Sauvegarde]
# ls -lh
total 28K
-rw-r--r-- 1 emilie emilie 12 29 avril 09:52 'Fichier 1.txt'
-rw-r--r-- 1 emilie emilie 10 29 avril 09:52 'Fichier 2.txt'
-rw-r--r-- 1 emilie emilie 10 29 avril 09:52 'Fichier 3.txt'
-rw-r--r-- 1 root root 45 29 avril 10:09 sauvegardecompression.tar.gz
-rw-r--r-- 1 root root 10K 29 avril 09:57 sauvegarde.tar
```

Le fichier compressé est plus léger à 45 octets que le premier archivage qui est à 10Ko.

- Commande *cpio*

Création de l'archivage avec la commande *cpio* :

```
(root@Emilie)-[/home/emilie/sauvegarde]
# ls | cpio -o | gzip > sauvegarde.cpio.gz
2 blocs

(root@Emilie)-[/home/emilie/sauvegarde]
# ls
Fichier1.txt Fichier2.txt Fichier3.txt sauvegarde.cpio sauvegarde.cpio.gz
```

Comparaison des tailles des fichiers :

```
(root@Emilie)-[/home/emilie/Sauvegarde]
# ls -l
total 24
-rw-r--r-- 1 emilie emilie 12 29 avril 09:52 Fichier1.txt
-rw-r--r-- 1 emilie emilie 10 29 avril 09:52 Fichier2.txt
-rw-r--r-- 1 emilie emilie 10 29 avril 09:52 Fichier3.txt
drwxr-xr-x 3 root root 4096 29 avril 10:49 home
-rw-r--r-- 1 root root 512 2 mai 19:55 sauvegarde.cpio
-rw-r--r-- 1 root root 185 2 mai 19:56 sauvegarde.cpio.gz
```

- Commande *pax*

Création de l'archivage avec la commande *pax* :

```
# pax -wzf archive.pax cnam-utc502-os
```

L'option *-z* permet d'utiliser la compression Gunzip. L'option *-j* est également disponible pour utiliser la compression Bzip2.

Comparaison des tailles des fichiers :

```
user@LabUTC502:~/Desktop$ ls
010editor-desktop.desktop cnam-utc502-os pyCharm.desktop repository
user@LabUTC502:~/Desktop$ pax -wzf archive.pax cnam-utc502-os/
user@LabUTC502:~/Desktop$ ls -lh
total 86M
-rwxr-xr-x 1 user user 145 Mar 13 11:29 010editor-desktop.desktop
-rw-r--r-- 1 user user 86M May 2 19:00 archive.pax
drwxr-xr-x 7 user user 4.0K May 2 18:58 cnam-utc502-os
-rwxr-xr-x 1 user user 201 Aug 1 2022 pyCharm.desktop
drwxr-xr-x 2 user user 4.0K May 2 18:58 repository
```

3. Restauration de l'intégralité

Pour la restauration il faut que le dossier de destination existe, sinon il faut le créer avec la commande *mkdir* dans l'arborescence souhaité.

- Commande *tar*

Utilisation de l'option -C qui indique le répertoire d'extraction :

```
(root@Emilie)-[/home/emilie/Sauvegarde]
# tar xvf sauvegarde.tar -C /home/emilie/restauration
home/emilie/Sauvegarde/
home/emilie/Sauvegarde/Fichier 3.txt
home/emilie/Sauvegarde/Fichier 2.txt
home/emilie/Sauvegarde/Fichier 1.txt
```

En vérifiant l'extraction nous constatons ceci :

```
(root@Emilie)-[/home/emilie/restauration]
# ls
home
```

Lors de l'archivage, c'est tout l'arborescence qui a été renseigné, ce qui fait que l'affichage sera pareil, dans le dossier nous retrouvons le dossier emilie puis la suite.

- Commande *cpio*

Décompression du fichier avec *cpio* :

```
(root@Emilie)-[/home/emilie/restauration]
# cpio -iv < /home/emilie/sauvegarde/sauvegarde.cpio
Fichier1.txt
Fichier2.txt
Fichier3.txt
sauvegarde.cpio
1 bloc

(root@Emilie)-[/home/emilie/restauration]
# ls
Fichier1.txt Fichier2.txt Fichier3.txt sauvegarde.cpio
```

Contrairement à la commande *tar*, ce sont seulement les fichiers qui ont été archivés.

- Commande *pax*

```
# pax -rf archive.pax          #pour restaurer l'archive dans un autre répertoire
# pax -rzf archive.pax        #pour décompresser/restaurer l'archive au format Gunzip
```

```
user@LabUTC502:~/Desktop/repository$ ls
archive.pax
user@LabUTC502:~/Desktop/repository$ pax -rzf archive.pax
user@LabUTC502:~/Desktop/repository$ ls -lh
total 86M
-rw-r--r-- 1 user user 86M May  2 19:00 archive.pax
drwxr-xr-x 7 user user 4.0K May  2 18:58 cnam-utc502-os
```

4. Restauration d'un fichier

- Commande *tar*

Dans le but de restaurer un fichier spécifique, la commande est la suivant :

```
# tar -xvf fichier_compressé --wild nom_fichier -C dossier_destination
```

```
(root@Emilie)-[/home/emilie/Sauvegarde]
# tar -xvf sauvegarde.tar --wildcards 'Fichier 1.txt' -C /home/emilie/restauration
tar: Fichier 1.txt : non trouvé dans l'archive
tar: Arrêt avec code d'échec à cause des erreurs précédentes
```

Cependant nous avons une erreur lorsque nous voulons extraire un fichier. Nous avons aussi testé en nommant le fichier autrement, mais sans réussite.

Nous avons ensuite testé d'extraire tous les fichiers dont les extensions sont *.txt* :

```
(root@Emilie)-[/home/emilie/Sauvegarde]
# tar -xvf sauvegarde.tar --wildcards '*.txt' -C /home/emilie/restauration
home/emilie/Sauvegarde/Fichier 3.txt
home/emilie/Sauvegarde/Fichier 2.txt
home/emilie/Sauvegarde/Fichier 1.txt
```

- Commande *cpio*

Nous constatons le même souci avec la commande *cpio* pour extraire seulement un fichier :

```
(root@Emilie)-[/home/emilie/sauvegarde]
# cpio -E Fichier1.txt < /home/emilie/sauvegarde/sauvegarde.cpio

cpio: You must specify one of -oipt options.
Essayez « cpio --help » ou « cpio --usage » pour plus d'informations.

(root@Emilie)-[/home/emilie/sauvegarde]
# cpio -iv -E 'Fichier1.txt' < /home/emilie/sauvegarde/sauvegarde.cpio

cpio: You must specify one of -oipt options.
Essayez « cpio --help » ou « cpio --usage » pour plus d'informations.
```

- Commande *pax*

Nous n'avons pas trouvé de commande adéquate pour *pax*.

Synthèse

L'archivage est une option intéressante pour créer des sauvegardes d'arborescences de fichiers. Les archives peuvent être compressées ou non pour gagner de la place. Bien évidemment la compression augmente le temps de création et de restauration de l'archive. Une fois l'archive créée, on se retrouve avec un seul fichier qui peut aisément être déplacé puis utilisé pour retrouver l'arborescence ainsi sauvegardée. Un autre avantage est de pouvoir extraire seulement un fichier ou un groupe de fichier spécifique (via son nom ou le type) de l'archivage.

La commande *cpio* permet de faire des archives. Cependant elle n'est pas récursive, récupère les noms de fichiers sur l'entrée standard et les sauvegarde sur la sortie standard. Il est donc nécessaire d'utiliser la commande *find* et faire une redirection dans un fichier. Contrairement à *tar* avec l'option *-p*, la commande *cpio* conserve les droits et propriétaires des fichiers.

On va donc plutôt utiliser *tar* quand on veut archiver des données qui pourront être utilisables par tout le monde. Ce qui est le cas la plupart du temps. Ceci est utilisé notamment par les développeurs quand ils donnent leurs codes source, ces archives sont souvent compressées et portent l'extension *.tgz*. L'utilisation de *Cpio* servira plutôt pour faire des sauvegardes de son système, pour pouvoir restaurer des données à l'identique.

En outre, la commande *tar* est un archivage dit classique pour une utilisation simplifiée des données, quant à *cpio* c'est une commande plus avancée proposant des options plus complètes. Il est très important de spécifier l'extension, par exemple : *.tar.gz* ou *.tar.bz2* ainsi que d'autres extension pour le type de compression et surtout utiliser les bonnes commandes pour l'extraction.

Manipulation 12 : Sauvegarde incrémentale

Pour cette manipulation, nous avons testé 2 commandes, avec *tar* et *pax*

- Partie avec *tar*

1. Sauvegarde intégrale

Création des fichiers avec la commande *touch* :

```
# touch fichier1.txt fichier2.txt
```

```
(root@Emilie)-[/home/emilie/sauvegardes]
# ls
fichier1.txt  fichier2.txt
```

Sauvegarde intégrale avec *tar* et les options *-cvzf*:

```
(root@Emilie)-[~]
# tar -cvzf sauvegardeintegrale.tar /home/emilie/sauvegardes
tar: Suppression de « / » au début des noms des membres
/home/emilie/sauvegardes/
/home/emilie/sauvegardes/fichier2.txt
/home/emilie/sauvegardes/fichier3.txt
/home/emilie/sauvegardes/fichier1.txt
```

```
(root@Emilie)-[~]
# ls -l
total 4
-rw-r--r-- 1 root root 241  2 mai  21:05 sauvegardeintegrale.tar
```

Nous regardons le contenu de la sauvegarde avec l'option *-tf*:

```
(root@Emilie)-[/home/emilie/sauvegardes]
# tar -tf sauvegardeintegrale.tar
home/emilie/sauvegardes/
home/emilie/sauvegardes/fichier2.txt
home/emilie/sauvegardes/sauvegardeintegrale.tar
home/emilie/sauvegardes/fichier1.txt
```

2. Créations, modifications et suppressions de fichiers

Création des nouveaux fichiers :

```
(root@Emilie)-[/home/emilie/sauvegards]  
# touch fichier3.txt fichier4.txt
```

Modification du fichier 3 :

```
(root@Emilie)-[/home/emilie/sauvegards]  
# vi fichier3.txt
```

Suppression du fichier 3 :

```
(root@Emilie)-[/home/emilie/sauvegards]  
# rm fichier3.txt
```

3. Sauvegarde incrémentale et vérification du contenu de la sauvegarde

Création d'un nouveau fichier pour qu'il soit sauvegardé à son tour :

```
(root@Emilie)-[/home/emilie/sauvegards]  
# touch fichier5.txt
```

Création d'un fichier caché qui va servir à comparer les anciens fichiers et les nouveaux. Dans le cas présent ce sera ceux créés avant le 02 Mai à 20h30. On appellera le fichier « comparaison ».

```
(root@Emilie)-[~]  
# touch -t 05022030 .comparaison
```

Création de la sauvegarde incrémentale :

```
(root@Emilie)-[~]  
# find ~ -newer .comparaison | tar -c -T - -f /home/emilie/sauvegards/sauvegardeincrementale.tar
```

Nous regardons le contenu de la sauvegarde :

```
(root@Emilie)-[/home/emilie/sauvegards]
# tar -tf sauvegardeincrementale.tar
root/
root/.face.icon
root/.bashrc.original
root/.bashrc
root/.viminfo
root/.cache/
root/.cache/zcompdump
root/.profile
root/sauvegardeintegrale.tar
root/.ssh/
root/.face
root/.zshrc
root/.zsh_history
root/.viminfo
```

4. Restauration intégrale de l'archive dans un répertoire différent

Restauration intégrale avec la commande suivante :

```
tar -xvf /home/emilie/sauvegarde/sauvegardeincrementale.tar /home/emilie/restauration
```

- Partie avec *pax*

1. Sauvegarde intégrale

Comme lors de l'utilisation de la commande précédente, nous créons des fichiers pour faire la manipulation. Pour faire l'archivage nous utilisons la commande *pax* et les options *-wf* :

```
user@LabUTC502:~/Desktop/arbo$ echo "Ceci est mon premier fichier" >> file1.txt
user@LabUTC502:~/Desktop/arbo$ echo "Ceci est mon second fichier" >> file2.txt
user@LabUTC502:~/Desktop/arbo$ echo "Ceci est mon troisième fichier" >> file3.txt
user@LabUTC502:~/Desktop/arbo$ ls
file1.txt file2.txt file3.txt
user@LabUTC502:~/Desktop/arbo$ cat file3.txt
Ceci est mon troisième fichier
user@LabUTC502:~/Desktop/arbo$ cd ..
user@LabUTC502:~/Desktop$ ls
010editor-desktop.desktop arbo pyCharm.desktop
user@LabUTC502:~/Desktop$ pax -wf archive.pax arbo/
user@LabUTC502:~/Desktop$ ls
010editor-desktop.desktop arbo archive.pax pyCharm.desktop
```

2. Création, modification, suppression de fichier et sauvegarde incrémentielle

Nous créons, modifions des fichiers et sauvegardons de façon partielle. Avec *pax*, c'est l'option *-u* qui permet de faire une sauvegarde incrémentielle :

```
user@LabUTC502:~/Desktop/arbo$ rm file3.txt
user@LabUTC502:~/Desktop/arbo$ echo "Ceci est mon tout premier fichier" > file1.txt
user@LabUTC502:~/Desktop/arbo$ echo "Ceci est mon quatrième fichier" >> file4.txt
user@LabUTC502:~/Desktop/arbo$ ls
file1.txt file2.txt file4.txt
user@LabUTC502:~/Desktop/arbo$ cat file1.txt
Ceci est mon tout premier fichier
user@LabUTC502:~/Desktop/arbo$ cd ..
user@LabUTC502:~/Desktop$ pax -wuf archive2.pax arbo/
user@LabUTC502:~/Desktop$ cd arbo/
user@LabUTC502:~/Desktop/arbo$ rm file4.txt
user@LabUTC502:~/Desktop/arbo$ echo "Ceci est mon deuxième fichier" > file2.txt
user@LabUTC502:~/Desktop/arbo$ echo "Ceci est mon cinquième fichier" >> file5.txt
user@LabUTC502:~/Desktop/arbo$ ls
file1.txt file2.txt file5.txt
user@LabUTC502:~/Desktop/arbo$ cat file2.txt
Ceci est mon deuxième fichier
user@LabUTC502:~/Desktop/arbo$ cd ..
user@LabUTC502:~/Desktop$ pax -wuf archive3.pax arbo/
user@LabUTC502:~/Desktop$ ls
010editor-desktop.desktop arbo archive2.pax archive3.pax archive.pax pyCharm.desktop
```

3. Restauration intégrale

Nous restaurons complètement des fichiers dans un autre dossier. Les différentes modifications apparaissent bien au fil des restaurations successives :

```
user@LabUTC502:~/Desktop/restauration$ ls
archive2.pax archive3.pax archive.pax
user@LabUTC502:~/Desktop/restauration$ pax -rf archive.pax
user@LabUTC502:~/Desktop/restauration$ ls arbo/
file1.txt file2.txt file3.txt
user@LabUTC502:~/Desktop/restauration$ cat arbo/file1.txt
Ceci est mon premier fichier
user@LabUTC502:~/Desktop/restauration$ cat arbo/file2.txt
Ceci est mon second fichier
user@LabUTC502:~/Desktop/restauration$ pax -rf archive2.pax
user@LabUTC502:~/Desktop/restauration$ ls arbo/
file1.txt file2.txt file3.txt file4.txt
user@LabUTC502:~/Desktop/restauration$ cat arbo/file1.txt
Ceci est mon tout premier fichier
user@LabUTC502:~/Desktop/restauration$ pax -rf archive3.pax
user@LabUTC502:~/Desktop/restauration$ ls arbo/
file1.txt file2.txt file3.txt file4.txt file5.txt
user@LabUTC502:~/Desktop/restauration$ cat arbo/file1.txt
Ceci est mon tout premier fichier
user@LabUTC502:~/Desktop/restauration$ cat arbo/file2.txt
Ceci est mon deuxième fichier
```

Synthèse

La question de la sauvegarde des données est une question sensible, d'autant plus qu'à terme cela prend beaucoup de place. Ce que l'on voudrait idéalement c'est de pouvoir à chaque fois sauvegarder seulement les données qui ont changé. C'est justement ce que propose de faire la sauvegarde incrémentale.

Grâce à la sauvegarde incrémentale on peut gagner énormément de place tout en faisant des sauvegardes très régulières. Le volume de données ne s'accroît qu'en cas de nouveautés, indépendamment de la régularité.

Cette stratégie nécessite une organisation rigoureuse ! Une sauvegarde complète est la somme de la sauvegarde initiale et de toutes les sauvegardes partielles, il est nécessaire de n'en perdre aucune sous peine de corrompre ses données.

Manipulation 13 : Intégrité d'un FS

Pour commencer, nous vérifions du partitionnement des disques pour démontrer la bonne partition :

```
(root@kali)-[/media/kali/manip13]
# lsblk -fe7
NAME        FSTYPE FSVER LABEL        UUID                                   FSAVAIL FSUSE% MOUNTPOINTS
sda
├─sda1 ext4   1.0   root         f02c792a-f423-4d70-b817-1bdc006babf2 59.8G   18% /
sdb
├─sdb1 LVM2_m LVM2 0          gyxCEU-LEZM-hwOP-nQ8o-ELIJ-Lx0B-IOGg3D
├─sdb2 LVM2_m LVM2 0          20WXcc-53Vc-dtcD-JDjR-mtwo-IZsX-hgp6M1
├─VG1-volume1
│   └─ext4 1.0          a1c0c444-f18b-4d02-a2dd-e3adc5f14fb5 762.3M   0% /utilisateur1
└─sdb3 ext3   1.0   manip13      6c7fe8d5-1a07-433a-b754-6379fda39a8b 230.4M   49% /media/kali/manip13
sr0      iso966 Joliet VBox_GAs_7.0.0 2022-10-06-17-50-34-86 0        100% /media/kali/VBox_GAs_7.0.0
```

Nous procédons au démontage de la partition avec la commande *umount* :

```
# umount -l /dev/sdb3
```

Contrôle de la partition à l'aide de la commande *fsck* :

```
# fsck /dev/sdb3
fsck from util-linux 2.38.1
e2fsck 1.46.6-rc1 (12-Sep-2022)
manip13: clean, 63/139656 files, 299124/563200 blocks
```

Contrôle de la partition avec plus de détails :

```
(root@kali)-[/media/kali]
# fsck -fv -C0 /dev/sdb3
fsck from util-linux 2.38.1
e2fsck 1.46.6-rc1 (12-Sep-2022)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information

    63 inodes used (0.05%, out of 139656)
    23 non-contiguous files (36.5%)
    0 non-contiguous directories (0.0%)
    # of inodes with ind/dind/tind blocks: 27/14/1
299124 blocks used (53.11%, out of 563200)
    0 bad blocks
    0 large files

    45 regular files
    9 directories
    0 character device files
    0 block device files
    0 fifos
    0 links
    0 symbolic links (0 fast symbolic links)
    0 sockets

    54 files
```

Utilisation de la commande *debugfs* avec l'option *-R stats* :

```
# debugfs -R stats /dev/sdb3
Filesystem volume name:   manip13
Last mounted on:         /media/kali/manip13
Filesystem UUID:         6c7fe8d5-1a07-433a-b754-6379fda39a8b
Filesystem magic number: 0xEF53
Filesystem revision #:   1 (dynamic)
Filesystem features:     has_journal ext_attr resize_inode dir_index filetype sparse_super large_file
Filesystem flags:        signed_directory_hash
Default mount options:   user_xattr acl
Filesystem state:        clean
Errors behavior:         Continue
Filesystem OS type:      Linux
Inode count:             139656
Block count:             563200
Reserved block count:    28160
Overhead clusters:      41471
Free blocks:            264076
Free inodes:            139593
First block:            1
Block size:             1024
Fragment size:          1024
Reserved GDT blocks:     254
Blocks per group:        8192
Fragments per group:     8192
Inodes per group:        2024
Inode blocks per group:  506
Filesystem created:      Fri Apr 28 19:31:48 2023
Last mount time:         Fri Apr 28 19:48:25 2023
Last write time:         Sun Apr 30 14:34:10 2023
Mount count:             0
Maximum mount count:     -1
Last checked:            Sun Apr 30 14:34:10 2023
Check interval:          0 (<none>)
Lifetime writes:         253 MB
Reserved blocks uid:     0 (user root)
Reserved blocks gid:     0 (group root)
First inode:             11
Inode size:             256
Required extra isize:    32
Desired extra isize:     32
Journal inode:           8
Default directory hash:  half_md4
Directory Hash Seed:     191156b3-85b8-4905-aeaf-ec4e5af564ab
Journal backup:          inode blocks
Directories:             9
 Group 0: block bitmap at 259, inode bitmap at 260, inode table at 261
          2291 free blocks, 2012 free inodes, 2 used directories
 Group 1: block bitmap at 8451, inode bitmap at 8452, inode table at 8453
          2306 free blocks, 2024 free inodes, 0 used directories
 Group 2: block bitmap at 16385, inode bitmap at 16386, inode table at 16387
          1540 free blocks, 2024 free inodes, 0 used directories
```

La commande *e2fsck* nous sert à vérifier les FS qui sont au format *ext* :

```
(root@kali)-[/media/kali]
# e2fsck -b 259 /dev/sdb3
e2fsck 1.46.6-rc1 (12-Sep-2022)
e2fsck: Bad magic number in super-block while trying to open /dev/sdb3

The superblock could not be read or does not describe a valid ext2/ext3/ext4
filesystem. If the device is valid and it really contains an ext2/ext3/ext4
filesystem (and not swap or ufs or something else), then the superblock
is corrupt, and you might try running e2fsck with an alternate superblock:
    e2fsck -b 8193 <device>
or
    e2fsck -b 32768 <device>

/dev/sdb3 contains a ext3 file system labelled 'manip13'
    last mounted on /media/kali/manip13 on Fri Apr 28 19:48:25 2023
```


A l'aide de la commande *tune2fs*, nous vérifions la ligne « maximum mount count » qui correspond au nombre de montage :

```
└─# tune2fs -l /dev/sda1
tune2fs 1.46.6-rc1 (12-Sep-2022)
Filesystem volume name:   root
Last mounted on:         /
Filesystem UUID:          f02c792a-f423-4d70-b817-1bdc006babf2
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index filetype needs_recovery extent 64bit flex_bg s
parse_super large_file huge_file dir_nlink extra_isize metadata_csum
Filesystem flags:         signed_directory_hash
Default mount options:    user_xattr acl
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              5251072
Block count:              20995837
Reserved block count:    1049791
Overhead clusters:       475050
Free blocks:              16742164
Free inodes:              4782700
First block:              0
Block size:               4096
Fragment size:            4096
Group descriptor size:    64
Reserved GDT blocks:     1024
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         8192
Inode blocks per group:   512
Flex block group size:    16
Filesystem created:       Mon Dec  5 14:52:11 2022
Last mount time:          Wed Apr 26 23:05:40 2023
Last write time:          Wed Apr 26 23:05:25 2023
Mount count:              10
Maximum mount count:      -1
```

Nous remarquons que le compteur maximum de « mount » est à -1. Nous allons modifier cette valeur à 1 avec la commande *tune2fs* :

```
# tune2fs -c 1 -i 0d /dev/sda1
tune2fs 1.46.6-rc1 (12-Sep-2022)
Setting maximal mount count to 1
Setting interval between checks to 0 seconds
```

Nous vérifions que la valeur est passé à 1 en utilisant la même commande *tune2fs -l* :

```
└─(root@kali)-[/home/kali]
└─# tune2fs -l /dev/sda1 | egrep "count|Last"
Last mounted on:         /
Inode count:              5251072
Block count:              20995837
Reserved block count:    1049791
Last mount time:         Sun Apr 30 15:39:11 2023
Last write time:         Sun Apr 30 15:38:43 2023
Mount count:             1
Maximum mount count:     1
Last checked:            Sun Apr 30 15:38:43 2023
```


Synthèse

La commande *fsck* permet de vérifier l'intégrité d'un système de fichiers. La conformité des données est bien sûr étroitement liée à la validité du support qui les stocke. Ce support pouvant être amené à subir de nombreuses opérations diverses susceptibles de faire apparaître des dysfonctionnements, il est important d'être en mesure de procéder à une telle vérification, de façon périodique si possible. Selon les besoins cette vérification peut même être faite à chaque montage du système de fichiers.

Manipulation 14 : Debug d'un FS

Pour repérer le numéro d'inode d'un fichier, il faut utiliser la commande *ls* avec l'option *-i* :

```
$ ls -i a_modif.txt
3541484 a_modif.txt
```

Ayant récupéré le numéro d'inode, nous pouvons utiliser la commande *debugfs* en utilisant l'option *-w* pour être en écriture afin de pouvoir modifier des blocs, sur la partition dans lequel se trouve le fichier. Pour nous ce sera sur */dev/sdb3* :

```
# debugfs -w /dev/sdb3/
```

Afin de modifier les blocs, nous rentrerons la commande *mi* qui signifie « Modify Inode ». En rouge ce que nous avons renseigné :

```
Debugfs: mi<14>
```

```
Mode [0100644]
User ID [1000]
Group ID [1000]
Size [25]
Creation time [1683231228]
Modification time [1683231228]
Access time [1683231228]
Deletion time [0] 56465
Link count [1] 0
Block count high [0] 54
Block count [2] 654
File flags [0x0] 654
Generation [0x2290d511]
File acl [0]
High 32bits of size [0]
Fragment address [0]
Direct Block #0 [6145]
Direct Block #1 [0]
Direct Block #2 [0]
Direct Block #3 [0]
Direct Block #4 [0]
Direct Block #5 [0]
Direct Block #6 [0]
Direct Block #7 [0]
Direct Block #8 [0]
Direct Block #9 [0]
Direct Block #10 [0]
Direct Block #11 [0]
Indirect Block [0]
Double Indirect Block [0]
Triple Indirect Block [0]
```

Pour vérifier l'état du fichier, nous démontons la partition puis nous lançons une vérification avec la commande *fsck* :

```
# umount -l /dev/sdb3
```

```
(root@kali)-[/home/kali]
# fsck -fv -C0 /dev/sdb3
fsck from util-linux 2.38.1
e2fsck 1.46.6-rc1 (12-Sep-2022)
Pass 1: Checking inodes, blocks, and sizes
Inodes that were part of a corrupted orphan linked list found. Fix<y>? yes
Inode 14 was part of the orphaned inode list. FIXED.

Pass 2: Checking directory structure
Entry 'a_modif.txt' in / (2) has deleted/unused inode 14. Clear<y>? yes

Pass 3: Checking directory connectivity
Pass 4: Checking reference counts

Pass 5: Checking group summary information
Block bitmap differences: -6145
Fix<y>? yes

Free blocks count wrong for group #0 (2290, counted=2291).
Fix<y>? yes
Free blocks count wrong (264075, counted=264076).
Fix<y>? yes
Inode bitmap differences: -14
Fix<y>? yes

Free inodes count wrong for group #0 (2011, counted=2012).
Fix<y>? yes
Free inodes count wrong (139592, counted=139593).
Fix<y>? yes

manip13: ***** FILE SYSTEM WAS MODIFIED *****

    63 inodes used (0.05%, out of 139656)
    23 non-contiguous files (36.5%)
    0 non-contiguous directories (0.0%)
    # of inodes with ind/dind/tind blocks: 27/14/1
  299124 blocks used (53.11%, out of 563200)
    0 bad blocks
    0 large files

    45 regular files
    9 directories
    0 character device files
    0 block device files
    0 fifos
    0 links
    0 symbolic links (0 fast symbolic links)
    0 sockets

    54 files
```

Nous constatons que l'inode 14 est en état "corrupted". Après correction, nous avons monté le FS, ce qui nous a permis de constater que le fichier « a_modif.txt » n'apparaît plus avec la commande *ls* :

```
# ls /media/kali/manip13
a_archiver archive_non_compressé.cpio dossier_desarchive lost+found test
```

Il est tout de même possible d'accéder directement à l'inode pour le modifier comme à l'étape au-dessus :

```

debugfs: debugfs: mi <14>
Mode [0100644]
User ID [1000]
Group ID [1000]
Size [24]
Creation time [1683231228]
Modification time [1683231228]
Access time [1683231228]
Deletion time [0]
Link count [1]
Block count high [0]
Block count [45]
File flags [0xffb0]
Generation [0x4845aca9]
File acl [0]
High 32bits of size [0]
Fragment address [0]
Direct Block #0 [6145] 46
Direct Block #1 [46] 654
Direct Block #2 [456]
Direct Block #3 [0]
Direct Block #4 [654546]
Direct Block #5 [654]
Direct Block #6 [65465464]
Direct Block #7 [54654]
Direct Block #8 [6546546]
Direct Block #9 [55]
Direct Block #10 [5]
Direct Block #11 [0]
Indirect Block [0]
Double Indirect Block [0]
Triple Indirect Block [0]

```

Après avoir monté le FS, il est possible d'accéder au fichier après avoir effectué une vérification du FS. Le fichier est de-nouveau présent, mais reste illisible :

```

(root@kali)-[/home/kali]
# e2fsck -f /dev/sdb3
e2fsck 1.46.6-rc1 (12-Sep-2022)
Pass 1: Checking inodes, blocks, and sizes
Inode 14 has imagic flag set. Clear<y>? yes
Inode 14 has encrypt flag but no encryption extended attribute.
Clear flag<y>? yes
Inode 14 has illegal block(s). Clear<y>? yes
Illegal block #4 (654546) in inode 14. CLEARED.
Illegal block #6 (65465464) in inode 14. CLEARED.
Illegal block #8 (6546546) in inode 14. CLEARED.
Inode 14 has INDEX_FL flag set but is not a directory.
Clear HTree index<y>? yes
Inode 14, i_size is 24, should be 11264. Fix<y>? yes
Inode 14, i_blocks is 45, should be 14. Fix<y>? yes

Running additional passes to resolve blocks claimed by more than one inode...
Pass 1B: Rescanning for multiply-claimed blocks
Multiply-claimed block(s) in inode 7: 5 46 55
Multiply-claimed block(s) in inode 14: 46 654 456 654 55 5
Pass 1C: Scanning directories for inodes with multiply-claimed blocks
Pass 1D: Reconciling multiply-claimed blocks
(There are 1 inodes containing multiply-claimed blocks.)

File /a_modif.txt (inode #14, mod time Thu May 4 22:13:48 2023)
has 6 multiply-claimed block(s), shared with 2 file(s):
<filesystem metadata>
<The group descriptor inode> (inode #7, mod time Fri Apr 28 19:37:53 2023)
Clone multiply-claimed blocks<y>? yes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
Block bitmap differences: +54654
Fix<y>? yes
Free blocks count wrong for group #0 (2291, counted=2285).
Fix<y>? yes
Free blocks count wrong for group #6 (3114, counted=3113).
Fix ('a' enables 'yes' to all) <y>? yes
Free blocks count wrong (264076, counted=264069).
Fix ('a' enables 'yes' to all) <y>? yes
Inode bitmap differences: +14
Fix ('a' enables 'yes' to all) <y>? yes
Free inodes count wrong for group #0 (2012, counted=2011).
Fix ('a' enables 'yes' to all) <y>? yes
Free inodes count wrong (139593, counted=139592).
Fix<y>? yes

manip13: ***** FILE SYSTEM WAS MODIFIED *****
manip13: 64/139656 files (37.5% non-contiguous), 299131/563200 blocks

```

Synthèse

La commande *debugfs* permet d'interagir avec un système de fichiers endommagé ou corrompu. Le système de fichiers peut subir des dommages qui modifient les données. Dans ce cas on a souvent besoin de retrouver l'état initial de ces données, avant modification. Bien que ce ne soit pas toujours garanti, la meilleure façon de pouvoir récupérer cet état est de faire le point sur les blocs endommagés et de les modifier.

En poussant un peu, il serait possible de récupérer des données.

Manipulation 15 :

1. Formatage au format ext2

La commande pour formater est *mkfs*.

Pour réaliser la commande correctement, il faut que la clé soit démontée, de ce fait, faire cela sur son arborescence de base en */dev/sdb*, ce qui donne la commande suivante :

```
# mkfs.ext2 /dev/sdb
mke2fs 1.46.6 (1-Feb-2023)
/dev/sdb contains a ntfs file system labelled 'data4'
Proceed anyway? (y,N) y
Creating filesystem with 985097 4k blocks and 246512 inodes
Filesystem UUID: a914eea0-2c2e-4431-baf6-c5c75ffecb75
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

Vérification de la clé en faisant la commande *df* afin d'afficher que la partition voulue :

```
# df -Th | grep sdb
/dev/sdb      ext2      3,7G      24K      3,6G      1% /media/user/usb
```

Une autre commande est aussi possible pour vérifier avec *file* :

```
# file -s /dev/sdb
/dev/sdb: Linux rev 1.0 ext2 filesystem data, UUID=54489001-207a-4bd3-9dfd-3504068a2efb (large files)
```

La clé USB est passé au format ext2

2. Passage au format ext3

Passage du format de la clé de ext2 à ext3 sans formater avec la commande *tune2fs* :

```
# tune2fs -j /dev/sdb
tune2fs 1.46.6 (1-Feb-2023)
Creating journal inode: 2048
```

Une fois fini, nous vérifions avec la commande précédente :

```
# file -s /dev/sdb
/dev/sdb: Linux rev 1.0 ext3 filesystem data, UUID=54489001-207a-4bd3-9dfd-3504068a2efb (large files)
```

La clé est passé en ext3 sans être formater.

Pour repasser au format ext2, nous utilisons la même commande `tune2fs` avec l'option `-O` :

```
# tune2fs -O ^has_journal /dev/sdb
tune2fs 1.46.6 (1-Feb-2023)

(root@KaliVM)-[/home/user]
# file -s /dev/sdb
/dev/sdb: Linux rev 1.0 ext2 filesystem data, UUID=54489001-207a-4bd3-9dfd-3504068a2efb (large files)
```

Nous remarquons que l'UUID de la clé reste identique. La différence que l'on remarque est la demande de création d'inode pour le passage en ext3. Pour revenir en ext2 il n'y a pas de message.

Synthèse

L'avantage avec la commande `tune2fs` est de pouvoir changer le format ou le FS d'un périphérique sans formater celle-ci. Cela peut servir lors d'un passage d'un FS plus ancien vers un plus récent sans pouvoir créer de sauvegarde au préalable.

La différence majeure entre les deux formats ext2 et ext3 se trouve dans le fait que ext3 supporte la journalisation qui permet de garder un historique de modifications. Pour permettre la journalisation, ext3 nécessite différentes informations sur les fichiers (propriétaire, permissions, date de modifications...) Lorsque l'on convertit une partition ext2 en ext3 on a donc besoin d'inodes pour tenir compte de ses informations ce qui justifie cette création d'inode dans le sens ext2 → ext3 mais pas dans le sens ext3 → ext2.

Synthèse générale

La mémoire d'un ordinateur peut contenir de nombreuses données, qui sont stockées sous la forme de fichiers. Très rapidement ces fichiers deviennent présents en très grand nombre, il est donc indispensable de les organiser et de les stocker d'une façon qui permet d'en simplifier le stockage, la modification, la suppression ainsi que la consultation. C'est là le rôle des systèmes de fichiers (abrégé FS pour File System) qui proposent une structure hiérarchique également nommée arborescence.

Il existe différents types de FS. Selon les différents systèmes d'exploitation déjà, (Linux : Extended File System, Windows : New Technology File System, MacOS : Apple File System), mais aussi au sein d'un même type de système d'exploitation (pour Linux : ext2, ext3, ext4, ReiserFS...). Chacun de ses FS présente ses particularités, qui sont autant de forces ou/et de faiblesses. Le choix d'un système de fichier détermine les conventions de nommage des fichiers (le nombre maximal de caractères, les caractères autorisés, parfois la taille du suffixe de nom de fichier) mais également le chemin d'accès à un fichier en définissant la structure des répertoires qui vont contenir ces fichiers. Un FS peut ou non permettre la journalisation et dispose de métadonnées qui caractérisent plus facilement les fichiers à gérer.

Les données sont stockées physiquement sur des volumes qui peuvent être de différentes nature : disque dur, SSD, clé USB, partitions de disque dur, etc. Pour simplifier l'utilisation de l'espace, il faut que tous ces volumes apparaissent à l'utilisateur comme faisant partie de la même arborescence. De la même façon, plusieurs FS peuvent être utilisés sur le même système (ext3 et ext4 par exemple), mais pas sur le même volume. On peut donc utiliser tel ou tel FS sur tel ou tel volume et les monter ensemble pour fournir un ensemble de mémoire cohérent et fonctionnel. Les volumes sont montés comme des périphériques, il est donc possible d'en ajouter, d'en supprimer, de les renommer ou les monter à la volée pour certains d'entre eux. Ils peuvent être manipulés manuellement ou automatiquement selon les besoins.

Bien qu'un volume comportant un FS se présente souvent à l'utilisateur comme un simple répertoire dans l'arborescence générale, la différence entre répertoire et FS est très significative. On ne peut démonter ou ajouter qu'un volume à part entière. Il est donc nécessaire de prendre connaissance des points de montage composant la structure pour la comprendre avant de la manipuler. A cet égard, la commande *mount* permet de très nombreuses opérations sur les volumes. Elle permet entre-autres de :

- Spécifier un FS pour un volume donné
- Monter/démonter un volume, que ce soit en lecture seule, écriture seule ou lecture/écriture
- Déplacer une hiérarchie de fichier
- Déplacer un point de montage
- Interdire l'exécution de fichiers binaires
- Monter automatiquement des FS
- Réserver l'accès d'un FS à un utilisateur ou un groupe donné
- ... et bien plus encore

Grâce à l'UUID du périphérique concerné, on peut aussi n'autoriser que certains de ces périphériques à être montés avec pour principal objectif d'empêcher la copie ou l'injection non autorisée de données.

Nous avons vu lors du montage automatique dans le fichier *fstab* qu'il est possible de spécifier l'UUID du périphérique à monter pour des raisons de performance et de sécurité. Il est tout à fait possible de bloquer le montage automatique, ou plutôt d'interdire tout nouveau périphérique à la lecture (ce qui n'implique pas d'écriture ni exécution). Cette mise en place permet d'éviter de faire des copies de document d'une entreprise ou d'injecter des fichiers corrompus.

Bien évidemment, une information capitale concernant les volumes est le nombre de blocs mémoire disponibles, ainsi que le pourcentage du volume utilisé. Un volume plein ne permet plus aucune opération, il est donc possible de le saturer et de perdre accès à toutes les informations qui y sont contenues. Pour remédier à cette problématique, habituellement 5% de l'espace de chaque FS est réservé à l'utilisateur root. Ainsi il est toujours possible pour les administrateurs d'y avoir accès et d'effectuer des actions telles que la suppression de données, le débogage ou encore le contrôle d'intégrité.

L'arborescence générale d'un système est donc composée de différents volumes présents dans le matériel de l'ordinateur, mais peut aussi être composée de systèmes de fichiers en réseau. C'est le cas par exemple du système NFS (pour Network File System). Un serveur NFS peut exporter tout ou une partie de son système de fichiers et le mettre à disposition à d'autres machines. Toute la gestion des droits doit, dans ce cas, être également exportée. Habituellement les ressources NFS sont montées au démarrage grâce au fichier */etc/fstab*. Ce type de système favorise grandement le partage de données et le travail collaboratif. Le système HDFS fonctionne sur le même principe mais permet en plus la réplication de données pour sécuriser la pérennité des informations.

De la même façon que l'on est limité par la quantité et le type de mémoire que possède l'ordinateur pour réaliser certaines actions (RAM ou ROM), on est limité par la taille et le type de volume permettant de stocker les données. Comme pour la mémoire qui peut être gérée sous forme de mémoire virtuelle, il est possible grâce à LVM de bénéficier du concept de "volumes virtuels". LVM est à la fois une méthode et un logiciel de gestion d'espaces de stockage. Les différents volumes tels que les disques durs, partitions sont autant de volumes physiques (on peut aussi considérer les volumes RAID). Ces volumes physiques peuvent être regroupés sous forme de groupes de volumes. Ces groupes de volumes sont alors découpés en volumes logiques, sorte de partitions virtuelles, beaucoup plus flexibles que leurs volumes physiques. Comme des volumes classiques, ces derniers peuvent être montés, démontés, cependant les volumes logiques sont facilement redimensionnables.

Un volume RAID représente un autre ensemble de techniques permettant d'obtenir des "volumes virtuels". Ce concept est plus particulièrement dédié à la réplication de données pour favoriser les performances, la sécurité et la tolérance aux pannes. Toutes nouvelles informations stockées sur un volume RAID est en réalité copiée sur plusieurs volumes synchronisés entre eux. Bien souvent ces volumes se trouvent sur des machines différentes, machines situées à des emplacements géographiques différents afin qu'en cas de force majeure (incendie par exemple) une copie intègre des données soit toujours disponible. Évidemment ce système présuppose la disponibilité d'un réseau pour synchroniser les différentes instances composant le volume en question.

La conservation des données est une problématique essentielle et le concept de sauvegarde est familier à tout le monde, de la multinationale tenant à ne pas perdre les données alimentant ses algorithmes de big-data, qu'à l'utilisateur lambda souhaitant conserver ses photos personnelles. Pour réaliser des sauvegardes efficaces, étant le moins coûteuses possible, il est important de bien réfléchir à son plan de sauvegarde. Les considérations suivantes doivent être prises en compte :

- Données importantes et données secondaires
- Fréquence de sauvegarde
- Support de sauvegarde
- Solution graphique/ligne de commande
- Type de restauration
- Type de sauvegarde (complète/différentielle/incrémentale)

Une sauvegarde est une opération coûteuse en termes de temps et de stockage mémoire. Une sauvegarde complète est plus facile à organiser et à restaurer mais est particulièrement gourmande en espace disque et longue à effectuer. A contrario, une sauvegarde incrémentale va être plus rapide et moins consommatrice d'espace puisse qu'on ne sauvegarde que ce qui est nécessaire, mais à chaque sauvegarde correspond un nouveau fichier. Celui-ci sera nécessaire de le restaurer avec les autres fichiers qui lui sont liés, exigeant ainsi une organisation importante pour être capable de mener à bien la restauration complète.

Certains supports de données tel que les CDs peuvent être numérisés et utilisés en tant que tel. Le fait de créer une image d'un disque (qu'il soit optique ou magnétique) permet de s'affranchir du support physique tout en pouvant quand même en consulter le contenu. Les bénéfices sont évidents en termes de stockage, mais aussi de partage de données.

Un volume de données est exposé à de nombreux événements au cours de son utilisation. Il subit de très nombreuses opérations de lectures et d'écritures, il s'use. Il est parfois également exposé à des conditions extérieures qui peuvent contribuer à le dégrader (incident, surchauffe, chocs lors de déplacement...), ainsi qu'à des événements logiciels (bugs, interruption d'un appel système...). Si l'intégrité physique du volume est mise en cause, bien évidemment l'intégrité des données qu'il contient est menacée. Il existe naturellement des outils permettant de contrôler l'intégralité des blocs mémoires contenus dans un volume, ainsi que d'autres outils permettant de résoudre les éventuels bugs et corruptions liées à des opérations échouées pour tenter de faire l'entretien de ces données virtuelles. Bien souvent ces outils permettent une utilisation programmatique et régulière et peuvent alerter les utilisateurs en cas de problématiques avérées.

Parmi toutes les stratégies disponibles pour assurer et accélérer l'accès aux données, les systèmes de fichiers ne sont pas en reste. La journalisation permet, selon les FS, d'optimiser les contrôles d'intégrité. Cette optimisation réduit le temps d'accès et de mise à disposition des informations du volume en cas de redémarrage du système.

De nombreuses opérations de dépannage, de modification ou d'administration nécessitent le démontage puis le remontage des volumes concernés. Si le volume contient des données requises pour faire fonctionner le système d'exploitation, il est difficile d'utiliser les différents outils nécessaires. Dans ce cas, il peut être important d'avoir recours à un live USB qui embarquera le strict minimum pour faire les opérations sur les FS et les volumes et ainsi pouvoir agir même sur les volumes nécessaires au bon fonctionnement de la machine.