



ZFS





From Gentoo Wiki

ZFS is a next generation filesystem (/wiki/Filesystem) created by Matthew Ahrens and Jeff Bonwick. It was designed around a few key ideas:

- Administration of storage should be simple.
- Redundancy should be handled by the filesystem.
- File-systems should never be taken offline for repair.
- Automated simulations of worst case scenarios before shipping code is important.
- Data integrity is paramount.

Development of ZFS started in 2001 at Sun Microsystems. It was released under the CDDL (<http://opensource.org/licenses/CDDL-1.0>) in 2005 as part of Open Solaris. Pawel Jakub Dawidek ported ZFS to FreeBSD in 2007. Brian Behlendorf at LLNL (<http://www.llnl.gov/>) started the ZFSOnLinux project in 2008 to port ZFS to Linux for High Performance Computing. Oracle purchased Sun Microsystems in 2010 and discontinued Open Solaris later that year. The Illumos project started to replace Open Solaris and roughly 2/3 of the core ZFS team resigned, including Matthew Ahrens and Jeff Bonwick. Most of them took jobs at companies and continue to develop Open Source ZFS as part of the Illumos project. The 1/3 of the ZFS core team at Oracle that did not resign continue development of an incompatible proprietary branch of ZFS in Oracle Solaris. The first release of Solaris included a few innovative changes that were under development prior to the mass resignation. Subsequent releases of Solaris have included fewer and less ambitious changes. Significant innovation continues in the open source branch of ZFS developed in Illumos. Today, a growing community continues development of the open source branch of ZFS across multiple platforms, including FreeBSD, Illumos, Linux and Mac OS X.

Resources

-  Home (http://www.open-zfs.org/wiki/Main_Page)
-  Wikipedia (<https://en.wikipedia.org/wiki/ZFS>)
-  GitHub (<https://github.com/zfsonlinux/zfs>)
-  Official documentation (<http://www.open-zfs.org/wiki/Documentation>)

Contents

- 1 Features
- 2 Installation
 - 2.1 Kernel
 - 2.2 Modules
 - 2.3 USE flags
 - 2.4 ARC

- 2.4.1 Adjusting ARC memory usage
- 3 Installing into the kernel directory (for static installs)
- 4 Usage
 - 4.1 Preparation
 - 4.2 Zpools
 - 4.2.1 import/export Zpool
 - 4.2.2 One hard drive
 - 4.2.3 Two hard drives (MIRROR)
 - 4.2.4 Three hard drives (RAIDZ1)
 - 4.2.5 Four hard drives (RAIDZ2)
 - 4.2.6 Four hard drives (STRIPED MIRROR)
 - 4.2.7 Spares/Replace vdev
 - 4.2.8 Zpool version update
 - 4.2.9 Zpool tips/tricks
 - 4.3 Volumes
 - 4.3.1 Create Volumes
 - 4.3.2 Mount/umount volumes
 - 4.3.3 Remove volumes
 - 4.3.4 Properties
 - 4.3.5 Set mountpoint
 - 4.3.6 NFS volume share
 - 4.4 Snapshots
 - 4.4.1 Creating
 - 4.4.2 Listing
 - 4.4.3 Rollback
 - 4.4.4 Cloning
 - 4.4.5 Removal
 - 4.5 Maintenance
 - 4.5.1 Scrubbing
 - 4.5.2 Log Files
 - 4.5.3 Monitor I/O
- 5 ZFS root
- 6 Caveats
- 7 See also
- 8 External resources

Features

A detailed list of features can be found in a separate article (</wiki/ZFS/Features>).

Installation

Kernel

sys-kernel/spl (<https://packages.gentoo.org/packages/sys-kernel/spl>) requires Zlib kernel support (module or builtin).

KERNEL

```
General Setup --->
  GCC plug ins --->
    [ ] Randomize layout of sensitive kernel structures
Cryptographic API --->
  <*> Deflate compression algorithm
Security options --->
  [ ] Harden common str/mem functions against buffer overflows
```

Notes:

- if building of sys-kernel/spl (<https://packages.gentoo.org/packages/sys-kernel/spl>) is complaining about CONFIG_ZLIB_DEFLATE option
- if spl fails in the configure phase, as it checks whether memory shrinkers, and fails with the line '->count_objects callback exists
- Randomizing layout of sensitive kernel structures (CONFIG_GCC_PLUGIN_RANDSTRUCT) will cause "unsupported stack pointer realignment

Modules

There are out-of-tree Linux kernel modules available from the ZFSOnLinux Project (<http://www.zfsonlinux.org/>). The current release is version 0.6.5.8 (zpool version 5000). This succeeds 0.6.1, which was the first release considered "ready for wide scale deployment on everything from desktops to super computers" (<http://www.h-online.com/open/news/item/ZFS-on-Linux-is-ready-for-wide-scale-deployment-1832848.html>), by the ZFSOnLinux Project.

Note

All changes to the GIT repository are subject to regression tests by LLNL (<https://www.llnl.gov/>).

To install ZFS on Gentoo Linux requires `~amd64` keyword for `sys-fs/zfs` (<https://packages.gentoo.org/packages/sys-fs/zfs>) and its dependencies `sys-fs/zfs-kmod` (<https://packages.gentoo.org/packages/sys-fs/zfs-kmod>) and `sys-kernel/spl` (<https://packages.gentoo.org/packages/sys-kernel/spl>):

```
root # echo "sys-kernel/spl ~amd64" >> /etc/portage/package.accept_keywords
root # echo "sys-fs/zfs-kmod ~amd64" >> /etc/portage/package.accept_keywords
root # echo "sys-fs/zfs ~amd64" >> /etc/portage/package.accept_keywords
root # emerge -av zfs
```

The latest upstream versions require keywording the live ebuilds (optional):

```
root # echo "=sys-kernel/spl-9999 **" >> /etc/portage/package.accept_keywords
root # echo "=sys-fs/zfs-kmod-9999 **" >> /etc/portage/package.accept_keywords
root # echo "=sys-fs/zfs-9999 **" >> /etc/portage/package.accept_keywords
```

Add the `zfs` scripts to the run levels to do initialization at boot:

```
root # rc-update add zfs-import boot
root # rc-update add zfs-mount boot
root # rc-update add zfs-share default
root # rc-update add zfs-zed default
```

Note

Only the first two are necessary for most setups. `zfs-share` is for people using NFS shares while `zfs-zed` is for the ZFS Event Daemon that handles disk replacement via hotspares and email notification of failures.

Note

For those who want to use ZFS as root file system, as well as those who put their swaps on ZFS, they might add `zfs-import` and `zfs-mount` to `sysinit` level to make the file system accessible during boot or shutdown process.

USE flags

Cannot load package information. Is the atom `sys-fs/zfs` correct?

ARC

ZFSOnLinux uses ARC (http://en.wikipedia.org/wiki/Adaptive_replacement_cache) page replacement algorithm instead of the Last Recently Used page replacement algorithm used by other filesystems. This has a better hit rate, therefore providing better performance. The implementation of ARC in ZFS differs from the original paper in that the amount of memory used as cache can vary. This permits memory used by ARC to be

reclaimed when the system is under memory pressure (via the kernel's shrinker mechanism) and grow when the system has memory to spare. The minimum and maximum amount of memory allocated to ARC varies based on your system memory. The default minimum is 1/32 of all memory, or 64MB, whichever is more. The default maximum is the larger of 1/2 of system memory or 64MB.

The manner in which Linux accounts for memory used by ARC differs from memory used by the page cache. Specifically, memory used by ARC is included under "used" rather than "cached" in the output used by the `free` program. This in no way prevents the memory from being released when the system is low on memory. However, it can give the impression that ARC (and by extension ZFS) will use all of system memory if given the opportunity.

Adjusting ARC memory usage

The minimum and maximum memory usage of ARC is tunable via `zfs_arc_min` and `zfs_arc_max` respectively. These properties can be set any of three ways. The first is at runtime (new in 0.6.2):

```
root # echo 536870912 >> /sys/module/zfs/parameters/zfs_arc_max
```

Note

This sysfs value became writable in ZFSOnLinux 0.6.2. Changes through sysfs do not persist across boots. Also, the value in sysfs will be 0 when this value has not been manually configured. The current setting can be viewed by looking at `c_max` in `/proc/spl/kstat/zfs/arcstats`

The second is via `/etc/modprobe.d/zfs.conf`:

```
root # echo "options zfs zfs_arc_max=536870912" >> /etc/modprobe.d/zfs.conf
```

Note

If using genkernel to load ZFS, this value must be set before genkernel is run to ensure that the file is copied into the initramfs.

The third is on the kernel commandline by specifying `"zfs.zfs_arc_max=536870912"` (for 512MB).

Similarly, the same can be done to adjust `zfs_arc_min`.

Installing into the kernel directory (for static installs)

This example uses 9999, but just change it to the latest ~ or stable (when that happens) and you should be good. The only issue you may run into is having `zfs` and `zfs-kmod` out of sync with each other. Just try to avoid that :D

This will generate the needed files, and copy them into the kernel sources directory.

```
root # env EXTRA_ECONF='--enable-linux-builtin' ebuild /usr/portage/sys-kernel/spl/spl-9999.ebuild clean configure
```

```
root # (cd /var/tmp/portage/sys-kernel/spl-9999/work/spl-9999 && ./copy-builtin /usr/src/linux)
```

```
root # env EXTRA_ECONF='--with-spl=/usr/src/linux --enable-linux-builtin --with-spl-obj=/usr/src/linux' ebuild /usr/portage/sys-fs/zfs-kmod/zfs-kmod-9999.ebuild clean configure
```

```
root # (cd /var/tmp/portage/sys-fs/zfs-kmod-9999/work/zfs-kmod-9999/ && ./copy-builtin /usr/src/linux)
```

After this, you just need to edit the kernel config to enable `CONFIG_SPL` and `CONFIG_ZFS` and emerge the `zfs` binaries.

```
root # mkdir -p /etc/portage/profile
root # echo 'sys-fs/zfs -kernel-builtin' >> /etc/portage/profile/package.use.mask
root # echo 'sys-fs/zfs kernel-builtin' >> /etc/portage/package.use/zfs.conf
root # emerge --oneshot --verbose sys-fs/zfs
```

The echo's only need to be run once, but the emerge needs to be run every time you install a new version of zfs.

Usage

ZFS includes already all programs to manage the hardware and the file systems, there are no additional tools needed.

Preparation

ZFS supports the use of either block devices or files. Administration is the same in both cases, but for production use, the ZFS developers recommend the use of block devices (preferably whole disks). To take full advantage of block devices on Advanced Format disks, it is highly recommended to read the ZFS on Linux FAQ (<https://github.com/zfsonlinux/zfs/wiki/faq#advanced-format-disks>) before creating your pool. To go through the different commands and scenarios we can use files in place of block devices.

The following commands create 2GB sparse image files in `/var/lib/zfs_img/` that we use as our hard drives. This uses at most 8GB disk space, but in practice will use very little because only written areas are allocated:

```
root # mkdir /var/lib/zfs_img
root # truncate -s 2G /var/lib/zfs_img/zfs0.img
root # truncate -s 2G /var/lib/zfs_img/zfs1.img
root # truncate -s 2G /var/lib/zfs_img/zfs2.img
root # truncate -s 2G /var/lib/zfs_img/zfs3.img
```

Now we check which loopback devices are in use:

Note

On pool export, all of the files will be released and the folder `/var/lib/zfs_img` can be deleted

Zpools

The program `/usr/sbin/zpool` is used with any operation regarding *zpools*.

import/export Zpool

To export (unmount) an existing zpool named *zfs_test* into the file system, you can use the following command:

```
root # zpool export zfs_test
```

To import (mount) the zpool named *zfs_test* use this command:

```
root # zpool import zfs_test
```

The root mountpoint of *zfs_test* is a property and can be changed the same way as for volumes. To import (mount) the zpool named *zfs_test* root on */mnt/gentoo*, use this command:

```
root # zpool import -R /mnt/gentoo zfs_test
```

Note

ZFS will automatically search on the hard drives for the zpool named *zfs_test*

To search for all zpools available in the system issue the command:

```
root # zpool import
```

One hard drive

Create a new zpool named *zfs_test* with one hard drive:

```
root # zpool create zfs_test /var/lib/zfs_img/zfs0.img
```

The zpool will automatically be mounted, default is the root file system aka */zfs_test*

```
root # zpool status
```

To delete a zpool use this command:

```
root # zpool destroy zfs_test
```

Important

ZFS will not ask if you are sure.

Two hard drives (MIRROR)

In ZFS you can have several hard drives in a MIRROR, where equal copies exist on each storage. This increases the performance and redundancy. To create a new zpool named *zfs_test* with two hard drives as MIRROR:

```
root # zpool create zfs_test mirror /var/lib/zfs_img/zfs0.img /var/lib/zfs_img/zfs1.img
```

Note

of the two hard drives only 2GB are effective useable so *total_space * 1/n*

```
root # zpool status
```

To delete the zpool:

```
root # zpool destroy zfs_test
```

Three hard drives (RAIDZ1)

RAIDZ1 is the equivalent to RAID5, where data is written to the first two drives and a parity onto the third. You need at least three hard drives, one can fail and the zpool is still ONLINE but the faulty drive should be replaced as soon as possible.

To create a pool with RAIDZ1 and three hard drives:

```
root # zpool create zfs_test raidz1 /var/lib/zfs_img/zfs0.img /var/lib/zfs_img/zfs1.img /var/lib/zfs_img/zfs2.img
```

Note

of the three hard drives only 4GB are effective useable so $total_space * (1-1/n)$

```
root # zpool status
```

To delete the zpool:

```
root # zpool destroy zfs_test
```

Four hard drives (RAIDZ2)

RAIDZ2 is the equivalent to RAID6, where data is written to the first two drives and a parity onto the next two. You need at least four hard drives, two can fail and the zpool is still ONLINE but the faulty drives should be replaced as soon as possible.

To create a pool with RAIDZ2 and four hard drives:

```
root # zpool create zfs_test raidz2 /var/lib/zfs_img/zfs0.img /var/lib/zfs_img/zfs1.img /var/lib/zfs_img/zfs2.img /var/lib/zfs_img/zfs3.img
```

Note

of the four hard drives only 4GB are effective useable so $total_space * (1-2/n)$

```
root # zpool status
```

To delete the zpool:

```
root # zpool destroy zfs_test
```

Four hard drives (STRIPED MIRROR)

A STRIPED MIRROR is the equivalent to RAID10, where data is striped across sets of disks then the striped data is mirrored. You need at least four hard drives; this configuration provides redundancy and an increase in read speed. You can lose all disks but one per mirror.

To create a STRIPED MIRROR pool with four hard drives:

```
root # zpool create zfs_test mirror /var/lib/zfs_img/zfs0.img /var/lib/zfs_img/zfs1.img mirror /var/lib/zfs_img/zfs2.img /var/lib/zfs_img/zfs3.img
```

Note

of the four hard drives only 4GB are useable so $total_space * (1-2/n)$


```
root # zpool status
```

```
pool: zfs_test
state: ONLINE
scan: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
zfs_test	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
/var/lib/zfs_img/zfs0.img	ONLINE	0	0	0
/var/lib/zfs_img/zfs1.img	ONLINE	0	0	0
mirror-1	ONLINE	0	0	0
/var/lib/zfs_img/zfs2.img	ONLINE	0	0	0
/var/lib/zfs_img/zfs3.img	ONLINE	0	0	0

```
errors: No known data errors
```

To delete the zpool:

```
root # zpool destroy zfs_test
```

Spares/Replace vdev

You can add *hot-spares* into your zpool. In case a failure, those are already installed and available to replace faulty vdevs.

In this example, we use RAIDZ1 with three hard drives and a zpool named *zfs_test*:

```
root # zpool add zfs_test spare /var/lib/zfs_img/zfs3.img
```

```
root # zpool status
```

The status of */dev/loop3* will stay *AVAIL* until it is set to be *online*, now we let */var/lib/zfs_img/zfs0.img* fail:

```
root # zpool offline zfs_test /var/lib/zfs_img/zfs0.img
```

```
root # zpool status
```

```
pool: zfs_test
state: ONLINE
scan: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
zfs_test	ONLINE	0	0	0
raidz1-0	ONLINE	0	0	0
/var/lib/zfs_img/zfs0.img	ONLINE	0	0	0
/var/lib/zfs_img/zfs1.img	ONLINE	0	0	0
/var/lib/zfs_img/zfs2.img	ONLINE	0	0	0
spares				
/var/lib/zfs_img/zfs3.img				

```
errors: No known data errors
```

We replace /var/lib/zfs_img/zfs0.img with our spare /var/lib/zfs_img/zfs3.img:

```
root # zpool replace zfs_test /var/lib/zfs_img/zfs0.img /var/lib/zfs_img/zfs3.img
```

```
root # zpool status
```

```
pool: zfs_test
state: ONLINE
scan: resilvered 62K in 0h0m with 0 errors on Sun Sep 1 15:41:41 2013
config:
```

NAME	STATE	READ	WRITE	CKSUM
zfs_test	ONLINE	0	0	0
raidz1-0	ONLINE	0	0	0
spare-0	ONLINE	0	0	0
/var/lib/zfs_img/zfs0.img	ONLINE	0	0	0
/var/lib/zfs_img/zfs3.img	ONLINE	0	0	0
/var/lib/zfs_img/zfs1.img	ONLINE	0	0	0
/var/lib/zfs_img/zfs2.img	ONLINE	0	0	0
spares				
/var/lib/zfs_img/zfs3.img	INUSE	currently in use		

```
errors: No known data errors
```

The original vdev will automatically get removed asynchronously. If this is not the case, the old vdev may need to be detached with the "zpool detach" command. Later you will see it leave the zpool status output:

```
root # zpool status
```

```
pool: zfs_test
state: ONLINE
scan: resilvered 62K in 0h0m with 0 errors on Sun Sep  1 15:41:41 2013
config:
```

NAME	STATE	READ	WRITE	CKSUM
zfs_test	ONLINE	0	0	0
raidz1-0	ONLINE	0	0	0
/var/lib/zfs_img/zfs3.img	ONLINE	0	0	0
/var/lib/zfs_img/zfs1.img	ONLINE	0	0	0
/var/lib/zfs_img/zfs2.img	ONLINE	0	0	0

```
errors: No known data errors
```

Note

ZFS automatically resilvered onto /var/lib/zfs_img/zfs0.img and the zpool had no downtime

Now we start a manual scrub:

```
root # zpool scrub zfs_test
```

```
root # zpool status
```

```
pool: zfs_test
state: ONLINE
scan: scrub repaired 0 in 0h0m with 0 errors on Sun Sep  1 15:57:31 2013
config:
```

NAME	STATE	READ	WRITE	CKSUM
zfs_test	ONLINE	0	0	0
raidz1-0	ONLINE	0	0	0
/var/lib/zfs_img/zfs3.img	ONLINE	0	0	0
/var/lib/zfs_img/zfs1.img	ONLINE	0	0	0
/var/lib/zfs_img/zfs2.img	ONLINE	0	0	0

```
errors: No known data errors
```

Zpool version update

With every update of `sys-fs/zfs` (<https://packages.gentoo.org/packages/sys-fs/zfs>), you are likely to also get a more recent ZFS version. Also the status of your zpools will indicate a warning that a new version is available and the zpools could be upgraded. To display the current version on a zpool:

```
root # zpool upgrade -v
```

This system supports ZFS pool feature flags.

The following features are supported:

FEAT DESCRIPTION

```
-----
async_destroy          (read-only compatible)
    Destroy filesystems asynchronously.
empty_bpobj            (read-only compatible)
    Snapshots use less space.
lz4_compress
    LZ4 compression algorithm support.
```

The following legacy versions are also supported:

VER DESCRIPTION

```
--- -----
1  Initial ZFS version
2  Ditto blocks (replicated metadata)
3  Hot spares and double parity RAID-Z
4  zpool history
5  Compression using the gzip algorithm
6  bootfs pool property
7  Separate intent log devices
8  Delegated administration
9  refquota and refreservation properties
10 Cache devices
11 Improved scrub performance
12 Snapshot properties
13 snapused property
14 passthrough-x aclinherit
15 user/group space accounting
16 stmf property support
17 Triple-parity RAID-Z
18 Snapshot user holds
19 Log device removal
20 Compression using zle (zero-length encoding)
21 Deduplication
22 Received properties
23 Slim ZIL
```

- 24 System attributes
- 25 Improved scrub stats
- 26 Improved snapshot deletion performance
- 27 Improved snapshot creation performance
- 28 Multiple vdev replacements

For more information on a particular version, including supported releases, see the ZFS Administration Guide.

⚠ Warning

systems with a lower version installed will not be able to import a zpool of a higher version

To upgrade the version of zpool *zfs_test*:

```
root # zpool upgrade zfs_test
```

To upgrade the version of all zpools in the system:

```
root # zpool upgrade -a
```

Zpool tips/tricks

- You cannot shrink a zpool and remove vdevs after its initial creation.
- It is possible to add more vdevs to a MIRROR after its initial creation. Use the following command (/dev/loop0 is the first drive in the MIRROR):

```
root # zpool attach zfs_test /dev/loop0 /dev/loop2
```

- More than 9 vdevs in one RAIDZ could cause performance regression. For example it is better to use 2xRAIDZ with each five vdevs rather than 1xRAIDZ with 10 vdevs in a zpool
- RAIDZ1 and RAIDZ2 cannot be resized after initial creation (you may only add additional hot spares). You can, however, replace the hard drives with bigger ones (one at a time), e.g. replace 1T drives with 2T drives to double the available space in the zpool.
- It is possible to mix MIRROR, RAIDZ1 and RAIDZ2 in a zpool. For example to add two more vdevs in a MIRROR in a zpool with RAIDZ1 named *zfs_test*, use:

```
root # zpool add -f zfs_test mirror /dev/loop4 /dev/loop5
```

📌 Note

this needs the -f option

- It is possible to restore a destroyed zpool, by reimporting it straight after the accident happened:

```
root # zpool import -D
```

```
pool: zfs_test
id: 12744221975042547640
state: ONLINE (DESTROYED)
action: The pool can be imported using its name or numeric identifier.
```

Note

the option -D searches on all hard drives for existing zpools

Volumes

The program `/usr/sbin/zfs` is used for any operation regarding *volumes*. To control the size of a volume you can set quota and you can reserve a certain amount of storage within a zpool. By default zpool uses the full storage size.

Create Volumes

We use our zpool *zfs_test* to create a new volume called *volume1*:

```
root # zfs create zfs_test/volume1
```

The volume will be mounted automatically as `/zfs_test/volumes1/`

```
root # zfs list
```

Mount/umount volumes

Volumes can be mounted with the following command, the mountpoint is defined by the property *mountpoint* of the volume:

```
root # zfs mount zfs_test/volume1
```

To unmount the volume:

```
root # zfs unmount zfs_test/volume1
```

The folder `/zfs_test/volume1` stays without the volume behind it. If you write data to it and then try to mount the volume again, you will see the following error message:

CODE

```
cannot mount '/zfs_test/volume1': directory is not empty
```

Remove volumes

To remove volumes *volume1* from zpool *zfs_test*:

```
root # zfs destroy zfs_test/volume1
```

```
root # zfs list
```

Note

you cannot destroy a volume if there exist any snapshots of it

Properties

Properties for volumes are inherited from the zpool. So you can either change the property on the zpool for all volumes or specifically per individual volume or a mix of both.

To set a property for a volume:

```
root # zfs set <property> zfs_test/volume1
```

To show the setting for a particular property on a volume:

```
root # zfs get <property> zfs_test/volume1
```

Note

The properties are used on a volume e.g. compression, the higher is the version of this volume

You can get a list of all properties set on any zpool with the following command:

```
root # zfs get all
```

This is a partial list of properties that can be set on either zpools or volumes, for a full list see *man zfs*:

Property	Value	Function
quota=	20m,none	set a quota of 20MB for the volume
reservation=	20m,none	reserves 20MB for the volume within it's zpool
compression=	zle,gzip,on,off	uses the given compression method or the default method for compression which should be gzip
sharenfs=	on,off,ro,nfsoptions	shares the volume via NFS
exec=	on,off	controls if programs can be executed on the volume
setuid=	on,off	controls if SUID or GUID can be set on the volume
readonly=	on,off	sets read only attribute to on/off
atime=	on,off	update access times for files in the volume
dedup=	on,off	sets deduplication on or off
mountpoint=	none,path	sets the mountpoint for the volume below the zpool or elsewhere in the file system, a mountpoint set to <i>none</i> prevents the volume from being mounted

Set mountpoint

Set the mountpoint for a volume, use the following command:

```
root # zfs set mountpoint=/mnt/data zfs_test/volume1
```

The volume will be automatically moved to /mnt/data

NFS volume share

Activate NFS share on volume:

```
root # zfs set sharenfs=on zfs_test/volume2
```

```
root # exportfs
```

Per default the volume is shared using the exportfs command in the following manner. See exportfs(8) (<https://linux.die.net/man/8/exportfs>) and exports(5) (<https://linux.die.net/man/5/exports>) for more information.

CODE sharenfs default options

```
/usr/sbin/exportfs -i -o sec=sys,rw,no_subtree_check,no_root_squash,mountpoint *:<mountpoint of dataset>
```

Otherwise, the command is invoked with options equivalent to the contents of this property:

```
root # zfs set sharenfs="no_root_squash,rw=@192.168.11.0/24" zfs_test/volume2
```

```
root # exportfs
```

To stop sharing the volume:

```
root # zfs set sharenfs=off zfs_test/volume2
```

```
root # exportfs
```

Snapshots

Snapshots are volumes which have no initial size and save changes made to another volume. With increasing changes between the snapshot and the original volume it grows in size.

Creating

To create a snapshot of a volume, use the following command:

```
root # zfs snapshot zfs_test/volume1@22082011
```

Note

volume1@22082011 is the full name of the snapshot, everything after the @ symbol can be any alphanumeric combination

Every time a file in *volume1* changes, the old data of the file will be linked into the snapshot.

Listing

List all available snapshots:

```
root # zfs list -t snapshot -o name,creation
```

Rollback

To rollback a full volume to a previous state:

```
root # zfs rollback zfs_test/volume1@21082011
```

Note

if there are other snapshots in between, then you have to use the -r option. This would remove all snapshots between the one you want to rollback and the original volume

Cloning

ZFS can clone snapshots to new volumes, so you can access the files from previous states individually:

```
root # zfs clone zfs_test/volume1@21082011 zfs_test/volume1_restore
```

In the folder /zfs_test/volume1_restore can now be worked on in the version of a previous state

Removal

Remove snapshots of a volume with the following command:

```
root # zfs destroy zfs_test/volume1@21082011
```

Maintenance

Scrubbing

Start a scrubbing for zpool *zfs_test*:

```
root # zpool scrub zfs_test
```

Note

this might take some time and is quite I/O intensive

Log Files

To check the history of commands that were executed:

```
root # zpool history
```

Monitor I/O

Monitor I/O activity on all zpools (refreshes every 6 seconds):

```
root # zpool iostat 6
```

ZFS root

To boot from a ZFS volume as your root filesystem you will need a ZFS capable kernel and an initial ramdisk (initramfs) which has the ZFS userspace utilities. The easiest way to set this up is as follows.

First, make sure you have compiled a kernel with ZFS support and used `make install` to copy it to `/boot/` and `make modules_install` to make the modules available at boot time.

Then, install the very latest (`git HEAD`) version of `genkernel` .

```
root # emerge -av =sys-kernel/genkernel-9999
```

```
root # genkernel initramfs --zfs
```

Then configure your boot loader to use the new initramfs and kernel. I use grub.

```
root # emerge -av =sys-boot/grub-9999
```

Tell grub that you want to use ZFS, and which volume to boot from.

FILE `/etc/default/grub` **Specify real_root device**

```
GRUB_CMDLINE_LINUX="dozfs real_root=ZFS=mypool/myvolume"
```

Finally, create the grub configuration and install grub to your boot device.

```
root # grub-mkconfig -o /boot/grub/grub.cfg
```

```
root # grub-install /dev/nvme0n1 # or your alternative boot device
```

You should now be able to reboot in to Gentoo running on ZFS root.

Caveats

- Swap: Due to the Linux kernel's design, swap is recommended.
 - 2013 information (out of date?): "However, swap on a zvol can deadlock under situations involving high memory pressure. It is better to place swap on a dedicated device until [zfsonlinux/zfs#1526](https://github.com/zfsonlinux/zfs/issues/1526) (<https://github.com/zfsonlinux/zfs/issues/1526>) is fixed."
- Memory fragmentation: Memory fragmentation on Linux can cause memory allocations to consume more memory than is actually measured, which means that actual ARC memory usage can exceed `zfs_arc_max` by a constant factor. This effect will be dramatically reduced once [zfsonlinux/zfs#75](https://github.com/zfsonlinux/zfs/issues/75) (<https://github.com/zfsonlinux/zfs/issues/75>) is fixed. Recent versions of ZFS on Linux include the `arcstats.py` script which allows you to monitor ARC usage.

See also

- [btrfs \(/wiki/Btrfs\)](/wiki/Btrfs) - A file system similar to ZFS, but without licensing issues.

External resources

- zfs-fuse.net (<http://zfs-fuse.net/>)
- [ZFS on Linux](http://zfsonlinux.org/) (<http://zfsonlinux.org/>)
- [ZFS Best Practices Guide](https://www.serverfocus.org/zfs-best-practices-guide) (<https://www.serverfocus.org/zfs-best-practices-guide>)
- [ZFS Evil Tuning Guide](https://www.serverfocus.org/zfs-evil-tuning-guide) (<https://www.serverfocus.org/zfs-evil-tuning-guide>)
- [article about ZFS on Linux/Gentoo \(german\)](http://www.pro-linux.de/artikel/2/1181/zfs-unter-linux.html) (<http://www.pro-linux.de/artikel/2/1181/zfs-unter-linux.html>)
- [ZFS Administration](https://pthree.org/2012/12/04/zfs-administration-part-i-vdevs/) (<https://pthree.org/2012/12/04/zfs-administration-part-i-vdevs/>)

Retrieved from "<http://wiki.gentoo.org/index.php?title=ZFS&oldid=706506> (<http://wiki.gentoo.org/index.php?title=ZFS&oldid=706506>)"

Categories (/wiki/Special:Categories):

[Pages with syntax highlighting errors](/index.php?title=Category:Pages_with_syntax_highlighting_errors&action=edit&redlink=1) (/index.php?title=Category:Pages_with_syntax_highlighting_errors&action=edit&redlink=1)

[Filesystems](/wiki/Category:Filesystems) (/wiki/Category:Filesystems)

- This page was last modified on 28 January 2018, at 04:05.

© 2001–2018 Gentoo Foundation, Inc.

Gentoo is a trademark of the Gentoo Foundation, Inc. The contents of this document, unless otherwise expressly stated, are licensed under the [CC-BY-SA-3.0](https://creativecommons.org/licenses/by-sa/3.0/) (<https://creativecommons.org/licenses/by-sa/3.0/>) license. The [Gentoo Name and Logo Usage Guidelines](https://www.gentoo.org/inside-gentoo/foundation/name-logo-guidelines.html) (<https://www.gentoo.org/inside-gentoo/foundation/name-logo-guidelines.html>) apply.