NAME

zfs - configure ZFS datasets

SYNOPSIS

zfs -?V
zfs version
zfs subcommand [*arguments*]

DESCRIPTION

The **zfs** command configures ZFS datasets within a ZFS storage pool, as described in zpool(8). A dataset is identified by a unique path within the ZFS namespace:

pool[/*component*]/*component*

for example:

```
rpool/var/log
```

The maximum length of a dataset name is **ZFS_MAX_DATASET_NAME_LEN -** 1 ASCII characters (currently 255) satisfying **[A-Za-z_.:/ -]**. Additionally snapshots are allowed to contain a single **@** character, while bookmarks are allowed to contain a single **#** character. **/** is used as separator between components. The maximum amount of nesting allowed in a path is **zfs_max_dataset_nesting** levels deep. ZFS tunables (**zfs_***) are explained in zfs(4).

A dataset can be one of the following:

file system Can be mounted within the standard system namespace and behaves like other file systems. While ZFS file systems are designed to be POSIX-compliant, known issues exist that prevent compliance in some cases. Applications that depend on standards conformance might fail due to non-standard behavior when checking file system free space.

volume A logical volume exported as a raw or block device. This type of dataset should only be used when a block device is required. File systems are typically used in most environments.

snapshot A read-only version of a file system or volume at a given point in time. It is specified as *filesystem@name* or *volume@name*.

bookmark Much like a **snapshot**, but without the hold on on-disk data. It can be used as the source of a send (but not for a receive). It is specified as *filesystem#name* or *volume#name*.

See zfsconcepts(7) for details.

Properties

Properties are divided into two types: native properties and user-defined (or "user") properties. Native properties either export internal statistics or control ZFS behavior. In addition, native properties are either editable or read-only. User properties have no effect on ZFS behavior, but you can use them to annotate datasets in a way that is meaningful in your environment. For more information about properties, see zfsprops(7).

Encryption

Enabling the **encryption** feature allows for the creation of encrypted filesystems and volumes. ZFS will encrypt file and zvol data, file attributes, ACLs, permission bits, directory listings, FUID mappings, and **userused**/**groupused**/**projectused** data. For an overview of encryption, see zfs-load-key(8).

SUBCOMMANDS

All subcommands that modify state are logged persistently to the pool in their original form.

zfs -?

Displays a help message.

zfs -V, **--version**

zfs version

Displays the software version of the **zfs** userland utility and the zfs kernel module.

Dataset Management

zfs-list(8)

Lists the property information for the given datasets in tabular form.

zfs-create(8)

Creates a new ZFS file system or volume.

zfs-destroy(8)

Destroys the given dataset(s), snapshot(s), or bookmark.

zfs-rename(8)

Renames the given dataset (filesystem or snapshot).

zfs-upgrade(8)
Manage upgrading the on-disk version of filesystems.

Snapshots

zfs-snapshot(8)
Creates snapshots with the given names.

zfs-rollback(8)
Roll back the given dataset to a previous snapshot.

zfs-hold(8)/zfs-release(8)
Add or remove a hold reference to the specified snapshot or snapshots. If a hold exists on a snapshot, attempts to destroy that snapshot by using the **zfs destroy** command return **EBUSY**.

zfs-diff(8)
Display the difference between a snapshot of a given filesystem and another snapshot of that filesystem from a later time or the current contents of the filesystem.

Clones

zfs-clone(8)
Creates a clone of the given snapshot.

zfs-promote(8)
Promotes a clone file system to no longer be dependent on its "origin" snapshot.

Send & Receive

zfs-send(8)
Generate a send stream, which may be of a filesystem, and may be incremental from a bookmark.

zfs-receive(8)
Creates a snapshot whose contents are as specified in the stream provided on standard input. If a full stream is received, then a new file system is created as well. Streams are created using the zfs-send(8) subcommand, which by default creates a full stream.

zfs-bookmark(8)
Creates a new bookmark of the given snapshot or bookmark. Bookmarks mark the point in time when the snapshot was created, and can be used as the incremental source for a **zfs send** command.

zfs-redact(8)
Generate a new redaction bookmark. This feature can be used to allow clones of a filesystem to be

made available on a remote system, in the case where their parent need not (or needs to not) be usable.

Properties

zfs-get(8)
: Displays properties for the given datasets.

zfs-set(8)
: Sets the property or list of properties to the given value(s) for each dataset.

zfs-inherit(8)
: Clears the specified property, causing it to be inherited from an ancestor, restored to default if no ancestor has the property set, or with the **-S** option reverted to the received value if one exists.

Quotas

zfs-userspace(8)/zfs-groupspace(8)/zfs-projectspace(8)
: Displays space consumed by, and quotas on, each user, group, or project in the specified filesystem or snapshot.

zfs-project(8)
: List, set, or clear project ID and/or inherit flag on the files or directories.

Mountpoints

zfs-mount(8)
: Displays all ZFS file systems currently mounted, or mount ZFS filesystem on a path described by its **mountpoint** property.

zfs-unmount(8)
: Unmounts currently mounted ZFS file systems.

Shares

zfs-share(8)
: Shares available ZFS file systems.

zfs-unshare(8)
: Unshares currently shared ZFS file systems.

Delegated Administration

zfs-allow(8)
: Delegate permissions on the specified filesystem or volume.

zfs-unallow(8)
Remove delegated permissions on the specified filesystem or volume.

Encryption

zfs-change-key(8)
Add or change an encryption key on the specified dataset.

zfs-load-key(8)
Load the key for the specified encrypted dataset, enabling access.

zfs-unload-key(8)
Unload a key for the specified dataset, removing the ability to access the dataset.

Channel Programs

zfs-program(8)
Execute ZFS administrative operations programmatically via a Lua script-language channel program.

Jails

zfs-jail(8)
Attaches a filesystem to a jail.

zfs-unjail(8)
Detaches a filesystem from a jail.

Waiting

zfs-wait(8)
Wait for background activity in a filesystem to complete.

EXIT STATUS

The **zfs** utility exits **0** on success, **1** if an error occurs, and **2** if invalid command line options were specified.

EXAMPLES

Example 1: Creating a ZFS File System Hierarchy

The following commands create a file system named *pool/home* and a file system named *pool/home/bob*. The mount point */export/home* is set for the parent file system, and is automatically inherited by the child file system.

```
# zfs create pool/home
# zfs set mountpoint=/export/home pool/home
# zfs create pool/home/bob
```

Example 2: Creating a ZFS Snapshot

The following command creates a snapshot named *yesterday*. This snapshot is mounted on demand in the *.zfs/snapshot* directory at the root of the *pool/home/bob* file system.

```
# zfs snapshot pool/home/bob@yesterday
```

Example 3: Creating and Destroying Multiple Snapshots

The following command creates snapshots named *yesterday* of *pool/home* and all of its descendent file systems. Each snapshot is mounted on demand in the *.zfs/snapshot* directory at the root of its file system. The second command destroys the newly created snapshots.

```
# zfs snapshot -r pool/home@yesterday
# zfs destroy -r pool/home@yesterday
```

Example 4: Disabling and Enabling File System Compression

The following command disables the **compression** property for all file systems under *pool/home*. The next command explicitly enables **compression** for *pool/home/anne*.

```
# zfs set compression=off pool/home
# zfs set compression=on pool/home/anne
```

Example 5: Listing ZFS Datasets

The following command lists all active file systems and volumes in the system. Snapshots are displayed if **listsnaps**=**on**. The default is **off**. See zpoolprops(7) for more information on pool properties.

```
# zfs list
NAME              USED  AVAIL  REFER  MOUNTPOINT
pool              450K   457G    18K  /pool
pool/home         315K   457G    21K  /export/home
pool/home/anne     18K   457G    18K  /export/home/anne
pool/home/bob     276K   457G   276K  /export/home/bob
```

Example 6: Setting a Quota on a ZFS File System

The following command sets a quota of 50 Gbytes for *pool/home/bob*:

```
# zfs set quota=50G pool/home/bob
```

Example 7: Listing ZFS Properties

The following command lists all properties for *pool/home/bob*:

```
# zfs get all pool/home/bob
NAME           PROPERTY              VALUE                  SOURCE
pool/home/bob  type                  filesystem             -
pool/home/bob  creation              Tue Jul 21 15:53 2009  -
pool/home/bob  used                  21K                    -
pool/home/bob  available             20.0G                  -
```

```
      pool/home/bob  referenced          21K                -
      pool/home/bob  compressratio       1.00x              -
      pool/home/bob  mounted             yes                -
      pool/home/bob  quota               20G                local
      pool/home/bob  reservation         none               default
      pool/home/bob  recordsize          128K               default
      pool/home/bob  mountpoint          /pool/home/bob     default
      pool/home/bob  sharenfs            off                default
      pool/home/bob  checksum            on                 default
      pool/home/bob  compression         on                 local
      pool/home/bob  atime               on                 default
      pool/home/bob  devices             on                 default
      pool/home/bob  exec                on                 default
      pool/home/bob  setuid              on                 default
      pool/home/bob  readonly            off                default
      pool/home/bob  zoned               off                default
      pool/home/bob  snapdir             hidden             default
      pool/home/bob  acltype             off                default
      pool/home/bob  aclmode             discard            default
      pool/home/bob  aclinherit          restricted         default
      pool/home/bob  canmount            on                 default
      pool/home/bob  xattr               on                 default
      pool/home/bob  copies              1                  default
      pool/home/bob  version             4                  -
      pool/home/bob  utf8only            off                -
      pool/home/bob  normalization       none               -
      pool/home/bob  casesensitivity     sensitive          -
      pool/home/bob  vscan               off                default
      pool/home/bob  nbmand              off                default
      pool/home/bob  sharesmb            off                default
      pool/home/bob  refquota            none               default
      pool/home/bob  refreservation      none               default
      pool/home/bob  primarycache        all                default
      pool/home/bob  secondarycache      all                default
      pool/home/bob  usedbysnapshots     0                  -
      pool/home/bob  usedbydataset       21K                -
      pool/home/bob  usedbychildren      0                  -
      pool/home/bob  usedbyrefreservation  0                -
```

The following command gets a single property value:

```
# zfs get -H -o value compression pool/home/bob
on
```

The following command lists all properties with local settings for *pool/home/bob*:

```
# zfs get -r -s local -o name,property,value all pool/home/bob
NAME           PROPERTY              VALUE
pool/home/bob  quota                 20G
pool/home/bob  compression           on
```

Example 8: Rolling Back a ZFS File System

The following command reverts the contents of *pool/home/anne* to the snapshot named *yesterday*, deleting all intermediate snapshots:

```
# zfs rollback -r pool/home/anne@yesterday
```

Example 9: Creating a ZFS Clone

The following command creates a writable file system whose initial contents are the same as *pool/home/bob@yesterday*.

```
# zfs clone pool/home/bob@yesterday pool/clone
```

Example 10: Promoting a ZFS Clone

The following commands illustrate how to test out changes to a file system, and then replace the original file system with the changed one, using clones, clone promotion, and renaming:

```
# zfs create pool/project/production
  populate /pool/project/production with data
# zfs snapshot pool/project/production@today
# zfs clone pool/project/production@today pool/project/beta
  make changes to /pool/project/beta and test them
# zfs promote pool/project/beta
# zfs rename pool/project/production pool/project/legacy
# zfs rename pool/project/beta pool/project/production
  once the legacy version is no longer needed, it can be destroyed
# zfs destroy pool/project/legacy
```

Example 11: Inheriting ZFS Properties

The following command causes *pool/home/bob* and *pool/home/anne* to inherit the **checksum** property from their parent.

```
# zfs inherit checksum pool/home/bob pool/home/anne
```

Example 12: Remotely Replicating ZFS Data

The following commands send a full stream and then an incremental stream to a remote machine, restoring them into *poolB/received/fs@a* and *poolB/received/fs@b*, respectively. *poolB* must contain the file system *poolB/received*, and must not initially contain *poolB/received/fs*.

```
# zfs send pool/fs@a |
  ssh host zfs receive poolB/received/fs@a
# zfs send -i a pool/fs@b |
  ssh host zfs receive poolB/received/fs
```

Example 13: Using the **zfs receive -d** Option

The following command sends a full stream of *poolA/fsA/fsB@snap* to a remote machine, receiving it into *poolB/received/fsA/fsB@snap*. The *fsA/fsB@snap* portion of the received snapshot’s name is determined from the name of the sent snapshot. *poolB* must contain the file system *poolB/received*. If *poolB/received/fsA* does not exist, it is created as an empty file system.

```
# zfs send poolA/fsA/fsB@snap |
  ssh host zfs receive -d poolB/received
```

Example 14: Setting User Properties

The following example sets the user-defined *com.example*:*department* property for a dataset:

```
# zfs set com.example:department=12345 tank/accounting
```

Example 15: Performing a Rolling Snapshot

The following example shows how to maintain a history of snapshots with a consistent naming scheme. To keep a week’s worth of snapshots, the user destroys the oldest snapshot, renames the remaining snapshots, and then creates a new snapshot, as follows:

```
# zfs destroy -r pool/users@7daysago
# zfs rename -r pool/users@6daysago @7daysago
# zfs rename -r pool/users@5daysago @6daysago
# zfs rename -r pool/users@4daysago @5daysago
# zfs rename -r pool/users@3daysago @4daysago
# zfs rename -r pool/users@2daysago @3daysago
# zfs rename -r pool/users@yesterday @2daysago
# zfs rename -r pool/users@today @yesterday
# zfs snapshot -r pool/users@today
```

Example 16: Setting sharenfs Property Options on a ZFS File System

The following commands show how to set **sharenfs** property options to enable read-write access for a set of IP addresses and to enable root access for system "neo" on the *tank/home* file system:

```
# zfs set sharenfs=’rw=@123.123.0.0/16:[::1],root=neo’ tank/home
```

If you are using DNS for host name resolution, specify the fully-qualified hostname.

Example 17: Delegating ZFS Administration Permissions on a ZFS Dataset

The following example shows how to set permissions so that user *cindys* can create, destroy, mount, and take snapshots on *tank/cindys*. The permissions on *tank/cindys* are also displayed.

```
# zfs allow cindys create,destroy,mount,snapshot tank/cindys
# zfs allow tank/cindys
---- Permissions on tank/cindys --------------------------------------
Local+Descendent permissions:
        user cindys create,destroy,mount,snapshot
```

Because the *tank/cindys* mount point permission is set to 755 by default, user *cindys* will be unable to mount file systems under *tank/cindys*. Add an ACE similar to the following syntax to provide mount point access:

```
# chmod A+user:cindys:add_subdirectory:allow /tank/cindys
```

Example 18: Delegating Create Time Permissions on a ZFS Dataset

The following example shows how to grant anyone in the group *staff* to create file systems in *tank/users*. This syntax also allows staff members to destroy their own file systems, but not destroy anyone else's file system. The permissions on *tank/users* are also displayed.

```
# zfs allow staff create,mount tank/users
# zfs allow -c destroy tank/users
# zfs allow tank/users
---- Permissions on tank/users ---------------------------------------
Permission sets:
        destroy
Local+Descendent permissions:
        group staff create,mount
```

Example 19: Defining and Granting a Permission Set on a ZFS Dataset

The following example shows how to define and grant a permission set on the *tank/users* file system. The permissions on *tank/users* are also displayed.

```
# zfs allow -s @pset create,destroy,snapshot,mount tank/users
# zfs allow staff @pset tank/users
# zfs allow tank/users
---- Permissions on tank/users ---------------------------------------
Permission sets:
        @pset create,destroy,mount,snapshot
Local+Descendent permissions:
        group staff @pset
```

Example 20: Delegating Property Permissions on a ZFS Dataset

The following example shows to grant the ability to set quotas and reservations on the *users/home* file system. The permissions on *users/home* are also displayed.

```
# zfs allow cindys quota,reservation users/home
# zfs allow users/home
---- Permissions on users/home ---------------------------------------
Local+Descendent permissions:
        user cindys quota,reservation
cindys% zfs set quota=10G users/home/marks
cindys% zfs get quota users/home/marks
NAME              PROPERTY  VALUE  SOURCE
users/home/marks  quota     10G    local
```

Example 21: Removing ZFS Delegated Permissions on a ZFS Dataset
The following example shows how to remove the snapshot permission from the *staff* group on the **tank/users** file system. The permissions on **tank/users** are also displayed.

```
# zfs unallow staff snapshot tank/users
# zfs allow tank/users
---- Permissions on tank/users ---------------------------------------
Permission sets:
        @pset create,destroy,mount,snapshot
Local+Descendent permissions:
        group staff @pset
```

Example 22: Showing the differences between a snapshot and a ZFS Dataset
The following example shows how to see what has changed between a prior snapshot of a ZFS dataset and its current state. The **-F** option is used to indicate type information for the files affected.

```
# zfs diff -F tank/test@before tank/test
M       /       /tank/test/
M       F       /tank/test/linked      (+1)
R       F       /tank/test/oldname -> /tank/test/newname
-       F       /tank/test/deleted
+       F       /tank/test/created
M       F       /tank/test/modified
```

Example 23: Creating a bookmark
The following example creates a bookmark to a snapshot. This bookmark can then be used instead of a snapshot in send streams.

```
# zfs bookmark rpool@snapshot rpool#bookmark
```

Example 24: Setting **sharesmb** Property Options on a ZFS File System

The following example show how to share SMB filesystem through ZFS. Note that a user and their password must be given.

```
# smbmount //127.0.0.1/share_tmp /mnt/tmp -o user=workgroup/turbo,password=obrut,uid=1000
```

Minimal */etc/samba/smb.conf* configuration is required, as follows.

Samba will need to bind to the loopback interface for the ZFS utilities to communicate with Samba. This is the default behavior for most Linux distributions.

Samba must be able to authenticate a user. This can be done in a number of ways (passwd(5), LDAP, smbpasswd(5), &c.). How to do this is outside the scope of this document - refer to smb.conf(5) for more information.

See the *USERSHARES* section for all configuration options, in case you need to modify any options of the share afterwards. Do note that any changes done with the net(8) command will be undone if the share is ever unshared (like via a reboot).

ENVIRONMENT VARIABLES

ZFS_COLOR Use ANSI color in **zfs diff** and **zfs list** output.

ZFS_MOUNT_HELPER Cause **zfs mount** to use mount(8) to mount ZFS datasets. This option is provided for backwards compatibility with older ZFS versions.

ZFS_SET_PIPE_MAX Tells **zfs** to set the maximum pipe size for sends/recieves. Disabled by default on Linux due to an unfixed deadlock in Linux's pipe size handling code.

ZFS_MODULE_TIMEOUT Time, in seconds, to wait for */dev/zfs* to appear. Defaults to **10**, max **600** (10 minutes). If <**0**, wait forever; if **0**, don't wait.

INTERFACE STABILITY

Committed.

SEE ALSO

attr(1), gzip(1), ssh(1), chmod(2), fsync(2), stat(2), write(2), acl(5), attributes(5), exports(5), zfsconcepts(7), zfsprops(7), exportfs(8), mount(8), net(8), selinux(8), zfs-allow(8), zfs-bookmark(8), zfs-change-key(8), zfs-clone(8), zfs-create(8), zfs-destroy(8), zfs-diff(8), zfs-get(8), zfs-groupspace(8), zfs-hold(8), zfs-inherit(8), zfs-jail(8), zfs-list(8), zfs-load-key(8), zfs-mount(8), zfs-program(8), zfs-project(8), zfs-projectspace(8), zfs-promote(8), zfs-receive(8), zfs-redact(8), zfs-release(8), zfs-rename(8), zfs-rollback(8), zfs-send(8), zfs-set(8), zfs-share(8), zfs-snapshot(8), zfs-unallow(8), zfs-unjail(8), zfs-unload-key(8), zfs-unmount(8), zfs-unshare(8), zfs-upgrade(8), zfs-userspace(8), zfs-wait(8), zpool(8)