

# Introduction aux systèmes de fichiers et à ZFS

Stéphane D'Alu  
stephane.dalu@insa-lyon.fr

sdalu@sdalu.com

2019-2020

Version : 0.92



## Première partie I

# Volume manager et Systèmes de fichiers

- 1 Matériel
- 2 Concepts
  - Partitions et labels
  - RAID
  - Intégrité et chiffrement
  - Instantanés et clones
  - Compression
  - Cache
  - Consistence
  - Utilisation
- 3 Volume manager
  - LVM + device-mapper
  - Vinum / GEOM
- 4 Systèmes de fichiers

## Bloc

Disque vs Mémoire NAND vs Système de fichiers

**bloc** ensemble de données de taille constante

Sa signification exacte dépend du contexte :

► Disque

**bloc** unité de données transferable (aka : secteur disque)

📏 Tailles usuelles : 512 ou 4k

► Mémoire NAND (SSD)

**page** unité de programmation (écriture)

📏 Tailles usuelles : 512, 2k, 4k, 8k, 16k

**bloc** unité d'effacement

📏 Tailles usuelles : 32, 64, 128 pages

► Système de fichiers

**bloc** unité d'allocation

📏 Tailles usuelles : multiple de blocs disque

# Sommaire

## 1 Matériel

## 2 Concepts

- Partitions et labels
- RAID
- Intégrité et chiffrement
- Instantanés et clones
- Compression
- Cache
- Consistance
- Utilisation

## 3 Volume manager

- LVM + device-mapper
- Vinum / GEOM

## 4 Systèmes de fichiers

# Disques

## Technologies et interfaces

Technologie :

Type	Capacité	Latence	Débit	Prix <sup>1</sup>
mécanique	10 TiB	4 ms	200 MB/s	400 €
<i>solid-state</i>	2 TiB	80 µs	3500 MB/s	400 €



NVMe SSD

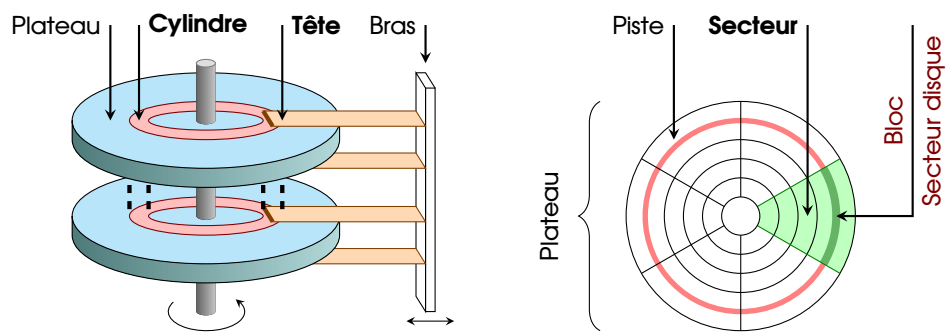
Interface :

Type	Version	Date	Débit	Entreprise	Status
NVMe (PCIe)	1.3c (Gen3 x4)	2018	3900 MB/s	✓	
SAS	4	2017	2400 MB/s	✓	
SATA	3.2	2013	1969 MB/s		
SCSI	Ultra-640	2003	640 MB/s	✓	obsolète
IDE	ATA-7	2005	133 MB/s		obsolète

1. Prix public moyen en 2018, interface SATA

# Secteur disque

Les origines



## ► Cylinder-Head-Sector (CHS)

Norme	Cylindre	Tête	Secteur	Adressage	Max
CHS	10 bits	8 bits	6 bits	24 bits	8 GiB
ECHS (E-IDE / ATA-2)	16 bits	4 bits	8 bits	38 bits	128 TiB

## ► Logical Block Address (LBA)

- Transition depuis CHS :  $LBA = (c \cdot N_{heads} + h) \cdot N_{sectors} + (s - 1)$
- Norme ATA-6 : adressage sur 48 bits

# Secteur disque

Quelle taille ?

À l'origine 512 octets, maintenant 4096 octets

Pourquoi changer la taille ?

⇒ Augmenter la capacité

- du disque en diminuant le nombre d'espace entre les secteurs
- adressable avec une même plage d'adresse

Différents types de secteur :

**natif** correspond à la représentation interne

**émulé** le disque ment sur sa représentation interne

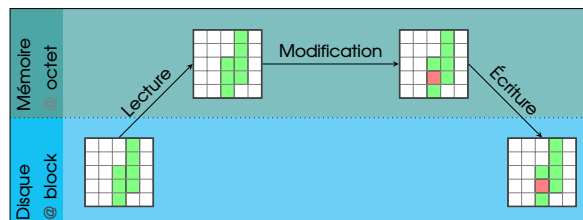
⇒ 512e, 4kn, 512n

# Secteur disque / bloc disque

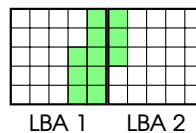
Unité de données

secteur disque plus petite unité de données transférable (lecture/écriture)

- ▶ un secteur disque (bloc) est lu et écrit atomiquement
- ▶ l'écriture de données plus petites que le secteur nécessite une opération de lecture **et** une d'écriture



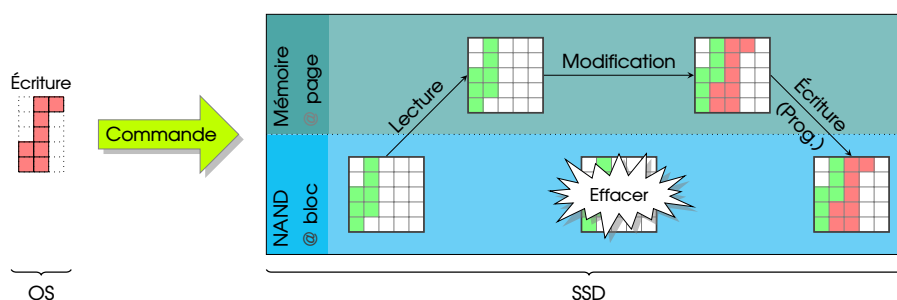
- ▶ des données plus petites que le secteur peuvent nécessiter deux secteurs



# Disques à base de mémoire

SSD, clef USB, carte mémoire, ...

Une mémoire NAND (disque SSD, clef USB) à besoin d'effacer son bloc mémoire avant de pouvoir y écrire des pages (~ blocs disque) de données (cycle *read-erase-modify-write* ou *erase-write*).



Écriture de pages sur un bloc non-effacé d'une mémoire NAND

👉 Usure liée à l'effacement

# Disques mécaniques (magnétiques)

PMR, SMR, ...

- ▶ De nombreuses évolutions
  - ▶ Têtes de lecture / écriture
  - ▶ L'enregistrement de l'information magnétique
  - ▶ L'organisation des pistes
- ▶ Technologies actuellement disponibles :
  - PMR Perpendicular Magnetic Recording (2005)
  - SMR Shingled Magnetic Recording (2013)
- ▶ L'écriture est plus large que la lecture (taille des têtes)

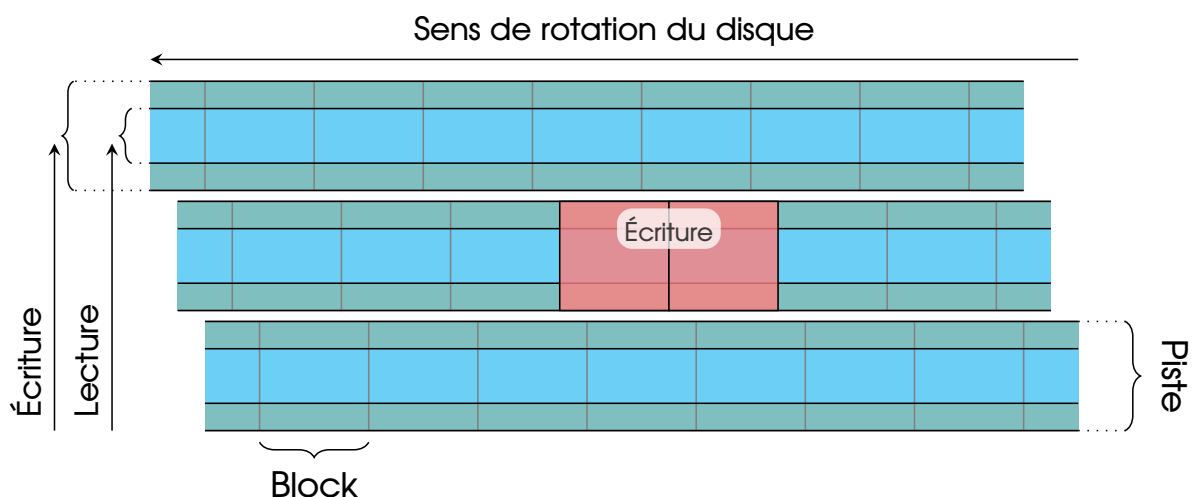
## Attention

- ▶ Les vendeurs ne précisent pas nécessairement la technologie utilisée ! (Western Digital et Toshiba ont vendus des SMR comme disques NAS)
- ▶ Performances catastrophique des SMR sous ZFS (lors de la reconstruction)

# Disques mécaniques (magnétiques)

Perpendicular Magnetic Recording (PMR)

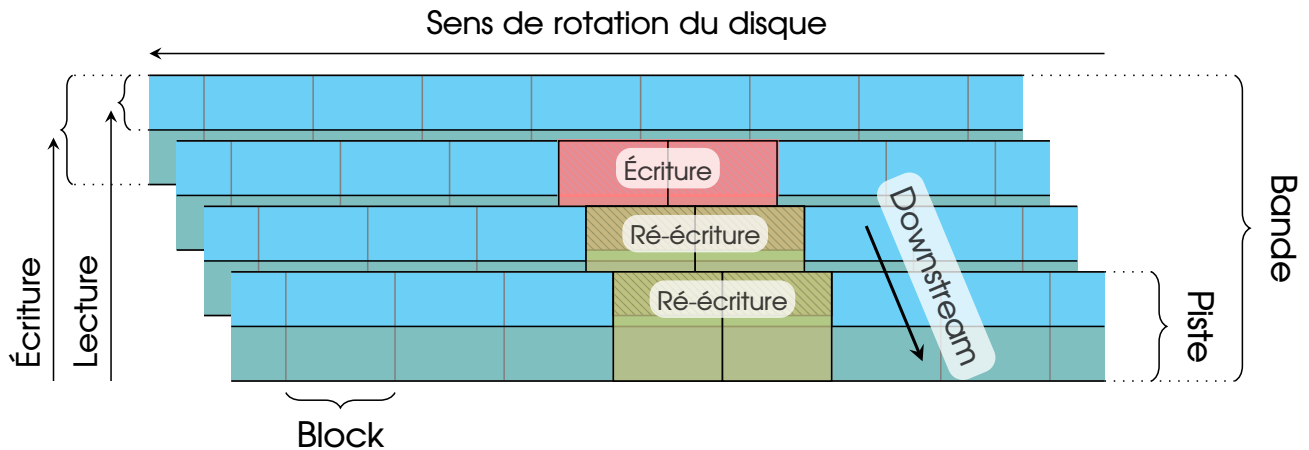
- ▶ Perpendicular Magnetic Recording (PMR)



# Disques mécaniques (magnétiques)

## Shingled Magnetic Recording (SMR)

- ▶ Shingled Magnetic Recording (SMR)
- ▶ Device-Managed / Host aware / Host managed
- ▶ Écrire un block  $\Rightarrow$  effacer une région
  - $\Rightarrow$  Mauvaise performance en écriture



# TRIM / UNMAP


## Libéré-délivré

Commande : TRIM (ATA) ou UNMAP (SCSI)

- ▶ Notifie le disque dur qu'un bloc disque n'est plus utilisé
- ▶ Optimisation des écritures
  - ▶ Cycles de programmation (SSD)
  - ▶ Ré-écriture (SMR)

## Effacer une mémoire NAND l'use !

 de 100 à 1.000.000 cycles programmation/effacement (PE)

 Problème des fichiers souvent modifiés  
(1 écriture/minute  $\Rightarrow \simeq 500.000$  cycles PE par an)

 Critique pour les équipements à base de micro-contrôleurs

$\Rightarrow$  Réduire l'usure en ré-assignant les blocs (table d'indirection) et choisissant ceux ayant subi le moins d'effacement

Stratégies de sélection du bloc à réassigner :

- ▶ dynamique : le bloc en cours de ré-écriture est considéré (*simple*)
- ▶ statique : l'ensemble des blocs sont analysés (*complexe*)

## Sommaire

### 1 Matériel

### 2 Concepts

- Partitions et labels
- RAID
- Intégrité et chiffrement
- Instantanés et clones
- Compression
- Cache
- Consistance
- Utilisation

### 3 Volume manager

- LVM + device-mapper
- Vinum / GEOM

### 4 Systèmes de fichiers



# Partitions

## GPT vs MBR

**partition** région du disque gérée séparément par le système d'exploitation

La description des partitions est définie dans la GPT ou le MBR.

Les partitions permettent entre autres d'avoir sur un même disque :

- ▶ des systèmes de fichiers différents
- ▶ plusieurs systèmes d'exploitation
- ▶ une séparation entre données (système, utilisateurs, . . .)

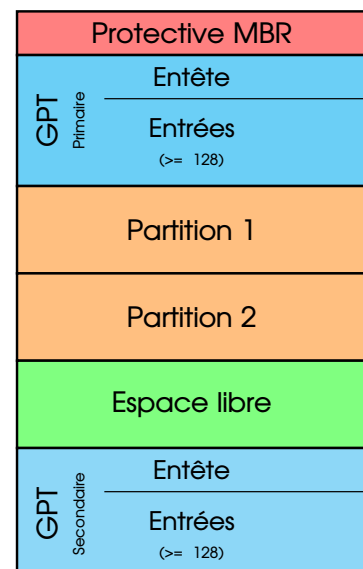
Nom	LBA	Partitions	Types	Label	GUID	Utilisation	Remarque
GPT	$2^{64}$	$\geq 128$	$2^{128}$	✓	✓	UEFI	
MBR	$2^{32}$	4	$2^8$			BIOS	fin de vie

Comparaison GPT / MBR

# GUID Partition Table (GPT)

## Caractéristiques et structuration du disque

- ▶ Protégé contre une utilisation par le BIOS (Protective MBR)
- ▶ Identification du disque par GUID
- ▶ Identification des partitions par : index, label, GUID
- ▶ Existence d'une GPT de secours
- ▶ Protection par CRC (entête + liste des entrées)
- ▶ Limitée à 128 entrées par les implémentations



Utilisation de l'espace disque

# Labels

Identifier un disque ou une partition

**label** chaîne de caractères identifiant un disque ou une partition indépendamment de l'ordre d'insertion

Générateur de labels au sein du système d'exploitation :

- ▶ GUID Partition Table  
Partition : label et GUID
- ▶ Système de fichiers  
UFS    `tunefs -L`                    NTF    `ntfslabel`  
Ext4   `e2label`                        exFAT   `extfatlabel`
- ▶ Numéro de série du disque
- ▶ Label explicite (FreeBSD : `glabel`)

# JBOD et RAID

Aggrégation de disques

**JBOD** Just a Bunch of Disks

**RAID** Redundant Array of Inexpensive Independent Disks

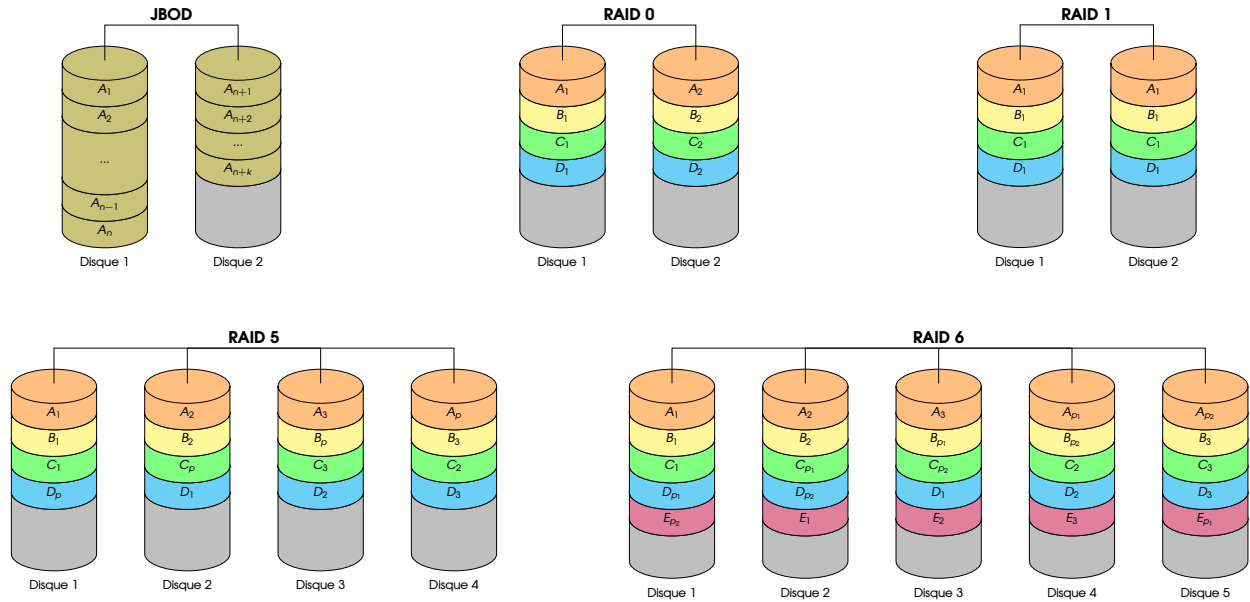
Appellation	Agrégation	Redondance	Disques		
			Min.	Panne	Efficacité
JBOD	concaténation	aucune	1	0	1
RAID 0 ( <i>stripping</i> )	par bandes	aucune	2	0	1
RAID 1 ( <i>mirroring</i> )	par bandes	duplication	2	$n - 1$	$1/n$
RAID 5	par bandes	parité répartie	3	1	$1 - 1/n$
RAID 6	par bandes	parité répartie	4	2	$1 - 1/2n$

Aggrégations de disques les plus courantes

# JBOD et RAID

Distribution des données sur les disques

Selon la configuration choisie, il est possible d'augmenter et/ou :  
la fiabilité, la capacité, le débit.

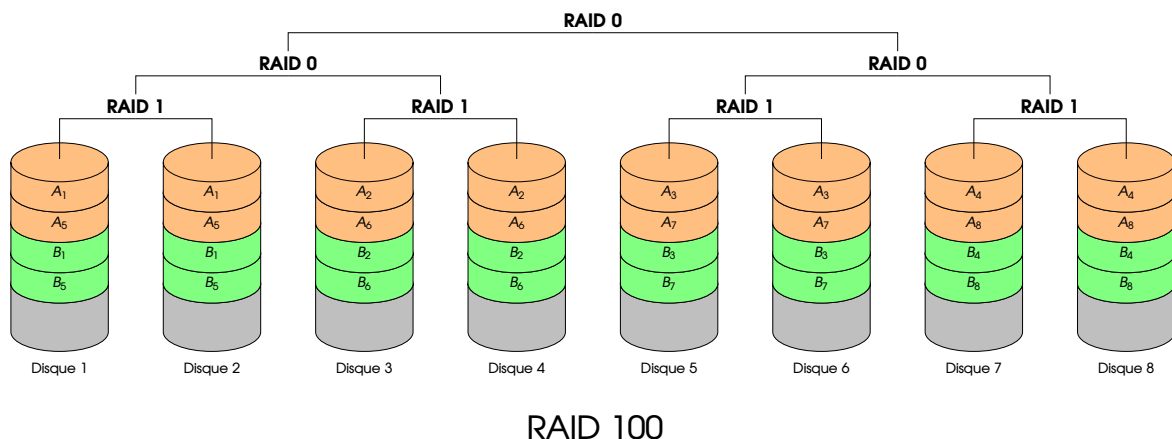


# JBOD et RAID

Combiner les agrégations de disques

Il est possible de combiner les agrégations entre elles  
(certaines combinaisons ne sont pas pertinentes) :

- RAID 10 = RAID 1 + RAID 0
- RAID 50 = RAID 5 + RAID 0
- RAID 100 = RAID 1 + RAID 0 + RAID 0
- JBOD + RAID 5



RAID 100

# RAID... quelques défauts

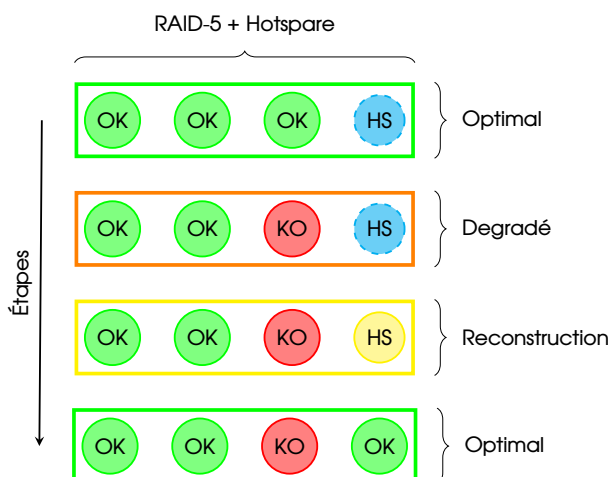
Pertes de données, gestion de l'espace disque

- ▶ possible corruption en cas de coupure de courant (*write-hole*)  
⇒ journalisation des écritures (*write-ahead logging*)
- ▶ pannes simultanées de disques (présence des mêmes défauts)  
⇒ ne pas acheter les mêmes disques (marque/série/version)
- ▶ temps de reconstruction important et probabilité de pannes  
⇒ augmenter la redondance
- ▶ fiabilité du *write-back cache*  
⇒ désactiver, avoir un cache avec batterie
- ▶ utilisation du disque entier (perte d'espace disque)  
⇒ RAID logiciel (possible sur des partitions)
- ▶ Pas de conversion possible entre RAID

# Hotspare

Ou comment gérer l'urgence

*hotspare* disque de rechange intégré au système et permettant le remplacement automatique d'un disque défectueux.



Séquence d'utilisation d'un *hotspare*

Avantage :

- ▶ réduction de la durée de fonctionnement en mode non-optimal

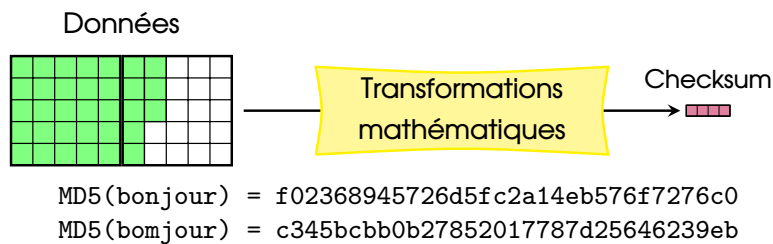
Inconvénient :

- ▶ monopolise un disque

# Checksum (Somme de contrôle)

SHA, SKEIN, . . .

**checksum** nombre dérivé d'un bloc de données, dont le but est d'en détecter une modification.



Nom	Bits de sortie	Pseudo-aléatoire	Remarque
Fletcher	16, 32, 64		rapide
MD5	128	✓	compromis
SHA-1	160	✓	compromis
SHA-2	224, 256, 384, 512	✓	
Skein	taille arbitraire	✓	

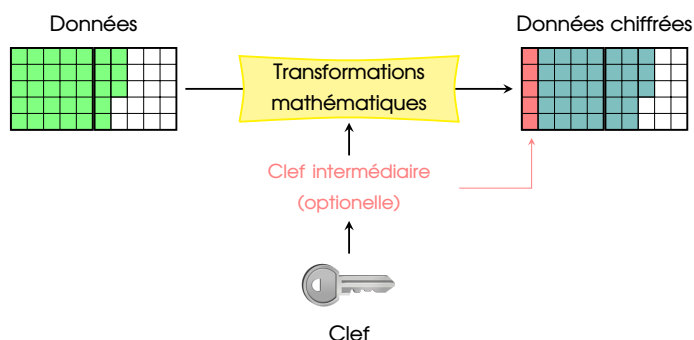
Fonctions de *checksum* / *hash* courantes

# Chiffrement

Pourquoi chiffrer ses données ?

**Chiffrement** Procédé grâce auquel on souhaite rendre la compréhension d'un document impossible à toute personne qui n'a pas la clé de (dé)chiffrement

- ▶ Protection des données contre :
  - ▶ le vol ou la perte de matériel (ordinateur portable)
  - ▶ l'espionnage



Clef intermédiaire (chiffrée) :

- ▶ clef utilisateur plus courte
- ▶ clefs multiples

# Chiffrement

AES, Twofish, Serpent, ...

Coût du chiffrement :

- ▶ entrées/sorties ralenties
- ▶ consommation énergétique accrue

Nom	Block	Clef			Certification			HW
		128	192	256	CRYPTREC	NESSIE	``AES finalist``	
AES	128	✓	✓	✓	✓	✓	✓	AES-NI
Camellia	128	✓	✓	✓	✓	✓		
Twofish	128	✓	✓	✓			✓	
Serpent	128	✓	✓	✓			✓	

Principaux algorithmes de chiffrement

# Chiffrement

L'accès aux données

Accéder aux données :

- ▶ Quand ? ⇒ lors du montage, au boot
- ▶ Comment ? ⇒ déchiffrement
  - ▶ mot de passe
  - ▶ dongle
  - ▶ authentification biométrique
  - ▶ TPM

## Attention

- ▶ mot de passe oublié ⇒ données perdues
- ▶ un disque ``monté`` est accessible
- ▶ 1 ou 2 secteurs défectueux (clef intermédiaire) ⇒ disque inutilisable (penser à faire une sauvegarde des clefs)

# Intégrité / Authenticité

## Hash-based Message Authentication Code

**Intégrité** assure l'absence de modification des données

**Authenticité** assure l'origine des données

L'**intégrité** est obtenue par le calcul d'un *checksum*, et l'**authenticité** par sa signature

⇒ le *checksum* seul ne protège pas d'un attaquant

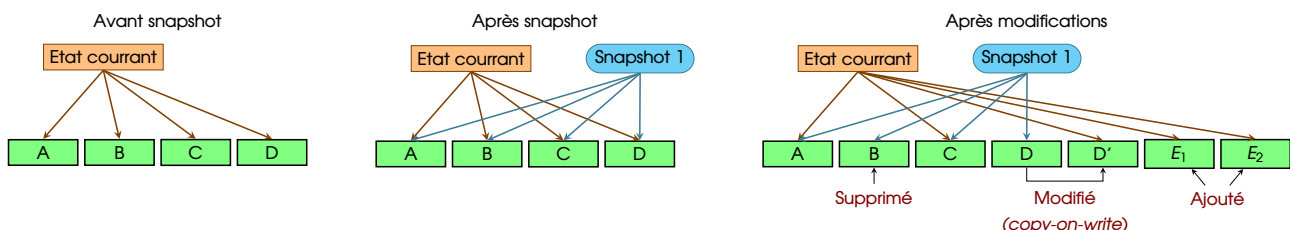
Coût **CPU** et **disque** important :

- ▶ ajout du *checksum*
- ▶ création de sa signature

# Snapshot (Instantané)

Figurer l'état du système de fichiers

**Instantané** état immuable (*i.e.* : lecture seule) du système (de fichiers) à un instant donné (opération atomique)



- ▶ création de sauvegardes à faible coût (*copy-on-write*)
- ▶ avoir un état cohérent pour de l'archivage (processus long)  
ex : `rsync` sur plusieurs TiB de données
- ▶ réalisation d'une *time-machine*  
ex : avoir une copie des différentes versions sur plusieurs jours
- ▶ pouvoir revenir en arrière (*rollback*)  
ex : réinitialiser le système après une mise-à-jour non fonctionnelle
- ▶ étape intermédiaire du clonage

# Clone

Un *snapshot* en lecture/écriture

**clone** *snapshot* dont l'état n'est plus immuable  
car accessible en écriture

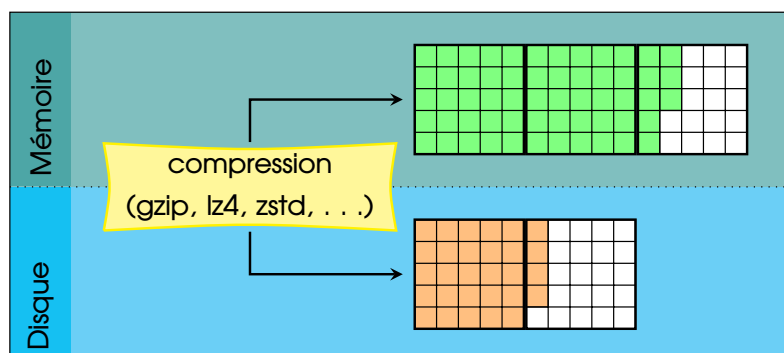
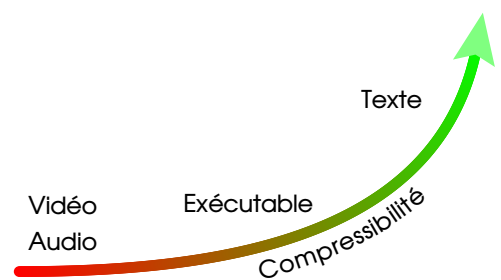
- ▶ avoir un état cohérent pour de l'archivage (processus long)  
ex : clonage d'une base de données pour y exécuter un *dump*
- ▶ dupliquer des machines virtuelles ou environnements d'exécution
- ▶ créer une "branche" parallèle pour du test / développement / reproductibilité / ...

# Compression

CPU vs Espace disque

Améliore en fonction de la compressibilité :

- ▶ la capacité disque utilisable
- ▶ le débit en lecture
- ▶ le débit en écriture (incertain)  
(le coût CPU peut être important)





# Compression

zlib, lz4, zstd, . . .

Choisir la méthode de compression en fonction des données et de l'utilisation qui en est habituellement faite :

- ▶ Archivage : zstd, zlib
- ▶ Utilisation courante : lz4
- ▶ Vidéo/Audio : pas de compression

Compresseur <sup>2</sup>	Version	Ratio	Compression	Décompression
zstd	1.3.4	2.877	470 MB/s	1 380 MB/s
zlib	1.2.11	2.743	110 MB/s	400 MB/s
brotili	1.0.2	2.701	410 MB/s	430 MB/s
lz4	1.8.1	2.101	750 MB/s	3 700 MB/s
snappy	1.1.4	2.091	530 MB/s	1 800 MB/s

Comparaison des différents algorithmes de compression

2. <http://facebook.github.io/zstd/#benchmarks>

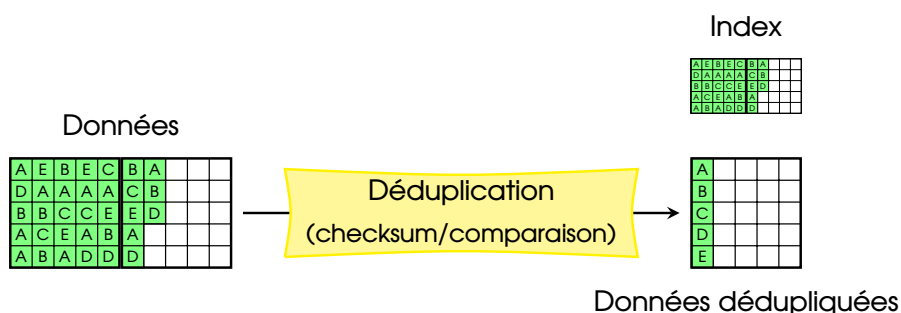
# Déduplication

Principe de fonctionnement

**Déduplication** compression par élimination des blocs de données identiques

Principe de fonctionnement :

- ▶ découpage par bloc
- ▶ discrimination des données différentes par checksum
- ▶ vérification de données identiques par comparaison (optionnel)
- ▶ table d'index pour la reconstruction



# Déduplication

Se fier au checksum ?

Hypothèses :

- ▶ fonction de hashage **pseudo-aleatoire** sur 256bits
- ▶ capacité de stockage de 1000 TiB
- ▶ secteur disque de 4 KiB

Probabilité de collision ( $n = 1000 \text{ TiB} / 4 \text{ KiB} = 250 \cdot 1024^3$ ,  $H = 2^{256}$ ) :  
 $\Rightarrow p \approx n^2 / 2H \approx 3.1 \cdot 10^{-55}$

## Paradoxe des anniversaires

Soit  $n$  éléments (élèves ou blocs disque) pouvant avoir  $H$  valeurs différentes (jour anniversaire ou valeur de hashage), et  $p$  la probabilité de collision.

- ▶  $p = 1 - \frac{H!}{(H-n)! \cdot H^n} \approx 1 - e^{-\frac{n^2}{2H}} \approx \frac{n^2}{2H}$
- ▶  $n \approx \sqrt{2H \cdot \ln \frac{1}{1-p}} \approx \sqrt{2H \cdot p}$

# Déduplication

Est-ce vraiment intéressant ?

Intérêt principal pour le stockage de données fortement identiques :

- ▶ machines virtuelles  
 $\Rightarrow$  Mitigé par : clonage de snapshots
- ▶ sauvegardes  
 $\Rightarrow$  Mitigé par : snapshots et sauvegardes incrémentales

Coût :

- ▶ fonction de hashage  $\Rightarrow$  CPU
- ▶ table d'indexation  $\Rightarrow$  mémoire  
(copie en mémoire pour raison de performance)

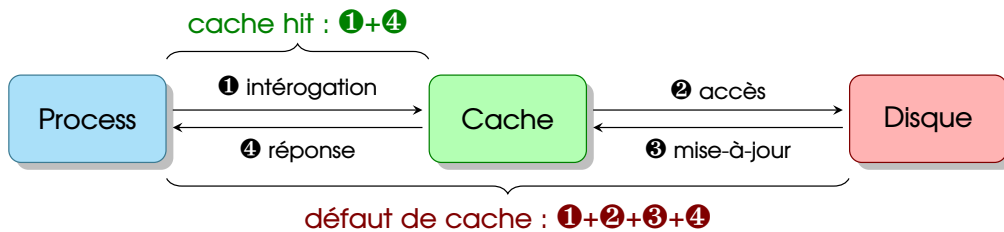
Dédupliquer les blocs disque ? ... Oui, mais en dernier recours

Bien envisager les solutions à base de snapshots et clones avant

# Cache

Accélérer les entrée/sortie

**cache** utilisation d'une mémoire extrêmement rapide mais disponible en faible quantité pour accélérer l'accès à une mémoire peu performante mais disponible en grande quantité



Stratégies de gestion du cache :

- ▶ Least Recently Used (LRU)
- ▶ Adaptive Replacement Cache (ARC)

1 récemment utilisés

3 → métadatas des évictions

2 fréquemment utilisés

4 → métadatas des évictions

# Consistence (1)

Un crash ? Et alors...

Kernel panic, coupure de courant, . . .

- 1 éviter la corruption
- 2 minimiser le temps de redémarrage (vérification/correction)

Les stratégies :

- ▶ **Ne rien faire.... et essayer de recoler les morceaux :**

Parcourir l'ensemble de la structure et y apporter les corrections nécessaires (e.g., `lost+found`).

⇒ Long et pas toujours efficace.

- ▶ **Journalisation :**

Enregistrer dans le journal (une petite zone disque) la liste des changements à effectuer (*write-ahead*), puis les réaliser sur le système de fichiers. En cas de crash, on rejoue le journal.

⇒ Le journal doit être résistant aux pannes (checksum + flush)

⇒ Les données sont écrites deux fois.

# Consistence (2)

Un crash ? Et alors...

- ▶ **Soft-Update :**

Réordonne les écritures pour éviter les inconsistences ou ne générer qu'une fuite d'espace disque. Après un crash, un processus (`fsck`) effectue en tâche de fond un *garbage-collection* pour récupérer l'espace disque.

- ▶ **Copy-on-Write (CoW) et modèle transactionnel :**

Les données ne sont jamais écrasées, et les séquences d'opérations sont soit committées soit complètement ignorées.

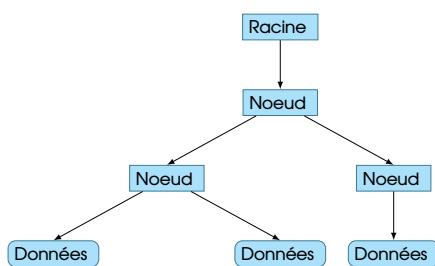
## Attention

En aucun cas l'absence de perte de données n'est garantie, le but étant de préserver l'intégrité des meta-données du système de fichiers (i.e., sa structure).

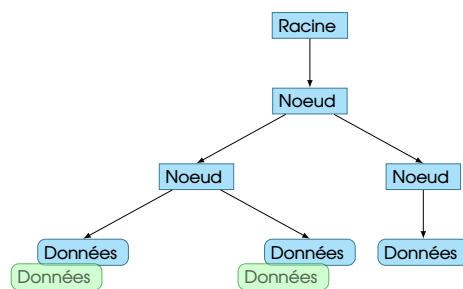
# Copy-on-Write (CoW)

Gestion et optimisation des ressources modifiables

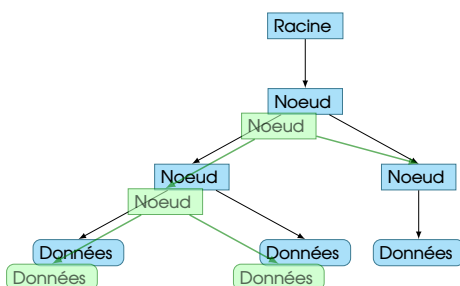
État initial



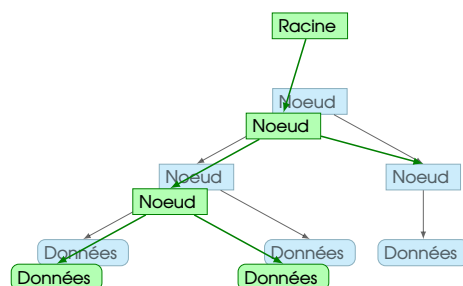
Modification des données



Modification de la structure



Basculement vers le nouvel état

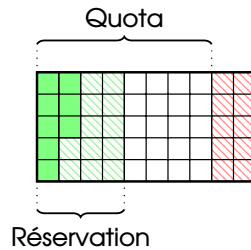


# Quota et Réserveation

## Gestion de la ressource disque

**quota** limite l'utilisation des ressources

**réserveation** garantie un minimum de ressource



La gestion des ressources peut être effectuée par rapport à :

- ▶ des utilisateurs
- ▶ des groupes d'utilisateurs
- ▶ un système de fichiers (cas de ZFS, Btrfs)

# Liens

*hardlink, symlink, varsym*

**lien** moyen d'associer différents noms au même fichier de données

- ▶ **Lien matériel** (*hardlink*) ⇒ inode

☞ compteur de référence

```
$ ln foo bar
$ ls -l bar
-rw-r--r--  2 sdalu  users  603589 Oct 24 17:55 bar
```

- ▶ **Lien symbolique** (*symlink*) ⇒ chemin

☞ possibilité de lien « pendouillant » (*dangling symlink*)

☞ *varsym* accepte la substitution de variable (NetBSD, DragonFly)

```
$ ln -s foo bar
$ ls -l bar
lrwxr-xr-x  1 sdalu  users           3 Oct 24 17:56 bar -> foo
```

Comparer : `rm foo ; cp zog foo ; cat bar`

# Encodage des noms de fichier

ASCII, UTF-8, latin-1, ...

- ▶ Internationalisation :
  - ▶ US-ASCII
  - ▶ Extended ASCII (ISO/IEC 8859 : Latin-1, Latin-2, ...)
  - ▶ CJK (Big5, Shift-JIS, ...)
  - ▶ **Unicode** (UTF-8, UTF-16, UTF-32 ; normalisation : NFD, NFC, NFKC, NFKD)

☞ un caractère n'est pas forcément égal à un octet

- ▶ Sensibilité à la casse (ex : `fichier` vs `FiChiEr`)

- ▶ Caractères illégaux :

- ▶ Unix : `/` `NUL`
- ▶ Windows : `%` `<` `>` `:` `"` `/` `\` `|` `?` `*` `NUL`

## Sélection d'un mauvais encodage

UTF-8 NFC : `répertoire` ⇒ ASCII-8bits : `r\303\251pertoire`

# Contrôle d'accès (ACL)

Unix, POSIX.1e, NFSv4

**ACL** permet d'autoriser ou de restreindre l'accès à des fichiers / répertoires.

- ▶ permissions ``UNIX``  
ex : `rwxr-x---`
- ▶ POSIX.1e (IEEE POSIX.1e draft 17, 1997)  
ex : `tag:qualifier:permissions`
- ▶ NFSv4 (RFC 5661, 2010)  
ex : `tag@qualifier:permissions:inheritance:type`

## Unix vs POSIX.1e vs NFSv4

- ▶ POSIX.1e est une extension des permissions ``Unix``
- ▶ les ACLs de POSIX.1e sont exprimables par des ACLs NFSv4

# Contrôle d'accès (ACL)

Liste des permissions (NFSv4)

Id	Nom long	Action	Cible
r	read_data	lecture	données
w	write_data	écriture	données
x	execute	exécution	
p	append_data	ajout	données
D	delete_child	suppression	
d	delete	suppression	
a	read_attributes	lecture	attributs
A	write_attributes	écriture	attributs
R	read_xattr	lecture	attributs étendus
W	write_xattr	écriture	attributs étendus
c	read_acl	lecture	ACL
C	write_acl	écriture	ACL
o	write_owner	écriture	propriétaire
s	synchronize	synchro	

Liste des permissions

# Contrôle d'accès (ACL)

Exemples

⇒ Les ACLs sont manipulés avec `getfacl` et `setfacl`

## Exemple: Unix

```
rwxr-x---
```

## Exemple: POSIX.1e

```
user::rwx
group::r-x
group:green:rwx
mask::rwx
other::r-x
```

## Exemple: NFSv4

```
owner@:rwxp--aARWcCos:-----:allow
group@:r-x---a-R-c--s:-----:allow
everyone@:r-x---a-R-c--s:-----:allow
```

- 1 Matériel
- 2 Concepts
  - Partitions et labels
  - RAID
  - Intégrité et chiffrement
  - Instantanés et clones
  - Compression
  - Cache
  - Consistance
  - Utilisation
- 3 Volume manager
  - LVM + device-mapper
  - Vinum / GEOM
- 4 Systèmes de fichiers

## Volume Manager

Du disque au volume

**Volume manager** méthode et logiciel de gestion de l'utilisation des espaces de stockage

Les disques sont **trop petits, trop lents** et **pas assez fiables**

- ▶ JBOD & RAID
- ▶ cache
- ▶ journalisation
- ▶ multipath

Besoins de sécurité, flexibilité :

- ▶ instantanés
- ▶ chiffrement, intégrité, authenticité
- ▶ provisionnement léger



# Logical Volume Manager (LVM) (Linux)

## Volume manager

☞ LVM utilise le *framework* **device-mapper** pour fournir ses services

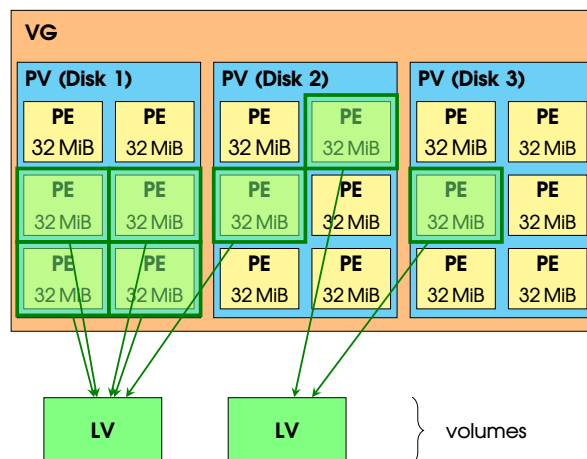
LVM est composé des objets :

PV disque physique

VG groupe de volumes

PE *extent* physique

LV volume logique



# Device-mapper (Linux)

## Framework d'accès disque

**Device-mapper** est un *framework* d'accès aux périphériques disques :

- ▶ création de volume disque de « plus haut niveau »
- ▶ possibilité d'appliquer device-mapper sur les volumes ainsi créés
- ▶ fichier de configuration centralisé (+ métadonnées sur les disques)

Fonctionnalités apportées par les modules :

- ▶ JBOD (concat)
- ▶ RAID : 0 (stripe), 1 (mirror), 5, 6
- ▶ multipath
- ▶ cache
- ▶ instantanés
- ▶ chiffrement, intégrité, authenticité
- ▶ provisionnement léger
- ▶ blocs contigus
- ▶ tests (délais, erreurs)

# Vinum (FreeBSD)

Volume manager historique

**Vinum** est le volume manager historique de FreeBSD :

☞ disponible depuis FreeBSD 3.0 (1998)

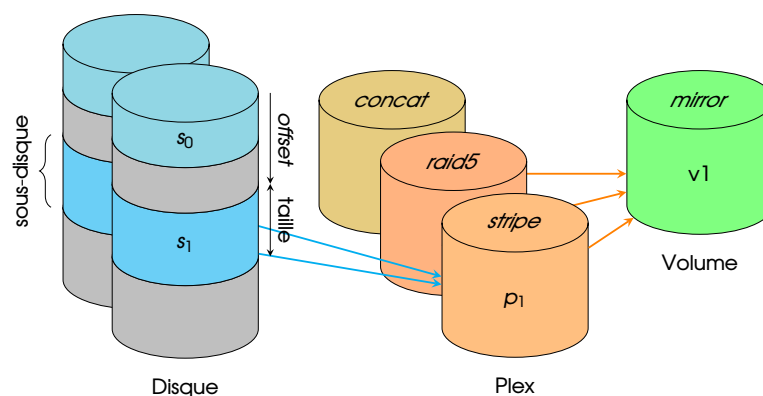
⇒ préférer ZFS ou une utilisation directe de GEOM

**drive** disque (ou périphérique de type bloc)

**sd** sous-disque (morceau de disque : *offset* + *taille*)

**plex** organisation de sous-disques (**raid5**, **stripe**, **concat**)

**volume** disque virtuel, composé de *plex* (**mirror**)



# GEOM (FreeBSD)

Framework d'accès disque

**GEOM** est un *framework* d'accès aux périphérique de type disque :

- ▶ fonctionne sur le modèle producteur/consommateur (⇒ chainnable)
- ▶ la configuration peut persister (sur le dernier bloc)

Fonctionnalités apportées par les différents modules :

- ▶ partitions (MBR, GPT, ...)
- ▶ JBOD (concat)
- ▶ RAID : 0 (stripe), 1 (mirror), 3, 5
- ▶ label
- ▶ multipath
- ▶ journalisation
- ▶ cache
- ▶ chiffrement, intégrité, authenticité
- ▶ secret partagé
- ▶ provisionnement léger
- ▶ périphérique distant
- ▶ périphérique en mémoire
- ▶ vinum
- ▶ tests (nop, sched)

- 1 Matériel
- 2 Concepts
  - Partitions et labels
  - RAID
  - Intégrité et chiffrement
  - Instantanés et clones
  - Compression
  - Cache
  - Consistance
  - Utilisation
- 3 Volume manager
  - LVM + device-mapper
  - Vinum / GEOM
- 4 Systèmes de fichiers

## Système de fichiers

Vers une spécialisation

On peut distinguer plusieurs classes de système de fichiers, ceux :

- ▶ pour disques durs : UFS, Ext2/3/4, XFS, . . .
- ▶ pour disques optiques : ISO 9660, UDF
- ▶ pour mémoires flash (sans *wear-leveling*) : F2FS, JFFS2, UBIFS, . . .
- ▶ en mémoire : tmpfs, ramfs, shmfs, . . .
- ▶ intégrés à un gestionnaire de volumes : ZFS, Btrfs
- ▶ . . .

### *wear-leveling*

- ▶ les SSD intègrent du *wear-leveling*
- ▶ en fonction de la qualité des cartes SD, préférer F2FS à Ext4

# Systèmes de fichiers

Critères de comparaison : la structure interne

- ▶ Limites
  - ▶ capacity max. du volume
  - ▶ nb. max. de fichiers
  - ▶ taille max. du fichier
  - ▶ taille max. du nom de fichier
  - ▶ taille max. du chemin
- ▶ Allocation
  - ▶ bloc de taille variable
  - ▶ *block suballocation / tail packing*
  - ▶ extents / bitmap
  - ▶ *allocate-on-flush*
  - ▶ inodes dynamiques
- ▶ Consistence
  - ▶ somme de controle
  - ▶ Copy-on-Write
  - ▶ journalisation
- ▶ Divers
  - ▶ XIP (*eXecute In Place*)
  - ▶ TRIM support

# Systèmes de fichiers

Critères de comparaison : les fonctionnalités

- ▶ Estampille temporelle
  - ▶ création
  - ▶ dernier accès
  - ▶ dernière modification
  - ▶ dernier archivage
- ▶ Nom de fichiers
  - ▶ caractères autorisés
  - ▶ sensibilité à la casse
  - ▶ préservation de la casse
- ▶ Liens
  - ▶ lien matériel (*hard link*)
  - ▶ lien symbolique (*symlink, varsym*)
- ▶ Composition des données
  - ▶ attributs étendus / forks / flux de données alternatifs
  - ▶ fichiers partiels (*spares files*)
- ▶ Sécurité
  - ▶ propriétaire
  - ▶ permissions
  - ▶ ACL (POSIX.1e, NFSv4)
  - ▶ MAC labels
  - ▶ File change log
- ▶ Gestion des données
  - ▶ instantanés, clones
  - ▶ chiffrement, intégrité
  - ▶ somme de controle
  - ▶ compression transparente
  - ▶ déduplication
- ▶ Changement de capacité
  - ▶ en-ligne / hors-ligne
  - ▶ agrandir / réduire

# Systèmes de fichiers

## Comparatif rapide

Caractéristiques	UFS2	Ext4	exFAT	XFS	Btrfs	ZFS
Capacité	8 ZiB	1 EiB	128 PiB	8 EiB	16 EiB	$2^{128}$
Taille fichiers	8 ZiB	16 TiB	128 PiB	8 EiB	16 EiB	16 EiB
Nb de fichiers		$2^{32}$	$2.7 \cdot 10^6$	$2^{64}$	$2^{64}$	$2^{48}$ /rép.
Noms de fichiers	255	255	255	255	255	255
Instantanés	✓			📁	✓	✓
Chiffrement		✓			📁	📁
Compression					✓	✓
TRIM	✓	✓		✓	✓	✓
CoW				📁	✓	✓
Journalisation	✓	✓		✓		✓
Inode dyn. alloc.				✓	✓	✓

Caractéristiques des principaux systèmes de fichiers (2019)

🔗 [https://en.wikipedia.org/wiki/Comparison\\_of\\_file\\_systems](https://en.wikipedia.org/wiki/Comparison_of_file_systems)

## Système de fichiers "traditionnel" (1)

Ext2/3/4, UFS, FFS, . . .

Le système de fichiers est composé en interne des éléments suivants :

**bloc** espace disque de taille fixe servant au stockage de l'information

📁 un multiple du bloc disque

**extent** zone disque contigüe exprimée en blocs

**superblock** bloc contenant la configuration du système de fichiers

**inode** index node, c'est la structure identifiant un fichier

Extensions :

▶ journalisation

▶ NFSv4

▶ soft-update

▶ POSIX.1e

Ses caractéristiques sont fixées à la création :

- ▶ nombre d'inodes (⇒ nombre max de fichiers)
- ▶ nombre et taille des blocs (⇒ espace disque disponible)
- ▶ nombre de *superblocks* (⇒ redondance, en cas de corruption)

Possibilité de modifier les caractéristiques hors-ligne :

- 1 démonter le système de fichiers (*i.e.* : hors-ligne)  
⇒ pour la partition système démarrer sur une clef USB bootable
- 2 redimensionner la partition (si possible !)  
⇒ dev-mapper (LVM) sous Linux peut simplifier ce processus
- 3 redimensionner le système de fichiers  
UFS `growfs` Ext2/3/4 `resize2fs`

⇒ L'outil `gparted` automatise ce processus.

## Deuxième partie II

# ZFS

# Sommaire

5 Présentation

6 ZPOOL

7 ZFS

- Volume

- Système de fichiers

# Sommaire

5 Présentation

6 ZPOOL

7 ZFS

- Volume

- Système de fichiers

ZFS, qu'est-ce ?

- ⇒ un *volume manager* **et** un système de fichiers
- ⇒ des limitations inatteignables (disque :  $2^{128}$ , taille fichier :  $2^{64}$ , entrées :  $2^{48}$ , ...)
- ⇒ conçu pour éviter la perte de données (*bit-rot*, matériel défectueux, ...)

Les fonctionnalités :

- ▶ "JBOD", RAID-Z, miroir
- ▶ copie multiple
- ▶ *checksum*
- ▶ *hot spare*
- ▶ label automatique
- ▶ consistant (CoW + journalisation)
- ▶ auto-correctif
- ▶ snapshots, clones, et rollback
- ▶ compression, déduplication
- ▶ **chiffrement, authenticité**
- ▶ quota et réservation
- ▶ ACL : Unix et NFSv4
- ▶ nom de fichiers : UTF-8, ASCII-8bit, choix de la casse

## Origines et évolution (1)

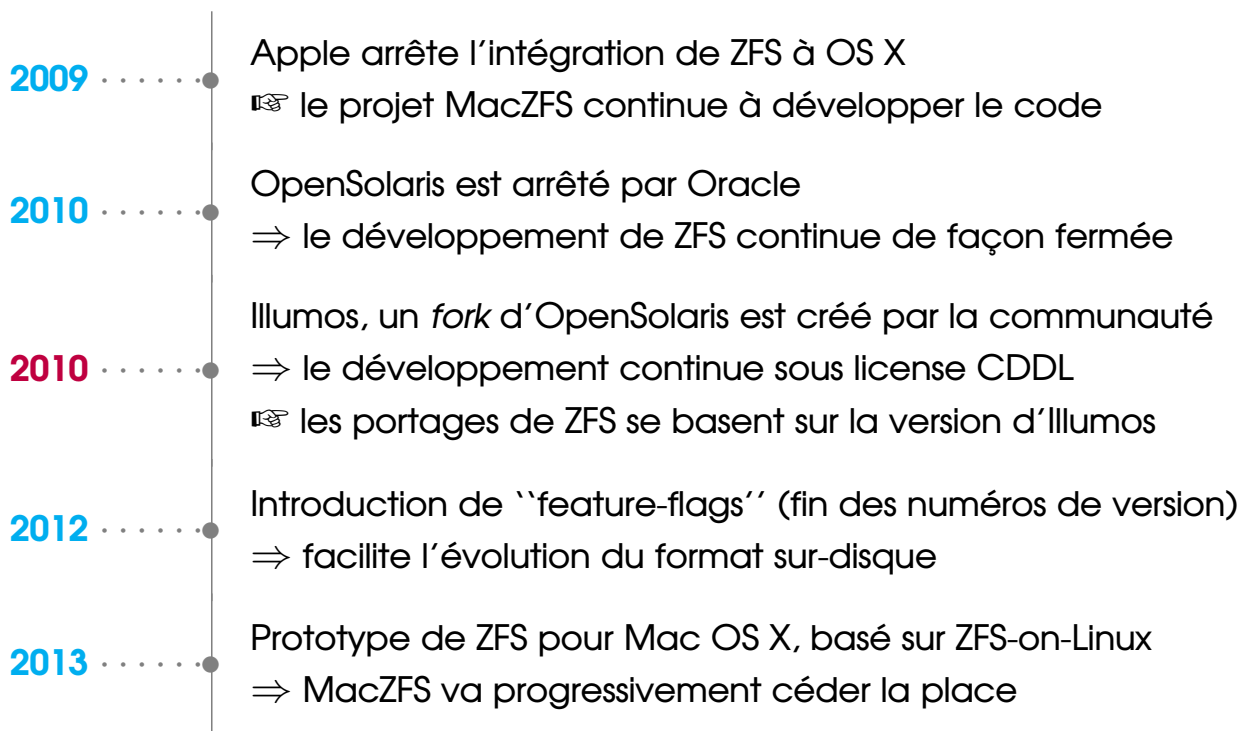
De ZFS à OpenZFS

- 2001** ..... ● Début du projet par deux ingénieurs de Sun Microsystems
- 2005** ..... ● Diffusion du code source dans OpenSolaris (license CDDL)
- 2006** ..... ● Début du portage sous Linux via FUSE  
☞ incompatibilité déclarée entre la licence CDDL et GPL
- 2007** ..... ● Apple commence son portage sous Mac OS X
- 2007** ..... ● NetApp attaque Sun pour violation de patente (WAFL)
- 2008** ..... ● ZFS est intégré FreeBSD 7.0
- 2008** ..... ● Début d'un portage native sous Linux (ZFS-on-Linux)
- 2009** ..... ● Rachat de Sun Microsystems par Oracle



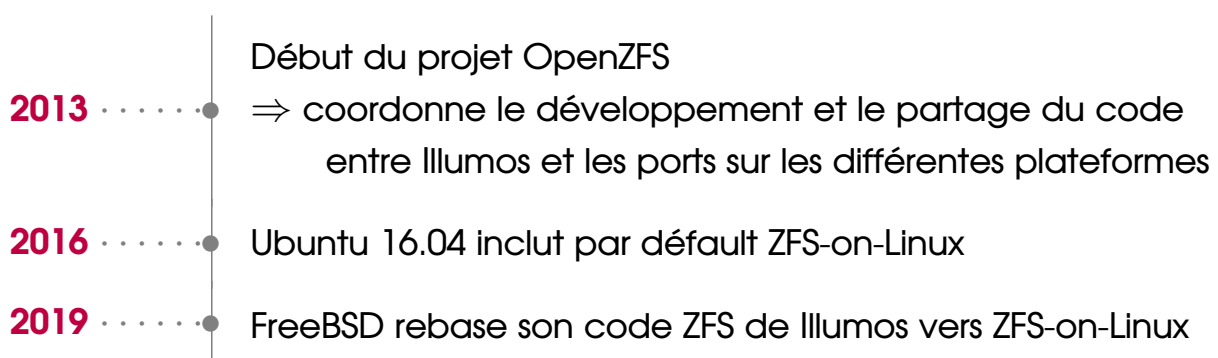
## Origines et évolution (2)

De ZFS à OpenZFS



## Origines et évolution (3)

De ZFS à OpenZFS



### Prochaines évolutions attendues

- ▶ réplication automatique
- ▶ chiffrement
- ▶ ZFS on Windows

# Disponibilité

Illumos, FreeBSD, Linux, OS X, NetBSD, ...

OS	Natif	Téléchargement
Illumos	✓	
FreeBSD	✓	
Linux		<a href="https://zfsonlinux.org/">https://zfsonlinux.org/</a>
Mac OS X		<a href="https://openzfsonosx.org/">https://openzfsonosx.org/</a>
NetBSD	✓	
Windows		<a href="https://github.com/openzfsonwindows/ZFSin">https://github.com/openzfsonwindows/ZFSin</a>

Disponibilité d'OpenZFS sur les différentes plateformes

## Monter son NAS (Network-attached storage)

FreeNAS, une version spécialisée de FreeBSD, offre les fonctionnalités d'un NAS basé sur ZFS. Il est configurable via une interface graphique.

# Comment utiliser ZFS ?

Se référer aux manuels ;)

```
$ man zpool # Gestion des disques (Volume Manager)
$ man zfs   # Création de systèmes de fichiers et volumes
```

## ZFS vs OpenZFS

Suite à l'abandon de la version open-source par Oracle, ZFS (Oracle) et OpenZFS (CDDL) divergent légèrement en fonctionnalité et dans la syntaxe.

C'est **OpenZFS** qui va être traité ici, et appelé ZFS par abus de langage.

# Un peu de vocabulaire

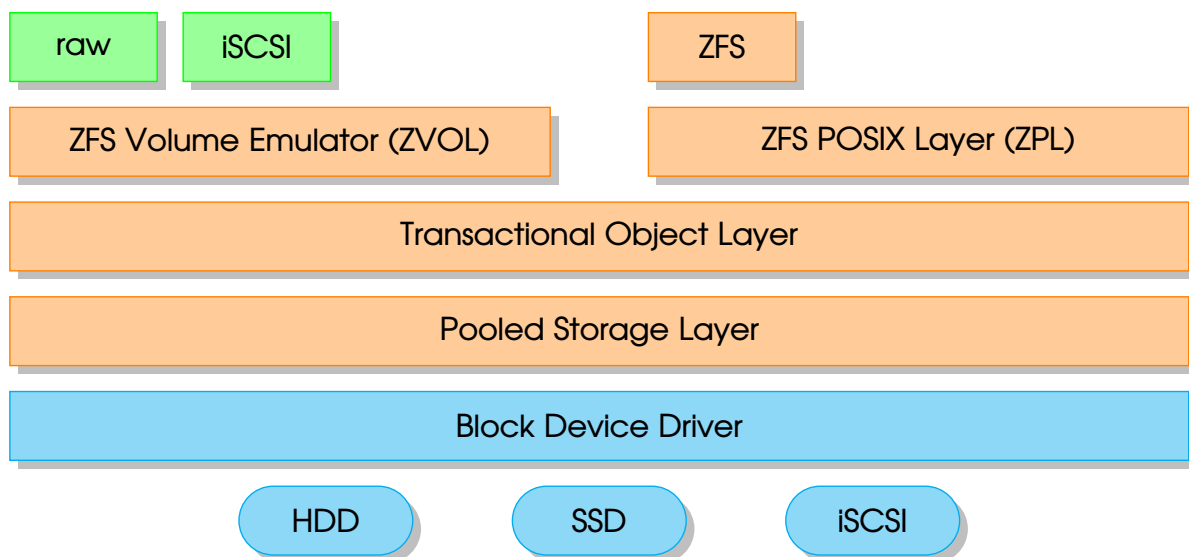
ARC ... ZPL

Nom	Signification
zpool	groupement de périphériques (vdev)
vdev	périphérique virtuel (intégrant ou non de la réplication)
raid-z	RAID avec copie-en-écriture sur blocs disque
zvol	volume (périphérique de type bloc)
ZPL	ZFS POSIX Layer
dataset	système de fichiers, volume, instantané, ou clone
snapshot	instantané
TXG	groupement de transactions
ZIL	journal des écritures synchrones (ZFS Intent Log)
DDT	table de déduplication
ARC	cache en lecture (mémoire) (Adaptive Replacement Cache)
L2ARC	cache en lecture de second niveau (disque)
ZIO	ZFS I/O
ZAP	ZFS Attribute Processor

## ZFS terminologie

# Point de vue du système d'exploitation

Les différentes couches



System view of the ZFS layers

# Version

Les évolutions de zpool et zfs

Les évolutions du format « sur disque » du *pool* et des différents systèmes de fichiers sont identifiés par un numéro de version.

⇒ une rétro-compatibilité partielle (lecture-seule) est possible

☞ La mise-à-jour s'effectue par la sous commande : **upgrade**

Parmi les fonctionnalités ainsi ajoutées depuis la création de ZFS :

**zpool** ditto, historique, compression, ZIL, *accounting*, déduplication, . . .

**zfs** sensibilité à la casse, quota, attributs système, . . .

## Pool : version vs feature

Le format du *pool*, pour les versions récentes, n'est plus défini par un numéro de version, mais en termes de fonctionnalités :

- ▶ `feature@feature_name = disabled | enabled | active`
- ▶ `unsupported@feature_guid`

# Performances

Bien connaître son système et ses données

Dégradation des performances à cause :

- ▶ besoin mémoire important : 2 GiB minimum !
- ▶ dégradation des performances après 80% de remplissage (CoW)  
(basculement de l'optimisation : vitesse → espace)
- ▶ utilisation de RAID matériel (à éviter !)  
(risque de perte de données (*write-hole*), latence supplémentaire)
- ▶ mauvaise utilisation de la déduplication
- ▶ absence de cache (ou mauvaise utilisation)
- ▶ absence de disque dédié à journalisation (ZFS Intent Log)
- ▶ le système de fichiers est à optimiser en fonction des données

⇒ **nécessite de connaître son système et ses données pour avoir des performances optimales**

L'attribut `ashift` définit le bloc minimal utilisé pour accéder aux disques :

- ▶ fixée à la création des vdev, **ne peut être modifiée**
- ▶ déterminé par le système en fonction des disques qui le composent  
⇒ sa valeur peut être incorrecte si le disque ment (ex : 512 vs 4K)
- ▶ les partitions doivent avoir un alignement adéquat ( $2^{\text{ashift}}$ )

### Mauvaise valeur

- ▶ amplification de l'écriture (*write-amplification*)
- ▶ réapparition du *write-hole* dû au cycle lecture-modification-écriture

### Les disques mentent...

De nombreux disques SSD indiquent un secteur à 512 B alors qu'ils gèrent en interne des blocs de 4 KiB voir 8 KiB

### Exemple

```
# FreeBSD: utilisation de sysctl ou gnop
sysctl vfs.zfs.min_auto_ashift=12 # définit le ashift minimum
gnop create -S4k da1             # ment sur la taille du secteur
zpool create tank ...
```

```
# Linux: existence de la pseudo-propriété ashift
zpool create -o ashift=12 tank ...
```

```
# Vérifier
zdb -C tank
```

### Linux

La pseudo-propriété `ashift` est problématique car elle est appliquée au *pool* à la place du *vdev*

Les différents états possibles pour un vdev :

- degraded** suffisamment de réplicats pour fonctionner malgré un ou plusieurs périphériques hors-lignes/dégradés/défectueux
- faulted** réplicat insuffisant pour fonctionner du à un ou plusieurs périphériques hors-lignes/dégradés/défectueux
- offline** explicitement marqué hors-ligne
- online** présent et fonctionnel
- removed** retiré pendant que le système est en cours d'exécution
- unavail** ne peut pas être ouvert

Le *pool* en fonction de ses *vdev* peut être dans l'un des états suivants : *online*, *degraded*, ou *faulted*

## Sommaire

### 5 Présentation

### 6 ZPOOL

### 7 ZFS

- Volume
- Système de fichiers

# Création d'un *pool*

Les différents types de *vdev*

- zpool** un *pool* ZFS est un ensemble de périphérique virtuels (*vdev*).
- vdev** périphérique virtuel pouvant fournir des fonctionnalités, une tolérance aux pannes, ou des performances accrues.

Les différents types de *vdev* possibles :

- |          |          |                |         |
|----------|----------|----------------|---------|
| Stockage | ▶ disk   | Fonctionnalité | ▶ spare |
|          | ▶ file   |                | ▶ log   |
|          | ▶ mirror |                | ▶ cache |
|          | ▶ raidz  |                |         |

## Répartition des données dans le *pool*

Les données sont réparties, lors de l'écriture, entre les différents *vdev* (servant de stockage) suivant un mode *best effort*.

# Création d'un *pool*

Disques et performances

Amélioration des performances en utilisant un *vdev* dédié pour :

- ▶ traiter les *données synchrones* (ZIL) (écriture) ⇒ log
- ▶ servir de cache (lecture) ⇒ cache

⇒ privilégier un SSD pour le log et le cache.

	ZIL	« pool »	L2ARC
Utilisation	écriture synchrone	données	accès au cache
Besoin	latence ↘ iops ↗		latence ↘ iops ↗
Opération	écriture	lecture/écriture	lecture
Disque	SSD	HDD	SSD
Taille	≈ 10 GiB		≈ 256 GiB

Choisir son disque pour les éléments du *pool* ZFS

# Création d'un *pool*

Ligne de commande

## Syntaxe

```
zpool create [-fnd] [-R root] [-t tempname] [-m mountpoint]  
[-o property=value]... [-O fs-property=value]...  
pool vdev...
```

## Exemples

```
# "JBOD"  
$ zpool create tank da1 da2  
# "JBOD" of mirror and "RAID5"  
$ zpool create tank mirror da1 da2 raidz da2 da4 da5  
# "RAID 5" + log  
$ zpool create tank raidz da1 da2 da3 log da4  
# "RAID 6" + cache  
$ zpool create tank raidz2 da1 da2 da3 da4 cache da5  
# "RAID 5" + log + cache + hotspare  
$ zpool create tank raidz da1 da2 da3 \  
    log mirror da4 da5 cache da6 spare d7
```

# Destruction d'un *pool*

... et comment le récupérer

Pour détruire un *pool* il doit être actuellement importé par le système.

Un *pool* détruit par erreur est récupérable (**import**) si :

- ▶ les données n'ont pas été écrasées
- ▶ les labels n'ont pas été détruits (**labelclear**)

## Syntaxe

```
zpool destroy [-f] pool  
zpool labelclear [-f] device
```

## Exemple

```
zpool destroy tank          # destruction du pool tank  
zpool import -f -D tank     # récupération du pool détruit
```



# Ajouter des éléments au *pool*

Étendre le stockage, ajouter des fonctionnalités

Ajouter de nouveaux éléments (*vdev*) au *pool* permet de :

- ▶ augmenter la capacité de stockage
- ▶ ajouter/étendre un ZIL séparé (SLOG)
- ▶ ajouter/étendre le cache (L2ARC)
- ▶ ajouter des hot-spare

## Syntaxe

```
zpool add [-fn] pool vdev ...
```

## Exemples

```
zpool add tank mirror da1 da2 # ajout d'un miroir (stockage)
zpool add tank log da1       # ajout d'un SLOG (ZIL)
zpool add tank cache da1     # ajout d'un cache
zpool add tank spare da1     # ajout d'un hot-spare
```

# Supprimer des éléments du *pool*

... à éviter sur l'espace de stockage

Peuvent être supprimés :

- ▶ les disques participant à log, cache, et spare
- ▶ les disques et mirror-xy à la racine du *pool*  
⇒ nécessite un remapping et un déplacement des blocs

☞ Utiliser `detach` pour supprimer un disque participant à un miroir.

## Syntaxe

```
zpool remove [-np] pool device...
```

## Exemple

```
zpool create tank mirror da1 da2 mirror da3 da4
zpool remove tank mirror-1
```

# Miroir

Des opérations spécifiques : `attach`, `detach`, `split`

Le miroir bénéficie d'opérations spécifiques :

`attach` transforme en un miroir à  $n + 1$  disques

`detach` transforme en un miroir à  $n - 1$  disques

`split` crée un nouveau *pool* en extrayant 1 disque de chaque miroir  
⇒ possibilité de spécifier le disque à extraire

☞ Les miroirs sont composés de 2 à  $n$  disques.

## Syntaxe

```
zpool attach [-f] pool device new_device
zpool detach pool device
zpool split [-n] [-R altroot] [-o mntopts] [-o property=value]
pool newpool [device...]
```

# Miroir

Exemple d'utilisation

## Exemple

```
# Création du pool
zpool create tank da1
# Transformation en un miroir à 2 disques
zpool attach tank da1 da2
# Extension du pool avec un miroir à 3 disques
# (mélange entre 2-disques et 3-disques nécessite l'option -f)
zpool add -f tank mirror da3 da4 da5
# Réduction du miroir à 3 disques en un miroir à 2 disques
zpool detach tank da5
# Scission du pool en deux
zpool split tank fish
```

## Réaliser un double

L'ajout de disques (`attach`) et la scission du *pool* (`split`) peut être une façon simple de créer une copie du *pool*.

# Import / Export

Prise en compte des *pool* par le système

Lorsqu'un *pool* est importé il est pris en compte par le système :

- ▶ systèmes de fichiers (datasets, snapshots) et volumes rendus accessibles
- ▶ *pool* persistant après un *reboot*
- ▶ consommation de ressources mémoires

## Syntaxe

```
zpool import [-o mntopts] [-o property=value]...  
              [--rewind-to-checkpoint] [-d dir | -c cachefile]  
              [-D] [-f] [-m] [-N] [-R root] [-F [-n]] -a  
zpool export [-f] pool...
```

## Exemple

```
zpool export tank # Exporte le pool tank  
zpool import      # Liste les pool importables  
zpool import tank # Importe le pool tank
```

# Checkpoint

Faire un instantané du *pool*

Le *checkpoint* permet de créer un instantané sur le *pool* :

- ▶ possibilité de rembobiner, de façon non destructive, jusqu'au *checkpoint*
- ▶ un seul *checkpoint* possible par *pool*
- ▶ seul un *import* permet de revenir à l'état du *checkpoint*

## Checkpoint sur un *pool* utilisé pour démarrer le système

Si ZFS est utilisé pour la racine (ie : / ) du système de fichiers et qu'un *checkpoint* est effectué sur ce *pool*, rembobiner jusqu'à celui-ci est problématique car il est nécessaire de faire un *export/import*.

## Syntaxe

```
zpool checkpoint [-d, --discard] pool  
zpool import [-o mntopts] [-o property=value]...  
              [--rewind-to-checkpoint] [-d dir | -c cachefile]  
              [-D] [-f] [-m] [-N] [-R root] [-F [-n]] -a
```

### Exemple

```
# Création du checkpoint
zpool checkpoint tank

# Exportation du pool (nécessaire pour faire un import)
zpool export tank
# Importation en lecture seule jusqu'au checkpoint
# => permet de regarder le pool à l'instant du checkpoint
zpool import -o readonly=on --rewind-to-checkpoint tank

# Exportation du pool (nécessaire pour faire un import)
zpool export tank
# Rembobine jusqu'au checkpoint
# => destruction des données situées après celui-ci
zpool import --rewind-to-checkpoint tank
# Suppression du checkpoint
zpool checkpoint -d tank
```

## Vérification de l'état du pool

### Vérifier et réparer

**scrub** parcourt l'ensemble des blocs du *pool* pour en vérifier l'intégrité via le *checksum*.

Le but de la vérification est de :

- 1 identifier les données corrompues (structure, fichiers, ...)
- 2 réparer les données si elles sont répliquées (miroir, raidz, copies, ditto, ...)

☞ Il s'agit d'une opération intensive sur les E/S

☞ Les données référencées par un *checkpoint* ne sont pas réparées

### Syntaxe

```
zpool scrub [-s | -p] pool...
```

### Exemple

```
zpool scrub tank
```

# Gestion des disques

online/offline

Un disque peut manuellement être passé « hors-ligne » ou « en-ligne » :

- ▶ forcer le changement d'état d'un vdev marqué en faute
- ▶ préparer une maintenance sur un disque
  - ☞ utilisation de `smartctl` pour détecter une défaillance
- ▶ activer le redimensionnement du disque
  - ☞ nécessaire si la propriété `autoexpand` du `pool` est à `off`

## Syntaxe

```
zpool offline [-t] pool device...
zpool online [-e] pool device...
```

## Exemple

```
zpool offline tank da1      # Met le disque da1 hors-ligne
zpool online tank da1      # Met le disque da1 en-ligne
zpool online -e tank da1    # Expansion demandée
```

# Gestion des disques

Remplacement

Un disque peut être remplacé car :

- ▶ il est défectueux
- ▶ on souhaite augmenter la capacité du `pool`

## Attention

Le disque doit avoir une capacité au moins égale à celui qu'il remplace  
⇒ Lors de la création partitionner le disque pour réduire la taille utilisable

## Syntaxe

```
zpool replace [-f] pool device [new_device]
```

## Exemple

```
zpool replace tank da1 da2 # Remplace le disque da1 par da2
```

# Information sur les *pool*

## Structure et statistiques

Les informations suivantes sont disponibles :

- ▶ synthèse et états des différents *pool* (`list`, `status`)
- ▶ statistiques d'entrée/sortie (`iostat`)

🔗 les propriétés fournissent des informations supplémentaires

## Syntaxe

```
zpool list [-Hpv] [-o property[,...]] [-T d|u] [pool]...  
           [interval [count]]  
zpool status [-vx] [-T d|u] [pool]... [interval [count]]  
zpool iostat [-T d|u] [-v] [pool]...
```

## Exemples

```
zpool list           # Information synthétique sur les pools  
zpool status tank   # Status et structure du pool tank  
zpool iostat -v     # Statistique des E/S par pool et vdev
```

# Propriétés

## Informé et configurer

Des propriétés sont associées au *pool*, elles permettent de :

- ▶ connaître les fonctionnalités disponibles (`feature@...`)
- ▶ connaître l'état du *pool*
- ▶ spécifier un comportement

## Syntaxe

```
zpool get [-Hp] [-o field[,...]] all | property[,...] pool...  
zpool set property=value pool
```

## Exemples

```
zpool get all tank   # Affiche toutes les propriétés  
zpool get guid tank  # Affiche la propriété 'guid'  
zpool set dedupditto=1000 tank # Change le seuil de blocs ..  
                               # .. supplémentaires en cas de deduplication
```

# Propriétés

Un petit aperçu des propriétés disponibles

Nom	Description
<code>guid</code>	identification unique
<code>health</code>	état du pool
<code>size</code>	espace total
<code>allocated</code>	espace utilisé
<code>freeing</code>	espace en cours de libération
<code>fragmentation</code>	pourcentage de fragmentation
<code>dedupratio</code>	ratio de blocs dédupliqués
<code>dedupditto</code>	seuil de création de copies supplémentaires
<code>bootfs</code>	dataset utilisé pour le boot (si supporté par l'OS)
<code>failmode</code>	action à effectuer en cas de <i>pool</i> inaccessible
<code>feature@...</code>	fonctionnalité supportée/activée

Quelques propriétés du *pool*

# Historique des commandes

Qui à fait quoi et quand

Chaque *pool* à son historique des commandes.

L'historique contient :

- ▶ date d'exécution
- ▶ commande exécutée
- ▶ utilisateur ayant exécuté la commande
- ▶ machine sur laquelle était importé le *pool*

## Syntaxe

```
zpool history [-il] [pool]...
```

## Exemple

```
zpool history tank
```

## 5 Présentation

## 6 ZPOOL

## 7 ZFS

- Volume
- Système de fichiers

# Datasets et marque-page

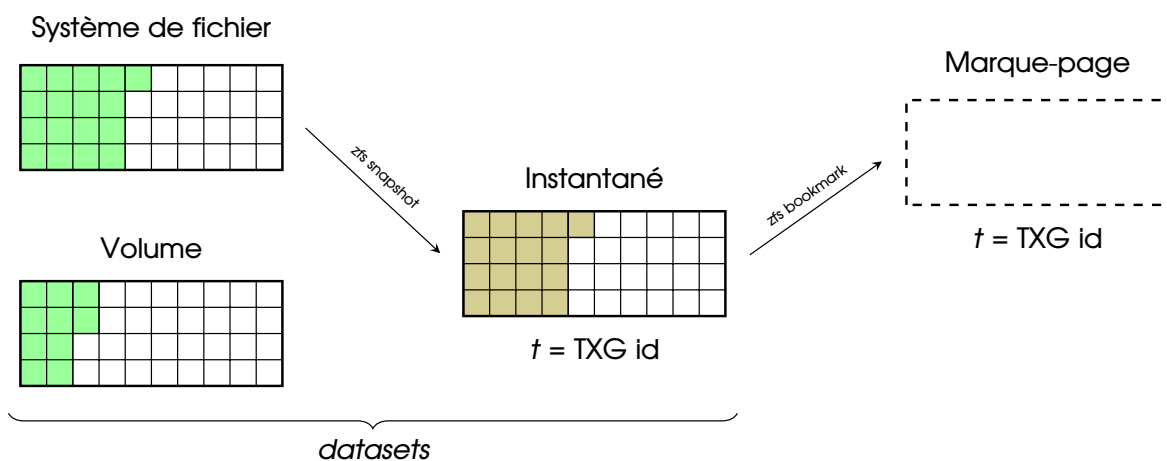
Système de fichiers, volume (et instantané)

**système de fichiers** structuration des données en répertoires/fichiers

**volume** périphérique de type block (disque)

**instantané** une « copie » dans un état immuable  
(d'un système de fichiers ou d'un volume)

**marque-page** instantané *sans* données



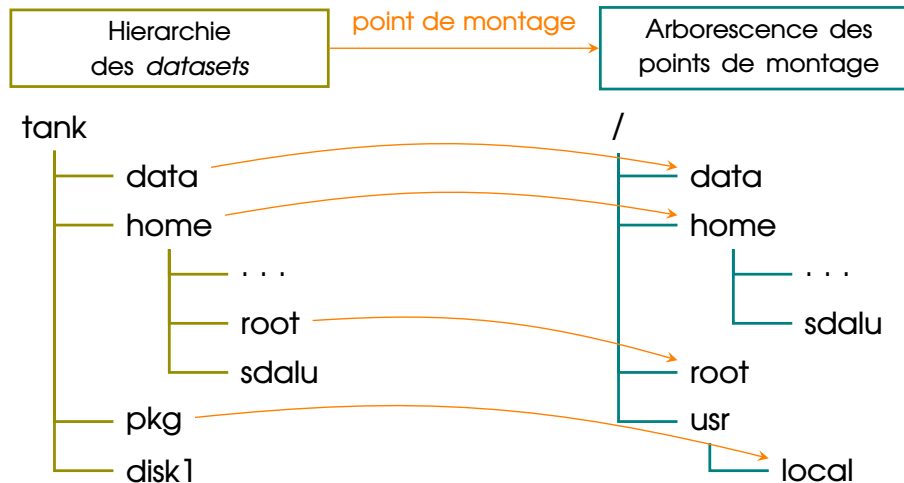


# Datasets

## Propriétés et hiérarchies

Les *datasets* sont :

- ▶ associés à des propriétés (clefs / valeurs)
- ▶ structurés de façon hiérarchique
  - ☞ différent de l'arborescence des points de montage



# Création

## Ligne de commande

### Syntaxe

```
zfs create [-pu] [-o property=value]... filesystem  
zfs create [-ps] [-b blocksize] [-o property=value]... -V size volume
```

☞ les propriétés suivantes doivent être positionnées à la création :

- ▶ volume : **volblocksize**
- ▶ système de fichiers : **casesensitivity**, **utf8only** et **normalization**

### Exemples

```
# Création de systèmes de fichiers  
zfs create tank/tmp  
zfs create -o casesensitivity=insensitive tank/samba  
zfs create -o utf8only=on -o normalization=formC tank/home  
  
# Création de volumes  
zfs create -V 5G tank/swap # volume de 5G, pré-alloué  
zfs create -s -b 4096 -V 20G tank/disk1 # sans allocation, bloc = 4096
```

# Destruction

... et protection

- ☞ poser un *checkpoint* sur le *pool* peut être intéressant avant de procéder à des destructions importantes ou massives de *datasets*
- ☞ si un instantané est verrouillé (*hold*) il ne peut être détruit immédiatement

## Syntaxe

```
zfs destroy [-fnpRrv] filesystem|volume
zfs destroy [-dnpRrv] snapshot[%snapname] [,...]
zfs destroy filesystem|volume#bookmark
```

## Exemple

```
# Destruction d'un dataset: tank/tmp
zfs destroy tank/tmp
# Destruction des instantanés et descendants entre 1980 et 2010
zfs destroy -r tank/home@1980%2010
# Destruction différée de tank/vm et de tous ceux qui dépendent de lui
zfs destroy -Rd tank/vm
```

# Propriétés

Information et configuration

- ▶ Propriétés natives
  - ▶ Accéder à de l'information sur les *datasets* (lecture-seul)
  - ▶ Configurer les *datasets*
  - ☞ varie suivant le type de *dataset*
  - ☞ s'appliquent sur les données futurs
- ▶ Propriétés utilisateurs
  - ▶ Un moyen d'annoter les *datasets*
  - ☞ format `module:property`
  - ☞ toujours hérités et accessibles en lecture/écriture

## Syntaxe

```
zfs set property=value [property=value]... filesystem|volume|snapshot...
zfs get [-r|-d depth] [-Hp] [-o all | field[,field]...]
[-t type[,type]...] [-s source[,source]...]
all | property[,property]... filesystem|volume|snapshot...
zfs inherit [-rS] property filesystem|volume|snapshot...
```

# Propriétés

## Sources

Les propriétés proviennent de différentes sources :

- local** positionée localement (ex : `zfs set`)
- default** valeur par défaut (ex : `zfs inherit`)
- temporary** positionée localement, non persistante (ex : options de montage)
- inherited** provient du *dataset* parent
- received** reçue lors d'un transfert (ex : `zfs inherit -S`, `zfs send -p`)
- none** état intrinsèque (lecture seule)

## Exemples: sources des propriétés

```
zfs set copies=2 tank/data           # local
zfs inherit copies tank/data         # inherited / default
zfs inherit -S copies tank/data      # received / inherited / default
zfs send -p ... | zfs recv tank/data # received
zfs mount -oro tank/data             # temporary
```

# Propriétés

## Exemples

### Exemples

```
# Mise en place d'une propriété
zfs set exec=off tank/data           # Interdit l'exécution
zfs set mountpoint=none tank/data    # Désactive le point de montage
zfs set compress=lz4 tank/data       # Active la compression
zfs set foo:bar=toto tank/data       # Annotation avec la clef foo:bar

# Retour à la valeur par "défaut" (reçue, héritée, défaut)
zfs inherit -S copies tank/data      # Nombre de copies, reçu par le dataset
zfs inherit mountpoint tank/data     # Le point de montage est hérité

# Connaître la valeur des propriétés
zfs get checksum tank/data           # Lit la propriété checksum
zfs get all tank/data                 # Liste toutes les propriétés
zfs get -Hp -t snapshot all          # Propriétés des instantanés (parsable)
zfs get -s local all                  # Propriétés modifiées localement
```

 Pour la liste complète : `man zfs`

# Somme de contrôle

Détecter les corruptions

Propriété	R/W	Rôle
<code>checksum</code>	✓	somme de contrôle ou valeur de hashage pseudo-random = <code>on</code>   <code>off</code>   <code>fletcher2</code>   <code>fletcher4</code>   <code>sha256</code>   <code>sha512</code>   <code>skein</code>

👉 Ne pas désactiver le *checksum*

👉 Le hashage de la déduplication est utilisé si activée

	Algorithme	Caractéristique
Somme de contrôle	<code>fletcher</code>	rapide
Hash pseudo-random	<code>sha</code> , <code>skein</code>	fixé à 256 bits dans <code>zfs</code>

## Exemple

```
zfs set checksum=skein tank/data
```

# Copies

Un peu de redondance

- ▶ permet d'effectuer de la redondance sur un *pool* de type « JBOD »
- ▶ distribue les copies sur des *vdev* différents si possible

Propriété	R/W	Rôle
<code>copies</code>	✓	nombre de copies = <code>1</code>   <code>2</code>   <code>3</code>

👉 Pas forcément utile si un miroir ou `raid-z` est déjà en place

## Exemple

```
zfs set copies=2 tank/precious
```

# Compression

Réduire l'espace consommé

Propriété	R/W	Rôle
<code>compression</code>	✓	algorithme de compression = on   off   lzjb   gzip   gzip- <i>N</i>   zle   lz4
<code>compressratio</code>		ratio de compression
<code>refcompressratio</code>		ratio de compression ( <i>dataset</i> )

👉 La compression n'est effective que pour les futures données

## Exemple

```
zfs set compression=lz4 tank/data # Active la compression
zfs get refcompressratio tank/data # Ratio de la compression
```

# Déduplication

Quand les données sont fortement identiques

- ▶ L'algorithme de hashage de la déduplication (`dedup`), l'emporte sur celui de la somme de contrôle (`checksum`)
- ▶ Les blocs dédupliés sont communs à tout le *pool*
- ▶ Possibilité d'ajouter de la redondance à la déduplication (`dedupditto`)

Propriété	R/W	Rôle
<code>dedup</code>	✓	algorithme de déduplication = on   off   verify   sha256[,verify]   sha512[,verify]   skein[,verify]
<code>dedupditto</code>	✓	1 déduplication pour <i>n</i> répétitions = <i>number</i>
<code>dedupratio</code>		ratio de déduplication

# Déduplication

Couteux en mémoire (métadonnées)

- ☞ La déduplication est une opération couteuse en mémoire  
⇒  $\simeq 400\text{ B}$  (`sizeof(ddt_entry_t)`) par bloc dédupliqué

## Exemple: Histogramme de déduplication

```
zdb -S tank # Simulation de l'histogramme
zdb -DD tank # Estimation de l'histogramme actuel
zpool status -D tank # Histogramme actuel
```

Simulated DDT histogram:

bucket	allocated				referenced			
	blocks	LSIZE	PSIZE	DSIZE	blocks	LSIZE	PSIZE	DSIZE
1	15.6M	186G	179G	200G	15.6M	186G	179G	200G
2	4.04M	23.8G	22.7G	27.5G	8.84M	52.1G	49.9G	60.1G
4	585K	2.55G	2.52G	2.90G	2.67M	11.6G	11.4G	13.3G
...	...	...	...	...	...	...	...	...
16K	1	4K	4K	4K	20.6K	82.2M	82.2M	82.2M
32K	1	16K	16K	16K	39.6K	633M	633M	633M
Total	20.3M	212G	205G	231G	28.7M	257G	248G	282G

dedup = 1.23, compress = 1.04, copies = 1.14, dedup \* compress / copies = 1.12

# Déduplication

Mise en place

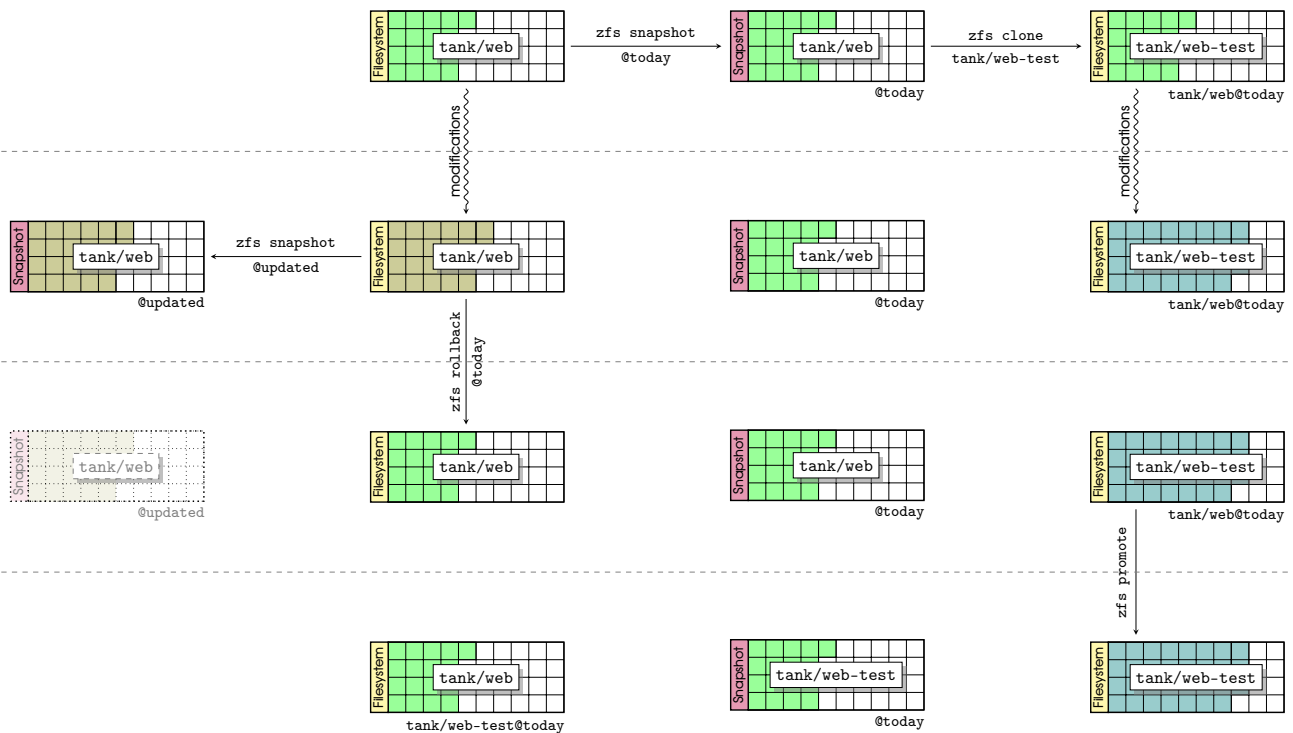
- ☞ Avoir suffisamment de mémoire pour des performances optimales  
⇒ empiriquement compter 5 GiB pour 1 TiB de disque
  - ☞ Tester que les données sont bien déduplicables
  - ☞ Utiliser `verify` en fonction de la probabilité de collision  
(ou de la confiance dans l'algorithme de hashage)
- ▶ Algorithme de hashage par défaut : `sha256`

## Exemple

```
zpool set dedupditto=1000 tank # redondance de déduplication
zfs set dedup=sha512 tank/vm # hashage seul (moitié de sha512)
zfs set dedup=sha256,verify tank/vm # hashage + vérification
```

# Instantanés, clones et manipulations

## Vue d'ensemble



# Instantanés

## Usage

Avoir une copie figée du système de fichiers :

- ▶ revenir en arrière (rollback)
- ▶ restaurer des fichiers
- ▶ créer une branche (clone)
- ▶ sauvegarder incrémentalement
- ▶ connaître les modifications (diff)

## Syntaxe

```
zfs snapshot [-r] [-o property=value]...  
filesystem@snapname | volume@snapname...
```

## Exemples

```
# Réalise un instantané  
zfs snapshot tank/data@2010-06-15  
# Réalise un instantané de façon récursive (sur tous les arguments)  
zfs snapshot -r tank/foo@today tank/toto@aujourd'hui
```

# Instantanés

## Accès et restauration de fichiers

Les instantanés sont accessibles (lecture-seule) :

- ▶ système de fichiers : `.zfs/snapshot/snapname/`
  - 👉 le répertoire `.zfs` se situe au niveau du point de montage
  - 👉 la visibilité du répertoire `.zfs` dépend de la propriété `snapdir`
- ▶ volume : `/dev/zvol/dataset@snapname`

## Exemple: restauration de fichiers

```
zfs create -o mountpoint=/data tank/data # Création du dataset
# .... # <Travail>
zfs snapshot tank/data@2010-06-15 # Réalisation d'un instantané
# .... # <Travail>
# .... # fichier modifié par erreur

# Restauration du fichier plan.txt
cd /data
cp .zfs/snapshot/2010-06-15/world-domination/plan.txt world-domination/
```

# Empêcher la destruction d'un instantané

## Pose de verroux

- ▶ Empêcher la destruction d'un instantané pendant une fenêtre temporelle
  - ⇒ permettre à un processus de s'exécuter correctement (malgré l'interaction d'autres scripts système)
  - 👉 effectuer une sauvegarde, des vérifications, ...
- 👉 Les commandes peuvent être appliquées récursivement
- 👉 Tous les verroux doivent être retirés pour supprimer l'instantané
- 👉 La suppression échoue si elle n'est pas différée
- 👉 **Renommer les instantanés reste possible**

## Syntaxe

```
zfs hold [-r] tag snapshot...
zfs holds [-r] snapshot...
zfs release [-r] tag snapshot...
zfs destroy [-dnpRrv] filesystem|volume@snapname [%snapname] ...
```



# Empêcher la destruction d'un instantané

Exemple d'utilisation

## Exemple

```
# Pose le verrou "backup"
zfs hold backup tank/web@today

# Informations sur les verrous posés
zfs holds tank/web@today # Liste les verrous
zfs get -Hp userrefs tank/web@today | cut -f 3 # Nombre de verrous

# Tentative de destruction infructueuse
# => cannot destroy snapshot tank/web@today: dataset is busy
zfs destroy tank/web@today

# Demande de destruction différée
zfs destroy -d tank/web@today

# Supprime le verrou "backup"
zfs release backup tank/web@today
```

# Revenir en arrière

rollback

- ▶ permet de revenir à un état antérieur (identifié par un instantané)
- ▶ la commande n'est pas récursive dans la hiérarchie des *datasets*  
⇒ parcourir les différents *datasets* à la main
- ▶ envisager l'utilisation d'un *checkpoint* sur le *pool* en cas d'erreur

## Syntaxe

```
zfs rollback [-rRf] snapshot
```

## Exemples

```
# Revient à l'état: @today
zfs rollback tank/web@today
# en détruisant: instantanés et marque-pages plus récents
zfs rollback -r tank/web@today
# en détruisant: instantanés, marque-pages et *clones* plus récents
zfs rollback -R tank/web@today
```

# Clones

Création d'une branche accessible en écriture

- ▶ dépendance implicite avec l'instantané d'origine
  - ⇒ création de la propriété `origin`
  - ☞ l'instantané d'origine ne peut être détruit tant que le clone existe
  - ☞ possibilité d'inversion de dépendance (`promote`)
- ▶ accessible en écriture
  - ☞ dépend de la propriété `readonly` (explicite ou héritée)

## Syntaxe

```
zfs clone [-p] [-o property=value]... snapshot filesystem|volume
```

## Exemples

```
# Création de l'instantané servant d'origine
zfs snapshot tank/web@today
# Création du clone (branche accessible en écriture)
zfs clone tank/web@today tank/web-test
```

# Promouvoir un clone

`promote`

- ▶ inverse la dépendance entre instantané d'origine et clone
- ▶ permet la destruction du *dataset* d'où le clone est origine

## Syntaxe

```
zfs promote clone-filesystem
```

## Exemple

```
# Création de l'instantané servant d'origine
zfs snapshot tank/web@today
# Création du clone (branche accessible en écriture)
zfs clone tank/web@today tank/web-test
# Promotion du clone
# => inversion de l'origine entre tank/web-test et tank/web
zfs promote tank/web-test # Promotion du clone
# Destruction du dataset ayant servi à créer le clone
zfs destroy tank/web
```

# Réorganiser, renommer

rename

- ▶ réorganiser la hierarchie des *datasets*
- ▶ corriger un nom
- 👉 les marque-pages ne peuvent être renommés (pour l'instant)

## Syntaxe

```
zfs rename [-f] filesystem|volume|snapshot filesystem|volume|snapshot  
zfs rename [-f] -p filesystem|volume filesystem|volume  
zfs rename -r snapshot snapshot  
zfs rename -u [-p] filesystem filesystem
```

## Exemples

```
# Renome un dataset (volume, système de fichiers ou instantané)  
zfs rename tank/foo tank/bar  
# Renome des instantanés de façon récursive  
zfs rename -r tank/foo@today tank/foo@yesterday  
# Créé les datasets parents si ils n'existent pas  
zfs rename -p tank/foo tank/a/b/c/d/foo
```

# Quota et réservation

Gestion des ressources

- quota** impose une limite sur l'espace disque utilisable
- reservation** garanti un espace disque minimum utilisable

Les quotas et réservations sont :

- ▶ non applicables sur les volumes (zvol)
- ▶ calculés en différé pour les utilisateurs / groupes
- ▶ mis en place par des propriétés

Applicable sur	Quota	Réservation
<i>dataset</i> et descendants	<code>quota</code>	<code>reservation</code>
<i>dataset</i>	<code>refquota</code>	<code>refreservation</code>
utilisateur et <i>dataset</i>	<code>userquota@...</code>	
groupe et <i>dataset</i>	<code>groupquota@...</code>	

Propriétés utilisées pour quota et réservation

# Quota et réservation

## Syntaxe et exemples

### Syntaxe

```
zfs set property=value [property=value]... filesystem...
zfs userspace [-Hinp] [-o field[,field]...] [-s field]... [-S field]...
               [-t type[,type]...] filesystem|snapshot
zfs groupspace [-Hinp] [-o field[,field]...] [-s field]... [-S field]...
               [-t type[,type]...] filesystem|snapshot
```

### Exemples

```
# Modifications de quotas et reservations
zfs set quota=1T tank/home           # Quota de 1To
zfs set refreservation=1G tank/home/sdalu # Ref reservation de 1Go
zfs set userquota@sdalu=500M tank/shared # Quota utilisateur de 500M
zfs set userquota@dalv=none tank/shared  # Suppr. quota utilisateur
zfs set groupquota@users=1G tank/shared  # Quota de groupe
# Liste les quotas utilisateurs/groupes pour un dataset donné
zfs userspace tank/shared
zfs groupspace tank/shared
```

# Espace disque

## Où sont utilisés les blocs ?

**used**  $\sum$  **usedby**

**logicalused** équivalent de la propriété **used** mais en ignorant les effets de la compression et des copies

**usedbydataset** espace utilisé par le *dataset* et excluant les données faisant partie de la *refreservation*

**usedbysnapshots** espace utilisé par les instantanés du *dataset*

**usedbychildren** espace utilisé par les descendants du *dataset*

**usedbyrefreservation** espace utilisé comptant dans la *refreservation*

**userused** espace utilisé par un utilisateur

**groupused** espace utilisé par un groupe

👉 **usedbysnapshots** n'est pas nécessairement la somme de l'espace utilisé par chaque instantané (espace partagé entre instantanés)

# Espace disque

Libérer de l'espace, est-ce possible ?

Libérer de l'espace... effacer un fichier ?

Pas si facile car :

- ▶ effacer un fichier peut consommer de l'espace (CoW)
- ▶ les blocs peuvent être partagés entre :
  - ▶ instantanés
  - ▶ clones
  - ▶ déduplication
- ▶ l'espace utilisé dans la réservation n'est pas restitué
- ▶ un *checkpoint* sur le *pool* retient les blocs
- ▶ la compression restitue moins d'espace que prévu

Mais plus d'espace que prévu peut être libéré :

- ▶ existence de copies multiples

# Optimisations

Cache, synchronisation, ...

Propriété	R/W	Rôle
<code>primarycache</code>	✓	ARC = <b>all</b>   none   metadata
<code>secondarycache</code>	✓	L2ARC = <b>all</b>   none   metadata
<code>redundant_metadata</code>	✓	copies supplémentaires des méta-données = <b>all</b>   most
<code>logbias</code>	✓	gestion des écritures synchrones : latence / débit = <b>latency</b>   throughput
<code>sync</code>	✓	écriture synchrone = <b>standard</b>   always   disabled

## Exemple

```
zfs set sync=disabled tank/tmp
```

Les commandes `send` / `recv` permettent de mettre en place :

- ▶ sauvegarde (totale ou incrémentale)
- ▶ copie/transfert de données
- ▶ répllication distante

Points intéressants :

- ▶ incrément depuis marque-page (*bookmark*) ou instantané (*snapshot*)
- ▶ unidirectionnel ( $\Rightarrow$  adapté aux scripts)
- ▶ estimation de la taille des données à envoyer
- ▶ rapport de progression
- ▶ possibilité de redémarrer un envoi interrompu

### Syntaxe

```
zfs send [-DLPRcenpv] [-i snapshot | -I snapshot] snapshot  
zfs send [-Lce] [-i snapshot|bookmark] filesystem|volume|snapshot  
zfs send [-Penv] -t receive_resume_token
```

### Exemples

```
# Génère un flux de répllication jusqu'à la snapshot @today  
# Le flux créé est dédupliqué (-D) et inclus les blocs  
# larges (-L), les données embarquées (-e), et n'effectue  
# pas de décompression (-c)  
zfs send -RDLe tank/data@today  
  
# Génère un flux depuis le bookmark #yesterday  
# jusqu'à la version actuelle (--head--)  
zfs send -i #yesterday main/data
```

- 👉 Le *pool* de réception doit supporter les fonctionnalités incluses dans le flux
- 👉 Destruction des instantanés intermédiaires non présents dans le flux

### Syntaxe

```
zfs recv [-vnsFu] [-o origin=snapshot] filesystem|volume|snapshot
zfs recv [-vnsFu] [-d | -e] [-o origin=snapshot] filesystem
zfs recv -A filesystem|volume
```

### Exemples

```
# Génère le nom du dataset à partir de celui transmis
# - en excluant le premier élément: a/b/c -> b/c
zfs recv -d tank
# - en ne conservant que le dernier élément: a/b/c -> c
zfs recv -e tank
# Réception avec rollback jusqu'à l'instantané de départ
# présent dans le flux (-F), et reprise possible sur erreur (-s)
zfs recv -s -F tank/foo
```

### Exemple: Transfert réseau et reprise sur erreur

```
# Envoi un flux incremental de @yesterday a @today, incluant
# les snapshots intermédiaires, sur la machine 'remote-host'
# En cas d'erreur de transmission les données sont conservées
zfs send -I @yesterday tank/data@today |
  ssh remote-host zfs recv -s pool/backup

# La connection réseau a été interrompue, récupère
# le jeton permettant de reprendre la transmission
token='zfs get -Hp receive_resume_token pool/backup | cut -f3'

# Poursuite de la transmission à partir des informations
# contenues dans le jeton
zfs send -t $token | ssh remote-host zfs recv -s pool/backup

# Ou suppression des données partiellement transférées
# si l'on ne souhaite pas poursuivre
zfs recv -A car/backup
```

# Partager

iSCSI, NFS, Samba

Propriété	R/W	Type	Rôle
shareiscsi	✓	volume	partage de via iSCSI = on   <b>off</b>   <u>opts</u>
sharenfs	✓	fichiers	partage via NFS = on   <b>off</b>   <u>opts</u>
sharesmb	✓	fichiers	partage via Samba = on   <b>off</b>   <u>opts</u>

## Syntaxe

```
zfs share -a | filesystem  
zfs unshare -a | filesystem|mountpoint
```

## Exemple

```
zfs set sharenfs=192.168.1.2 main/home # Partage NFS avec options
```

# Volumes

Créer un disque...

Propriété	R/W	Rôle
volsize	✓	taille du volume (multiple de <b>volblocksize</b> ) = <u>size</u>
volmode	✓	interprétation du volume par l'OS = <b>default</b>   geom   dev   none
volblocksize	✓	taille du bloc disque
readonly	✓	lecture seule = <b>on</b>   off

- 👉 **volblocksize** est l'équivalent de **recordsize** pour le système de fichiers
- 👉 pas de pré-allocation dans le cas d'un volume partiel (option -s)

## Exemples

```
zfs create -b 4096 -V 2G tank/disk1 # Volume de 2Go, bloc disque: 4096  
zfs create -s -V 5G tank/disk1 # Volume partiel de 5Go  
zfs set volsize=2T tank/disk1 # Augmente la taille à 2To
```



# Système de fichiers

## Création

Propriété	R/W	Rôle
<code>recordsize</code>	✓	taille maximum du bloc

- 👉 `recordsize` est la taille maximum des blocs composant le fichier  
⇒ si le fichier est plus petit un bloc de taille inférieure est utilisé

## Exemples

```
# Création du système de fichiers
zfs create -o casesensitivity=sensitive # insensible à la casse
           -o mountpoint=/data         # montage sur /data
           -o recordsize=64k           # taille max des blocs: 64k
           main/data                   # dataset: main/data

# Modification des propriétés
zfs set exec=off tank/data            # pas d'exécution de fichier
zfs set setuid=off tank/data          # ignore le bit set-UID
```

# Système de fichiers

## Options de montage

Propriété	R/W	Rôle
<code>canmount</code>	✓	autorise le montage (👉 non héritée) = <code>on</code>   <code>off</code>   <code>noauto</code>
<code>mountpoint</code>	✓	point de montage = <u><code>path</code></u>   <code>none</code>   <code>legacy</code>
<code>readonly</code>	✓	lecture seule = <code>on</code>   <code>off</code>
<code>atime</code>	✓	mise à jour de l'heure d'accès = <code>on</code>   <code>off</code>
<code>exec</code>	✓	autorise l'exécution de programme = <code>on</code>   <code>off</code>
<code>setuid</code>	✓	respect du bit set-UID = <code>on</code>   <code>off</code>

# Système de fichiers

## Nom de fichiers et ACLs

Propriété	R/W	Rôle
<code>utf8only</code>	✓	rejette les noms de fichier comportant des caractères non présent dans UTF-8 = <code>on</code>   <code>off</code>
<code>normalization</code>	✓	normalise le nom de fichier (UTF-8) (C = composition   D = décomposition   K = compatibilité) = <code>none</code>   <code>formC</code>   <code>formD</code>   <code>formKC</code>   <code>formKD</code>
<code>casesensitivity</code>	✓	considère le nom de fichier différent si la casse diffère ☞ la casse est préservée dans tous les cas = <code>sensitive</code>   <code>insensitive</code>   <code>mixed</code>
<code>aclmode</code>	✓	comportement durant un <code>chmod(2)</code> = <code>discard</code>   <code>groupmask</code>   <code>passthrough</code>   <code>restricted</code>
<code>aclinherit</code>	✓	héritage des ACLs = <code>discard</code>   <code>noallow</code>   <code>restricted</code>   <code>passthrough</code>

# Comparaison de système de fichiers

Savoir si c'était mieux avant...

- Connaître les différences entre instantanés / systèmes de fichiers

## Syntaxe

```
zfs diff [-Fht] snapshot [snapshot|filesystem]
```

☞  Supression,  Ajout,  Modification,  Renommage

## Exemple

```
# Même dataset
zfs diff @yesterday tank/data          # Comparaison avec @yesterday
zfs diff -Ft @2010 tank/data@2011     # + type de fichier (F) et date (t)

# Ancêtre commun: web-test est un clone de web
zfs diff tank/web@2010 tank/web-test
```

## Acronymes (1)

- ACL** Access Control List.
- ARC** Adaptive Replacement Cache.
- ASCII** American Standard Code for Information Interchange.
- ATA** AT Attachment.
  
- BIOS** Basic Input/Output System.
- Btrfs** Btrfs.
  
- CDDL** Common Development and Distribution License.
- CHS** Cylinder-Head-Sector.
- CJK** Chinese Japanese Korean.
- CoW** Copy-on-Write.
- CRC** Cyclic Redundancy Check.
  
- EASCII** Extended ASCII.

## Acronymes (2)

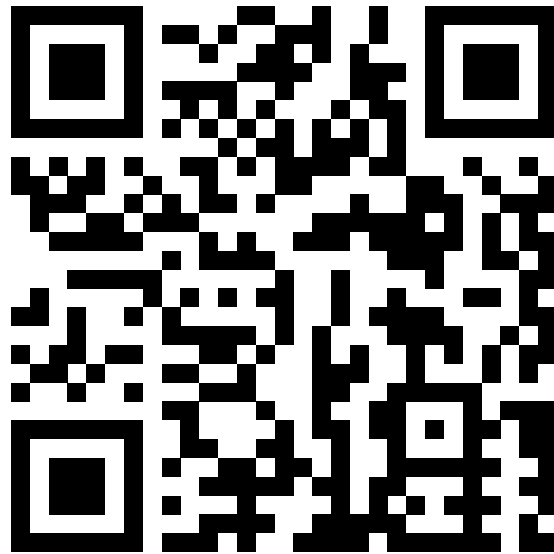
- FUSE** Filesystem in Userspace.
  
- GPL** General Public License.
- GPT** GUID Partition Table.
- GUID** Globally Unique Identifier.
  
- HMAC** Hash-based Message Authentication Code.
  
- IDE** Integrated Drive Electronics.
  
- JBOD** Just a Bunch of Disks.
  
- L2ARC** Level 2 Adaptive Replacement Cache.
- LBA** Logical Block Address.
- LRU** Least Recently Used.
- LVM** Logical Volume Manager.

## Acronymes (3)

- MBR** Master Boot Record.
- NAND** NOT-AND.
- NAS** Network-attached storage.
- NFC** Normalization Form Canonical Composition.
- NFD** Normalization Form Canonical Decomposition.
- NFKC** Normalization Form Compatibility Composition.
- NFKD** Normalization Form Compatibility Decomposition.
- NVMe** Non-Volatile Memory express.
- PMBR** Protective MBR.
- PMR** Perpendicular Magnetic Recording.
- RAID** Redundant Array of Independent Disks.
- SAS** Serial Attached SCSI.

## Acronymes (4)

- SATA** Serial Advanced Technology Attachment.
- SCSI** Small Computer System Interface.
- SMR** Shingled Magnetic Recording.
- SSD** Solid-State Disk.
- TPM** Trusted Platform Module.
- TXG** transaction group.
- UEFI** Unified Extensible Firmware Interface.
- WAFL** Write Anywhere File Layout.
- XIP** eXecute In Place.
- ZFS** ZFS.
- ZIL** ZFS Intent Log.



<http://www.sdalu.com/training/zfs/>