

4. ZFS (ZFS Datasets)

1. 在 /etc/fstab 文件中添加以下行，以配置 NFS 挂载：


```

# 挂载点 /home 使用 NFS 存储
/home nfs rw,hard,intr,noauto 192.168.1.100:/home 0 0
# 挂载点 /usr/local/var 使用 NFS 存储
/usr/local/var nfs rw,hard,intr,noauto 192.168.1.100:/usr/local/var 0 0

```

[illegible]















```
# 创建目录并设置权限
mkdir -p /var/lib/docker/images
chown root:root /var/lib/docker/images
chmod 700 /var/lib/docker/images















# 创建目录并设置权限
mkdir -p /home/ubuntu/.docker
chown ubuntu:ubuntu /home/ubuntu/.docker
chmod 700 /home/ubuntu/.docker

# 挂载UFS文件系统，使用tunefs(8)命令
mount -t UFS -o tunefs(8) /dev/disk1 /var/lib/docker/images
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ . ZFS □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ .

????? (Datasets)

[illegible]

zfs(8)

.

????? ?? (Dataset Types)

如何 实现 ?

我们 在 这个 项目 中 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 . 我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

如何 实现 (Viewing Datasets)

zfs list 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .

```
$ zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
mypool                              420M  17.9G   96K    none
mypool/R00T                         418M  17.9G   96K    none
mypool/R00T/default                 418M  17.9G  418M    /
...
```

我们 使用 了 很多 不同的 技术 , 但 我们 在 这个 项目 中 使用 了 很多 不同的 技术 .


```
$ zfs create mypool/lamb/baby
```

이제 우리는 'lamb' / 'baby' 데이터셋을 생성했습니다. , 데이터셋은 'lamb' 데이터셋의 하위 데이터셋입니다.

?? ??? (Creating Volumes)

-V 옵션은 데이터셋을 생성할 때 함께 생성할 수 있는 ZFS 볼륨을 지정합니다. **zfs create** 명령을 사용하여 볼륨을 생성합니다. 이 명령은 다음과 같습니다.

```
$ zfs create -V 4G mypool/avolume
```

Zvols은 ZFS 데이터셋에 연결된 전용 블록 장치입니다. **-t volume** 옵션을 사용하여 볼륨을 생성합니다. **zfs list** 명령을 사용하여 볼륨을 확인합니다.

```
$ zfs list mypool/avolume
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
mypool/avolume	4.13G	17.9G	64K	-

ZFS 데이터셋은 ZFS 볼륨을 생성할 때 함께 생성할 수 있는 ZFS 데이터셋입니다. 이 4GB zvol은 4.13GB의 데이터를 저장할 수 있습니다.

이제 zvol을 사용하여 데이터셋을 생성합니다. **/dev/zvol**은 zvol의 디바이스 경로입니다. 이 명령은 다음과 같습니다.

```
$ ls -al /dev/zvol/mypool/avolume
```

```
crw-r----- 1 root operator 0x4d Mar 27 20:22 /dev/zvol/mypool/avolume
```

이제 **newfs(8)** 명령을 사용하여 zvol에 파일 시스템을 생성합니다. 이 명령은 다음과 같습니다.

????? ?? ?? (Renaming Datasets)

zfs rename 명령을 사용하여 데이터셋을 재명명합니다. 이 명령은 다음과 같습니다.

```
$ zfs rename db/production db/old
```

```
$ zfs rename db/testing db/production
```

이제 **-f** 옵션을 사용하여 데이터셋을 재명명합니다. 이 명령은 다음과 같습니다.

이제 우리는...

NAME	PROPERTY	VALUE	SOURCE
mypool/lamb	compression	lz4	inherited from mypool

SOURCE 00 0 000000 . 00 000 0 000 ZFS 000000 0000 000 000000 . 00
000 0000 0 000000 0 000 00000 00000 00000 . 00 000 000000
0000 0 000000 , 000000 000 0000 0 000 0000 000 000000 . 000
000 0 00 0000 00 "00 /00 00 "00 00000 00 00 0000000 00000 .

zfs get

NAME	PROPERTY	VALUE	SOURCE
mypool/lamb	type	filesystem	-
mypool/lamb	creation	Fri Mar 27 20:05 2015	-
mypool/lamb	used	192K	-
...			

```
$ zfs get quota,reservation zroot/home
```

NAME	PROPERTY	VALUE	SOURCE
zroot/home	quota	none	local
zroot/home	reservation	none	default

NAME	QUOTA	RESERV
db	none	none
zroot	none	none
zroot/R00T	none	none
zroot/R00T/default	none	none

```
...
zroot/var/log          100G      20G
...
```

zfs 的 属性 分为 本地 属性 和 全局 属性 两种 类型。

?? ?? (Changing Properties)

zfs set 命令 用于 设置 属性。 本地 属性 和 全局 属性 都 可以 设置。 例如， 设置 本地 属性 **compression** 为 **off**。

```
$ zfs set compression=off mypool/lamb/baby
```

zfs get 命令 用于 获取 属性。

```
$ zfs get compression mypool/lamb/baby

NAME                PROPERTY          VALUE          SOURCE
mypool/lamb/baby    compression       off            local
```

zfs 的 属性 分为 本地 属性 和 全局 属性 两种 类型。 本地 属性 是指 只 对 特定 的 zfs 实例 生效 的属性， 例如 **compression**。 全局 属性 是指 对 所有 的 zfs 实例 生效 的属性， 例如 **mountpoint**。 本地 属性 的 名称 通常 以 小写字母 开头， 而 全局 属性 的 名称 通常 以 大写字母 开头。 例如， 设置 本地 属性 **compression** 为 **off**， 使用 **zfs set compression=off mypool/lamb/baby**。 设置 全局 属性 **mountpoint** 为 **/mnt/zfs**， 使用 **zfs set mountpoint=/mnt/zfs mypool**。

?? ?? ?? (Read-Only Properties)

ZFS 的 属性 分为 本地 属性 和 全局 属性 两种 类型。 本地 属性 是指 只 对 特定 的 zfs 实例 生效 的属性， 例如 **compression**。 全局 属性 是指 对 所有 的 zfs 实例 生效 的属性， 例如 **mountpoint**。 本地 属性 的 名称 通常 以 小写字母 开头， 而 全局 属性 的 名称 通常 以 大写字母 开头。 例如， 设置 本地 属性 **compression** 为 **off**， 使用 **zfs set compression=off mypool/lamb/baby**。 设置 全局 属性 **mountpoint** 为 **/mnt/zfs**， 使用 **zfs set mountpoint=/mnt/zfs mypool**。

?? ??? ? (Filesystem Properties)

zfs 的 属性 分为 本地 属性 和 全局 属性 两种 类型。 本地 属性 是指 只 对 特定 的 zfs 实例 生效 的属性， 例如 **compression**。 全局 属性 是指 对 所有 的 zfs 实例 生效 的属性， 例如 **mountpoint**。 本地 属性 的 名称 通常 以 小写字母 开头， 而 全局 属性 的 名称 通常 以 大写字母 开头。 例如， 设置 本地 属性 **compression** 为 **off**， 使用 **zfs set compression=off mypool/lamb/baby**。 设置 全局 属性 **mountpoint** 为 **/mnt/zfs**， 使用 **zfs set mountpoint=/mnt/zfs mypool**。

atime

zfs 的 属性 分为 本地 属性 和 全局 属性 两种 类型。 本地 属性 是指 只 对 特定 的 zfs 实例 生效 的属性， 例如 **compression**。 全局 属性 是指 对 所有 的 zfs 实例 生效 的属性， 例如 **mountpoint**。 本地 属性 的 名称 通常 以 小写字母 开头， 而 全局 属性 的 名称 通常 以 大写字母 开头。 例如， 设置 本地 属性 **compression** 为 **off**， 使用 **zfs set compression=off mypool/lamb/baby**。 设置 全局 属性 **mountpoint** 为 **/mnt/zfs**， 使用 **zfs set mountpoint=/mnt/zfs mypool**。

我们使用 `com.allanjude` 命名空间，并设置 `backup_ignore` 属性。

我们使用 `ZFS` 的 `zfs` 命令来设置属性。

```

$ zfs set com.allanjude:backup_ignore=on mypool/lamb
```

Jude 使用 `zfs` 命令来设置属性。

父/子关系 (Parent/Child Relationships)

我们使用 `zfs` 命令来查看属性。

```

$ zfs get -r compression mypool/lamb
```

NAME	PROPERTY	VALUE	SOURCE
mypool/lamb	compression	lz4	inherited from mypool
mypool/lamb/baby	compression	off	local

我们使用 `zfs` 命令来查看属性。

zfs inherit 命令用于继承属性。

```

$ zfs inherit compression mypool/lamb/baby
$ zfs get -r compression mypool/lamb
```

NAME	PROPERTY	VALUE	SOURCE
mypool/lamb	compression	lz4	inherited from mypool
mypool/lamb/baby	compression	lz4	inherited from mypool

我们使用 `zfs` 命令来设置属性。

我们使用 `zfs` 命令来查看属性。

```

$ zfs set compression=gzip-9 mypool/lamb
$ zfs get -r compression mypool/lamb
```

NAME	PROPERTY	VALUE	SOURCE
mypool/lamb	compression	gzip-9	local
mypool/lamb/baby	compression	gzip-9	inherited from mypool/lamb

1. 我们使用 `gzip-9` 属性来压缩文件。

2. 继承和重命名 (Inheritance and Renaming)

我们使用 `zfs create` 命令来创建新的文件系统。

我们使用 `zfs get` 命令来查看文件系统的属性。

```

$ zfs create mypool/second
$ zfs get compress mypool/second
NAME          PROPERTY      VALUE  SOURCE
mypool/second  compression   lz4    inherited from mypool
```

我们使用 `zfs rename` 命令来重命名文件系统。

```

$ zfs rename mypool/lamb/baby mypool/second/baby
$ zfs get -r compression mypool/second
NAME          PROPERTY      VALUE  SOURCE
mypool/second  compression   lz4    inherited from mypool
mypool/second/baby  compression   lz4    inherited from mypool
```

我们使用 `zfs inherit` 命令来继承属性。

我们使用 `zfs set` 命令来设置属性。

3. 移除属性 (Removing Properties)

我们使用 `zfs unset` 命令来移除属性。

我们使用 `zfs inherit` 命令来继承属性。

```

$ zfs inherit com.allanjude:backup_ignore mypool/lamb
```

我们使用 `zfs get` 命令来查看文件系统的属性。

zfs inherit 命令 用于 设置 文件系统 的 属性 继承 策略。 默认 情况下， 子 文件系统 会 继承 父 文件系统的 属性。

```
$ zfs inherit -r compression mypool
```

zfs inherit 命令 还可以 用于 设置 文件系统 的 其他 属性， 例如 压缩 策略。

ZFS 挂载与卸载 (Mounting ZFS Filesystems)

在 FreeBSD 中， ZFS 文件系统的 挂载 和 卸载 操作 通常 通过 编辑 `/etc/fstab` 文件 来实现。 该文件 用于 配置 系统 的 启动 时 自动 挂载 的 文件系统。

要 挂载 ZFS 文件系统， 需要 指定 挂载点 (mountpoint)。 挂载点 是指 文件系统 在 系统 中的 位置。 例如， 如果 要 挂载 名为 `zroot/usr/home` 的 文件系统， 则 需要 指定 挂载点 为 `/usr/home`。

```
$ zfs get mountpoint zroot/usr/home
```

NAME	PROPERTY	VALUE	SOURCE
zroot/usr/home	mountpoint	/usr/home	inherited from zroot/usr

在 `/etc/fstab` 文件中， 需要 指定 文件系统 的 名称、 挂载点、 文件系统 类型、 挂载选项 和 挂载频率。 例如， 以下 是 挂载 `zroot/usr/home` 文件系统的 配置 示例：

在 FreeBSD 中， ZFS 文件系统的 挂载 和 卸载 操作 通常 通过 编辑 `/etc/fstab` 文件 来实现。 该文件 用于 配置 系统 的 启动 时 自动 挂载 的 文件系统。

要 挂载 ZFS 文件系统， 需要 指定 挂载点 (mountpoint)。 挂载点 是指 文件系统 在 系统 中的 位置。 例如， 如果 要 挂载 名为 `zroot/usr/home` 的 文件系统， 则 需要 指定 挂载点 为 `/usr/home`。

在 `/etc/fstab` 文件中， 需要 指定 文件系统 的 名称、 挂载点、 文件系统 类型、 挂载选项 和 挂载频率。 例如， 以下 是 挂载 `zroot/usr/home` 文件系统的 配置 示例：

在 FreeBSD 中， ZFS 文件系统的 挂载 和 卸载 操作 通常 通过 编辑 `/etc/fstab` 文件 来实现。 该文件 用于 配置 系统 的 启动 时 自动 挂载 的 文件系统。

要 挂载 ZFS 文件系统， 需要 指定 挂载点 (mountpoint)。 挂载点 是指 文件系统 在 系统 中的 位置。 例如， 如果 要 挂载 名为 `zroot/usr/home` 的 文件系统， 则 需要 指定 挂载点 为 `/usr/home`。

canmount off 表示 数据 集 可以 被 挂载 到 任何 目录 下 。 如果 数据 集 被 挂载 到 一个 目录 下 ， 那么 该 目录 下 的 所有 文件 和 目录 都 可以 被 访问 。

如果 数据 集 被 挂载 到 一个 目录 下 ， 那么 该 目录 下 的 所有 文件 和 目录 都 可以 被 访问 。

canmount 表示 数据 集 可以 被 挂载 到 任何 目录 下 。 如果 数据 集 被 挂载 到 一个 目录 下 ， 那么 该 目录 下 的 所有 文件 和 目录 都 可以 被 访问 。

??? ???? ?? ????? (Datasets without Mount Points)

ZFS 数据 集 可以 被 挂载 到 任何 目录 下 。 如果 数据 集 被 挂载 到 一个 目录 下 ， 那么 该 目录 下 的 所有 文件 和 目录 都 可以 被 访问 。

FreeBSD 10.1 数据 集 可以 被 挂载 到 任何 目录 下 。 如果 数据 集 被 挂载 到 一个 目录 下 ， 那么 该 目录 下 的 所有 文件 和 目录 都 可以 被 访问 。

```
$ zfs mount
zroot/R00T/default /
zroot/tmp /tmp
zroot/usr/home /usr/home
zroot/usr/ports /usr/ports z
root/usr/src /usr/src
...
```

/usr 数据 集 可以 被 挂载 到 任何 目录 下 。 如果 数据 集 被 挂载 到 一个 目录 下 ， 那么 该 目录 下 的 所有 文件 和 目录 都 可以 被 访问 。

zfs list 表示 数据 集 可以 被 挂载 到 任何 目录 下 。 如果 数据 集 被 挂载 到 一个 目录 下 ， 那么 该 目录 下 的 所有 文件 和 目录 都 可以 被 访问 。

```
$ zfs list -o name,canmount,mountpoint -r zroot/usr
NAME          CANMOUNT  MOUNTPOINT
zroot/usr      off       /usr
zroot/usr/home on        /usr/home
zroot/usr/ports on        /usr/ports
zroot/usr/src  on        /usr/src
```

canmount off 表示 数据 集 可以 被 挂载 到 任何 目录 下 。 如果 数据 集 被 挂载 到 一个 目录 下 ， 那么 该 目录 下 的 所有 文件 和 目录 都 可以 被 访问 。

如果 数据 集 被 挂载 到 一个 目录 下 ， 那么 该 目录 下 的 所有 文件 和 目录 都 可以 被 访问 。

??? ??? ???? ?? ?? ????? (Multiple Datasets with the Same Mount Point)

canmount off 表示 数据 集 可以 被 挂载 到 任何 目录 下 。 如果 数据 集 被 挂载 到 一个 目录 下 ， 那么 该 目录 下 的 所有 文件 和 目录 都 可以 被 访问 。

canmount off 表示 数据 集 可以 被 挂载 到 任何 目录 下 。 如果 数据 集 被 挂载 到 一个 目录 下 ， 那么 该 目录 下 的 所有 文件 和 目录 都 可以 被 访问 。

FreeBSD 的 10 个 分区 方案 中 的 一个 方案 是 将 分区 方案 设计 为 以下 方案 。

FreeBSD 的 10 个 分区 方案 中 的 一个 方案 是 将 分区 方案 设计 为 以下 方案 。

在 10 个 分区 方案 中 的 一个 方案 是 将 分区 方案 设计 为 以下 方案 。

在 10 个 分区 方案 中 的 一个 方案 是 将 分区 方案 设计 为 以下 方案 。

```
$ zfs create db/programs # zfs create db/data
```

在 10 个 分区 方案 中 的 一个 方案 是 将 分区 方案 设计 为 以下 方案 。

```
$ zfs set canmount=off db/programs
$ zfs set mountpoint=/opt db/programs
```

在 10 个 分区 方案 中 的 一个 方案 是 将 分区 方案 设计 为 以下 方案 。

```
$ zfs set readonly=on db/programs
```

在 10 个 分区 方案 中 的 一个 方案 是 将 分区 方案 设计 为 以下 方案 。

```
$ zfs set canmount=off db/data
$ zfs set mountpoint=/opt db/data
$ zfs set setuid=off db/data
$ zfs set exec=off db/data
```

在 10 个 分区 方案 中 的 一个 方案 是 将 分区 方案 设计 为 以下 方案 。

```
$ zfs create db/programs/bin
$ zfs create db/programs/sbin
$ zfs create db/data/test
$ zfs create db/data/production
```

在 10 个 分区 方案 中 的 一个 方案 是 将 分区 方案 设计 为 以下 方案 。

When `setuid` is used, the process runs with the permissions of the user that created the pool. This is not always what you want. For example, if you create a pool as root, but then want to use it as a regular user, you may want to change the permissions. This can be done with the `zfs set` command.

How to create pools without mount points (Pools without Mount Points)

By default, ZFS pools are created with a mount point. However, you can create a pool without a mount point. This is useful for pools that are used for storage, but not for mounting.

```
$ zfs set mountpoint=none mypool
```

When you create a pool without a mount point, you can still use the pool for storage. However, you will need to manually mount and unmount the pool. This is done with the `zfs mount` and `zfs unmount` commands.

```
$ zfs set mountpoint=/someplace mypool/lamb
```

When you create a pool with a mount point, you can use the pool for storage and mounting. However, you will need to manually mount and unmount the pool.

How to manually mount and unmount filesystems (Manually Mounting and Unmounting Filesystems)

When you create a pool with a mount point, you can use the pool for storage and mounting. However, you will need to manually mount and unmount the pool. This is done with the `zfs mount` and `zfs unmount` commands.

```
$ zfs mount mypool/usr/src
```

When you create a pool with a mount point, you can use the pool for storage and mounting. However, you will need to manually mount and unmount the pool. This is done with the `zfs mount` and `zfs unmount` commands.

```
$ zfs unmount mypool/second
```

When you create a pool with a mount point, you can use the pool for storage and mounting. However, you will need to manually mount and unmount the pool. This is done with the `zfs mount` and `zfs unmount` commands.

```
$ zfs mount -o mountpoint=/mnt mypool/lamb
```

When you create a pool with a mount point, you can use the pool for storage and mounting. However, you will need to manually mount and unmount the pool. This is done with the `zfs mount` and `zfs unmount` commands.

How to use ZFS and /etc/fstab (ZFS and /etc/fstab)

When you create a pool with a mount point, you can use the pool for storage and mounting. However, you will need to manually mount and unmount the pool. This is done with the `zfs mount` and `zfs unmount` commands.

```
$ zfs set mountpoint=legacy mypool/second
```

When you create a pool with a mount point, you can use the pool for storage and mounting. However, you will need to manually mount and unmount the pool. This is done with the `zfs mount` and `zfs unmount` commands.

```
$ mount -t zfs mypool/second /tmp/second
```

將 `/etc/fstab` 的 ZFS 內容複製到新的檔案中。將內容中的 `noatime, noexec, readonly` 改為 `ro, nosuid` 並加入 `fsck` 選項。(將 `atime, exec, rw, suid` 改為 `noatime, noexec, readonly, nosuid`)。將 `scratch/junk` 加入 `/etc/fstab` 中。

```
scratch/junk /tmp nosuid 2 0
```

將 `scratch/junk` 加入 `/etc/fstab` 中。

ZFS ?? ?? (Tweaking ZFS Volumes)

Zvol 的 `volsize` 選項是用來指定 Zvol 的總大小。Zvol 的 `volblocksize` 選項是用來指定 Zvol 的區塊大小。

?? ?? (Space Reservations)

zvol 的 **volsize** 選項是用來指定 Zvol 的總大小。Zvol 的 **volblocksize** 選項是用來指定 Zvol 的區塊大小。Zvol 的 **volmode** 選項是用來指定 Zvol 的模式。Zvol 的 **volmode** 選項可以用來指定 Zvol 的模式為 `geom(4)` 或 `volmode`。

將 `volmode` 改為 `geom(4)`。Zvol 的 `volmode` 選項可以用來指定 Zvol 的模式為 `geom(4)` 或 `volmode`。

Zvol 的 `volmode` 選項可以用來指定 Zvol 的模式為 `geom(4)` 或 `volmode`。Zvol 的 `volmode` 選項可以用來指定 Zvol 的模式為 `geom(4)` 或 `volmode`。

將 `volmode` 改為 `geom(4)`。Zvol 的 `volmode` 選項可以用來指定 Zvol 的模式為 `geom(4)` 或 `volmode`。

zfs create -V 選項是用來指定 Zvol 的總大小。Zvol 的 **-s** 選項是用來指定 Zvol 的區塊大小。

Zvol ?? (Zvol Mode)

FreeBSD 的 `geom(4)` 模式是用來指定 Zvol 的模式。Zvol 的 **geom(4)** 模式是用來指定 Zvol 的模式。

通过 **volmode** 指定 `dev` 设备名称，如 `/dev/sda`。通过 `geom` 指定 GEOM 设备名称，如 `geom`。通过 `dev` 指定设备名称，如 `dev`。通过 `none` 指定无设备。

volmode 指定 `none` 表示无设备。ZFS 通过 `volmode` 指定设备名称，如 `geom`，通过 `dev` 指定设备名称，如 `dev`。

volmode 指定 `default` 表示默认设备。通过 `sysctl vfs.zfs.vol.mode` 指定设备名称。通过 `zvol` 指定设备名称，如 `1`，通过 `geom`，通过 `2`，通过 `dev`，通过 `3`，通过 `none`。

通过 `zfs rename` 指定设备名称，如 `zvol`。通过 `zfs rename` 指定设备名称，如 `zvol`。

Dataset Integrity

ZFS 通过 `VDEV` 指定设备名称，如 `geom`。通过 `RAID-Z` 指定设备名称，如 `geom`。通过 `RAID-Z` 指定设备名称，如 `geom`。通过 `RAID-Z` 指定设备名称，如 `geom`。

Checksums

ZFS 通过 `on` 指定设备名称，如 `OpenZFS`。通过 `on` 指定设备名称，如 `OpenZFS`。通过 `on` 指定设备名称，如 `OpenZFS`。

通过 `on` 指定设备名称，如 `OpenZFS`。通过 `on` 指定设备名称，如 `OpenZFS`。通过 `on` 指定设备名称，如 `OpenZFS`。

通过 `fletcher4` 指定设备名称，如 `fletcher4`。通过 `fletcher4` 指定设备名称，如 `fletcher4`。通过 `fletcher4` 指定设备名称，如 `fletcher4`。

`off` 指定设备名称，如 `off`。通过 `off` 指定设备名称，如 `off`。

`noparity` 指定设备名称，如 `noparity`。通过 `noparity` 指定设备名称，如 `noparity`。通过 `noparity` 指定设备名称，如 `noparity`。

通过 `ZFS` 指定设备名称，如 `fletcher2`。通过 `ZFS` 指定设备名称，如 `fletcher2`。通过 `ZFS` 指定设备名称，如 `fletcher2`。

去重 (deduplication) 使用 sha256 校验 数据完整性。

??? (Copies)

ZFS 通过 `copies` 属性指定数据副本数量。默认情况下，`copies` 为 3，表示每个块都有 3 个副本。可以通过 `zfs set copies=2 mypool/lamb` 将副本数设置为 2。

使用 `dd` 命令生成数据并写入 ZFS 池。例如：
`dd if=/dev/random of=/lamb/random1 bs=1m count=10`

验证副本数量：
`zfs get copies mypool/lamb`

```
$ dd if=/dev/random of=/lamb/random1 bs=1m count=10
10+0 records in
10+0 records out
10485760 bytes transferred in 0.144787 secs (72421935 bytes/sec)
$ zfs set copies=2 mypool/lamb
```

使用 `zfs list` 查看 ZFS 池的详细信息，包括副本数。

```
$ zfs list mypool/lamb
NAME          USED  AVAIL  REFER  MOUNTPOINT
mypool/lamb  10.2M  13.7G  10.1M   /lamb
```

再次使用 `dd` 命令生成数据并写入 ZFS 池。例如：
`dd if=/dev/random of=/lamb/random2 bs=1m count=10`

```
$ dd if=/dev/random of=/lamb/random2 bs=1m count=10
10+0 records in
10+0 records out
10485760 bytes transferred in 0.141795 secs (73950181 bytes/sec)
```

再次验证副本数量。

```
$ zfs list mypool/lamb
NAME          USED  AVAIL  REFER  MOUNTPOINT
```

```
mypool/lamb 30.2M 13.7G 30.1M /lamb
```











□ □ □□ 30MB □ , □ □ □□ □□ □□ 10, □ □ 10MB □ □ 2□ 20□□ .

ls(1) □ □ □ □ □□ :

```
$ ls -l /lamb/random*
-rw-r--r-- 1 root wheel 10485760 Apr 6 15:27 /lamb/random1
-rw-r--r-- 1 root wheel 10485760 Apr 6 15:29 /lamb/random2
```

????? ??? (Metadata Redundancy)

이러한 구성을 사용하면, VDEV 구성을 RAID-Z로 구성하여, 데이터의 중복성을 높일 수 있습니다. 이 구성은 3개의 VDEV를 사용하여, RAID-Z 구성을 구성합니다. 이 구성은 3개의 VDEV를 사용하여, RAID-Z 구성을 구성합니다. 이 구성은 3개의 VDEV를 사용하여, RAID-Z 구성을 구성합니다.

redundant_metadata           .

redundant_metadata all() ZFS .
 .

redundant_metadata most ZFS
 .
 ,
 100
 .

redundant_metadata *most*  **copies** 3       ,
ZFS    6     4  .

□ □□ □□□□□ □□ □□□□□ □□ □□□□□ □□ □□ □□ □□ □□□□□□ .
 □□□□ □□ □□□ □□ □□□□□ □□□□ □□ □□□□□□ □□ □□ □□ □□□ □□
 □□□□□ □□ □□ □□ □□ □□ □□ □□ □ □□□□ . □ □□ □□ □□ □□ □□ □□ □□ □□
 □□□ □□□□□ .

 .