19.2. Quick Start Guide

There is a startup mechanism that allows FreeBSD to mount ZFS pools 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.

19.2.1. 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 that the `example` pool has been created and mounted. It is now accessible as a file system. Files can be created on it and users can browse it:

```
# 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
```

However, this pool is not taking advantage of any ZFS features. 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`.

Compression can be disabled 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  Mounte
/dev/ad0s1a          2026030  235234  1628714    13%    /
devfs                      1       1        0   100%    /dev
/dev/ad0s1d         54098308 1032864 48737580     2%    /usr
example             17547008       0 17547008     0%    /examp
example/compressed  17547008       0 17547008     0%    /examp
```

The pool and file system may also be observed by viewing the output from `mount`:

```
# mount
/dev/ad0s1a on / (ufs, local)
devfs on /dev (devfs, local)
/dev/ad0s1d on /usr (ufs, local, soft-updates)
example on /example (zfs, local)
example/compressed on /example/compressed (zfs, local)
```

After creation, ZFS datasets can be used like any file systems. However, many other features are available which can be set on a per-dataset basis. In the example below, a new file system called `data` is created. Important files will be stored here, so it is configured to keep two copies of each data block:

```
# zfs create example/data
# zfs set copies=2 example/data
```

It is now possible to see the data and space utilization by issuing `df`:

```
# df
Filesystem         1K-blocks    Used    Avail Capacity  Mounte
/dev/ad0s1a          2026030  235234  1628714    13%    /
devfs                      1       1        0   100%    /dev
/dev/ad0s1d         54098308 1032864 48737580     2%    /usr
```

```
example               17547008        0 17547008     0%    /examp
example/compressed    17547008        0 17547008     0%    /examp
example/data          17547008        0 17547008     0%    /examp
```

Notice that each file system on the pool has the same amount of available space. This is the reason for using `df` in these examples, to show that the file systems use only the amount of space they need and all draw from the same pool. ZFS eliminates concepts such as volumes and partitions, and allows multiple file systems to occupy the same pool.

To destroy the file systems and then destroy the pool as it is no longer needed:

```
# zfs destroy example/compressed
# zfs destroy example/data
# zpool destroy example
```

19.2.2. RAID-Z

Disks fail. One method of avoiding data loss from disk failure is to implement RAID. ZFS supports this feature in its pool design. RAID-Z

pools require three or more disks but provide more usable space than mirrored pools.

This example creates a RAID-Z pool, specifying the disks to add to the pool:

```
# zpool create storage raidz da0 da1 da2
```

> **Note:**
> Sun™ recommends that the number of devices used in a RAID-Z configuration be between three and nine. For environments requiring a single pool consisting of 10 disks or more, consider breaking it up into smaller RAID-Z groups. If only two disks are available and redundancy is a requirement, consider using a ZFS mirror. Refer to zpool(8) for more details.

The previous example created the `storage` zpool. This example makes a new file system called `home` in that pool:

```
# zfs create storage/home
```

Compression and keeping extra copies of directories and files can be enabled:

```
# zfs set copies=2 storage/home
# zfs set compression=gzip storage/home
```

To make this the new home directory for users, copy the user data to this directory and create the appropriate symbolic links:

```
# cp -rp /home/* /storage/home
# rm -rf /home /usr/home
# ln -s /storage/home /home
# ln -s /storage/home /usr/home
```

Users data is now stored on the freshly-created `/storage/home`. Test by adding a new user and logging in as that user.

Try creating a file system snapshot which can be rolled back later:

```
# zfs snapshot storage/home@08-30-08
```

Snapshots can only be made of a full file system, not a single directory or file.

The `@` character is a delimiter between the file system name or the volume name. If an important directory has been accidentally deleted, the file system can be backed up, then rolled back to an earlier snapshot when the directory still existed:

```
# zfs rollback storage/home@08-30-08
```

To list all available snapshots, run `ls` in the file system's `.zfs/snapshot` directory. For example, to see the previously taken snapshot:

```
# ls /storage/home/.zfs/snapshot
```

It is possible to write a script to perform regular snapshots on user data. However, over time, snapshots can consume a great deal of disk space. The previous snapshot can be removed using the command:

```
# zfs destroy storage/home@08-30-08
```

After testing, `/storage/home` can be made the real `/home` using this command:

```
# zfs set mountpoint=/home storage/home
```

Run `df` and `mount` to confirm that the system now treats the file system as the real `/home`:

```
# mount
/dev/ad0s1a on / (ufs, local)
devfs on /dev (devfs, local)
/dev/ad0s1d on /usr (ufs, local, soft-updates)
storage on /storage (zfs, local)
storage/home on /home (zfs, local)
# df
Filesystem   1K-blocks    Used    Avail Capacity  Mounted on
/dev/ad0s1a    2026030  235240  1628708    13%    /
devfs                1       1        0   100%    /dev
/dev/ad0s1d   54098308 1032826 48737618     2%    /usr
storage       26320512       0 26320512     0%    /storage
storage/home  26320512       0 26320512     0%    /home
```

This completes the RAID-Z configuration. Daily status updates about the file systems created can be generated as part of the nightly periodic(8) runs. Add this line to `/etc/periodic.conf`:

```
daily_status_zfs_enable="YES"
```

19.2.3. Recovering RAID-Z

Every software RAID has a method of monitoring its `state`. The status of RAID-Z devices may be viewed with this command:

```
# zpool status -x
```

If all pools are Online and everything is normal, the message shows:

```
all pools are healthy
```

If there is an issue, perhaps a disk is in the Offline state, the pool state will look similar to:

```
  pool: storage
 state: DEGRADED
status: One or more devices has been taken offline by the admi
        Sufficient replicas exist for the pool to continue fur
        degraded state.
action: Online the device using 'zpool online' or replace the
        'zpool replace'.
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        storage     DEGRADED     0     0     0
          raidz1    DEGRADED     0     0     0
            da0     ONLINE       0     0     0
            da1     OFFLINE      0     0     0
            da2     ONLINE       0     0     0

errors: No known data errors
```

This indicates that the device was previously taken offline by the administrator with this command:

```
# zpool offline storage da1
```

Now the system can be powered down to replace `da1`. When the system is back online, the failed disk can replaced in the pool:

```
# zpool replace storage da1
```

From here, the status may be checked again, this time without `-x` so that all pools are shown:

```
# zpool status storage
 pool: storage
 state: ONLINE
 scrub: resilver completed with 0 errors on Sat Aug 30 19:44:1
config:

        NAME        STATE     READ WRITE CKSUM
        storage     ONLINE       0     0     0
          raidz1    ONLINE       0     0     0
            da0     ONLINE       0     0     0
            da1     ONLINE       0     0     0
            da2     ONLINE       0     0     0

errors: No known data errors
```

In this example, everything is normal.

19.2.4. Data Verification

ZFS uses checksums to verify the integrity of stored data. These are enabled automatically upon creation of file systems.

> **Warning:**
> Checksums can be disabled, but it is *not* recommended! Checksums take very little storage space and provide data integrity. Many ZFS features will not work properly with checksums disabled. There is no noticeable performance gain from disabling these checksums.

Checksum verification is known as *scrubbing*. Verify the data integrity of the `storage` pool with this command:

```
# zpool scrub storage
```

The duration of a scrub depends on the amount of data stored. Larger amounts of data will take proportionally longer to verify. Scrubs are very I/O intensive, and only one scrub is allowed to run at a time. After the scrub completes, the status can be viewed with `status`:

```
# zpool status storage
 pool: storage
 state: ONLINE
 scrub: scrub completed with 0 errors on Sat Jan 26 19:57:37 2
config:

        NAME        STATE     READ WRITE CKSUM
        storage     ONLINE       0     0     0
          raidz1    ONLINE       0     0     0
            da0     ONLINE       0     0     0
            da1     ONLINE       0     0     0
            da2     ONLINE       0     0     0

errors: No known data errors
```

The completion date of the last scrub operation is displayed to help track when another scrub is required. Routine scrubs help protect data from silent corruption and ensure the integrity of the pool.

Refer to zfs(8) and zpool(8) for other ZFS options.

Prev | Up | Next
Chapter 19. The Z File System (ZFS) | Home | 19.3. zpool Administration