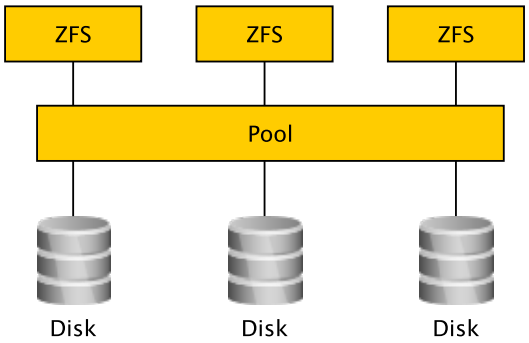
OpenZFS Cheat Sheet



Storage Pool Configurations

Single Disk **Not redundant**, mounted as `/mypool`, there is no need for an entry in `/etc/fstab`. `zpool create mypool disk1`

Stripe (RAID-0) **No redundancy, loss of all data if any one disk dies**. High performance, full disk space of all disks usable. `zpool create mystripe disk1 disk2`

Mirror (RAID-1) Survives failure of one drive, VDEV shown as `mirror-0` in `zpool status` output, parallel reads of all disks, slow writes on all disks, lower total capacity compared to RAID-0. `zpool create mymirror mirror disk1 disk2`

Single-Parity (RAID-Z1) Requires at least 3 disks, survives one failing disk, parity information gets distributed across all disks, fast reads and writes with comparable performance, 66% total capacity. Shown as `raidz1-0` VDEV in `zpool status` output. `zpool create paritypool raidz disk1 disk2 disk3`

Double-Parity (RAID-Z2) survives the failure of two disks per VDEV, slower than RAID-Z1, 4 disks required, 50% capacity. `zpool create myraidz2 raidz2 disk1 disk2 disk3 disk4`

RAID10 Min. of 4 disks needed, survives failure of 2 disks (one per VDEV) high read speeds, write speed half of that, capacity 50%. Good compromise between redundancy, capacity and performance. `zpool create myr10 mirror disk1 disk2 mirror disk3 disk4`

Triple-Parity (RAID-Z3) Little slower than RAID-Z2, redundancy of three disks per VDEV, min. 5 disks required, 40% capacity. `zpool create myz3 raidz3 disk1 disk2 disk3 disk4 disk5`

Display Pool Status

`zpool status` Display pool status including disk configuration, any errors and applicable updates, last scrub (if any).

`zpool list` Show pool capacity, used and free space.

`zpool iostat` Display pool I/O statistics.
Options display individual devices (`-v`), I/O latency (`-w`), request size histograms (`-r`), wait (`-l`) and queue statistics (`-q`).

`zpool history` Administrative command history of the pool. Options include long form (`-l`) or events like transaction groups (`-i`).

`zpool get` Properties and values, change `default`s via `zpool set`.

Special VDEVs

Level 2 ARC (L2ARC)
A fast read cache for cases where data does not fit in the ZFS main memory cache (ARC: adaptive replacement cache) anymore. Reads will then served out of the L2ARC, which requires a fast storage device (flash) to see benefits. Use the `cache` keyword to add such a device to a pool. `zpool add mypool cache /dev/nda0`

ZFS Intent Log (ZIL) Changes synchronous into asynchronous writes (does not affect reads). Applications get faster confirmation of written data and hence acts like a database transaction log. Gets cleared once the writes have reached the underlying pool storage media. Requires fast storage media, but does not need to be big. Use the `log` keyword to add to a pool: `zpool add mypool log /dev/nda1`

Spare Does not take I/O until replaced (manually or by external failure management software) with a failed pool disk. Add to a pool with the `spare` keyword: `zpool add mypool spare /dev/nda2`

Pool Extension

Turn single disk pool into RAID1
The single disk pool device `nda0` gets the mirror partner disk `nda1` attached: `zpool attach mypool mirror /dev/nda0 /dev/nda1`

Replace failed disk Exchange the failed pool disk `nda2` with device `nda3`: `zpool replace mypool /dev/nda2 /dev/nda2`

Pool Import/Export

Export Finishes and closes I/O operations, then unmounts the pool from the filesystem hierarchy. Attach to a different system and import there to reconstruct the pool state. `zpool export mypool`

Import Imports pools into the running system.

- Scan for ZFS pool signatures: `zpool import`
- Import a pool via ID or its name: `zpool import mypool`
- Renames a pool: `zpool import oldname newname`
- Mount the imported pool under a different path to prevent shadowing existing ones: `zpool import -R /media mypool`
- Import readonly: `zpool import -o readonly mypool`
- Try restoring recently destroyed pool: `zpool import -D mypool`

ZFS Datasets

Datasets sit on top of the pool and consume its space. Addressed like this: `mypool/data`. Each pool creates a default dataset with its own name as top level. **Example:** `zpool create test disk1` creates a pool and dataset called `test`, which may hold child datasets beneath. Mounts the top-level pool as the `/test` dataset.

Create new dataset Creates a new dataset called `ds`, inherits most properties of the parent dataset: `zfs create mypool/ds`
Create multiple datasets: `zfs create -p mypool/home/fred`

Display datasets
List the used, available space, and mountpoint: `zfs list`

- Show dataset by ZFS path: `zfs list mypool/ds`
- List datasets and any children: `zfs list -r mypool/ds`
- Limit display depth of datasets: `zfs list -d 1 mypool/home`
- Display dataset name only: `zfs list -o name mypool/home`
- Remove output headers, too: `zfs list -Ho name mypool/home`
- Change display order: `zfs list -o used,avail,refer,name`
- Sort output by column: `zfs list -rs refer mypool/home`
- Reverse sort order: `zfs list -rS refer mypool/home`

Display available pool space Lists available and used space per dataset and children, including snapshots and any reservations: `zfs list -o space` **Recommendation:** use instead of `df -h`.

Rename dataset Rename dataset or move to a different place within the pool hierarchy, comparable to the Unix `mv` command. `zfs rename mypool/home/fred mypool/home/eva`

Destroy dataset Simulate first: `zfs destroy -nv mypool/old`
Output: `would destroy mypool/old`
Remove and show result: `zfs destroy -v mypool/old`
Output: `will destroy mypool/old`
Recursively destroy datasets: `zfs destroy -r mypool/testdata`

Properties

Dataset use is flexible by inheriting most of their parents properties. Children may override these if necessary. Change only `default` properties in the `SOURCE` column. The same goes for pool properties.

Display properties There are various ways to display properties:

- Show pool properties: `zpool get all mypool`
- Display all properties of a dataset: `zfs get all pool/dataset`
- List single property (`capacity`): `zpool get capacity mypool`
- List multiple properties: `zpool get capacity,health mypool`

Change properties
Deactivate access time property: `zfs set atime=off mypool`
Change mountpoint: `zfs set mountpoint=/media mypool/ds`

Custom Properties Define custom dataset property as a `key=value pair`. Datasets inherit them, but can change their value.

- Creation: `zfs set warranty:expires=2048/04/20 mypool`
- List custom property: `zfs get warranty:expires mypool`
- Reset the value: `zfs inherit warranty:expires mypool`
- Remove property: `zfs inherit -r warranty:expires mypool`

Scrub

ZFS stores checksums alongside the data and in the whole dataset hierarchy. Different factors like I/O errors, malformed drivers or defective RAM, broken cables, etc. cause the checksum calculation to not match anymore. Given enough redundancy at the pool level (VDEVs), ZFS finds these errors and corrects them (self healing). Run monthly `scrub`s on the pool to re-calculate the checksums. If they do not match, ZFS asks the redundant VDEV for it and corrects the data if they match. `zpool scrub mypool` The `zpool status` output displays the progress of the scrub operation and any errors that were found. There is no `fsck` in ZFS since it is unnecessary.

Volumes

ZFS Volumes uses contiguous pool space to export via iSCSI over the network. When formatting a volume with a non-ZFS filesystem, then this filesystem uses all underlying ZFS features automatically.

Volume creation
Create a 10 GB ZFS volume: `zfs create -V 10G mypool/vol1`

Create sparse volume An overprovisioned volume that does not use the reserved space right away, but fills it up over time. `zfs create -V 1P -s mypool/sparse1PBvol`

Quota

Each dataset may use up the entire disk space of the pool. Quotas restrict that to certain user defined amounts. Writes that run over the quota are strictly stopped by ZFS to enforce the quotas.

Define Quota
Datasets do not have a quota by default. Set a quota for a specific dataset using `zfs set`: `zfs set quota=10G mypool/dataset` A quota restricts the dataset and any children (existing and future ones). Datasets share the total quota amongst themselves. If a quota should apply to the parent dataset alone, then use: `zfs set refquota=10G mypool/dataset` ZFS quotas may also restrict certain daemons or users in the system: `zfs set userquota@fred=10G mypool/home/fred` A group quota restricts a collection of users as a whole: `zfs set groupquota@projectX=100G mypool/projectX`

Display quotas Query the quota property using `zfs get`:

- List quotas for a given dataset: `zfs get quota mypool/dataset`
- Same for the `refquota`: `zfs get refquota mypool/home/fred`
- Display a userquota: `zfs userspace mypool/home/fred`
- Show group quotas: `zfs groupspace mypool/projectX`

Remove quota Return dataset to having no quota applied (same for `refquota`): `zfs set quota=none mypool/home/fred`

Reservation

Reservations guarantee a certain amount of available disk space in the pool, regardless of how much other datasets use. This reduces the total pool space for the reservation. This prevents any one dataset to use up the whole pool space and allows capacity planning.

Define reservation
The `reservation` property controls the amount of disk space to reserve for a dataset: `zfs set reservation=100G mypool/home` Reservations apply to child datasets, too. Set the reservation to the dataset alone: `zfs set refreservation=10G mypool/home/eve`

Display reservation
Use the `reservation` property to show the dataset reservation: `zfs get reservation mypool/home/eve` Likewise, `zfs list -o space` shows the used reservation in the `USEDREFRESERV` column.

Remove reservation Set `reservation` or `refreservation` to `none`: `zfs set reservation=none mypool/home/eve`

Snapshots

Snapshots provide a quick way of preserving a readonly state of the dataset at a certain time. Roll back a snapshot to return the dataset to the snapshot's creation time. Restoring individual files from the snapshot is also possible without a complete rollback. A hidden `.zfs` directory allows readonly access to individual files for this purpose.

Snapshot creation `zfs snapshot mypool/ds@mysnapshot`
Short form: `zfs snap mypool/ds@mysnapshot`
Recursive snapshot: `zfs snap -r mypool/ds@mysnapshot`

Display snapshots
List all dataset snapshots: `zfs list -t snap mypool/ds`
List snapshots with all children: `zfs list -rt snap mypool/ds`

Show writes since last snapshot
The `written` property shows the amount of data written to the dataset since taking the last snapshot. In combination with `used` and `referenced` it gives a good overview of the necessary pool space: `zfs list -rt all -o name,used,refer,written mypool`

Compare snapshots Use `zfs diff` to compare changes in the dataset since a particular snapshot: `zfs diff mypool/ds@backup` The table below describes the output:

Character	Description
+	File added
−	File deleted
M	File modified
R	File renamed

Applied to directories means changes in the metadata. Compare two snapshots: `zfs diff mypool/ds@backup1 mypool/ds@backup2`

Snapshot rollback Throws away the current dataset state and returns to the last snapshot state. Removes all created data since that particular snapshot. `zfs rollback mypool/ds@backup2` Rolling back to an earlier snapshot requires the removal of all intermediary snapshots: `zfs rollback -r mypool/ds@backup1`

Snapshot mounting Mount a snapshot into the filesystem as a readonly dataset: `mount -t zfs mypool/ds@backup /mnt/backup`

Delete snapshot range
Execute a verbose (`-v`) dry run (`-n`) first to see what would get deleted: `zfs destroy -vn mypool/ds@backup` If this is the desired action, remove the dry run parameter `-n` to execute: `zfs destroy -v mypool/ds@backup` Recursively delete snapshots using `-r`: `zfs destroy -rv mypool/ds@backup`

Delete range
For snapshots `@a` - `@e`, define different ranges for deletion:

- Delete `@b` through `@d` (inclusive): `zfs destroy -v mypool@b%d`
- Delete starting from snapshot `@b`: `zfs destroy -v mypool@b%`
- Delete snapshots before `@b`: `zfs destroy -v mypool%@b`

ZFS Holds Preserve a snapshot by creating a hold referenced by a custom tag (description). ZFS allows multiple tags and only if the last tag is gone, the snapshot may be removed.

- Create new hold: `zfs hold keepme mypool/home@important`
- List holds recursively: `zfs holds -r mypool/home@important`
- Lift a hold again by tag: `zfs release mypool/home@important`

Clones

A clone represents a writeable version of a snapshot. All the data from the snapshot they are based on are present in the clone, too.

Clone creation Create a clone from an existing snapshot using `zfs clone`: `zfs clone mypool/ds@backup mypool/myclone` The `origin` property of a clone contains the originating snapshot for reference: `zfs get origin mypool/myclone`

Make clone independent ZFS prevents removing a snapshot as long as dependent clones still exist. Resolve this by reversing the dependency of clone and snapshot by promoting a clone to a full dataset: `zfs promote mypool/myclone` The `origin` property disappears and the snapshot is associated with the clone now.

Clone removal
Delete clones like regular datasets: `zfs destroy mypool/myclone`

Encryption

ZFS allows encrypting datasets with individual keys. Some metadata remains unencrypted to allow scrub and other operations to run.

Create encrypted dataset Set dataset encryption passphrase: `zfs create -o encryption=on -o keyformat=passphrase -o keylocation=prompt mypool/secret`

Load status Get status: `zfs get keystatus mypool/secret`

Load key Unencrypt dataset: `zfs load-key mypool/secret`

Delegation

ZFS allows delegating ZFS commands to non-root users.

Delegate permission to user Grant user the `atime` property permission on dataset: `zfs allow -u joe atime mypool/dataset`

Display delegations
Show existing dataset delegations: `zfs allow mypool/dataset`

Remove delegation Use `zfs unallow` to remove `compression` permission: `zfs unallow -u joe compression mypool/dataset`

Delegate permission to group
Target a group: `zfs allow -g mygroup atime mypool/dataset`

Delegate permission to delegate
ZFS permits delegating those permissions that the delegating user possesses: `zfs allow -u jill allow mypool/dataset`

Create a set of permissions A permission set allows combining multiple delegations under a user-defined name: `zfs allow -s @myset mount,snapshot,rollback,destroy mypool/dataset`

Use permission set for delegations Apply permission set to grant permissions: `zfs allow -u jill @myset mypool/dataset`

Sending and Receiving Snapshots

ZFS allows transferring snapshots as a byte stream locally or over the network to the same or a different pool for backup purposes.

Write snapshot to a file Write a byte stream representation of the data to a file: `zfs send mypool/ds@backup > dsbackup` Restore this file to the same or a different pool using `zfs receive`. Display more details about the transfer using the `-v` parameter: `zfs send -v mypool/ds@backup > target`

Create dataset from snapshot
Use `receive` (or: `recv`) to re-create dataset from snapshot: `zfs recv mypool/backup < dsbackup` Display transfer details using the `-v` parameter: `zfs recv -v mypool/backup < dsbackup`

Replication without intermediary file
A pipe between `zfs send` and `zfs recv` combines the input and output: `zfs send mypool/ds@backup|zfs recv mypool/new`

Replication via SSH
Send snapshots over SSH to another pool on a different host: `zfs send poolA/ds@backup|ssh host zfs recv poolB/new`

Send permissions Allow non-root user `sender` to send datasets: `zfs allow -u sender send,snapshot mypool/source`

Receive permissions
Non-root user `receiver` requires more permissions: `zfs allow -u receiver compression,mountpoint,mount,create,receive mypool/destination`

Benedict Reuschling (`benedict@reuschling.org`), March 2025