Chapter 22. The Z File System (ZFS)

Table of Contents

ZFS is an advanced file system designed to solve major problems found in previous storage subsystem software.

Originally developed at Sun™, ongoing open source ZFS development has moved to the OpenZFS Project.

ZFS has three major design goals:

- Data integrity: All data includes a checksum of the data. ZFS calculates checksums and writes them along with the data. When reading that data later, ZFS recalculates the checksums. If the checksums do not match, meaning detecting one or more data errors, ZFS will attempt to automatically correct errors when ditto-, mirror-, or parity-blocks are available.

- Pooled storage: adding physical storage devices to a pool, and allocating storage space from that shared pool. Space is available to all file systems and volumes, and increases by adding new storage devices to the pool.

- Performance: caching mechanisms provide increased performance. ARC is an advanced memory-based read cache. ZFS provides a second level disk-based read cache with L2ARC, and a disk-based synchronous write cache named ZIL.

A complete list of features and terminology is in ZFS Features and Terminology.

22.1. What Makes ZFS Different

More than a file system, ZFS is fundamentally different from traditional file systems. Combining the traditionally separate roles of volume manager and file system provides ZFS with unique advantages. The file system is now aware of the underlying structure of the disks. Traditional file systems could exist on a single disk alone at a time. If there were two disks then creating two separate file systems was necessary. A traditional hardware RAID configuration avoided this problem by presenting the operating system with a single logical disk made up of the space provided by physical disks on top of which the operating system placed a file system. Even with software RAID solutions like those provided by GEOM, the UFS file system living on top of the RAID believes it's dealing with a single device. ZFS' combination of the volume manager and the file system solves this and allows the creation of file systems that all share a pool of available storage. One big advantage of ZFS' awareness of the physical disk layout is that existing file systems grow automatically when adding extra disks to the pool. This new space then becomes available to the file systems. ZFS can also apply different properties to each file system. This makes it useful to create separate file systems and datasets instead of a single monolithic file system.

22.2. Quick Start Guide

FreeBSD can mount ZFS pools and datasets during system initialization. To enable it, add this line to **/etc/rc.conf**:

```
zfs_enable="YES"
```

Then start the service:

```
# service zfs start
```

The examples in this section assume three SCSI disks with the device names **da0**, **da1**, and **da2**. Users of SATA hardware should instead use **ada** device names.

22.2.1. Single Disk Pool

To create a simple, non-redundant pool using a single disk device:

```
# zpool create example /dev/da0
```

To view the new pool, review the output of `df`:

```
# df
Filesystem  1K-blocks    Used    Avail Capacity  Mounted on
/dev/ad0s1a   2026030  235230  1628718    13%    /
devfs               1       1        0   100%    /dev
/dev/ad0s1d  54098308 1032846 48737598     2%    /usr
example      17547136       0 17547136     0%    /example
```

This output shows creating and mounting of the `example` pool, and that is now accessible as a file system. Create files for users to browse:

```
# cd /example
# ls
# touch testfile
# ls -al
total 4
drwxr-xr-x   2 root  wheel    3 Aug 29 23:15 .
drwxr-xr-x  21 root  wheel  512 Aug 29 23:12 ..
-rw-r--r--   1 root  wheel    0 Aug 29 23:15 testfile
```

This pool is not using any advanced ZFS features and properties yet. To create a dataset on this pool with compression enabled:

```
# zfs create example/compressed
# zfs set compression=gzip example/compressed
```

The `example/compressed` dataset is now a ZFS compressed file system. Try copying some large files to **/example/compressed**.

Disable compression with:

```
# zfs set compression=off example/compressed
```

To unmount a file system, use `zfs umount` and then verify with `df`:

```
# zfs umount example/compressed
# df
Filesystem  1K-blocks    Used    Avail Capacity  Mounted on
/dev/ad0s1a   2026030  235232  1628716    13%    /
devfs               1       1        0   100%    /dev
/dev/ad0s1d  54098308 1032864 48737580     2%    /usr
example      17547008       0 17547008     0%    /example
```

To re-mount the file system to make it accessible again, use `zfs mount` and verify with `df`:

```
# zfs mount example/compressed
# df
Filesystem         1K-blocks    Used    Avail Capacity  Mounted on
/dev/ad0s1a          2026030  235234  1628714    13%    /
devfs                      1       1        0   100%    /dev
/dev/ad0s1d         54098308 1032864 48737580     2%    /usr
example             17547008       0 17547008     0%    /example
example/compressed  17547008       0 17547008     0%    /example/compressed
```

Running `mount` shows the pool and file systems: