FICHE DE RÉVISION — GESTION DES DISQUES SOUS LINUX

TSSR — Identification, partitionnement, formatage, montage, LVM et RAID

1. Vue d'ensemble — le parcours d'un disque

```
1. IDENTIFIER   →  2. PARTITIONNER  →  3. FORMATER  →  4. MONTER
   lsblk             fdisk / parted       mkfs.xxx        mount
   fdisk -l          (MBR ou GPT)                         /etc/fstab
   blkid
```

Un disque brut n'est utilisable qu'après avoir traversé ces QUATRE étapes. Sauter une étape = disque inutilisable.

2. Identifier un disque

Commande	Rôle
`lsblk`	vue arborescente des disques / partitions, tailles, points de montage
`fdisk -l`	liste détaillée des disques et tables de partitions (root requis)
`blkid`	UUID et type de filesystem de chaque partition
`df -h`	espace utilisé / disponible sur les filesystems montés
`du -sh dossier`	taille réelle occupée par un dossier

2.1 Décoder un nom de périphérique — les 4 étapes

Étape	Ce qu'on lit	Exemple
1	**la connectique** : `h` = IDE, `s` = SCSI / SATA / SAS	`s`…
2	**le type de device** : la lettre suivante est un `d` = disk	`sd` = SATA Disk
3	**la lettre du disque physique**, allouée par le système	`sda` = disque 1, `sdb` = disque 2
4	**le numéro de partition**, ajouté en chiffre	`sda1`, `sda2`…

2.2 Convention de nommage

Préfixe	Signification
`sd` **+ lettre**	disque SATA, SCSI, SAS **ou USB** (`usb-storage` émule du SCSI) — `sda`, `sdb`…
`h` **(IDE)**	obsolète ; un disque IDE apparaît aujourd'hui aussi en `sdX` (driver `libata`)
`nvme0n1`, `nvme0n1p1`	disque NVMe — le `p` avant le numéro de partition est **obligatoire**
`vda`	disque virtuel (VirtIO — Hyper-V, KVM)
chiffre suffixe	numéro de partition : `sda1`, `sda2`…

> *Ce nommage n'est stocké nulle part de façon permanente. /dev est un système de fichiers virtuel, reconstruit en mémoire à chaque démarrage : le noyau attribue sda/sdb selon l'ordre de détection des bus, pas selon un identifiant fixe. Ce qui est réellement persistant sur le disque : la table de partitions et l'UUID du filesystem (écrit dans le superblock au mkfs).*

Toujours référencer une partition par UUID dans /etc/fstab. JAMAIS par /dev/sdX en dur.

3. Partitionner un disque

3.1 MBR vs GPT

	MBR (Master Boot Record)	GPT (Guid Partition Table)
Emplacements réservés	**4 entrées historiques**, occupées ou non	**128 partitions** par défaut
Limite de taille	2 To par disque (adressage LBA 32 bits)	pas de limite pratique
Étendue / logiques	oui, obligatoire au-delà de 4 partitions	n'existe pas

3.2 Le mécanisme étendue / logique (MBR uniquement)

Le MBR ne dispose que de **4 entrées historiques** dans sa table de partitions : c'est une **contrainte de conception d'origine**, pas une limite arbitraire. Ces 4 emplacements restent structurellement réservés, qu'ils soient tous occupés ou non.

Pour aller au-delà, une primaire devient **partition étendue** (un simple conteneur), à l'intérieur de laquelle on crée des **partitions logiques**, numérotées **à partir de 5** — jamais 3 ou 4, quel que soit le nombre réel de primaires, puisque 1 à 4 restent la propriété de cette structure historique.

```
sda1  (primaire)
sda2  (étendue — conteneur)
  ├─ sda5  (logique)
  └─ sda6  (logique)
```

> *GPT ne connaît AUCUNE de ces catégories. Toutes les partitions sont décrites de façon identique dans la table (jusqu'à 128 entrées par défaut) : ni primaire, ni étendue, ni logique, parce que la contrainte qui justifiait cette distinction n'existe plus.*

3.3 Les outils

```
# fdisk — MBR (et GPT depuis les versions récentes)
fdisk /dev/sdb
#   n = nouvelle partition   d = supprimer   p = afficher la table
#   t = changer le type      L = lister les codes disponibles
#   w = écrire (RIEN n'est appliqué avant w)
```

```
# parted — recommandé pour GPT / disques > 2 To
parted /dev/sdb
(parted) mklabel gpt
(parted) mkpart primary ext4 0% 100%
(parted) print
(parted) quit
```

Dans fdisk, RIEN n'est appliqué avant w. On peut tout casser puis quitter sans écrire.

3.4 Codes de type de partition

En hexadécimal, colonne `Id` de `fdisk -l` :

Code	Filesystem
07	**NTFS standard** — le bon code, à utiliser dans l'immense majorité des cas
0C	FAT32 (LBA)

Code	Filesystem
05	Étendue
83	Linux
82	Swap Linux
8e	LVM Linux
ef	EFI

PIÈGE CLASSIQUE : 86 et 87 ne sont PAS les codes NTFS standards.

Ce sont des restes des **volumes dynamiques Windows NT** (miroir / RAID logiciel — Windows NT ajoute `0x80` au type de base pour un ensemble tolérant aux pannes). Ils peuvent apparaître par erreur ou par héritage sur une NTFS pourtant parfaitement simple. Le bon code reste `07`.

> *Aucun rapport avec une limite de taille (2 To) : cette limite dépend de l'adressage LBA du MBR lui-même, pas du code de type. Correction si repéré : fdisk /dev/sdb → t → numéro de partition → 07 → w*

4. Formater — créer un système de fichiers

Une fois la partition créée, elle est **vide** : il faut y écrire un système de fichiers.

4.1 Le mécanisme -t : mkfs est un simple aiguillage

```
mkfs -t ext4 /dev/sdb1      # strictement équivalent à :
mkfs.ext4 /dev/sdb1
```

`mkfs` **ne sait rien faire seul** : c'est un **frontend générique**. L'option `-t TYPE` lui fait appeler le binaire `mkfs.TYPE` correspondant. Si ce binaire est absent, l'erreur `not found` vient de **l'outil spécialisé manquant**, pas de `mkfs` lui-même.

4.2 Commandes par filesystem

Filesystem	Commande	Paquet requis	Préinstallé ?
ext2 / ext3 / ext4	`mkfs.ext4 /dev/sdb1`	`e2fsprogs`	oui, quasi systématique
XFS	`mkfs.xfs /dev/sdb1`	`xfsprogs`	**non**, à installer
FAT32	`mkfs.vfat -F 32 /dev/sdb1`	`dosfstools`	souvent, à vérifier
NTFS	`mkfs.ntfs /dev/sdb1`	`ntfs-3g`	variable selon la distro

> *FAT32 : ne pas oublier -F 32. Sans cette option, mkfs.vfat déduit automatiquement FAT12 / FAT16 / FAT32 selon la taille de la partition.*

Réflexe systématique avant tout TP : which mkfs.TYPE pour vérifier la présence du binaire. Ne JAMAIS présumer qu'un outil est présent par défaut selon la distribution.

5. Monter un système de fichiers

5.1 Les deux informations minimales

- **La source** — le périphérique / la partition à monter. Généralement une partition (`/dev/sdb1`), mais aussi un UUID (`UUID=xxxx`), un partage réseau (`//serveur/partage` en CIFS) ou un fichier image.
- **Le point de montage** — un répertoire **déjà existant** : `mount` ne le crée pas automatiquement.

```
mkdir /mnt/data
mount /dev/sdb1 /mnt/data
umount /mnt/data
```

Le type de filesystem est **optionnel** : sans autre argument, `mount` le **détecte automatiquement** via une inspection du superblock. C'est pour ça que ces deux informations suffisent dans la grande majorité des cas, sans avoir besoin de préciser `-t ext4`.

5.2 Principe fondamental — rien n'est accessible sans montage

Un disque ou une partition, même correctement partitionné et formaté, reste INVISIBLE et INUTILISABLE tant qu'il n'est pas monté.

Contrairement à Windows (automontage, lettre attribuée par défaut), Linux exige une **action volontaire**, sauf :

- la racine `/`, montée automatiquement par le noyau au boot ;
- l'automontage géré par l'environnement de bureau sur un poste graphique (GNOME / KDE) — ce n'est **pas** une propriété du noyau lui-même.

> *Un point de montage est un simple dossier hôte. Pendant qu'un périphérique y est monté, toute action (permissions, contenu) s'applique au filesystem monté, pas au dossier hôte. Après umount, le dossier hôte retrouve son état propre : ce sont deux entités distinctes, temporairement superposées.*

5.3 Empilement vs hiérarchie — à ne pas confondre

Empilement (mount stacking) — monter plusieurs filesystems sur le **même** point de montage les **empile** sans les remplacer. Seul le **dernier monté** est visible ; les précédents restent montés mais **masqués**, et redeviennent visibles au fur et à mesure des démontages, en ordre **LIFO**.

```
mount /dev/sdb1 /mnt/test     # contenu de sdb1 visible
mount /dev/sdc1 /mnt/test     # sdb1 masqué, contenu de sdc1 visible
mount | grep /mnt/test        # les DEUX lignes apparaissent : rien démonté
umount /mnt/test              # démonte sdc1 → sdb1 redevient visible
```

Hiérarchie — monter des filesystems différents sur des **sous-dossiers distincts** : tous restent accessibles **simultanément**, sans masquage.

```
/mnt/administratif          ← sdb1 (ext4)
├── compta                  ← sdb6 (xfs)
└── RH                      ← sdb2 (xfs)
```

```
ls /mnt/administratif           # contenu de sdb1
ls /mnt/administratif/compta    # contenu de sdb6
ls /mnt/administratif/RH        # contenu de sdb2
```

Empilement = même point de montage, seul le dernier est visible (LIFO). Hiérarchie = sous-dossiers distincts, tout reste visible simultanément.

C'est exactement le genre de structure construite en environnement professionnel : un dossier `/administratif` qui sert de point d'entrée logique, avec des sous-dossiers `compta` et `RH` qui pointent en réalité vers des **partitions physiquement séparées**. L'utilisateur voit une seule arborescence cohérente, sans savoir que trois disques différents sont impliqués.

5.4 Montage permanent — /etc/fstab

```
UUID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx  /mnt/data  ext4  defaults  0  2
```

Colonne	Sens
1	périphérique (**UUID**, jamais `/dev/sdX` en dur)
2	point de montage
3	type de filesystem

Colonne	Sens
4	options de montage
5	`dump` (0 = ignoré)
6	ordre `fsck` (0 = pas vérifié, 1 = racine, 2 = autres)

```
mount -a
```

mount -a applique toutes les lignes de fstab sans redémarrer. TOUJOURS tester avant de rebooter : une erreur de syntaxe sur un point de montage critique peut empêcher le système de démarrer.

6. Cas particulier — monter du NTFS depuis Linux

6.1 Pourquoi ça a du sens

Dual-boot Windows / Linux, disque externe partagé entre les deux OS, récupération de données ou forensics, échange hôte / invité en environnement virtualisé.

6.2 Deux modèles de permissions incompatibles

	Linux (POSIX)	NTFS
Modèle	triplet User / Group / Other (`rwx`)	ACL Windows (DACL, SID)
Commande	`chmod`	pas d'équivalent direct

6.3 Comportement par défaut de ntfs-3g

Sans option, un montage `ntfs-3g` accorde **rwx à tout le monde**, quel que soit l'utilisateur affiché comme propriétaire. C'est un comportement **documenté et volontaire**, faute de correspondance native entre NTFS et les permissions Unix.

```
mount /dev/sdb2 /mnt/ntfs
ls -l /mnt/ntfs
# drwxrwxrwx ...   ← accès total pour tout le monde
```

chmod sur un tel montage peut sembler fonctionner OU être silencieusement ignoré, sans jamais renvoyer d'erreur.

POSIX autorise explicitement chaque driver à décider comment gérer un mécanisme de contrôle d'accès alternatif.

6.4 Corriger le comportement par défaut

```
mount -t ntfs-3g /dev/sdb2 /mnt/data \
      -o uid=1000,gid=1000,umask=022,default_permissions
```

ou dans `/etc/fstab` :

```
# tout sur UNE seule ligne dans /etc/fstab :
UUID=xxxx  /mnt/data  ntfs-3g
           uid=1000,gid=1000,umask=022,default_permissions  0  0
```

Option	Rôle
`ro` / **`rw`**	**read only** ou **read write**
`uid`	le propriétaire (ex. `1000` = paulo)

Option	Rôle
`gid`	le groupe propriétaire (ex. `users`)
`umask=022`	le **masque maximal** est `777`, l'umask est une **soustraction** → `777 - 022 = 755`

> *Un disque NTFS monté sur un serveur multi-utilisateurs sans précaution expose son contenu en écriture à absolument tout le monde. Vérifier avec mount | grep ntfs.*

7. Le swap

```
mkswap /dev/sdb2
swapon /dev/sdb2
swapon --show
free -h
```

Ligne `fstab` correspondante :

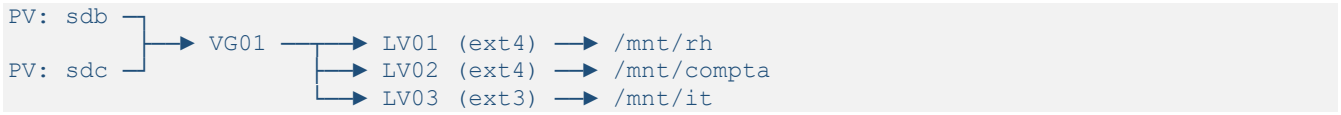
```
UUID=xxxx-xxxx    none    swap    sw    0    0
```

8. LVM — Logical Volume Manager

LVM ajoute une **couche d'abstraction** entre les disques physiques et les systèmes de fichiers, ce qui permet de **redimensionner un volume sans déplacer les données**.

```
Disque physique (PV) → Groupe de volumes (VG) → Volume logique (LV)
```

8.1 Exemple concret — mutualiser deux disques



Couche	Détail
PV	`sdb`, `sdc` — deux disques physiques distincts
VG	`VG01` — regroupe les deux disques en un seul pool d'espace
LV	`LV01`, `LV02`, `LV03` — trois volumes découpés dans ce pool
FS	ext4, ext4, ext3 — chaque LV peut avoir un filesystem différent
Montage	`/mnt/rh`, `/mnt/compta`, `/mnt/it` — chacun monté indépendamment

8.2 Les commandes

```
pvcreate /dev/sdb /dev/sdc          # déclare les disques en volumes physiques
vgcreate VG01 /dev/sdb /dev/sdc     # crée le groupe de volumes

lvcreate -L 20G -n LV01 VG01
lvcreate -L 30G -n LV02 VG01
lvcreate -L 15G -n LV03 VG01

mkfs.ext4 /dev/VG01/LV01            # FORMATER avant toute utilisation
mkfs.ext4 /dev/VG01/LV02
mkfs.ext3 /dev/VG01/LV03

mkdir -p /mnt/rh /mnt/compta /mnt/it
mount /dev/VG01/LV01 /mnt/rh
mount /dev/VG01/LV02 /mnt/compta
mount /dev/VG01/LV03 /mnt/it
```

Pour **ajouter un PV à un VG existant** :

```
vgextend VG01 /dev/sdd
```

> *L'intérêt concret : l'espace de sdb et sdc est mutualisé dans VG01, donc un LV peut dépasser la capacité d'un seul disque physique. Et chaque LV reste indépendant : redimensionner LV02 n'affecte ni LV01 ni LV03, même s'ils partagent le même pool sous-jacent.*

8.3 Extension à chaud — le vrai intérêt de LVM

```
lvextend -L +10G /dev/VG01/LV02
resize2fs /dev/VG01/LV02          # pour ext4
xfs_growfs /mnt/point_de_montage  # pour XFS (surtout pas resize2fs)
```

9. RAID logiciel — mdadm

Le RAID gère la TOLÉRANCE AUX PANNES et la performance. LVM gère la FLEXIBILITÉ de volume. Les deux sont complémentaires, pas concurrents.

```
mdadm --create /dev/md0 --level=5 --raid-devices=3 \
      /dev/sdb1 /dev/sdc1 /dev/sdd1
mdadm --detail /dev/md0
cat /proc/mdstat          # état de la synchronisation / santé du RAID
```

> *RAID5 : minimum 3 disques, tolère la perte d'un seul. Utiliser des disques CMR (pas SMR) pour éviter des reconstructions anormalement longues, voire des échecs de rebuild.*

10. Pourquoi ça marche ainsi — « Everything is a file »

Un disque physique est représenté par un **fichier spécial** de type **block device** :

```
ls -l /dev/sda
brw-rw---- 1 root disk 8, 0 ... /dev/sda
```

Le `b` (au lieu de `-` pour un fichier normal, `d` pour un dossier) signifie que ce fichier sert d'**interface** vers le disque physique. `mount` demande au noyau d'**interpréter le contenu brut de ce fichier spécial** comme un système de fichiers structuré, accessible via un point de montage.

11. Antisèche — commandes essentielles

Étape	Commande
Identifier	`lsblk`, `fdisk -l`, `blkid`
Partitionner	`fdisk` (MBR) / `parted` (GPT)
Changer le type	`fdisk` → `t` → code hexa
Formater	`mkfs.ext4`, `mkfs.xfs`, `mkfs.vfat -F 32`, `mkfs.ntfs`
Monter	`mount` / `umount`
Monter au boot	ligne `/etc/fstab` (UUID)
Tester fstab	`mount -a`
Vérifier un montage	`mount | grep chemin`
Espace disque	`df -h`, `du -sh`

Étape	Commande
Swap	`mkswap`, `swapon`, `free -h`
LVM	`pvcreate`, `vgcreate`, `lvcreate`, `vgextend`, `lvextend`
RAID	`mdadm --create`, `cat /proc/mdstat`
NTFS proprement	`mount -t ntfs-3g ... -o uid=,gid=,umask=,default_permissions`

12. Pièges et points à retenir pour l'ECF

Ordre imposé : IDENTIFIER → PARTITIONNER → FORMATER → MONTER.

- Dans `fdisk`, **rien n'est appliqué avant** `w`.
- MBR = **4 entrées**, 2 To max, logiques numérotées **à partir de 5**. GPT = 128 partitions, aucune notion de primaire / étendue / logique.
- Code NTFS = `07`. `86` / `87` sont des restes de **volumes dynamiques Windows NT**, pas une histoire de taille.
- `mkfs` n'est qu'un **aiguillage** vers `mkfs.TYPE` : `not found` = binaire spécialisé absent, pas `mkfs` en cause.
- FAT32 : sans `-F 32`, `mkfs.vfat` choisit tout seul entre FAT12 / 16 / 32.
- `mount` **ne crée pas** le point de montage : `mkdir` d'abord.
- `fstab` toujours par **UUID** — `/dev/sdX` peut changer au reboot, `/dev` étant virtuel et reconstruit à chaque démarrage.
- Tester avec `mount -a` **avant** de rebooter.
- Empilement = même point de montage, seul le dernier visible (LIFO). Hiérarchie = sous-dossiers distincts, tout reste visible.
- NTFS sans options = **rwx pour tout le monde**, et `chmod` peut être ignoré sans erreur.
- LVM : **PV** → **VG** → **LV**, et il faut **formater le LV** avant de l'utiliser.
- Extension à chaud = `lvextend` **puis** `resize2fs` (ext4) ou `xfs_growfs` (XFS).
- RAID5 = 3 disques minimum, tolère 1 perte. RAID ≠ LVM.