

Managing ZFS File Systems in Oracle® Solaris 11.3



Part No: E54801
November 2016

Part No: E54801

Copyright © 2006, 2016, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Référence: E54801

Copyright © 2006, 2016, Oracle et/ou ses affiliés. Tous droits réservés.

Ce logiciel et la documentation qui l'accompagne sont protégés par les lois sur la propriété intellectuelle. Ils sont concédés sous licence et soumis à des restrictions d'utilisation et de divulgation. Sauf stipulation expresse de votre contrat de licence ou de la loi, vous ne pouvez pas copier, reproduire, traduire, diffuser, modifier, accorder de licence, transmettre, distribuer, exposer, exécuter, publier ou afficher le logiciel, même partiellement, sous quelque forme et par quelque procédé que ce soit. Par ailleurs, il est interdit de procéder à toute ingénierie inverse du logiciel, de le désassembler ou de le décompiler, excepté à des fins d'interopérabilité avec des logiciels tiers ou tel que prescrit par la loi.

Les informations fournies dans ce document sont susceptibles de modification sans préavis. Par ailleurs, Oracle Corporation ne garantit pas qu'elles soient exemptes d'erreurs et vous invite, le cas échéant, à lui en faire part par écrit.

Si ce logiciel, ou la documentation qui l'accompagne, est livré sous licence au Gouvernement des Etats-Unis, ou à quiconque qui aurait souscrit la licence de ce logiciel pour le compte du Gouvernement des Etats-Unis, la notice suivante s'applique:

U.S. GOVERNMENT END USERS. Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

Ce logiciel ou matériel a été développé pour un usage général dans le cadre d'applications de gestion des informations. Ce logiciel ou matériel n'est pas conçu ni n'est destiné à être utilisé dans des applications à risque, notamment dans des applications pouvant causer des dommages corporels. Si vous utilisez ce logiciel ou matériel dans le cadre d'applications dangereuses, il est de votre responsabilité de prendre toutes les mesures de secours, de sauvegarde, de redondance et autres mesures nécessaires à son utilisation dans des conditions optimales de sécurité. Oracle Corporation et ses affiliés déclinent toute responsabilité quant aux dommages causés par l'utilisation de ce logiciel ou matériel pour ce type d'applications.

Oracle et Java sont des marques déposées d'Oracle Corporation et/ou de ses affiliés. Tout autre nom mentionné peut correspondre à des marques appartenant à d'autres propriétaires qu'Oracle.

Intel et Intel Xeon sont des marques ou des marques déposées d'Intel Corporation. Toutes les marques SPARC sont utilisées sous licence et sont des marques ou des marques déposées de SPARC International, Inc. AMD, Opteron, le logo AMD et le logo AMD Opteron sont des marques ou des marques déposées d'Advanced Micro Devices. UNIX est une marque déposée d'The Open Group.

Ce logiciel ou matériel et la documentation qui l'accompagne peuvent fournir des informations ou des liens donnant accès à des contenus, des produits et des services émanant de tiers. Oracle Corporation et ses affiliés déclinent toute responsabilité ou garantie expresse quant aux contenus, produits ou services émanant de tiers, sauf mention contraire stipulée dans un contrat entre vous et Oracle. En aucun cas, Oracle Corporation et ses affiliés ne sauraient être tenus pour responsables des pertes subies, des coûts occasionnés ou des dommages causés par l'accès à des contenus, produits ou services tiers, ou à leur utilisation, sauf mention contraire stipulée dans un contrat entre vous et Oracle.

Accessibilité de la documentation

Pour plus d'informations sur l'engagement d'Oracle pour l'accessibilité à la documentation, visitez le site Web Oracle Accessibility Program, à l'adresse <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> ou le site <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> si vous êtes malentendant.

Contents

Using This Documentation	13
 1 Introducing the Oracle Solaris ZFS File System	15
What's New in ZFS for Oracle Solaris 11.3?	15
Oracle Solaris ZFS Features	15
Components of a ZFS Storage Pool	16
Using Disks in a ZFS Storage Pool	17
Using Files in a ZFS Storage Pool	18
Redundancy Features of a ZFS Storage Pool	18
Mirrored Storage Pool Configuration	19
RAID-Z Storage Pool Configuration	19
ZFS Hybrid Storage Pool	20
Dynamic Striping in a Storage Pool	20
 2 Getting Started With Oracle Solaris ZFS	21
Hardware and Software Requirements	21
Planning the ZFS Implementation	22
Naming ZFS Components	22
Identifying Storage Requirements	22
Choosing the Type of Data Redundancy	23
Determining the ZFS File System Hierarchy	23
 3 Creating and Destroying Oracle Solaris ZFS Storage Pools	25
Creating ZFS Storage Pools	25
▼ How to Set Up ZFS on a System	26
Creating a Mirrored Storage Pool	30
Creating a RAID-Z Storage Pool	30
Creating a ZFS Storage Pool With Log Devices	31
Creating a ZFS Storage Pool With Cache Devices	32
Doing a Dry Run of Storage Pool Creation	33

Handling ZFS Storage Pool Creation Issues	34
Destroying ZFS Storage Pools	36
4 Managing Devices in Oracle Solaris ZFS Storage Pools	37
Adding Devices to a Storage Pool	37
Removing Devices From a Storage Pool	40
Attaching and Detaching Devices in a Storage Pool	41
Splitting a Mirrored Storage Pool to Create a New Pool	42
Taking Devices in a Storage Pool Offline or Returning Online	44
Clearing Storage Pool Device Errors	46
Replacing Devices in a Storage Pool	46
▼ How to Replace a Device in a Storage Pool	47
Working With Hot Spares in Storage Pools	48
Designating Hot Spares in a Storage Pool	49
Activating and Deactivating Hot Spares in Your Storage Pool	49
5 Managing Oracle Solaris ZFS Storage Pools	53
Managing ZFS Storage Pool Properties	53
Querying ZFS Storage Pool Status	55
Displaying Information About ZFS Storage Pools	55
Viewing I/O Statistics for ZFS Storage Pools	59
Determining the Health Status of ZFS Storage Pools	61
Migrating ZFS Storage Pools	65
Preparing for ZFS Storage Pool Migration	66
Exporting a ZFS Storage Pool	66
Determining Available Storage Pools to Import	66
Importing ZFS Storage Pools	68
Recovering Destroyed ZFS Storage Pools	70
Upgrading ZFS Storage Pools	72
6 Managing the ZFS Root Pool	75
Requirements for Configuring the ZFS Root Pool	75
ZFS Root Pool Space Requirements	75
ZFS Root Pool Configuration Recommendations	76
Installing the ZFS Root Pool	76
Managing a ZFS Root Pool	78
▼ How to Configure a Mirrored Root Pool (SPARC or x86/EFI (GPT))	78
▼ How to Configure a Mirrored Root Pool (SPARC or x86/VTOC)	79
▼ How to Update a ZFS Boot Environment	81

▼ How to Mount an Alternate BE	82
Replacing Disks in a ZFS Root Pool	82
Managing ZFS Swap and Dump Devices	86
Viewing Swap and Dump Information	87
Adjusting the Sizes of ZFS Swap and Dump Devices	89
Troubleshooting ZFS Dump Device Issues	90
Booting From a ZFS Root File System	90
Booting From an Alternate Root Pool Disk	91
Booting From a ZFS Root File System on a SPARC Based System	92
Booting From a ZFS Root File System on an x86 Based System	94
Booting for Recovery Purposes in a ZFS Root Environment	95
7 Managing Oracle Solaris ZFS File Systems	97
Introduction to ZFS File Systems	97
Creating, Destroying, and Renaming ZFS File Systems	98
▼ How to Create a ZFS File System	98
▼ How to Destroy a ZFS File System	99
▼ How to Rename a ZFS File System	100
Introducing ZFS Properties	101
ZFS Read-Only Native Properties	109
Settable ZFS Native Properties	110
ZFS User Properties	115
Querying ZFS File System Information	116
Listing Basic ZFS Information	116
Creating Complex ZFS Queries	117
Listing Incomplete ZFS Datasets	118
Creating Parsable Output with <code>zfs list</code>	119
Managing ZFS Properties	119
Setting ZFS Properties	119
Inheriting ZFS Properties	120
Querying ZFS Properties	121
Mounting ZFS File Systems	124
Managing ZFS Mount Points	124
Mounting ZFS File Systems	126
Using Temporary Mount Properties	128
Unmounting ZFS File Systems	128
Sharing and Unsharing ZFS File Systems	129
Legacy ZFS Sharing Syntax	130
New ZFS Sharing Syntax	131

ZFS Sharing Migration/Transition Issues	137
Troubleshooting ZFS File System Sharing Problems	138
Setting ZFS Quotas and Reservations	140
Setting Quotas on ZFS File Systems	141
Setting Reservations on ZFS File Systems	145
Compressing ZFS File Systems	146
Encrypting ZFS File Systems	147
Changing an Encrypted ZFS File System's Keys	149
Mounting an Encrypted ZFS File System	151
Upgrading Encrypted ZFS File Systems	151
Interactions Between ZFS Compression, Deduplication, and Encryption Properties	152
Examples of Encrypting ZFS File Systems	153
Migrating ZFS File Systems	154
▼ How to Migrate a File System to a ZFS File System	155
Upgrading ZFS File Systems	158
8 Working With Oracle Solaris ZFS Snapshots and Clones	159
Overview of ZFS Snapshots	159
Creating and Destroying ZFS Snapshots	160
Displaying and Accessing ZFS Snapshots	163
Rolling Back a ZFS Snapshot	165
Identifying ZFS Snapshot Differences (<code>zfs diff</code>)	165
Overview of ZFS Clones	167
Creating a ZFS Clone	168
Destroying a ZFS Clone	168
Replacing a ZFS File System With a ZFS Clone	169
Saving, Sending, and Receiving ZFS Data	170
Saving ZFS Data With Other Backup Products	171
Types of ZFS Snapshot Streams	171
Sending a ZFS Snapshot	173
Using Resumable Replication	174
Receiving a ZFS Snapshot	175
Applying Different Property Values to a ZFS Snapshot Stream	176
Sending and Receiving Complex ZFS Snapshot Streams	178
Remote Replication of ZFS Data	181
Monitoring ZFS Pool Operations	181
Copying ZFS Files	184

9 Using ACLs and Attributes to Protect Oracle Solaris ZFS Files	185
Oracle Solaris ACL Model	185
ACL Formats	185
ACL Entry Descriptions	186
ACL Inheritance	189
ACL Properties	190
Setting ACLs on ZFS Files	191
Command Syntax for Setting ACLs	191
Modifying ACLs on ZFS Files	193
ACL Interaction With Permission Bits	194
Setting ACL Inheritance on ZFS Files	197
Granting ACLs That Are Inherited by Files	197
Granting ACLs That Are Inherited by Both Files and Directories	198
Modifying ACL Inheritance With the ACL Inherit Mode	200
Applying Special Attributes to ZFS Files	204
Applying Immutability to a ZFS File	204
Preventing Accidental Deletions With the nounLink Attribute	205
Applying Read-Only Access to a ZFS File	205
Displaying and Changing ZFS File Attributes	205
10 Oracle Solaris ZFS Delegated Administration	207
Overview of ZFS Delegated Administration	207
Disabling ZFS Delegated Permissions	208
Delegating ZFS Permissions	208
Delegating ZFS Permissions (zfs allow)	211
Removing ZFS Delegated Permissions (zfs unallow)	212
Delegating ZFS Permissions Examples	212
Displaying ZFS Delegated Permissions Examples	216
Removing ZFS Delegated Permissions Examples	218
11 Oracle Solaris ZFS Advanced Topics	221
ZFS Volumes	221
Using a ZFS Volume as a Swap or Dump Device	222
Using a ZFS Volume as an iSCSI LUN	222
Using ZFS on an Oracle Solaris System With Zones Installed	224
Adding ZFS File Systems to a Non-Global Zone	225
Delegating Datasets to a Non-Global Zone	226
Adding ZFS Volumes to a Non-Global Zone	226
Using ZFS Storage Pools Within a Zone	227

Managing ZFS Properties Within a Zone	227
Understanding the zoned Property	228
Copying Zones to Other Systems	229
Using a ZFS Pool With an Alternate Root Location	230
Creating a ZFS Pool With an Alternate Root Location	230
Importing a Pool With an Alternate Root Location	231
Importing a Pool With a Temporary Name	231
12 Oracle Solaris ZFS Troubleshooting and Pool Recovery	233
Identifying ZFS Problems	233
Resolving General Hardware Problems	234
Identifying Hardware and Device Faults	234
System Reporting of ZFS Error Messages	235
Identifying Problems With ZFS Storage Pools	236
Determining If Problems Exist in a ZFS Storage Pool	237
Reviewing ZFS Storage Pool Status Information	237
Resolving ZFS Storage Device Problems	241
Resolving a Missing or Removed Device	241
Replacing or Repairing a Damaged Device	245
Changing Pool Devices	255
Resolving Data Problems in a ZFS Storage Pool	255
Resolving ZFS Space Issues	256
Checking ZFS File System Integrity	260
Repairing Corrupted ZFS Data	262
Identifying the Type of Data Corruption	263
Repairing a Corrupted File or Directory	264
Repairing ZFS Storage Pool-Wide Damage	266
Repairing a Damaged ZFS Configuration	267
Repairing an Unbootable System	268
13 Recommended Oracle Solaris ZFS Practices	269
Recommended Storage Pool Practices	269
General System Practices	269
ZFS Storage Pool Creation Practices	271
Storage Pool Practices for Performance	275
ZFS Storage Pool Maintenance and Monitoring Practices	276
Recommended File System Practices	277
Root File System Practices	277
File System Creation Practices	278

Monitoring ZFS File System Practices	278
A Oracle Solaris ZFS Version Descriptions	281
Overview of ZFS Versions	281
ZFS Pool Versions	281
ZFS File System Versions	283
Glossary	285
Index	289

Using This Documentation

- **Overview** – Describes how to set up and administer ZFS file systems.
- **Audience** – System administrators.
- **Required knowledge** – Basic Oracle Solaris or UNIX system administration experience and general file system administration experience.

Product Documentation Library

Documentation and resources for this product and related products are available at <http://www.oracle.com/pls/topic/lookup?ctx=E53394>.

Feedback

Provide feedback about this documentation at <http://www.oracle.com/goto/docfeedback>.

◆◆◆ 1 CHAPTER 1

Introducing the Oracle Solaris ZFS File System

This chapter provides an overview of the Oracle Solaris ZFS file system and its features and benefits. It covers the following information:

- [“What's New in ZFS for Oracle Solaris 11.3?” on page 15](#)
- [“Oracle Solaris ZFS Features” on page 15](#)
- [“Components of a ZFS Storage Pool” on page 16](#)
- [“Redundancy Features of a ZFS Storage Pool” on page 18](#)

What's New in ZFS for Oracle Solaris 11.3?

The following ZFS features are introduced in this Oracle Solaris release.

- You can set a default user or a default group quota to restrict disk space for a large common file system that is used by many different users. For more information, see [“Setting ZFS Quotas and Reservations” on page 140](#).
- You can display recursive ZFS snapshot differences by using the recursive (-r) option. For more information, see [“Displaying Recursive ZFS Snapshot Differences” on page 166](#).
- With the command `zpool monitor`, you can monitor the status and progress of ongoing operations that have been started on ZFS data such sending or receiving of a ZFS stream, data scrubbing and resilvering. For more information, see [“Monitoring ZFS Pool Operations” on page 181](#).
- You can now use LZ4 algorithm when compressing a dataset. For more information, see [“Compressing ZFS File Systems” on page 146](#).
- ZFS cache devices now persist across reboots. For more information, see [“Creating a ZFS Storage Pool With Cache Devices” on page 32](#)

Oracle Solaris ZFS Features

The Oracle Solaris ZFS file system provides features and benefits not found in other file systems. The following table compares the features of the ZFS file system with traditional file systems.

Note - For a more detailed discussion of the differences between ZFS and historical file systems, see [Oracle Solaris ZFS and Traditional File System Differences](#).

TABLE 1 Comparison of the ZFS File System and Traditional File Systems

ZFS File System	Traditional File Systems
Uses concept of <i>storage pools</i> created on devices. The pool size grows as more devices are added to the pool. The additional space is immediately available for use.	Constrained to one device and to the size of that device.
Volume manager unnecessary. Commands configure pools for data redundancy over multiple devices.	Requires volume manager to handle multiple devices to provide data redundancy, which adds to the complexity of administration.
Supports one file system per user or project for easier management.	Uses one file system to manage multiple subdirectories.
Set up and manage many file systems by issuing commands, and directly apply properties that can be inherited by the descendant file systems within the hierarchy. No need to edit the <code>/etc/vfstab</code> file. File system mounts or unmounts are automatic based on file system properties.	Complex administration due to device and size constraints. For example, every time you add a new file system, you must edit the <code>/etc/vfstab</code> file.
You can create a <i>snapshot</i> , a read-only copy of a file system or volume quickly and easily. Initially, snapshots consume no additional disk space within the pool.	
Metadata is allocated dynamically. No pre-allocation or predetermined limits are set. The number of supported file systems is limited only by the available disk space.	Pre-allocation of metadata results in immediate space cost at the creation of the file system. Pre-allocation also predetermines the total number of file systems that can be supported.
Uses transactional semantics, where data management uses <i>copy-on-write</i> semantics, not data overwrite. Any sequence of operations is either entirely committed or entirely ignored. During accidental loss of power or a system crash, most recently written pieces of data might be lost but the file system always remains consistent and uncorrupted.	Overwrites data in place. File system vulnerable to getting into an inconsistent state, for example, if the system loses power between the time a data block is allocated and when it is linked into a directory. Tools such as <code>fsck</code> command or journaling do not always guarantee a fix and can introduce unnecessary overhead.
All checksum verification and data recovery are performed at the file system layer, and are transparent to applications. All failures are detected and recovery can be performed.	Checksum verification, if provided, is performed on a per-block basis. Certain failures, such as writing a complete block to an incorrect location, can result in data that is incorrect but has no checksum errors.
Supports self-healing data through its varying levels of data redundancy. A bad data block can be repaired by replacing it with correct data from another redundant copy.	
ACL (Access Control List) model based on NFSv4 specifications to protect ZFS, similar to NT-style ACL. The model provides a much more granular set of access privileges. Richer inheritance semantics designate how access privileges are applied through the directory hierarchy.	In previous Oracle Solaris releases, ACL implementation was based on POSIX ACL specifications to protect UFS.

Components of a ZFS Storage Pool

This section describes the components used for creating ZFS pools.

Using Disks in a ZFS Storage Pool

The most basic element of a storage pool is physical storage. Physical storage can be any block device of at least 128 MB in size. Typically, this device is a hard drive that is visible to the system in the `/dev/dsk` directory.

A storage device can be a whole disk (`c1t0d0`) or an individual slice (`c0t0d0s7`). From management, reliability, and performance perspectives, using whole disks is the easiest and most efficient way to use ZFS. ZFS formats the whole disk to contain a single, large slice. No special disk formatting is required. With other methods, such as building pools from disk slices, LUNs in hardware RAID arrays, or volumes presented by software-based volume managers, management becomes increasingly complex and might provide less-than-optimal performance.



Caution - Because of potential complexity in managing slices for storage pools, avoid using slices.

The `format` command displays the partition table of disks. When Oracle Solaris is installed on a SPARC® system with GPT aware firmware, an EFI (GPT) label is applied to the disk. The partition table would be similar to the following example:

Current partition table (original):

Total disk sectors available: 143358287 + 16384 (reserved sectors)

Part	Tag	Flag	First Sector	Size	Last Sector
0	usr	wm	256	68.36GB	143358320
1	unassigned	wm	0	0	0
2	unassigned	wm	0	0	0
3	unassigned	wm	0	0	0
4	unassigned	wm	0	0	0
5	unassigned	wm	0	0	0
6	unassigned	wm	0	0	0
8	reserved	wm	143358321	8.00MB	143374704

When Oracle Solaris is installed on an x86 based system, in most cases a EFI (GPT) label is applied to root pool disks. The partition table would be similar to the following:

Current partition table (original):

Total disk sectors available: 27246525 + 16384 (reserved sectors)

Part	Tag	Flag	First Sector	Size	Last Sector
0	BIOS_boot	wm	256	256.00MB	524543
1	usr	wm	524544	12.74GB	27246558
2	unassigned	wm	0	0	0
3	unassigned	wm	0	0	0
4	unassigned	wm	0	0	0
5	unassigned	wm	0	0	0
6	unassigned	wm	0	0	0
8	reserved	wm	27246559	8.00MB	27262942

In the output, partition 0 (BIOS boot) contains required GPT boot information. Similar to partition 8, partition 0 requires no administration and should not be modified. The root file system is contained in partition 1.

Note - For more information about EFI labels, see [“EFI \(GPT\) Disk Label” in *Managing Devices in Oracle Solaris 11.3*](#).

On an x86 based system, the disk must have a valid Solaris fdisk partition. For more information about creating or changing an Oracle Solaris fdisk partition, see [“Configuring Disks” in *Managing Devices in Oracle Solaris 11.3*](#).

Disk names generally follow the /dev/dsk/cNtNdN naming convention. Some third-party drivers use a different naming convention or place disks in a location other than the /dev/dsk directory. To use these disks, you must manually label the disk and allocate it to ZFS.

You can specify disks by using either the full path or a shorthand name that consists of the device name within the /dev/dsk directory. The following examples show valid disk names:

- c1t0d0
- /dev/dsk/c1t0d0
- /dev/tools/disk

Using Files in a ZFS Storage Pool

With ZFS, you can use files as virtual devices in your storage pool. If you adopt this method, ensure that all files are specified as complete paths and are at least 64 MB in size.

This feature is useful for testing, such as experimenting with more complicated ZFS configurations when physical devices are insufficient. Do not use this feature in a production environment.

If you create a ZFS pool backed by files on a UFS file system, then you are implicitly relying on UFS to guarantee correctness and synchronous semantics. However, creating a ZFS pool backed by files or volumes that are created on another ZFS pool might cause a system deadlock or panic.

Redundancy Features of a ZFS Storage Pool

You should configure your storage pools using ZFS redundancy. Without redundancy, the risk of losing data is great. Moreover, without ZFS redundancy, the pool can only report data

inconsistencies, but cannot repair those inconsistencies. ZFS provides data redundancy and self-healing properties in mirrored and RAID-Z configurations.

Mirrored Storage Pool Configuration

A mirrored storage pool configuration requires at least two disks, preferably on separate controllers. A mirrored configuration can be simple or complex, where more than one mirror exists in each pool.

For information about creating simple or complex mirrored storage pools, see [“Creating a Mirrored Storage Pool” on page 30](#).

RAID-Z Storage Pool Configuration

ZFS supports a RAID-Z configuration with the following fault tolerance levels:

- Single-parity (raidz or raidz1) – Similar to RAID-5.
- Double-parity (raidz2) – Similar to RAID-6.
- Triple-parity (raidz3) – For more information, see http://blogs.oracle.com/ahl/entry/triple_parity_raid_z.

In RAID-Z, ZFS uses variable-width RAID stripes so that all writes are full-stripe writes. ZFS integrates file system and device management in such a way that the file system's metadata has enough information about the underlying data redundancy model to handle variable-width RAID stripes. Thus, RAID-Z avoids issues encountered in traditional RAID algorithms such as RAID-5's *write hole* problem.

ZFS provides self-healing data in a mirrored or RAID-Z configuration. When a bad data block is detected, ZFS fetches the correct data from another redundant copy and repairs the bad data by replacing it with the good copy.

A RAID-Z configuration with n disks of size x with p parity disks can hold approximately $(n-p)*x$ bytes and can withstand p devices failing before data integrity is compromised. You need at least two disks for a single-parity RAID-Z configuration and at least three disks for a double-parity RAID-Z configuration, and so on. For example, if you have three disks in a single-parity RAID-Z configuration, parity data occupies disk space equal to one of the three disks. Otherwise, no special hardware is required to create a RAID-Z configuration.

Just like mirrored configurations, RAID-Z configurations can either be simple or complex.

If you are creating a RAID-Z configuration with many disks, consider splitting the disks into multiple groupings. For example, a RAID-Z configuration with 14 disks is better split into

two 7-disk groupings. RAID-Z configurations with single-digit groupings of disks commonly perform better.

See the following sources for more information:

- [“Creating a RAID-Z Storage Pool” on page 30](#) – provides information about creating a RAID-Z storage pool.
- http://blogs.oracle.com/roch/entry/when_to_and_not_to – presents guidelines on choosing between a mirrored configuration or a RAID-Z configuration based on performance and disk space considerations.
- [Chapter 13, “Recommended Oracle Solaris ZFS Practices”](#) – covers additional RAID-Z storage pool recommendations.

ZFS Hybrid Storage Pool

The ZFS hybrid storage pool, available in Oracle's Sun Storage 7000 product series, combines DRAM, SSDs, and HDDs to improve performance and increase capacity, while reducing power consumption. With this product's management interface, you can select the ZFS redundancy configuration of the storage pool and easily manage other configuration options.

For more information, see the documentation in <http://www.oracle.com/technetwork/documentation/old-unified-ss-1882427.html>.

Dynamic Striping in a Storage Pool

ZFS dynamically stripes data across all top-level virtual devices. The decision about where to place data is done at write time, so no fixed-width stripes are created at allocation time.

When new virtual devices are added to a pool, ZFS gradually allocates data to the new device in order to maintain performance and disk space allocation policies. Each virtual device can also be a mirror or a RAID-Z device that contains other disk devices or files. This configuration gives you flexibility in controlling the fault characteristics of your pool. For example, you could create the following configurations out of four disks:

- Four disks using dynamic striping
- One four-way RAID-Z configuration
- Two two-way mirrors using dynamic striping

To ensure efficient use of ZFS, use top-level virtual devices of the same type with the same redundancy level in each device. Do not combine different types of virtual devices within the same pool, such as using a two-way mirror and a three-way RAID-Z configuration.

Getting Started With Oracle Solaris ZFS

This chapter provides information to help you set up a basic Oracle Solaris ZFS configuration. Later chapters provide more detailed information. By the end of this chapter, you will have a basic understanding of how the ZFS commands work and should be able to create a basic pool and file systems.

This chapter covers the following information:

- [“Hardware and Software Requirements” on page 21](#)
- [“Planning the ZFS Implementation” on page 22](#)

Hardware and Software Requirements

Ensure that you meet the following hardware requirements before using the ZFS software:

- A SPARC® or x86 based system that is running a supported Oracle Solaris release.
- Between 7 GB -13 GB of disk space. For information about how ZFS uses disk space, see [“ZFS Root Pool Space Requirements” on page 75](#).
- Sufficient memory to support your workload.
- Multiple controllers for mirrored pool configurations.

Additionally, to perform ZFS management tasks, you must assume a role with either of the following profiles:

- ZFS Storage Management – Provides the privilege to create, destroy, and manipulate devices within a ZFS storage pool
- ZFS File System Management – Provides the privilege to create, destroy, and modify ZFS file systems

Although you can use a superuser (root) account to configure ZFS, using RBAC (role-based access control) roles is a best-practice method. For more information about creating or assigning roles, see [Securing Users and Processes in Oracle Solaris 11.3](#).

In addition to using RBAC roles for administering ZFS file systems, you might also consider using ZFS delegated administration for distributed ZFS administration tasks. For more information, see [Chapter 10, “Oracle Solaris ZFS Delegated Administration”](#).

Planning the ZFS Implementation

This section describes factors you need to consider before configuring ZFS.

Naming ZFS Components

Each ZFS component, such as datasets and pools, must be named according to the following rules:

- Each component can contain only alphanumeric characters in addition to the following special characters:
 - Underscore (_)
 - Hyphen (-)
 - Colon (:)
 - Period (.)
 - Blank (" ")

Note - Tabs and other white spaces are not valid.

- Pool names must begin with a letter and can contain only alphanumeric characters as well as underscore (_), dash (-), and period (.). Note the following pool name restrictions:
 - The beginning sequence c[0-9] is not allowed.
 - The name log is reserved.
 - A name that begins with mirror, raidz, raidz1, raidz2, raidz3, or spare is not allowed because these names are reserved.
 - Pool names must not contain a percent sign (%).
- Dataset names must begin with an alphanumeric character.
- Dataset names must not contain a percent sign (%).
- Empty components are not allowed.

Identifying Storage Requirements

The pool describes the physical characteristics of the storage. You must create the pool before any file systems.

Before creating a storage pool, determine which devices will store your data. These devices must be disks of at least 128 MB in size, and they must not be in use by other parts of the

operating system. Allocate entire disks to ZFS rather than individual slices on a preformatted disk.

For more information about disks and how they are used and labeled, see [“Using Disks in a ZFS Storage Pool” on page 17](#).

Choosing the Type of Data Redundancy

ZFS supports multiple types of data redundancy, which determines the types of hardware failures the pool can withstand. ZFS supports nonredundant (striped) configurations, as well as mirroring and RAID-Z, which is a variation on RAID-5.

For more information about ZFS redundancy features, see [“Redundancy Features of a ZFS Storage Pool” on page 18](#).

Determining the ZFS File System Hierarchy

File system hierarchies are created on pools.

Hierarchies are simple, easy to understand, yet powerful mechanisms for organizing information. This section describes specific issues regarding hierarchy planning.

Selecting the File System Granularity

ZFS supports file systems organized into hierarchies, where each file system has only a single parent. The root of the hierarchy is always the pool name. ZFS supports property inheritance so that common you can set properties quickly and easily on entire trees of file systems by using hierarchies.

ZFS file systems provide a central point of administration. They are lightweight enough that you can establish one file system per user or project. This model enables you to control properties, snapshots, and backups on a per-user or per-project basis.

For more information about managing file systems, see [Chapter 7, “Managing Oracle Solaris ZFS File Systems”](#).

Grouping File Systems

You can group similar ZFS file systems into hierarchies under a common name. The hierarchy becomes the central point for administering and controlling file systems and their properties.

In [Example 1, “Configuring a Mirrored ZFS File System,” on page 28](#), the two file systems are placed under a file system named home.

Choosing File System Properties

You determine most file system characteristics properties. These properties control a variety of behaviors, including where the file systems are mounted, how they are shared, whether they use compression, or whether any quotas are in effect.

For more information about properties, see [“Introducing ZFS Properties” on page 101](#).

Creating and Destroying Oracle Solaris ZFS Storage Pools

This chapter describes how to create and destroy ZFS storage pools in Oracle Solaris. It covers the following topics:

- “[Creating ZFS Storage Pools](#)” on page 25
- “[Destroying ZFS Storage Pools](#)” on page 36

Creating ZFS Storage Pools

This section describes different ways of configuring storage pools. For information about root pools, see [Chapter 6, “Managing the ZFS Root Pool”](#).

When you create a storage pool, you configure virtual devices for the pool. A *virtual device* is an internal representation of the disk devices or files that are used to create the storage pool and describes the layout of physical storage and the storage pool's fault characteristics. A pool can have any number of virtual devices at the top of the configuration, known as a pool's *top-level vdev*.

If the top-level virtual device contains two or more physical devices, the configuration provides data redundancy as mirror or RAID-Z virtual devices. Because of the advantages of redundancy, you should create redundant storage pools. ZFS dynamically stripes data among all of the top-level virtual devices in a pool.

Even with a redundant configuration, make sure that you also schedule regular backups of your pool data to non-enterprise grade hardware. Storage pools with ZFS redundancy are not immune to hardware failures, power failures, or disconnected cables. Performing regular backups adds another layer of data protection to your enterprise.

After you create the storage pool, you can display information about it by using the following command:

```
# zpool status pool
```

For more options that you can use with the `zpool status` command, see “[Querying ZFS Storage Pool Status](#)” on page 55.

Observe the following restrictions when creating storage pools:

- Do not repartition or relabel disks that are part of an existing storage pool. Otherwise, you might have to reinstall the OS.
- Do not create a storage pool that contains components from another storage pool, such as files or volumes. Such a configuration can cause deadlocks.
- Do not create a pool to be shared across systems, which is an unsupported configuration. ZFS is not a cluster file system.

▼ How to Set Up ZFS on a System

1. **Become root or assume an equivalent role with the appropriate ZFS rights profile.**

For more information about the ZFS rights profiles, see [“Hardware and Software Requirements” on page 21](#).

2. **Create the ZFS pool.**

```
# zpool create pool keyword devices [keyword devices]
```

pool Name of the ZFS pool. The pool name must satisfy the naming requirements in [“Naming ZFS Components” on page 22](#)

keyword Further specifies the pool configuration such as type of redundancy, or whether log devices or cache will be used.

For the redundancy type, you use either the keyword `mirror` for a mirrored configuration or one of the following keywords for a RAID-Z configuration, depending on the parity you want: `raidz` or `raidz1` for single parity, `raidz2` for double parity, and `raidz3` for triple parity.

devices Specify the devices that are allocated for the pool. The devices cannot be in use or contain another file system. Otherwise, pool creation fails.

For more information about how device usage is determined, see [“Devices Actively Being Used” on page 34](#).

3. **(Optional) Display a list of ZFS pools on the system.**

```
# zpool list
```

4. **(Optional) Display the status of the pool.**

```
# zpool status pool
```

5. **Build the file system hierarchy.**

a. Create the basic file system.

The basic file system acts as a container for the individual file systems that are subsequently created.

```
# zfs create pool/filesystem
```

where *filesystem* is the name of the file system.

When used in a command, the file system name must always include the full path of the hierarchy: *pool/filesystem*. This rule also applies to subsequent children file systems that you create.

b. Set the properties to be shared by children file systems.

```
# zfs set property=value pool/filesystem
```

You can set multiple system properties.

Tip - You can simultaneously create a file system and set its properties by using the following syntax:

```
# zfs create -o property=value [-o property=value] pool/filesystem
```

c. Create the individual file systems that are grouped under the basic file system.

```
# zfs create pool/filesystem/fs1
# zfs create pool/filesystem/fs2
...
```

where *fs1*, *fs2*, and so on are individual file systems.

d. (Optional) Set properties specific to the individual file system.

```
# zfs set property=value pool/filesystem/fs1
```

6. (Optional) View the final results.

```
# zpool list
```

For more information about viewing pool status, see [“Querying ZFS Storage Pool Status” on page 55](#).

Example 1 Configuring a Mirrored ZFS File System

In the following example, a basic ZFS configuration is created on the system with the following specifications:

- Two disks are allocated to the ZFS file system: `c1t0d0` and `c2t0d0`.
- The pool `system1` uses mirroring.
- The file system `home` is created over the pool.
- The following properties are set for `home`: `mountpoint`, `share.nfs`, and `compression`.
- Two children file systems, `user1` and `user2` are created on `home`.
- A quota is set for `user2`. This property restricts the disk space available for `user2` regardless of the available disk space of the entire pool.

The command `zfs get used example` displays file system properties.

```
# zpool create system1 mirror c1t0d0 c2t0d0
# zpool list
NAME          SIZE      ALLOC    FREE     CAP    HEALTH   ALTROOT
system1       80G      137K     80G      0%    ONLINE   -

# zpool status system1

pool: system1
state: ONLINE
scrub: none requested
config:

NAME          STATE      READ  WRITE  CKSUM
system1       ONLINE    0     0     0
  mirror-0    ONLINE    0     0     0
    c1t0d0    ONLINE    0     0     0
    c2t0d0    ONLINE    0     0     0

errors: No known data errors

# zfs create system1/home
# zfs set mountpoint=/export/zfs system1/home
# zfs set share.nfs=on system1/home
# zfs set compression=on system1/home

# zfs get compression system1/home
NAME          PROPERTY    VALUE    SOURCE
system1/home  compression on        local

# zfs create system1/home/user1
# zfs create system1/home/user2

# zfs set quota=10G system1/home/user2
```

```
# zfs list
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
system1	92.0K	67.0G	9.5K	/system1
system1/home	24.0K	67.0G	8K	/export/zfs
system1/home/user1	8K	67.0G	8K	/export/zfs/user1
system1/home/user2	8K	10.0G	8K	/export/zfs/user2

Example 2 Configuring a RAID-Z ZFS File System

This example creates a RAID-Z file system and shows how you can specify disks by using either their shorthand device names or their full device names: the disk `c6t0d0` is the same as `/dev/dsk/c6t0d0`.

- Three disks are allocated to the ZFS file system: `c4t0d0`, `c5t0d0`, and `c6t0d0`.
- The pool `rdpool` uses a RAID-Z single-parity configuration.
- The file system base is created over the pool.

```
# zpool create rdpool raidz c4t0d0 c5t0d0 /dev/dsk/c6t0d0
# zpool list
```

NAME	SIZE	ALLOC	FREE	CAP	HEALTH	ALTROOT
rdpool	120G	205K	120G	0%	ONLINE	-

```
# zpool status -v rdpool
pool: rdpool
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
rdpool	ONLINE	0	0	0
raidz-0	ONLINE	0	0	0
c4t0d0	ONLINE	0	0	0
c5t0d0	ONLINE	0	0	0
c6t0d0	ONLINE	0	0	0

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

```
# zfs create rdpool/base
# zfs set mountpoint=/export/zfs rdpool/base
# zfs set share.nfs=on rdpool/base
# zfs set compression=on rdpool/base
```

```
# zfs get compression rdpool/home
```

NAME	PROPERTY	VALUE	SOURCE
rdpool/base	compression	on	local

```
# zfs create rdpool/base/user1
# zfs create rdpool/base/user2
```

```
# zfs set quota=10G rdpool/base/user2

# zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rdpool                              92.0K  67.0G   9.5K   /rdpool
rdpool/base                         24.0K  67.0G    8K   /export/zfs
rdpool/base/user1                   8K    67.0G   8K   /export/zfs/user1
rdpool/base/user2                   8K   10.0G   8K   /export/zfs/user2
```

Creating a Mirrored Storage Pool

To create a mirrored pool, use the `mirror` keyword. To configure multiple mirrors, repeat the keyword on the command line. The following command creates a pool `system1` with two top-level virtual devices.

```
zpool create system1 mirror c1d0 c2d0 mirror c3d0 c4d0
```

Both virtual devices are two-way mirrors. Data is dynamically striped across both mirrors, with data being redundant between each disk appropriately.

For more information about recommended mirrored configurations, see [Chapter 13, “Recommended Oracle Solaris ZFS Practices”](#).

You can perform the following operations on ZFS mirrored configurations:

- Add another top-level virtual device with a different set of disks. See [“Adding Devices to a Storage Pool” on page 37](#).
- Attach additional disks. See [“Attaching and Detaching Devices in a Storage Pool” on page 41](#).
- Replace disks. See [“Replacing Devices in a Storage Pool” on page 46](#).
- Detach disks. See [“Attaching and Detaching Devices in a Storage Pool” on page 41](#).
- Split a mirrored configuration to create a new, identical pool. See [“Splitting a Mirrored Storage Pool to Create a New Pool” on page 42](#).

For an example of how to configure ZFS with a mirrored storage pool, see [Example 1, “Configuring a Mirrored ZFS File System,” on page 28](#).

Creating a RAID-Z Storage Pool

To create a storage pool with a RAID-Z configuration, use one of the RAID-Z keywords depending on the parity that you want for the pool:

- `raidz` or `raidz1` for single-parity configuration.

- `raidz2` for double-parity configuration.
- `raidz3` for triple-parity configuration.

To create multiple RAID-Z top level virtual devices, repeat the keyword on the command line. The following command creates a pool `rdpool` with one top-level virtual device. The virtual device is a triple-parity RAID-Z configuration that consists of nine disks.

```
zpool create rdpool raidz3 c0t0d0 c1t0d0 c2t0d0 c3t0d0 c4t0d0 \ c5t0d0 c6t0d0  
c7t0d0 c8t0d0
```

For an example of how to configure ZFS with a RAID-Z storage pool, see [Example 2, “Configuring a RAID-Z ZFS File System,”](#) on page 29.

You can perform the following operations on ZFS RAID-Z configurations:

- Add another top-level virtual device with a different set of disks. See [“Adding Devices to a Storage Pool”](#) on page 37.
- Replace disks. See [“Replacing Devices in a Storage Pool”](#) on page 46.

You cannot perform the following operations on a RAID-Z configuration:

- Attach an additional disk.
- Detach a disk, except when you are replacing it with a spare disk, or when you need to detach a spare disk.
- Remove a device that is not a log device or a cache device.

Creating a ZFS Storage Pool With Log Devices

The ZFS intent log (ZIL) satisfies POSIX requirements for synchronous transactions. For example, databases often require their transactions to be on stable storage devices when returning from a system call. NFS and other applications can also use `fsync()` to ensure data stability.

By default, the ZIL is allocated from blocks within the main pool. However, you can obtain better performance by using separate intent log devices such as NVRAM or a dedicated disk.

Log devices for the ZFS intent log are not related to database log files. Deploying separate log devices can improve performance but the improvement also depends on the device type, the hardware configuration of the pool, and the application workload. For preliminary performance information, see http://blogs.oracle.com/perrin/entry/slog_blog_or_blogging_on.

You can configure mirrored log devices only for redundancy and not RAID-Z log devices. If an unmirrored log device fails, storing log blocks reverts to the storage pool. Further, you can add, replace, remove, attach, detach, import, and export log devices as part of the larger storage pool.

Consider the following points about ZFS log devices:

- The minimum size of a log device is the same as the minimum size of each device in a pool, which is 64 MB. The amount of logged data that might be stored on a log device is relatively small. Log blocks are freed when the log transaction or system call is committed.
- The maximum size of a log device should be approximately half the size of physical memory, which is the maximum amount of potential logged data that can be stored. For example, if a system has 16 GB of physical memory, consider a maximum log device size of 8 GB.

To create a storage pool with log devices, use the `log` keyword. The following example shows how to configure a mirrored storage pool called `datap` with a mirrored log device.

```
# zpool create datap mirror c0t5000C500335F95E3d0 c0t5000C500335F907Fd0 \  
    mirror c0t5000C500335BD117d0 c0t5000C500335DC60Fd0 \  
    log mirror c0t5000C500335E106Bd0 c0t5000C500335FC3E7d0
```

```
# zpool status datap
```

```
pool: datap  
state: ONLINE  
scrub: none requested  
config:
```

NAME	STATE	READ	WRITE	CKSUM
datap	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c0t5000C500335F95E3d0	ONLINE	0	0	0
c0t5000C500335F907Fd0	ONLINE	0	0	0
mirror-1	ONLINE	0	0	0
c0t5000C500335BD117d0	ONLINE	0	0	0
c0t5000C500335DC60Fd0	ONLINE	0	0	0
logs				
mirror-2	ONLINE	0	0	0
c0t5000C500335E106Bd0	ONLINE	0	0	0
c0t5000C500335FC3E7d0	ONLINE	0	0	0

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

Creating a ZFS Storage Pool With Cache Devices

Cache devices provide an additional layer of caching between main memory and disk. These devices provide the greatest performance improvement for random-read workloads of mostly static content.

To configure a storage pool with cache devices, use the `cache` keyword, for example:

```
# zpool create system1 mirror c2t0d0 c2t1d0 c2t3d0 cache c2t5d0 c2t8d0
```



```
# zpool status system1
pool: system1
state: ONLINE
scrub: none requested
config:

NAME            STATE      READ  WRITE  CKSUM
system1         ONLINE    0     0     0
  mirror-0      ONLINE    0     0     0
    c2t0d0      ONLINE    0     0     0
    c2t1d0      ONLINE    0     0     0
    c2t3d0      ONLINE    0     0     0
  cache
    c2t5d0      ONLINE    0     0     0
    c2t8d0      ONLINE    0     0     0

errors: No known data errors
```

You can add single or multiple cache devices to the pool, either while it is being created or after it is created, as shown in [Example 6, “Adding Cache Devices,” on page 39](#). However, you cannot create mirrored cache devices or create them as part of a RAID-Z configuration.

Note - If a read error is encountered on a cache device, that read I/O is reissued to the original storage pool device, which might be part of a mirrored or a RAID-Z configuration. The content of cache devices is considered volatile, similar to other system caches.

After cache devices are added, they gradually fill with content from main memory. The time before a cache device reaches full capacity varies depending on its size. Use the `zpool iostat` command to monitor capacity and reads, as shown in the following example:

```
# zpool iostat -v pool 5
```

For more information about the `zpool iostat` command, see [“Viewing I/O Statistics for ZFS Storage Pools” on page 59](#).

Doing a Dry Run of Storage Pool Creation

For testing purposes, you can simulate creating a pool without actually writing to the device. The `zpool create -n` command performs the device in-use checking and redundancy-level validation, and reports any errors in the process. If no errors are found, you see output similar to the following example:

```
# zpool create -n system1 mirror c1t0d0 c1t1d0
would create 'system1' with the following layout:

system1
```

```
mirror
  c1t0d0
  c1t1d0
```



Caution - Some errors, such as specifying the same device twice in the same configuration, cannot be detected without actually creating the pool. Therefore, the actual pool creation can still fail even if the dry run is successful.

Handling ZFS Storage Pool Creation Issues

This section groups descriptions of errors during pool creation by those relating to devices, redundancy, or mount points.

Some error messages suggest using the -f option to override the reported errors. However, as a general rule, you should repair errors instead of overriding them.

Devices Actively Being Used

When you create ZFS pools on specific devices, ZFS first determines if those devices are in use either by ZFS itself or some other part of the operating system. If the devices are in use, then appropriate error messages are displayed.

You must manually correct the errors reported by the following messages before attempting to create the pool again.

Mounted file system

The disk contains a file system that is currently mounted. To correct this error, use the `umount` command.

File system in `/etc/vfstab`

The disk contains a file system that is listed in the `/etc/vfstab` file but the file system is not currently mounted. To correct this error, remove or comment out the line in the `/etc/vfstab` file.

Dedicated dump device

The disk is in use as the dedicated dump device for the system. To correct this error, use the `dumpadm` command.

Part of a ZFS pool

The disk or file is part of an active ZFS storage pool. To correct this error, use the `zpool destroy` command to destroy the other pool provided that the pool is no longer needed. If that pool is still needed, use the `zpool detach` command to detach the disk from that pool. You can detach a disk only from a mirrored storage pool.

The following in-use checks serve as helpful warnings. You can override these warnings by using the `-f` option to create the pool:

Contains a file system

The disk contains a known file system but it is not mounted or being used.

Part of volume

The disk is part of a Solaris Volume Manager volume.

Part of exported ZFS pool

The disk is part of a storage pool that has been exported or manually removed from a system. In the latter case, the pool is reported as `potentially active` because the disk might be a network-attached drive in use by another system. Be cautious when overriding a potentially active pool.

Mismatched Redundancy Levels

Creating pools with virtual devices of different redundancy levels results in error messages similar to the following example:

```
# zpool create system1 mirror c1t0d0 c2t0d0 mirror c3t0d0 c4t0d0 c5t0d0
invalid vdev specification
use '-f' to override the following errors:
mismatched replication level: 2-way mirror and 3-way mirror vdevs are present
```

Similar error messages are generated if you create mirrored or RAID-Z pools using devices of different sizes.

Maintaining mismatched levels of redundancy results in unused disk space on the larger device, which is an inefficient use of ZFS. You should correct these errors instead of overriding them.

Default Mount Point for Storage Pools Not Empty

When a pool is created, the default mount point for the top-level file system is `/pool-name`. If this directory exists and contains data, an error occurs.

To create a pool with a different default mount point, use the `zpool create -m mountpoint` command. For example:

```
# zpool create system1 c1t0d0
default mountpoint '/system1' exists and is not empty
use '-m' option to provide a different default
# zpool create -m /export/zfs system1 c1t0d0
```

This command creates the new pool `system1` and the `system1` file system with a mount point of `/export/zfs`.

For more information about mount points, see [“Managing ZFS Mount Points” on page 124](#).

Destroying ZFS Storage Pools

You can destroy a pool even if the pool contains mounted datasets by using the `zpool destroy pool` command.



Caution - ZFS cannot always keep track of which devices are in use. Ensure that you are destroying the correct pool and you always have copies of your data. If you accidentally destroy the wrong pool, see [“Recovering Destroyed ZFS Storage Pools” on page 70](#).

When you destroy a pool, the pool is still available for import. Therefore, confidential data might remain on the disks that were part of the pool. To completely destroy the data, use a feature like the `format` utility's `analyze->purge` option on every disk in the destroyed pool.

To ensure data confidentiality, create encrypted ZFS file systems. Even if a destroyed pool is recovered, the data would remain inaccessible without the encryption keys. For more information, see [“Encrypting ZFS File Systems” on page 147](#).

When a pool with a mix of available and unavailable devices is destroyed, data is written to the available disks to indicate that the pool is no longer valid. This state information prevents the devices from being listed as a potential pool when you perform an import. Even with unavailable devices, the pool can still be destroyed. When the unavailable devices are repaired, they are reported as potentially active when you create a new pool and appear as valid devices when you search for pools to import.

A pool itself can become unavailable if a sufficient number of its devices are unavailable. The state of the pool's top-level virtual device is reported as `UNAVAIL`. In this case, you can destroy the pool only by using the `zpool destroy -f` command.

For more information about pool and device health, see [“Determining the Health Status of ZFS Storage Pools” on page 61](#).

Managing Devices in Oracle Solaris ZFS Storage Pools

This chapter discusses different tasks you can perform to manage the physical devices that you use for the ZFS pools on the system. It has the following sections:

- “Adding Devices to a Storage Pool” on page 37
- “Removing Devices From a Storage Pool” on page 40
- “Attaching and Detaching Devices in a Storage Pool” on page 41
- “Splitting a Mirrored Storage Pool to Create a New Pool” on page 42
- “Taking Devices in a Storage Pool Offline or Returning Online” on page 44
- “Clearing Storage Pool Device Errors” on page 46
- “Replacing Devices in a Storage Pool” on page 46
- “Designating Hot Spares in a Storage Pool” on page 49

Adding Devices to a Storage Pool

You can dynamically add disk space to a pool by adding a new top-level virtual device. This disk space is immediately available to all datasets in the pool.

The virtual device that you add should have the same level of redundancy as the existing virtual device. However, you can change the level of redundancy by using the `-f` option.

To add a new virtual device to a pool, use the `zpool add` command.

```
# zpool add pool keyword devices
```

Note - With `zpool add -n`, you can perform a dry run before actually adding devices.

EXAMPLE 3 Adding Disks to a Mirrored ZFS Configuration

In the following example, a mirror is added to a ZFS configuration that consists of two top-level mirrored devices.

```
# zpool add mpool mirror c0t3d0 c1t3d0
# zpool status mpool
pool: mpool
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM	
mpool	ONLINE	0	0	0	
mirror-0	ONLINE	0	0	0	
c0t1d0	ONLINE	0	0	0	
c1t1d0	ONLINE	0	0	0	
mirror-1	ONLINE	0	0	0	
c0t2d0	ONLINE	0	0	0	
c1t2d0	ONLINE	0	0	0	
mirror-2	ONLINE	0	0	0	Added mirrored device.
c0t3d0	ONLINE	0	0	0	
c1t3d0	ONLINE	0	0	0	

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

EXAMPLE 4 Adding Disks to a RAID-Z Configuration

This example shows how to add one RAID-Z device consisting of three disks to an existing RAID-Z storage pool that also contains three disks.

```
# zpool add rzpool raidz c2t2d0 c2t3d0 c2t4d0
# zpool status rzpool
pool: rzpool
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM	
rzpool	ONLINE	0	0	0	
raidz1-0	ONLINE	0	0	0	
c1t2d0	ONLINE	0	0	0	
c1t3d0	ONLINE	0	0	0	
c1t4d0	ONLINE	0	0	0	
raidz1-1	ONLINE	0	0	0	Added RAID-Z device.
c2t2d0	ONLINE	0	0	0	
c2t3d0	ONLINE	0	0	0	
c2t4d0	ONLINE	0	0	0	

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

EXAMPLE 5 Adding a Mirrored Log Device

This example shows how to add a mirrored log device to a mirrored storage pool.

```
# zpool add newpool log mirror c0t6d0 c0t7d0
# zpool status newpool
pool: newpool
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
newpool	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c0t4d0	ONLINE	0	0	0
c0t5d0	ONLINE	0	0	0
logs				
mirror-1	ONLINE	0	0	0
c0t6d0	ONLINE	0	0	0
c0t7d0	ONLINE	0	0	0

Added mirrored log device.

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

Note that mirrored log devices are provided with an identifier, such as `mirror-1` in the example. The identifier is useful when you remove log devices, as shown in [Example 7, “Removing a Mirrored Log Device,”](#) on page 40.

EXAMPLE 6 Adding Cache Devices

The following example shows how to add a cache device to a pool.

```
# zpool add system1 cache c2t5d0 c2t8d0
# zpool status system1
pool: system1
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
system1	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c2t0d0	ONLINE	0	0	0
c2t1d0	ONLINE	0	0	0
c2t3d0	ONLINE	0	0	0
cache				
c2t5d0	ONLINE	0	0	0
c2t8d0	ONLINE	0	0	0

Added cache device.

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

Removing Devices From a Storage Pool

EXAMPLE 7 Removing a Mirrored Log Device

This example shows how to remove the log device `mirror-1` that was created in [Example 5, “Adding a Mirrored Log Device,” on page 39](#). Note that if the log device is not redundant, then remove the device by referring to the device name, such as `c0t6d0`.

```
# zpool remove newpool mirror-1
# zpool status newpool
pool: newpool
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
newpool	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c0t4d0	ONLINE	0	0	0
c0t5d0	ONLINE	0	0	0

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

EXAMPLE 8 Removing Cache Devices

This example shows how to remove the cache device that was created in [Example 6, “Adding Cache Devices,” on page 39](#).

```
# zpool remove system1 c2t5d0 c2t8d0
# zpool status system1
pool: system1
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
system1	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c2t0d0	ONLINE	0	0	0
c2t1d0	ONLINE	0	0	0


```
c2t3d0  ONLINE      0      0      0
```

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

Attaching and Detaching Devices in a Storage Pool

To add a new device to an existing virtual device, use the following command:

```
# zpool attach pool existing-device new-device
```

You can use the `zpool detach` command to detach a device provided one of the following conditions applies:

- The device belongs to a mirrored pool configuration.
- In a RAID-Z configuration, the detached device is replaced by another physical device or by a spare.

Outside of these conditions, detaching a device generates an error similar to the following:

```
cannot detach c1t2d0: only applicable to mirror and replacing vdevs
```

The following examples show how to apply `zfs attach` commands.

EXAMPLE 9 Converting a Two-Way Mirrored Storage Pool to a Three-Way Mirrored Storage Pool

In this example, `mpool` is an existing two-way mirror pool. It is converted into a three-way mirror pool by attaching `c2t1d0`, the new device, to the existing device `c1t1d0`. The newly attached device is immediately resilvered.

```
# zpool attach mpool c1t1d0 c2t1d0
```

```
# zpool status mpool
```

```
pool: mpool
```

```
state: ONLINE
```

```
scrub: resilver completed after 0h0m with 0 errors on Fri Jan  8 12:59:20 2010
```

```
config:
```

NAME	STATE	READ	WRITE	CKSUM	
mpool	ONLINE	0	0	0	
mirror-0	ONLINE	0	0	0	
c0t1d0	ONLINE	0	0	0	
c1t1d0	ONLINE	0	0	0	
c2t1d0	ONLINE	0	0	0	592K resilvered Attached device creates a 3-way mirror pool

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

EXAMPLE 10 Converting a Nonredundant Storage Pool to a Mirrored Storage Pool

The `zpool attach` command also enables you to convert a storage pool or a log device from a nonredundant to a redundant configuration.

The following example shows the status of the nonredundant pool `system1` before and after it is converted into a redundant pool.

```
# zpool status system1
pool: system1
state: ONLINE
scrub: none requested
config:
NAME                STATE      READ  WRITE CKSUM
system1             ONLINE      0     0     0
    c0t1d0           ONLINE      0     0     0

errors: No known data errors
# zpool attach system1 c0t1d0 c1t1d0
# zpool status system1
pool: system1
state: ONLINE
scrub: resilver completed after 0h0m with 0 errors on Fri Jan  8 14:28:23 2010
config:

NAME                STATE      READ  WRITE CKSUM
system1             ONLINE      0     0     0      Pool becomes mirrored
    mirror-0         ONLINE      0     0     0
        c0t1d0       ONLINE      0     0     0
        c1t1d0       ONLINE      0     0     0      73.5K resilvered

errors: No known data errors
```

Splitting a Mirrored Storage Pool to Create a New Pool

You can quickly clone a mirrored ZFS storage pool by using the `zpool split` command. The new pool will have identical contents to the original mirrored ZFS storage pool. You can then import the new pool either to the same system or to another system. For more information about importing pools, see [“Importing ZFS Storage Pools” on page 68](#).

To split a pool, use the following command:

```
# zpool split pool new-pool [device]
```

Unless you specify the device, the `zpool split` command by default detaches the last disk of the pool's virtual device for the newly created pool. If a pool has multiple top-level virtual

devices, the command detaches a disk from each virtual device to create a new pool out of those disks.

Splitting a pool applies only to mirrored configurations. You cannot split a RAID-Z configured pool or a nonredundant pool.

If you split a pool that has only a single top-level device consisting of three disks, the new pool created out of the third disk is nonredundant. The remaining pool retains data redundancy with its two remaining disks. To convert the new pool into a redundant configuration, attach a new device to the pool.

For more procedures and examples about splitting a ZFS pool with the `zpool split` command, log in to your account at [My Oracle Support \(https://support.oracle.com\)](https://support.oracle.com) and see "How to Use 'zpool split' to Split an rpool (Doc ID 1637715.1)".

Ensure the following before splitting a mirrored pool:

- Data and application operations are quiesced.
- No resilvering is in progress.
- Your hardware is configured correctly. For related information about confirming your hardware's cache flush settings, see [“General System Practices” on page 269](#).

Before the actual split operation occurs, data in memory is flushed to the mirrored disks. After the data is flushed, the disk is detached from the pool and given a new pool GUID so that the pool can be imported on the same system on which it was split.

If the pool to be split has non-default file system mount points and the new pool is created on the same system, then you must use the `zpool split -R` option to identify an alternate root directory for the new pool so that any existing mount points do not conflict. For example:

```
# zpool split -R /system2 system1 system2
```

If you don't use the `zpool split -R` option and you can see that mount points conflict when you attempt to import the new pool, import the new pool with the `-R` option. If the new pool is created on a different system, then specifying an alternate root directory is not necessary unless mount point conflicts occur.

EXAMPLE 11 Splitting a Mirrored ZFS Pool

In this example, a mirrored storage pool called `poolA` with three disks is split. The two resulting pools are the mirrored pool `poolA` with two disks and the new pool `poolB` with one disk.

```
# zpool status poolA
pool: poolA
state: ONLINE
scan: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
------	-------	------	-------	-------

```
poolA      ONLINE      0      0      0
mirror-0   ONLINE      0      0      0
c0t0d0     ONLINE      0      0      0
c0t1d0     ONLINE      0      0      0
c0t2d0     ONLINE      0      0      0
```

errors: No known data errors

```
# zpool split poolA poolB
```

```
# zpool import luna
```

```
# zpool status poolA poolB
```

pool: luna

state: ONLINE

scan: none requested

config:

NAME	STATE	READ	WRITE	CKSUM
poolB	ONLINE	0	0	0
c0t2d0	ONLINE	0	0	0

errors: No known data errors

pool: poolA

state: ONLINE

scan: none requested

config:

NAME	STATE	READ	WRITE	CKSUM
poolA	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c0t0d0	ONLINE	0	0	0
c0t1d0	ONLINE	0	0	0

errors: No known data errors

With the new configuration, you can perform other operations. For example, you can import poolB on another system for backup purposes. After the backup is complete, you can destroy poolB and reattach the disk to poolA.

Taking Devices in a Storage Pool Offline or Returning Online

When a device in a storage pool becomes permanently unreliable or non-functional, you can take the device offline with the following command:

```
# zpool offline [option] pool device
```

where the name of *device* can either be the short name or the full path.

When you take a device offline, its OFFLINE state becomes persistent. For a nonpersistent OFFLINE state, use the `-t` option, which takes the device temporarily offline. When the system is rebooted, the device is automatically restored to the ONLINE state.



Caution - Do not take devices offline such that the pool itself becomes unavailable. For example, you cannot take offline two devices in a `raidz1` configuration, nor can you take offline a top-level virtual device. The following error message would be displayed:

```
cannot offline c0t5000C500335F95E3d0: no valid replicas
```

The OFFLINE state does not mean that the device is detached from the pool. Therefore, you cannot use that device for another pool. Otherwise, an error message is generated similar to the following example:

```
device is part of exported or potentially active ZFS pool. Please see zpool(8)
```

To use the device for another pool, first restore the device to the ONLINE state, and then destroy the pool to which the device belongs.

If you do not want to destroy the pool, replace the offlined device with a comparable device. The replaced device then becomes available for a different pool.

Note - You do not need to take devices offline to replace them.

To return a device online, use the following command:

```
# zpool online [option] pool device
```

Any data that has been written to the pool is resynchronized with the newly available device.

If you try to bring online a device whose state is UNAVAIL, a message about a faulted device is displayed, similar to the following example:

```
warning: device 'device' onlined, but remains in faulted state
use 'zpool clear' to restore a faulted device
```

Messages about faulted device might also be displayed on the console or written to the `/var/adm/messages` file.

For more information about replacing a faulted device, see [“Resolving a Missing or Removed Device” on page 241](#).

To expand a LUN, use the `zpool online -e` command. By default, a LUN that is added to a pool is not expanded to its full size unless the `autoexpand` pool property is enabled. If the property is disabled, use the command to expand the LUN automatically. You can run the command regardless of whether the LUN is offline or already online.

Clearing Storage Pool Device Errors

Pool device failures, such as temporarily losing connectivity, can cause errors that are also reported in a `zpool status` output. To clear such errors, use the following command:

```
# zpool clear pool [devices]
```

If devices are specified, the command clears only those errors that are associated with the devices. Otherwise, the command clears all device errors within the pool.

For more information about clearing `zpool` errors, see [“Clearing Transient or Persistent Device Errors” on page 247](#).

Replacing Devices in a Storage Pool

You can replace a device in a storage pool by using the `zpool replace` command.

```
# zpool replace pool replaced-device [new-device]
```

If you are installing a device's replacement on the same location in a redundant pool, you might only need to identify the replaced device. On some hardware, ZFS recognizes the new device on the same location. However, if you install the replacement on a different location, you must specify both the replaced device and the new device.

The automatic detection of replaced devices is hardware-dependent and might not be supported on all platforms. Furthermore, some hardware types support the `autoreplace` pool property. If this property is enabled, then a device's replacement on the same location is automatically formatted and replaced. Running the `zpool replace` command is unnecessary.

Note the following guidelines:

- A hot spare device, if available, automatically replaces a device that is removed while the system is running. After a failed disk is replaced, you might need to detach the spare by using the `zpool detach` command. For information about detaching a hot spare, see [“Activating and Deactivating Hot Spares in Your Storage Pool” on page 49](#).
- A device that is removed and then reinserted is automatically brought online. In this case, no replacement occurred. The replacement hot spare device is automatically removed after the reinserted device is brought back online.

When you replace a device with one that has a larger capacity, the new device is not automatically expanded to its full size after you add it to the pool. The `autoexpand` pool property determines the automatic expansion of a replacement LUN. By default, this property is disabled. You can enable it before or after the larger LUN is added to the pool.

On some systems with SATA disks, you must unconfigure a disk before you can take it offline. If you are replacing a disk in the same slot position on this system, then you can just run the `zpool replace` command as described in [Example 12, “Replacing Devices in a Mirrored Pool,” on page 48](#) the first example in this section. For an example of replacing a SATA disk, see [Example 58, “Replacing a SATA Disk in a ZFS Storage Pool,” on page 250](#).

Because of the resilvering of data onto new disks, replacing disks is time-consuming. Between disk replacements, run the `zpool scrub` command to ensure that the replacement devices are operational and that the data is written correctly.

For more information about replacing devices, see [“Resolving a Missing or Removed Device” on page 241](#) and [“Replacing or Repairing a Damaged Device” on page 245](#).

▼ How to Replace a Device in a Storage Pool

1. **If necessary, switch the device to the OFFLINE state.**

```
# zpool offline pool device
```

2. **Physically replace the device with a new device.**

In the case of redundant storage pools, ensure that the replacement device is equal to or larger than the smallest disk in the pool.

3. **Run the `format` command.**

Check the following in the output:

- Ensure that the new device is listed.
- Check whether the replacement device is marked as `WWN` to verify that the device ID has changed..

4. **Replace the device in the pool.**

If the new device has a new ID, include the ID in the command.

```
# zpool replace pool replaced-device [new-device-ID]
```

Note - If you are replacing multiple devices, make sure that each device is fully resilvered before you replace the next device.

5. **If necessary, bring the device online.**

```
# zpool online pool new-device
```

6. **If faulted device errors are reported, perform FMA procedures.**

- a. Run the command `fmadm faulty`.
- b. From the **Affects:** section of the output, identify the pool name and the GUID of the virtual device.
- c. Run the following command and provide the information from the previous step.

```
# fmadm repaired zfs://pool=name/vdev=guid
```

Example 12 Replacing Devices in a Mirrored Pool

In this example, two 16-GB disks in the pool `system1` are replaced with two 72-GB disks. The `autoexpand` property is enabled after the disk replacements to expand the full disk sizes.

```
# zpool status system1
pool: system1
state: ONLINE
scrub: none requested
config:

    NAME        STATE      READ WRITE CKSUM
    system1     ONLINE    0     0     0
      mirror    ONLINE    0     0     0
        c1t16d0 ONLINE    0     0     0
        c1t17d0 ONLINE    0     0     0

# zpool list system1
NAME      SIZE  ALLOC  FREE   CAP  HEALTH  ALTROOT
system1   16.8G  76.5K  16.7G   0%  ONLINE  -
# zpool replace system1 c1t16d0 c1t1d0
# zpool replace system1 c1t17d0 c1t2d0
# zpool list system1
NAME      SIZE  ALLOC  FREE   CAP  HEALTH  ALTROOT
system1   16.8G  88.5K  16.7G   0%  ONLINE  -
# zpool set autoexpand=on system1
# zpool list system1
NAME      SIZE  ALLOC  FREE   CAP  HEALTH  ALTROOT
system1   68.2G  117K  68.2G   0%  ONLINE  -
```

Working With Hot Spares in Storage Pools

The hot spares feature enables you to identify disks that could be used to replace a failed or faulted device in a storage pool. A hot spare device is inactive in a pool until the spare replaces the failed device.

Designating Hot Spares in a Storage Pool

Devices can be designated as hot spares in the following ways:

- When the pool is created.

```
# zpool create pool keyword devices spare devices
```

- After the pool is created.

```
# zpool add pool spare devices
```

To remove a hot spare, use the following command:

```
# zpool remove pool spare-device
```

Note - You cannot remove a hot spare that is currently being used by the pool.

A hot spare device must be equal to or larger than the size of the largest disk in the pool. Otherwise, the smaller spare device can still be designated as a hot spare. However, when that device is activated to replace a failed device, the operation fails with the following error message:

```
cannot replace disk3 with disk4: device is too small
```

Do not share a spare across multiple pools or multiple systems even if the device is visible for access by these systems. You can configure a disk to be shared among several pools provided that only a single system must control all of these pools. However, this practice is risky. For example, if pool A that is using the shared spare is exported, pool B could unknowingly use the spare while pool A is exported. When pool A is imported, data corruption could occur because both pools are using the same disk.

Activating and Deactivating Hot Spares in Your Storage Pool

You activate hot spares in the following ways:

- Manual replacement – Run the `zpool replace` command to replace a failed device. When a new device is inserted to replace the failed disk, you activate the new device by detaching the spare.
- Automatic replacement – An FMA agent detects a fault, determines spare availability, and automatically replaces the faulted device. A hot spare also replaces a device in the `UNAVAIL` state.

If you set the `autoreplace` pool property to `on`, the spare is automatically detached and returned to the spare pool when the new device is inserted and the online operation completes.

To deactivate a hot spare, perform one of the following actions:

- Remove the hot spare from the storage pool.
- Detach the hot spare after physically replacing a failed disk. See [Example 13, “Detaching a Hot Spare After the Failed Disk Is Replaced,”](#) on page 50.
- Swap in another hot spare either temporarily or permanently. See [Example 14, “Detaching a Failed Disk and Using the Hot Spare,”](#) on page 51.

EXAMPLE 13 Detaching a Hot Spare After the Failed Disk Is Replaced

This example assume the following configuration:

- In `system1`'s `mirror-1` configuration, disk `c0t5000C500335BA8C3d0` has failed. The following partial output shows the status of `mirror-1`:

```
# zpool status system1
.
mirror-1                DEGRADED      0      0      0
  c0t5000C500335BD117d0  ONLINE        0      0      0
  c0t5000C500335BA8C3d0  UNAVAIL       0      0      0    Failed disk
```
- The pool's spare `c0t5000C500335E106Bd0` is automatically activated to replace the failed disk.
- You physically replace the failed disk with a new device `c0t5000C500335DC60Fd0`.

The example begins with reconfiguring the pool with the new device. First, you run `zpool replace` to inform ZFS about the removed device. Then, if necessary, you run `zpool detach` to deactivate the spare and return it to the spare pool. The example ends with displaying the status of the new configuration and performing the appropriate FMA steps for fault devices, as shown in [Step 6 of “How to Replace a Device in a Storage Pool”](#) on page 47.

```
# zpool replace system1 c0t5000C500335BA8C3d0
# zpool detach system1 c0t5000C500335E106Bd0
# zpool status system1
.
.
mirror-1                ONLINE        0      0      0
  c0t5000C500335BD117d0  ONLINE        0      0      0
  c0t5000C500335DC60Fd0  ONLINE        0      0      0    Replacement device
spares
  c0t5000C500335E106Bd0  AVAIL                    Deactivated spare

# fmadm faulty
```

```
# fmadm repaired zfs://pool=name/vdev=guid
```

EXAMPLE 14 Detaching a Failed Disk and Using the Hot Spare

Instead of a new replacement device, you can use the spare device as a permanent replacement instead. In this case, you simply detach the failed disk. If the failed disk is subsequently repaired, then you can add it to the pool as a newly designated spare.

This example uses the same assumptions as [Example 13, “Detaching a Hot Spare After the Failed Disk Is Replaced,” on page 50](#).

- The mirror-1 configuration of the pool system1 is in a degraded state.

```
# zpool status system1
.
mirror-1          DEGRADED      0      0      0
c0t5000C500335BD117d0 ONLINE      0      0      0
c0t5000C500335BA8C3d0 UNAVAIL      0      0      0   Failed disk
```

- The pool's spare c0t5000C500335E106Bd0 is automatically activated to replace the failed disk.

The example begins with detaching the failed disk that has been replaced by the spare.

```
# zpool detach system1 c0t5000C500335BA8C3d0
# zpool status system1
.
.
mirror-1          ONLINE        0      0      0
c0t5000C500335BD117d0 ONLINE      0      0      0
c0t5000C500335E106Bd0 ONLINE      0      0      0   Spare replaces failed disk
```

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

Subsequently, you add the repaired disk back to the pool as the spare device. You complete the procedure by performing the appropriate FMA steps for fault devices.

```
# zpool add system1 spare c0t5000C500335BA8C3d0
# zpool status system1
.
.
mirror-1          ONLINE        0      0      0
c0t5000C500335BD117d0 ONLINE      0      0      0
c0t5000C500335E106Bd0 ONLINE      0      0      0   Former spare
spares
c0t5000C500335BA8C3d0  AVAIL              Repaired disk as spare
```

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

```
# fmadm faulty
# fmadm repaired zfs://pool=name/vdev=guid
```


Managing Oracle Solaris ZFS Storage Pools

This chapter describes how to administer storage pools in Oracle Solaris ZFS. It has the following sections:

- “Managing ZFS Storage Pool Properties” on page 53
- “Querying ZFS Storage Pool Status” on page 55
- “Migrating ZFS Storage Pools” on page 65
- “Upgrading ZFS Storage Pools” on page 72

Managing ZFS Storage Pool Properties

To manage ZFS pool properties, you use the following commands:

- `zpool get all pool` – Lists all the properties of the pool with their corresponding values.
- `zpool get property pool` – Lists the specified property of the pool with its corresponding value.
- `zpool set property=value pool` – Assigns a value to the pool's specified property.

If you attempt to set a pool property on a pool that is full, a message similar to the following is displayed:

```
# zpool set autoreplace=on system1
cannot set property for 'system1': out of space
```

For information on preventing pool space capacity problems, see [Chapter 13, “Recommended Oracle Solaris ZFS Practices”](#).

TABLE 2 ZFS Pool Property Descriptions

Property Name	Type	Default Value	Description
allocated	String	N/A	Read-only value that identifies the amount of storage space within the pool that has been physically allocated.
altroot	String	off	Identifies an alternate root directory. If set, this directory is prepended to any mount points within the pool. This property can be used in the following situations: when you are examining

Property Name	Type	Default Value	Description
			an unknown pool, if the mount points cannot be trusted, or in an alternate boot environment where the typical paths are not valid.
autoreplace	Boolean	off	Controls automatic device replacement. If set to off, you must initiate device replacement using the <code>zpool replace</code> command. If set to on, any new device found in the same physical location as a device that previously belonged to the pool is automatically formatted and replaced. The property abbreviation is <code>replace</code> .
bootfs	Boolean	N/A	Identifies the default bootable file system for the root pool. This property is typically set by the installation programs.
cachefile	String	N/A	Controls where pool configuration information is cached. All pools in the cache are automatically imported when the system boots. However, installation and clustering environments might require this information to be cached in a different location so that pools are not automatically imported. You can set this property to cache pool configuration information in a different location. This information can be imported later by using the <code>zpool import -c</code> command. For most ZFS configurations, this property is not used.
capacity	Number	N/A	Read-only value that identifies the percentage of pool space used. The property abbreviation is <code>cap</code> .
dedupditto	String	N/A	Sets a threshold for the reference count for a deduped block. If the count goes above the threshold, another ditto copy of the block is stored automatically.
dedupratio	String	N/A	Read-only deduplication ratio achieved for a pool, expressed as a multiplier.
delegation	Boolean	on	Controls whether a nonprivileged user can be granted access permissions that are defined for a file system. For more information, see Chapter 10, “Oracle Solaris ZFS Delegated Administration” .
failmode	String	wait	Controls the system behavior if a catastrophic pool failure occurs. This condition is typically a result of a loss of connectivity to the underlying storage device or devices or a failure of all devices within the pool. The behavior of such an event is determined by one of the following values: <ul style="list-style-type: none"> ■ <code>wait</code> – Blocks all I/O requests to the pool until device connectivity is restored and the errors are cleared by using the <code>zpool clear</code> command. In this state, I/O operations to the pool are blocked but read operations might succeed. A pool remains in the <code>wait</code> state until the device issue is resolved. ■ <code>continue</code> – Returns an EIO error to any new write I/O requests but allows reads to any of the remaining healthy devices. Any write requests that have yet to be committed to disk are blocked. After the device is reconnected or replaced, the errors must be cleared with the <code>zpool clear</code> command. ■ <code>panic</code> – Prints a message to the console and generates a system crash dump.

Property Name	Type	Default Value	Description
free	String	N/A	Read-only value that identifies the number of blocks within the pool that are not allocated.
guid	String	N/A	Read-only property that identifies the unique identifier for the pool.
health	String	N/A	Read-only property that identifies the current health of the pool, as either ONLINE, DEGRADED, SUSPENDED, REMOVED, or UNAVAIL.
listshares	String	off	Controls whether share information in this pool is displayed with the <code>zfs list</code> command. The default value is <code>off</code> .
listsnapshots	String	off	Controls whether snapshot information that is associated with this pool is displayed with the <code>zfs list</code> command. If this property is disabled, you can display snapshot information with the <code>zfs list -t snapshot</code> command.
readonly	Boolean	off	Identifies whether a pool can be modified. This property is enabled only when a pool is has been imported in read-only mode. If enabled, any synchronous data that exists only in the intent log will not be accessible until the pool is re-imported in read-write mode.
size	Number	N/A	Read-only property that identifies the total size of the storage pool.
version	Number	N/A	Identifies the current on-disk version of the pool. The preferred method of updating pools is with the <code>zpool upgrade</code> command, although you can use this property when you need a specific version for backwards compatibility. You can set this property to any number between 1 and the current version reported by the <code>zpool upgrade -v</code> command.

Querying ZFS Storage Pool Status

The `zpool list` command provides several ways to request information regarding pool status. The information available generally falls into three categories: basic usage information, I/O statistics, and health status. This section discusses all three types of storage pool information.

Displaying Information About ZFS Storage Pools

The `zpool list` command displays basic information about pools. You can use the command in the following ways:

- Without options: `zpool list [pool]`
If you do not specify a pool, then information for all pools is displayed.
- With options: `zpool list options [arguments]`

Displaying Information About All Storage Pools or a Specific Pool

The `zpool list [pool]` command displays the following pool information:

NAME	Name of the pool.
SIZE	Total size of the pool, equal to the sum of the sizes of all top-level virtual devices.
ALLOC	Amount of physical space allocated to all datasets and internal metadata. Note that this amount differs from the amount of disk space as reported at the file system level.
FREE	Amount of unallocated space in the pool.
CAP (CAPACITY)	Amount of disk space used, expressed as a percentage of the total disk space.
HEALTH	Current health status of the pool. For more information about pool health, see “Determining the Health Status of ZFS Storage Pools” on page 61 .
ALTROOT	Alternate root of the pool, if one exists. For more information about alternate root pools, see “Using a ZFS Pool With an Alternate Root Location” on page 230 .

The following example shows sample `zpool list` command output:

```
# zpool list
NAME                SIZE    ALLOC    FREE    CAP  HEALTH  ALTROOT
syspool1            80.0G   22.3G   47.7G   28%  ONLINE  -
syspool2            1.2T    384G    816G   32%  ONLINE  -
```

To obtain statistics for a specific pool, specify the pool name with the command.

Displaying Specific Storage Pool Statistics

You can select the specific pool information to be displayed by issuing options and arguments with the `zpool list` command.

The `-o` option enables you to filter which columns are displayed. The following example shows how to list only the name and size of each pool:

```
# zpool list -o name,size
```


NAME	SIZE
syspool1	80.0G
syspool2	1.2T

You can use the `zpool list` command as part of a shell script by issuing the combined `-Ho` options. The `-H` option suppresses display of column headings and instead displays tab-separated pool information. For example:

```
# zpool list -Ho name,size
syspool1 80.0G
syspool2 1.2T
```

The `-T` option enables you to gather time-stamped statistics about the pools. Use the following syntax:

```
# zpool list -T d interval [count]
```

`d` Specifies to use the standard date format when displaying the date.

`interval` Specifies the interval in seconds between which information is displayed.

`count` Specifies the number of times to report the information. If you do not specify `count` then the information is continuously refreshed at the specified interval until you press Ctl-C.

The following example displays pool information twice, with a 3-second gap between the reports. The output uses the standard format to display the date.

```
# zpool list -T d 3 2
Tue Nov 2 10:36:11 MDT 2010
NAME  SIZE  ALLOC  FREE   CAP  DEDUP  HEALTH  ALTROOT
pool  33.8G  83.5K  33.7G    0%  1.00x  ONLINE  -
rpool 33.8G  12.2G  21.5G   36%  1.00x  ONLINE  -
Tue Nov 2 10:36:14 MDT 2010
pool  33.8G  83.5K  33.7G    0%  1.00x  ONLINE  -
rpool 33.8G  12.2G  21.5G   36%  1.00x  ONLINE  -
```

Displaying Pool Devices by Physical Locations

Use the `zpool status -l` option to display information about the physical location of pool devices. Reviewing the physical location information is helpful when you need to physically remove or replace a disk.

In addition, you can use the `fmadm add-alias` command to include a disk alias name that helps you identify the physical location of disks in your environment. For example:

```
# fmadm add-alias SUN-Storage-J4400.1002QCQ015 Lab10Rack5disk
# zpool status -l system1
```

```
pool: system1
state: ONLINE
  scan: scrub repaired 0 in 0h0m with 0 errors on Fri Aug  3 16:00:35 2012
config:
```

NAME	STATE	READ	WRITE	CKSUM
system1	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_02/disk	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_20/disk	ONLINE	0	0	0
mirror-1	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_22/disk	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_14/disk	ONLINE	0	0	0
mirror-2	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_10/disk	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_16/disk	ONLINE	0	0	0
.				
.				
.				
spares				
/dev/chassis/Lab10Rack5.../DISK_17/disk	AVAIL			
/dev/chassis/Lab10Rack5.../DISK_12/disk	AVAIL			

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

Displaying ZFS Storage Pool Command History

Use the `zpool history` command to display the log of `zfs` and `zpool` command use. The log records when these commands were successfully used to modify pool state information or to troubleshoot an error condition.

Note the following information about the history log:

- You cannot disable the log. The log is saved persistently on disk and is preserved across system reboots.
- The log is implemented as a ring buffer. The minimum size is 128 KB. The maximum size is 32 MB.
- For smaller pools, the maximum size is capped at 1 percent of the pool size, where the size is determined at pool creation time.
- Because the log requires no administration, you do not need to tune the log size or its location.

The following example shows the `zfs` and `zpool` command history on the pool `system1`.

```
# zpool history system1
2012-01-25.16:35:32 zpool create -f system1 mirror c3t1d0 c3t2d0 spare c3t3d0
2012-02-17.13:04:10 zfs create system1/test
2012-02-17.13:05:01 zfs snapshot -r system1/test@snap1
```

Use the `-l` option to display a long format that includes the user name, the host name, and the zone in which the operation was performed. For example:

```
# zpool history -l system1
History for 'system1':
2012-01-25.16:35:32 zpool create -f system1 mirror c3t1d0 c3t2d0 spare c3t3d0
[user root on host1:global]
2012-02-17.13:04:10 zfs create system1/test [user root on host1:global]
2012-02-17.13:05:01 zfs snapshot -r system1/test@snap1 [user root on host1:global]
```

Use the `-i` option to display internal event information that can be used for diagnostic purposes. For example:

```
# zpool history -i system1
History for 'system1':
2012-01-25.16:35:32 zpool create -f system1 mirror c3t1d0 c3t2d0 spare c3t3d0
2012-01-25.16:35:32 [internal pool create txg:5] pool spa 33; zfs spa 33; zpl 5;
uts host1 5.11 11.1 sun4v
2012-02-17.13:04:10 zfs create system1/test
2012-02-17.13:04:10 [internal property set txg:66094] $share2=2 dataset = 34
2012-02-17.13:04:31 [internal snapshot txg:66095] dataset = 56
2012-02-17.13:05:01 zfs snapshot -r system1/test@snap1
2012-02-17.13:08:00 [internal user hold txg:66102] <.send-4736-1> temp = 1 ...
```

Viewing I/O Statistics for ZFS Storage Pools

To request I/O statistics for a pool or specific virtual devices, use the `zpool iostat` command. Similar to the `iostat` command, this command can display a static snapshot of all I/O activity, as well as updated statistics for every specified interval. The following statistics are reported:

alloc capacity	The amount of data currently stored in the pool or device. This amount differs from the amount of disk space available to actual file systems by a small margin due to internal implementation details.
free capacity	The amount of disk space available in the pool or device. Like the used statistic, this amount differs from the amount of disk space available to datasets by a small margin.
read operations	The number of read I/O operations sent to the pool or device, including metadata requests.
write operations	The number of write I/O operations sent to the pool or device.
read bandwidth	The bandwidth of all read operations (including metadata), expressed as units per second.
write bandwidth	The bandwidth of all write operations, expressed as units per second.

Listing Pool-Wide I/O Statistics

When issued with no options, the `zpool iostat` command displays the accumulated statistics since boot for all pools on the system. For example:

```
# zpool iostat
capacity      operations      bandwidth
pool          alloc   free    read  write   read  write
-----
rpool         6.05G  61.9G      0      0     786   107
system1       31.3G  36.7G      4      1    296K  86.1K
-----
```

Because these statistics are cumulative since boot, bandwidth might appear low if the pool is relatively idle. You can request a more accurate view of current bandwidth usage by specifying an interval. For example:

```
# zpool iostat system1 2
capacity      operations      bandwidth
pool          alloc   free    read  write   read  write
-----
system1       18.5G  49.5G      0     187      0  23.3M
system1       18.5G  49.5G      0     464      0  57.7M
system1       18.5G  49.5G      0     457      0  56.6M
system1       18.8G  49.2G      0     435      0  51.3M
```

In this example, the command displays usage statistics for the pool `system1` every two seconds until you press Control-C. Alternately, you can specify an additional count argument, which causes the command to terminate after the specified number of iterations.

For example, `zpool iostat 2 3` would print a summary every two seconds for three iterations, for a total of six seconds.

Listing Virtual Device I/O Statistics

The `zpool iostat -v` command can display I/O statistics for virtual devices. Use this command to identify abnormally slow devices or to observe the distribution of I/O generated by ZFS. For example:

```
# zpool iostat -v
capacity      operations      bandwidth
pool          alloc   free    read  write   read  write
-----
rpool         6.05G  61.9G      0      0     785   107
  mirror      6.05G  61.9G      0      0     785   107
    clt0d0s0    -      -      0      0     578   109
    clt1d0s0    -      -      0      0     595   109
```

system1	36.5G	31.5G	4	1	295K	146K
mirror	36.5G	31.5G	126	45	8.13M	4.01M
c1t2d0	-	-	0	3	100K	386K
c1t3d0	-	-	0	3	104K	386K

Note two important points when viewing I/O statistics for virtual devices:

- Statistics on disk space use are available only for top-level virtual devices. The way in which disk space is allocated among mirror and RAID-Z virtual devices is particular to the implementation and not easily expressed as a single number.
- The numbers might not add up exactly as you would expect. In particular, operations across RAID-Z and mirrored devices will not be exactly equal. This difference is particularly noticeable immediately after a pool is created because a significant amount of I/O is done directly to the disks as part of pool creation, which is not accounted for at the mirror level. Over time, these numbers gradually equalize. However, broken, unresponsive, or offline devices can affect this symmetry as well.

You can use `interval` and `count` when examining virtual device statistics.

You can also display physical location information about the pool's virtual devices. The following example shows sample output that has been truncated:

```
# zpool iostat -lv
          capacity      operations      bandwidth
pool      alloc  free    read  write    read  write
-----
export    2.39T  2.14T     13    27   42.7K   300K
mirror    490G  438G      2     5    8.53K   60.3K
  /dev/chassis/lab10rack15/SCSI_Device__2/disk - - 1 0 4.47K 60.3K
  /dev/chassis/lab10rack15/SCSI_Device__3/disk - - 1 0 4.45K 60.3K
mirror    490G  438G      2     5    8.62K   59.9K
  /dev/chassis/lab10rack15/SCSI_Device__4/disk - - 1 0 4.52K 59.9K
  /dev/chassis/lab10rack15/SCSI_Device__5/disk - - 1 0 4.48K 59.9K
```

Determining the Health Status of ZFS Storage Pools

You can display pool and device health by using the `zpool status` command. In addition, the `fmd` command also reports potential pool and device failures on the system console, and the `/var/adm/messages` file.

This section describes only how to determine pool and device health. For data recovery from unhealthy pools, see [Chapter 12, “Oracle Solaris ZFS Troubleshooting and Pool Recovery”](#).

A pool's health status is described by one of four states:

DEGRADED

A pool with one or more failed devices whose data is still available due to a redundant configuration.

ONLINE

A pool that has all devices operating normally.

SUSPENDED

A pool that is waiting for device connectivity to be restored. A **SUSPENDED** pool remains in the wait state until the device issue is resolved.

UNAVAIL

A pool with corrupted metadata, or one or more unavailable devices, and insufficient replicas to continue functioning.

Each pool device can fall into one of the following states:

DEGRADED	The virtual device has experienced a failure but can still function. This state is most common when a mirror or RAID-Z device has lost one or more constituent devices. The fault tolerance of the pool might be compromised because a subsequent fault in another device might be unrecoverable.
OFFLINE	The device has been explicitly taken offline by the administrator.
ONLINE	The device or virtual device is in normal working order even though some transient errors might still occur.
REMOVED	The device was physically removed while the system was running. Device removal detection is hardware-dependent and might not be supported on all platforms.
UNAVAIL	The device or virtual device cannot be opened. In some cases, pools with UNAVAIL devices appear in DEGRADED mode. If a top-level virtual device is UNAVAIL , then nothing in the pool can be accessed.

The health of a pool is determined from the health of all its top-level virtual devices. If all virtual devices are **ONLINE**, then the pool is also **ONLINE**. If any one of the virtual devices is **DEGRADED** or **UNAVAIL**, then the pool is also **DEGRADED**. If a top-level virtual device is **UNAVAIL** or **OFFLINE**, then the pool is also **UNAVAIL** or **SUSPENDED**. A pool in the **UNAVAIL** or **SUSPENDED** state is completely inaccessible. No data can be recovered until the necessary devices are attached or repaired. A pool in the **DEGRADED** state continues to run but you might not achieve the same level of data redundancy or data throughput than if the pool were online.

The `zpool status` command also displays the state of resilver and scrub operations as follows:

- Resilver or scrub operations are in progress.
- Resilver or scrub operations have been completed.
Resilver and scrub completion messages persist across system reboots.
- Operations have been cancelled.

Storage Pool Health Status

You can review pool health status by using one of the following `zpool status` command options:

- `zpool status -x [pool]` – Displays only the status pools that have errors or are otherwise unavailable.
- `zpool status -v [pool]` – Generates verbose output providing detailed information about the pools and their devices.

You should investigate any pool that is not in the **ONLINE** state for potential problems.

The following example shows how to generate a verbose status report about the pool system1.

```
# zpool status -v system1
pool: system1
state: DEGRADED
status: One or more devices are unavailable in response to persistent errors.
        Sufficient replicas exist for the pool to continue functioning in a
        degraded state.
action: Determine if the device needs to be replaced, and clear the errors
        using 'zpool clear' or 'fmadm repaired', or replace the device
        with 'zpool replace'.
scan: scrub repaired 0 in 0h0m with 0 errors on Wed Jun 20 15:38:08 2012
config:
```

NAME	STATE	READ	WRITE	CKSUM
system1	DEGRADED	0	0	0
mirror-0	DEGRADED	0	0	0
c0t5000C500335F95E3d0	ONLINE	0	0	0
c0t5000C500335F907Fd0	UNAVAIL	0	0	0
mirror-1	ONLINE	0	0	0
c0t5000C500335BD117d0	ONLINE	0	0	0
c0t5000C500335DC60Fd0	ONLINE	0	0	0

device details:

```
c0t5000C500335F907Fd0    UNAVAIL    cannot open
status: ZFS detected errors on this device.
        The device was missing.
```

see: *URL to My Oracle Support knowledge article* for recovery

errors: No known data errors

The READ and WRITE columns provide a count of I/O errors that occurred on the device, while the CKSUM column provides a count of uncorrectable checksum errors that occurred on the device. Both error counts indicate a potential device failure for which some corrective action is needed. If non-zero errors are reported for a top-level virtual device, portions of your data might have become inaccessible.

The output identifies problems as well as possible causes for the pool's current state. The output also includes a link to a knowledge article for up-to-date information about the best way to recover from the problem. From the output, you can determine which device is damaged and how to repair the pool.

For more information about diagnosing and repairing UNAVAIL pools and data, see [Chapter 12, “Oracle Solaris ZFS Troubleshooting and Pool Recovery”](#).

Gathering ZFS Storage Pool Status Information

You can use the `zpool status` interval and count options to gather statistics over a period of time. In addition, you can display a time stamp by using the `-T` option. For example:

```
# zpool status -T d 3 2
Wed Jun 20 16:10:09 MDT 2012
  pool: pond
  state: ONLINE
    scan: resilvered 9.50K in 0h0m with 0 errors on Wed Jun 20 16:07:34 2012
config:
```

NAME	STATE	READ	WRITE	CKSUM
pond	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c0t5000C500335F95E3d0	ONLINE	0	0	0
c0t5000C500335F907Fd0	ONLINE	0	0	0
mirror-1	ONLINE	0	0	0
c0t5000C500335BD117d0	ONLINE	0	0	0
c0t5000C500335DC60Fd0	ONLINE	0	0	0

errors: No known data errors

```
  pool: rpool
  state: ONLINE
    scan: scrub repaired 0 in 0h11m with 0 errors on Wed Jun 20 15:08:23 2012
config:
```

NAME	STATE	READ	WRITE	CKSUM
rpool	ONLINE	0	0	0


```

mirror-0          ONLINE      0      0      0
c0t5000C500335BA8C3d0s0 ONLINE      0      0      0
c0t5000C500335FC3E7d0s0 ONLINE      0      0      0

errors: No known data errors
Wed Jun 20 16:10:12 MDT 2012

pool: pond
state: ONLINE
scan: resilvered 9.50K in 0h0m with 0 errors on Wed Jun 20 16:07:34 2012
config:

NAME              STATE      READ WRITE CKSUM
pond              ONLINE      0      0      0
  mirror-0        ONLINE      0      0      0
    c0t5000C500335F95E3d0 ONLINE      0      0      0
    c0t5000C500335F907Fd0 ONLINE      0      0      0
  mirror-1        ONLINE      0      0      0
    c0t5000C500335BD117d0 ONLINE      0      0      0
    c0t5000C500335DC60Fd0 ONLINE      0      0      0

errors: No known data errors

pool: rpool
state: ONLINE
scan: scrub repaired 0 in 0h11m with 0 errors on Wed Jun 20 15:08:23 2012
config:

NAME              STATE      READ WRITE CKSUM
rpool             ONLINE      0      0      0
  mirror-0        ONLINE      0      0      0
    c0t5000C500335BA8C3d0s0 ONLINE      0      0      0
    c0t5000C500335FC3E7d0s0 ONLINE      0      0      0

errors: No known data errors

```

Migrating ZFS Storage Pools

Occasionally, you might need to move a storage pool between systems. You would disconnect the storage devices from the original system and reconnect them to the destination system either by physically recabling the devices or by using multiported devices such as the devices on a SAN.

ZFS enables you to export the pool from one system and import it on the destination system even if the systems are of different architectural endianness. For information about replicating or migrating file systems between different storage pools, which might reside on different systems, see [“Saving, Sending, and Receiving ZFS Data” on page 170](#).

Preparing for ZFS Storage Pool Migration

To migrate a pool, you must first export it. This operation flushes any unwritten data to disk, writes data to the disk indicating that the export is complete, and removes all information about the pool from the system.

If you do not explicitly export the pool but instead remove the disks manually, you can still import the resulting pool on another system. However, you might lose the last few seconds of data transactions. Also, because the devices are no longer present, the pool will appear as `UNAVAIL` on the original system. By default, the destination system cannot import a pool that has not been explicitly exported. This condition is necessary to prevent you from accidentally importing an active pool that consists of network-attached storage that is still in use on another system.

Exporting a ZFS Storage Pool

To export a pool, use the following command:

```
# zpool export [option] pool
```

The command first unmounts any mounted file systems within the pool. If any of the file systems fail to unmount, you can forcefully unmount them by using the `-f` option. However, if ZFS volumes in the pool are in use, the operation fails even with the `-f` option. To export a pool with a ZFS volume, first ensure that all consumers of the volume are no longer active.

For more information, see [“ZFS Volumes” on page 221](#).

After this command is executed, the pool is no longer visible on the system.

If devices are unavailable at the time of export, the devices cannot be identified as cleanly exported. If one of these devices is later attached to a system without any of the other working devices, it appears as potentially active.

Determining Available Storage Pools to Import

After the pool has been removed from the system, you can attach the devices to the target system. You do not need to attach them under the same device name. ZFS detects any moved or renamed devices and adjusts the configuration appropriately. Note that ZFS can handle some situations in which only some of the devices are available. However, a successful pool migration depends on the overall health of all the devices.

Use the following general command syntax for all pool import operations:

```
# zpool import [options] [pool|ID-number]
```

To discover available pools that can be imported, run the `zpool import` command without specifying pools. In the output, the pools are identified by names and unique number identifiers. If pools available for import share the same name, use the numeric identifier to import the correct pool.

If problems exist with a pool to be imported, the command output also provides the appropriate information to help you determine what action to take.

In the following example, one of the devices is missing but you can still import the pool because the mirrored data remains accessible.

```
# zpool import
pool: system1
  id: 4715259469716913940
  state: DEGRADED
status: One or more devices are unavailable.
action: The pool can be imported despite missing or damaged devices. The
        fault tolerance of the pool may be compromised if imported.
config:

    system1                                DEGRADED
      mirror-0                            DEGRADED
        c0t5000C500335E106Bd0            ONLINE
        c0t5000C500335FC3E7d0            UNAVAIL  cannot open

device details:

    c0t5000C500335FC3E7d0    UNAVAIL  cannot open
status: ZFS detected errors on this device.
        The device was missing.
```

In the following example, because two disks are missing from a RAID-Z virtual device, not enough redundant data exists to reconstruct the pool. With insufficient available devices, ZFS cannot import the pool.

```
# zpool import
pool: mothership
  id: 3702878663042245922
  state: UNAVAIL
status: One or more devices are unavailable.
action: The pool cannot be imported due to unavailable devices or data.
config:

    mothership    UNAVAIL  insufficient replicas
      raidz1-0    UNAVAIL  insufficient replicas
        c8t0d0    UNAVAIL  cannot open
        c8t1d0    UNAVAIL  cannot open
        c8t2d0    ONLINE
        c8t3d0    ONLINE

device details:
```

```
c8t0d0    UNAVAIL          cannot open
status: ZFS detected errors on this device.
        The device was missing.

c8t1d0    UNAVAIL          cannot open
status: ZFS detected errors on this device.
        The device was missing.
```

Importing ZFS Storage Pools

To import a specific pool, specify the pool name or its numeric identifier with the `zfs import` command. Additionally, you can rename a pool while importing it. For example:

```
# zpool import system1 mpool
```

This command imports the exported pool `system1` and renames it `mpool`. The new pool name is persistent.

Note - You cannot rename a pool directly. You can only change the name of a pool while exporting and importing the pool, which also renames the root dataset to the new pool name.



Caution - During an import operation, warnings occur if the pool might be in use on another system.

```
cannot import 'pool': pool may be in use on another system
use '-f' to import anyway
```

Do not attempt to import a pool that is active on one system to another system. ZFS is not a native cluster, distributed, or parallel file system and cannot provide concurrent access from multiple, different stems.

You can also import pools under an alternate root by using the `-R` option. For more information, see [“Using a ZFS Pool With an Alternate Root Location” on page 230](#).

Importing a Pool With a Missing Log Device

By default, a pool with a missing log device cannot be imported. You can use `zpool import -m` command to force a pool to be imported with a missing log device.

In the following example, the output indicates a missing mirrored log when you first import the pool `dozer`.

```
# zpool import dozer
The devices below are missing, use '-m' to import the pool anyway:
```

```
mirror-1 [log]
c3t3d0
c3t4d0
```

cannot import 'dozer': one or more devices is currently unavailable

To proceed with importing the pool with the missing mirrored log, use the `-m` option.

```
# zpool import -m dozer
# zpool status dozer
pool: dozer
state: DEGRADED
status: One or more devices could not be opened. Sufficient replicas exist for
the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
see: URL to My Oracle Support knowledge article
scan: scrub repaired 0 in 0h0m with 0 errors on Fri Oct 15 16:51:39 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM	
dozer	DEGRADED	0	0	0	
mirror-0	ONLINE	0	0	0	
c3t1d0	ONLINE	0	0	0	
c3t2d0	ONLINE	0	0	0	
logs					
mirror-1	UNAVAIL	0	0	0	insufficient replicas
13514061426445294202	UNAVAIL	0	0	0	was c3t3d0
16839344638582008929	UNAVAIL	0	0	0	was c3t4d0

The imported pool remains in a DEGRADED state. Based on the output recommendation, attach the missing log devices. Then, run the `zpool clear` command to clear the pool errors.

Importing a Pool in Read-Only Mode

If a pool is so damaged that it cannot be accessed, importing in read-only mode might enable you to recover the pool's data. For example:

```
# zpool import -o readonly=on system1
# zpool scrub system1
cannot scrub system1: pool is read-only
```

When a pool is imported in read-only mode, the following conditions apply:

- All file systems and volumes are mounted in read-only mode.
- Pool transaction processing is disabled. Any pending synchronous writes in the intent log are not made until the pool is imported in read-write mode.
- Setting a pool property during the read-only import is ignored.

You can set the pool back to read-write mode by exporting and importing the pool. For example:

```
# zpool export system1
# zpool import system1
# zpool scrub system1
```

Importing a Pool By Using a Specific Device Path

By default, the `zpool import` command searches devices only within the `/dev/dsk` directory. If devices exist in another directory, or you are using pools backed by files, you must use the `-d` option to search alternate directories. For example:

```
# zpool create mpool mirror /file/a /file/b
# zpool export mpool
# zpool import -d /file
pool: mpool
   id: 7318163511366751416
  state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

      mpool      ONLINE
      mirror-0   ONLINE
        /file/a  ONLINE
        /file/b  ONLINE
# zpool import -d /file mpool
```

If devices exist in multiple directories, you can specify multiple `-d` options.

The following command imports the pool `mpool` by identifying one of the pool's specific devices, `/dev/etc/c2t3d0`:

```
# zpool import -d /dev/etc/c2t3d0 mpool
# zpool status mpool
pool: mpool
state: ONLINE
scan: resilvered 952K in 0h0m with 0 errors on Fri Jun 29 16:22:06 2012
config:

NAME          STATE      READ WRITE CKSUM
mpool         ONLINE         0     0     0
mirror-0     ONLINE         0     0     0
c2t3d0       ONLINE         0     0     0
c2t1d0       ONLINE         0     0     0
```

Recovering Destroyed ZFS Storage Pools

You can use the `zpool import -D` command to recover a storage pool that has been destroyed.

In the following example, the pool `system1` is indicated as destroyed.

```
# zpool import -D
pool: system1
id: 5154272182900538157
state: ONLINE (DESTROYED)
action: The pool can be imported using its name or numeric identifier.
config:
```

```
system1      ONLINE
mirror-0     ONLINE
  c1t0d0     ONLINE
  c1t1d0     ONLINE
```

To recover the destroyed pool, import it with the `-D` option.

```
# zpool import -D system1
# zpool status system1
pool: system1
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
system1	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c1t0d0	ONLINE	0	0	0
c1t1d0	ONLINE	0	0	0

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

If one of the devices in the destroyed pool is unavailable, you might still recover the destroyed pool by including the `-f` option. In this scenario, you would import the degraded pool and then attempt to fix the device failure. For example:

```
# zpool import -D
pool: dozer
id: 4107023015970708695
state: DEGRADED (DESTROYED)
status: One or more devices are unavailable.
action: The pool can be imported despite missing or damaged devices. The
        fault tolerance of the pool may be compromised if imported.
config:
```

dozer	DEGRADED
raidz2-0	DEGRADED
c8t0d0	ONLINE
c8t1d0	ONLINE
c8t2d0	ONLINE
c8t3d0	UNAVAIL cannot open
c8t4d0	ONLINE

device details:

```
        c8t3d0    UNAVAIL          cannot open
        status: ZFS detected errors on this device.
              The device was missing.
# zpool import -Df dozer
# zpool status -x
  pool: dozer
  state: DEGRADED
status: One or more devices are unavailable in response to persistent errors.
       Sufficient replicas exist for the pool to continue functioning in a
       degraded state.
action: Determine if the device needs to be replaced, and clear the errors
       using 'zpool clear' or 'fmadm repaired', or replace the device
       with 'zpool replace'.
       Run 'zpool status -v' to see device specific details.
scan: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
dozer	DEGRADED	0	0	0
raidz2-0	DEGRADED	0	0	0
c8t0d0	ONLINE	0	0	0
c8t1d0	ONLINE	0	0	0
c8t2d0	ONLINE	0	0	0
4881130428504041127	UNAVAIL	0	0	0
c8t4d0	ONLINE	0	0	0

```
errors: No known data errors
# zpool online dozer c8t3d0
# zpool status -x
all pools are healthy
```

Upgrading ZFS Storage Pools

With the `zpool upgrade` command, you can upgrade ZFS storage pools from a previous Oracle Solaris release.

Before using the command, use the `zpool status` command to check whether the pools were configured with a ZFS version that is previous to the version currently on the system. Also, consider displaying the features of the current ZFS version on the system by using the `-v` option as shown below:

```
# zpool upgrade -v
```

The list of features would vary depending on the ZFS version number on the system. See [“ZFS Pool Versions” on page 281](#) for a complete list.

Use the `-a` option to upgrade the pools and take advantage of the latest ZFS features.

```
# zpool upgrade -a
```

After you upgrade the pools, they are no longer accessible on a system that is running a previous ZFS version.

EXAMPLE 15 Upgrading ZFS Pools

This example shows the actions to upgrade pools.

```
# zpool status
pool: system1
state: ONLINE
status: The pool is formatted using an older on-disk format. The pool can
still be used, but some features are unavailable.
action: Upgrade the pool using 'zpool upgrade'. Once this is done, the
pool will no longer be accessible on older software versions.
scrub: none requested
config:
      NAME      STATE    READ WRITE CKSUM
      system1   ONLINE      0     0     0
        mirror-0 ONLINE      0     0     0
          clt0d0 ONLINE      0     0     0
          clt1d0 ONLINE      0     0     0
errors: No known data errors
```

```
# zpool upgrade -v
```

This system is currently running ZFS pool version *version-number*.

The following versions are 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
.
.
    Additional features
```

```
# zpool upgrade -a
```


Managing the ZFS Root Pool

This chapter describes how to manage the Oracle Solaris ZFS root pool and its components. It covers the following topics:

- “Requirements for Configuring the ZFS Root Pool” on page 75
- “Managing a ZFS Root Pool” on page 78
- “Managing ZFS Swap and Dump Devices” on page 86
- “Booting From a ZFS Root File System” on page 90

For information about root pool recovery, see *Using Unified Archives for System Recovery and Cloning in Oracle Solaris 11.3*.

Requirements for Configuring the ZFS Root Pool

ZFS is the default root file system of Oracle Solaris. The root pool contains the boot environment (BE) and is automatically created during the installation.

ZFS Root Pool Space Requirements

The size of the swap dump volumes depend on the amount of physical memory. The minimum amount of pool space for a bootable ZFS root file system depends upon the amount of physical memory, the disk space available, and the number of BEs to be created.

The 7 GB-13 GB minimum disk space recommended in “[Hardware and Software Requirements](#)” on page 21 is consumed as follows:

- Swap area and dump device – The default sizes of the swap and dump volumes that the installation program creates vary based on variables such as the amount of system memory. The dump device size is approximately half the size of physical memory or greater, depending on the system's activity.

You can adjust the sizes of your swap and dump volumes during or after installation. The new sizes must support system operation. See “[Adjusting the Sizes of ZFS Swap and Dump Devices](#)” on page 89.

- Boot environment – A ZFS BE is approximately 4 GB-6 GB. Each ZFS BE that is cloned from another ZFS BE does not need additional disk space. The BE size will increase when it is updated, depending on the updates. All ZFS BEs in the same root pool use the same swap and dump devices.
- Oracle Solaris Components – All subdirectories of the root file system except /var that are part of the OS image must be in the root file system. All Oracle Solaris components except the swap and dump devices must reside in the root pool.

ZFS Root Pool Configuration Recommendations

Follow these guidelines when configuring the ZFS root pool:

- If you are using EFI (GPT) labeled disks, create the root pool on mirrored whole disks. If you are using SMI (VTOC) labeled disks, create the root pool on mirrored slices.
In most cases, x86 systems and SPARC systems with GPT aware firmware have EFI (GPT) labeled disks. Otherwise, SPARC systems have SMI (VTOC) labeled disks.
- Do not rename the root pool that is created after an initial installation because the system might become unbootable.
- Do not use a thinly provisioned VMware device for a root pool device.

Root pools have the following limitations:

- RAID-Z or striped configurations are not supported for root pools.
- Root pools cannot have a separate log device.
- You cannot configure multiple top-level virtual devices on root pools. However, you can expand a mirrored root pool by attaching additional devices.
- The gzip and lz4 compression algorithms are not supported on root pools.

Installing the ZFS Root Pool

As documented in [Installing Oracle Solaris 11.3 Systems](#), you can use Live Media, text installer, or Automated Installer (AI) with the AI manifest to install Oracle Solaris. All three methods automatically install a ZFS root pool on a single disk. The installation also configures swap and dump devices on ZFS volumes on the root pool.

The AI method offers more flexibility in installing the root pool. In the AI manifest, you can specify the disks to use to create a mirrored root pool as well as enable ZFS properties, as shown in [Example 16, “Modifying the AI Manifest to Customize Root Pool Installation,” on page 77](#).

After Oracle Solaris is completely installed, perform the following actions:

- If the installation created a root pool on a single disk, then manually convert the pool into a mirrored configuration. See [“How to Configure a Mirrored Root Pool \(SPARC or x86/VTOC\)” on page 79](#).
- Set a quota on the ZFS root file system to prevent the root file system from filling up. Currently, no ZFS root pool space is reserved as a safety net for a full file system. For example, if you have a 68 GB disk for the root pool, consider setting a 67 GB quota on the ZFS root file system (`rpool/ROOT/solaris`) to allow for 1 GB of remaining file system space. See [“Setting Quotas on ZFS File Systems” on page 141](#).
- Create a root pool recovery archive for disaster recovery or for migration purposes by using the Oracle Solaris archive utility. For more information, refer to [Using Unified Archives for System Recovery and Cloning in Oracle Solaris 11.3](#) and the `archiveadm(1M)` man page.

EXAMPLE 16 Modifying the AI Manifest to Customize Root Pool Installation

This example shows how to customize the AI manifest to perform the following:

- Create a mirrored root pool consisting of `c1t0d0` and `c2t0d0`.
- Enable the root pool's `listsnapshots` property.

```
<target>
<disk whole_disk="true" in_zpool="rpool" in_vdev="mirrored">
<disk_name name="c1t0d0" name_type="ctd"/>
</disk>
<disk whole_disk="true" in_zpool="rpool" in_vdev="mirrored">
<disk_name name="c2t0d0" name_type="ctd"/>
</disk>
<logical>
<zpool name="rpool" is_root="true">
<vdev name="mirrored" redundancy="mirror"/>
<!--
...
-->
<filesystem name="export" mountpoint="/export"/>
<filesystem name="export/home"/>
<pool_options>
<option name="listsnapshots" value="on"/>
</pool_options>
<be name="solaris"/>
</zpool>
</logical>
</target>
```

EXAMPLE 17 Sample Root Pool Configuration

The following example shows a mirrored root pool and file system configuration after an AI installation with a customized manifest.

```
# zpool status rpool
```

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

NAME	STATE	READ	WRITE	CKSUM
rpool	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c8t0d0	ONLINE	0	0	0
c8t1d0	ONLINE	0	0	0

```
# zfs list
```

NAME	USED	AVAIL	REFER	MOUNTPOINT
rpool	11.8G	55.1G	4.58M	/rpool
rpool/ROOT	3.57G	55.1G	31K	legacy
rpool/ROOT/solaris	3.57G	55.1G	3.40G	/
rpool/ROOT/solaris/var	165M	55.1G	163M	/var
rpool/VARSHARE	42.5K	55.1G	42.5K	/var/share
rpool/dump	6.19G	55.3G	6.00G	-
rpool/export	63K	55.1G	32K	/export
rpool/export/home	31K	55.1G	31K	/export/home
rpool/swap	2.06G	55.2G	2.00G	-

Managing a ZFS Root Pool

This section provides procedures for managing the ZFS root pool.

▼ How to Configure a Mirrored Root Pool (SPARC or x86/EFI (GPT))

This procedure describes how to convert the default root pool installation into a redundant configuration. This procedure applies to most x86 systems and SPARC systems with GPT-aware firmware whose disks have the EFI (GPT) label.

1. (Optional) Display the current root pool status.

```
# zpool status root-pool
```

2. Attach a second disk to configure a mirrored root pool.

```
# zpool attach root-pool current-disk new-disk
```

The correct disk labeling and the boot blocks are applied automatically.

3. View the root pool status to confirm that resilvering is complete.

If resilvering has been completed, the output includes a message similar to the following:

```
scan: resilvered 11.6G in 0h5m with 0 errors on Fri Jul 20 13:57:25 2014
```

4. If the new disk is larger than the current disk, enable the ZFS autoexpand property.

```
# zpool set autoexpand=on root-pool
```

The following example shows the difference in the rpool's disk space after the autoexpand property is enabled.

```
# zpool list rpool
NAME    SIZE  ALLOC   FREE  CAP  DEDUP  HEALTH  ALTROOT
rpool   29.8G   152K   29.7G   0%   1.00x  ONLINE  -
```

```
# zpool set autoexpand=on rpool
```

```
# zpool list rpool
NAME    SIZE  ALLOC   FREE  CAP  DEDUP  HEALTH  ALTROOT
rpool   279G   146K   279G   0%   1.00x  ONLINE  -
```

5. Verify that you can boot successfully from the new disk.

Note - Unexpected behavior might occur if the ZFS configuration consists of a root file system that is built on mirrored iSCSI targets and the second LUN is not available on the same iSCSI target or session as the boot disk. When the system is booted, the boot process would report that opening the second iSCSI LUN failed and the root pool is in a degraded state. However, this status is temporary. The issue automatically resolves after the ZFS performs a quick resilvering. The second LUN then goes online and the state of the root pool goes online as well.

▼ How to Configure a Mirrored Root Pool (SPARC or x86/VTOC)

This procedure describes how to convert the default root pool installation into a redundant configuration. This procedure applies to certain x86 systems and SPARC systems without GPT-aware firmware whose disks have the SMI (VTOC) label.

Before You Begin Prepare the second disk to attach to the root pool as follows:

- **SPARC:** Confirm that the disk has an SMI (VTOC) disk label and that slice 0 contains the bulk of the disk space. If you need to relabel the disk and create a slice 0, see [“How to Replace a ZFS Root Pool \(VTOC\)” in *Managing Devices in Oracle Solaris 11.3*](#).
- **x86:** Confirm that the disk has an fdisk partition, an SMI disk label, and a slice 0. If you need to repartition the disk and create a slice 0, see [“Modifying Slices or Partitions” in *Managing Devices in Oracle Solaris 11.3*](#).

1. (Optional) Display the current root pool status.

```
# zpool status root-pool
```

The configuration would display the disk's slice 0, as shown in the following example for rpool.

```
# zpool status rpool
```

```
pool: rpool
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
rpool	ONLINE	0	0	0
c2t0d0s0	ONLINE	0	0	0

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

2. Attach the second disk to configure a mirrored root pool.

```
# zpool attach root-pool current-disk new-disk
```

Make sure that you include the slice when specifying the disk, such as c2t0d0s0. The correct disk labeling and the boot blocks are applied automatically.

3. View the root pool status to confirm that resilvering is complete.

If resilvering has been completed, the output includes a message similar to the following:

```
scan: resilvered 11.6G in 0h5m with 0 errors on Fri Jul 20 13:57:25 2014
```

4. If the new disk is larger than the current disk, enable the ZFS autoexpand property.

```
# zpool set autoexpand=on root-pool
```

The following example shows the difference in the rpool's disk space after the autoexpand property is enabled.

```
# zpool list rpool
```

NAME	SIZE	ALLOC	FREE	CAP	DEDUP	HEALTH	ALTROOT
rpool	29.8G	152K	29.7G	0%	1.00x	ONLINE	-

```
# zpool set autoexpand=on rpool
```

```
# zpool list rpool
```

NAME	SIZE	ALLOC	FREE	CAP	DEDUP	HEALTH	ALTROOT
rpool	279G	146K	279G	0%	1.00x	ONLINE	-

5. Verify that you can boot successfully from the new disk.

Note - Unexpected behavior might occur if the ZFS configuration consists of a root file system that is built on mirrored iSCSI targets and the second LUN is not available on the same iSCSI target or session as the boot disk. When the system is booted, the boot process would report that opening the second iSCSI LUN failed and the root pool is in a degraded state. However, this status is temporary. The issue automatically resolves after the ZFS performs a quick resilvering. The second LUN then goes online and the state of the root pool goes online as well.

▼ How to Update a ZFS Boot Environment

By default, the ZFS BE is named `solaris`. The `pkg update` command updates the ZFS BE by creating and automatically activating a new BE, provided that significant differences exist between the current and updated BEs.

1. (Optional) View the current boot environment configuration.

The BE's Active field shows `N` to indicate that the BE is active, `R` to indicate that it becomes active after a system reboot, or both (`NR`).

```
# beadm list
BE      Active Mountpoint Space Policy Created
--      -
solaris NR    /          3.82G static 2012-07-19 13:44
```

2. Update the ZFS BE.

```
# pkg update
.
DOWNLOAD                                PKGS      FILES    XFER (MB)
Completed                                707/707 10529/10529 194.9/194.9
.
```

A new BE, `solaris-1`, is created automatically and activated.

3. Reboot the system to complete the BE activation. Then, confirm the BE status.

```
# init 6
.
.
# beadm list
BE      Active Mountpoint Space Policy Created
--      -
solaris -      -          46.95M static 2014-07-20 10:25
solaris-1 NR    /          3.82G static 2014-07-19 14:45
```

4. If an error occurs when booting the new BE, activate and boot the previous BE.

```
# beadm activate solaris
# init 6
```

You use the same `beadm activate BE` command syntax to activate an existing backup BE independent of any update operation.

▼ How to Mount an Alternate BE

1. **Become an administrator.**

2. **Mount the alternate BE.**

```
# beadm mount alt-BE /mnt
```

3. **Access the BE.**

```
# ls /mnt
```

4. **Unmount the alternate BE when you're finished.**

```
# beadm umount alt-BE
```

Replacing Disks in a ZFS Root Pool

You might need to replace a disk in the root pool for the following reasons:

- The root pool is too small and you want to replace it with a larger disk
- The root pool disk is failing. In a non-redundant pool, if the disk is failing and the system no longer boots, boot from another source such as a CD or the network. Then, replace the root pool disk.

After replacing a disk with the `zpool replace`, you must apply the boot blocks manually.

In a mirrored root pool configuration, you can replace a disk without having to boot from another source. You use the `zpool replace` to replace a failed disk. Or, you can use the `zpool attach` command if you have an additional disk.

If you are replacing root pool disks that have the SMI (VTOC) label, ensure that you fulfill the following requirements:

- SPARC: Confirm that the disk has an SMI (VTOC) disk label and a slice 0 that contains the bulk of the disk space. If you need to relabel the disk and create a slice 0, see [“How to Replace a ZFS Root Pool \(VTOC\)” in *Managing Devices in Oracle Solaris 11.3*](#).

- x86: Confirm that the disk has an `fdisk` partition, an SMI disk label, and a slice 0. If you need to repartition the disk and create a slice 0, see [“Modifying Slices or Partitions” in *Managing Devices in Oracle Solaris 11.3*](#).

▼ How to Replace a Disk in a ZFS Root Pool

1. **Physically connect the replacement disk.**
2. **Attach the new disk to the root pool.**

```
# zpool attach root-pool current-disk new-disk
```

where *current-disk* becomes the old disk to be detached at the end of this procedure.

The correct disk labeling and the boot blocks are applied automatically.

Note - If the disks have SMI (VTOC) labels, make sure that you include the slice when specifying the disk, such as `c2t0d0s0`.

3. **View the root pool status to confirm that resilvering is complete.**

If resilvering has been completed, the output includes a message similar to the following:

```
scan: resilvered 11.6G in 0h5m with 0 errors on Fri Jul 20 13:57:25 2014
```

4. **Verify that you can boot successfully from the new disk.**
5. **After a successful boot, detach the old disk.**

```
# zpool detach root-pool old-disk
```

Note - If the disks have SMI (VTOC) labels, make sure that you include the slice when specifying the disk, such as `c2t0d0s0`.

6. **If the attached disk is larger than the existing disk, enable the ZFS `autoexpand` property.**

```
# zpool set autoexpand=on root-pool
```

7. **Set the system to boot automatically from the new disk.**

- SPARC: Use either the `eeeprom` command or the `setenv` command from the boot PROM.
- x86: Reconfigure the system BIOS.

Example 18 Replacing a Disk in a ZFS Root Pool (SPARC or x86/EFI (GPT))

This example replaces `c2t0d0` in the root pool named `rpool`. It assumes that the replacement disk `c2t1d0` has been physically connected to the system.

```
# zpool attach rpool c2t0d0 c2t1d0
```

Make sure to wait until resilver is done before rebooting.

```
# zpool status rpool
```

```
pool: rpool
```

```
state: ONLINE
```

```
scan: resilvered 11.7G in 0h5m with 0 errors on Fri Jul 20 13:45:37 2012
```

```
config:
```

NAME	STATE	READ	WRITE	CKSUM
rpool	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c2t0d0	ONLINE	0	0	0
c2t1d0	ONLINE	0	0	0

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

After completing the boot test from the new disk `c2t1d0`, you would detach `c2t0d0` and, if necessary, enable the `autoexpand` property.

```
# zpool detach c2t0d0
```

```
# zpool list rpool
```

NAME	SIZE	ALLOC	FREE	CAP	DEDUP	HEALTH	ALTROOT
rpool	29.8G	152K	29.7G	0%	1.00x	ONLINE	-

```
# zpool set autoexpand=on rpool
```

```
# zpool list rpool
```

NAME	SIZE	ALLOC	FREE	CAP	DEDUP	HEALTH	ALTROOT
rpool	279G	146K	279G	0%	1.00x	ONLINE	-

You would complete the operation by setting the system to automatically boot from the new disk.

Example 19 Replacing SATA Disks in a Root Pool (SPARC or x86/EFI (GPT))

Systems with SATA disks require that before replacing a failed disk with the `zpool replace` command, you take the disk offline and unconfigure it. For example:

```
# zpool offline rpool c1t0d0
```

```
# cfgadm -c unconfigure c1::dsk/c1t0d0
```

You would physically remove the failed disk `c1t0d0` and insert the replacement disk `c1t0d0`. On some hardware, you do not have to bring the disk online or reconfigure the replacement disk after it is inserted.

```
# cfgadm -c configure cl::disk/c1t0d0
# zpool online rpool c1t0d0
# zpool replace rpool c1t0d0
# zpool status rpool
```

On x86 systems, after resilvering is completed, you would install the boot blocks.

```
x86# bootadm install-bootloader
```

Example 20 Replacing a Disk in a ZFS Root Pool (SPARC or x86/VTOC)

This example replaces `c2t0d0s0` in the root pool named `rpool`. It assumes that the replacement disk `c2t1d0s0` has been physically connected to the system.

```
# zpool attach rpool c2t0d0s0 c2t1d0s0
Make sure to wait until resilver is done before rebooting.

# zpool status rpool
pool: rpool
state: ONLINE
scan: resilvered 11.7G in 0h5m with 0 errors on Fri Jul 20 13:45:37 2012
config:

NAME        STATE  READ  WRITE CKSUM
rpool       ONLINE    0     0     0
  mirror-0  ONLINE    0     0     0
    c2t0d0s0 ONLINE    0     0     0
    c2t1d0s0 ONLINE    0     0     0
```

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

You would test booting from the new disk `c2t1d0s0`. You would also test booting from the old disk `c2t0d0s0` in case `c2t1d0s0` fails.

```
ok boot /pci@1f,700000/scsi@2/disk@1,0
```

```
ok boot /pci@1f,700000/scsi@2/disk@0,0
```

After completing the boot tests, you would detach `c2t0d0s0` and, if necessary, enable the `autoexpand` property.

```
# zpool detach c2t0d0s0

# zpool list rpool
NAME  SIZE  ALLOC  FREE  CAP  DEDUP  HEALTH  ALTROOT
rpool 29.8G  152K  29.7G   0%  1.00x  ONLINE  -

# zpool set autoexpand=on rpool

# zpool list rpool
NAME  SIZE  ALLOC  FREE  CAP  DEDUP  HEALTH  ALTROOT
```

```
rpool 279G 146K 279G 0% 1.00x ONLINE -
```

You would complete the operation by setting the system to automatically boot from the new disk.

Example 21 Replacing SATA Disks in a Root Pool (SPARC or x86 (VTOC))

Systems with SATA disks require that before replacing a failed disk with the `zpool replace` command, you take the disk offline and unconfigure it. For example:

```
# zpool offline rpool c1t0d0s0
# cfgadm -c unconfigure c1::disk/c1t0d0
```

You would physically remove the failed disk `c1t0d0` and insert the replacement disk `c1t0d0`. On some hardware, you do not have to bring the disk online or reconfigure the replacement disk after it is inserted.

```
# cfgadm -c configure c1::disk/c1t0d0
```

After confirming that the replacement disk `c1t0d0s0` has an SMI label and a slice 0, you would proceed with the replacement.

```
# zpool replace rpool c1t0d0s0
# zpool online rpool c1t0d0s0
# zpool status rpool
```

After resilvering is completed, you install the boot blocks.

```
# bootadm install-bootloader
```

Managing ZFS Swap and Dump Devices

The installation process automatically creates a swap area and a dump device on a ZFS volume in the ZFS root pool.

The dump device is used when the directory where crash dumps are saved has insufficient space, or if you ran the `dumpadm -n` command syntax. The `-n` modifies the dump configuration to not run `savecore` automatically running after a system reboot.

Certain systems avail of the deferred dump feature in the current Oracle Solaris release. With this feature, a system dump is preserved in memory across a system reboot to enable you to analyze the crash dump after the system reboots. For more information, see [“About Devices and the Oracle Hardware Management Pack” in *Managing Devices in Oracle Solaris 11.3*](#).

Note the following guidelines when managing swap and dump volumes:

- On non-root pools, swap and dump volumes are supported on either single-disk configurations or mirrored configurations.
- You must use separate ZFS volumes for the swap area and dump devices.
- Sparse volumes are not supported for swap volumes.
- Currently, using a swap file on a ZFS file system is not supported.

Viewing Swap and Dump Information

To view the swap area, use the `swap -l` command. For example:

```
# swap -l
swapfile          dev      swaplo  blocks      free
/dev/zvol/dsk/rpool/swap 145,2      16 16646128 16646128
```

To view the dump configuration, use the `dumpadm` command. For example:

```
# dumpadm
Dump content: kernel pages
Dump device: /dev/zvol/dsk/rpool/dump (dedicated)
Savecore directory: /var/crash/
Savecore enabled: yes
Save compressed: on
```

You can also manually create swap or dump volumes in a non-root pool. After creating a dump device on the non-root pool, you must also reset it by running the `dump -d` command.

In the following example, a dump device is created on the non-root pool `bpool`.

```
# zfs create -V 10g bpool/dump2
# dumpadm -d /dev/zvol/dsk/bpool/dump2
Dump content      : kernel with ZFS metadata
Dump device       : /dev/zvol/dsk/bpool/dump2 (dedicated)
Savecore directory: /var/crash
Savecore enabled  : yes
Save compressed   : on
```

For more information about dump and swap devices, see [Chapter 3, “Configuring Additional Swap Space”](#) in *Managing File Systems in Oracle Solaris 11.3* and *Troubleshooting System Administration Issues in Oracle Solaris 11.3*.

▼ How to Create a Swap Volume

This procedure applies to both root pools and non-root pools. If you need more swap space but the existing swap device is busy, just add another swap volume by using this same procedure.

1. **Create the swap volume.**

```
# zfs create -V size new-pool/swap
```

2. **With a text editor, update the `/etc/vfstab` entry for the new swap device.**

See [Example 22, “Manually Creating a Swap Volume,”](#) on page 88 for a sample entry.

3. **Activate the new swap volume if you want to switch to the new swap volume from an existing active swap volume.**

```
# swap -a path-to-new-swap-volume
```

4. **If necessary, reboot the system.**

Example 22 Manually Creating a Swap Volume

This example creates a new 4 GB swap volume in the pool `rpool`. This new swap volume is intended to replace an existing swap volume.

```
# zfs create -V 4g rpool2/swap2
# vi /etc/vfstab
/dev/zvol/dsk/rpool2/swap    -    -    swap    no    -    vfstab entry
# swap -a /dev/zvol/dsk/rpool/swap2
```

▼ How to Create a Dump Volume

This procedure applies whether you are using a root pool or a non-root pool.

1. **Create the dump volume.**

```
# zfs create -V size new-pool/dump
```

2. **Reset the dump device.**

```
# dumpadm -d dump-path
```

3. **If necessary, reboot the system.**

Example 23 Manually Creating a Dump Volume

This example creates a new 4 GB dump volume in the pool `rpool`.

```
# zfs create -V 4g rpool2/dump
# dumpadm -d /dev/zvol/dsk/rpool2/dump
```


Adjusting the Sizes of ZFS Swap and Dump Devices

After installation, you might need to adjust the size of swap and dump devices after installation. Or you might need to recreate the swap and dump volumes.

By default, when you specify *n* blocks for the swap size, the first page of the swap file is automatically skipped. Thus, the actual size that is assigned is *n*-1 blocks. To configure the swap file size differently, use the `swaplow` option with the `swap` command. For more information about the options for the `swap` command, see the [swap\(1M\)](#) man page.

For information on removing a swap device on an active system, see “How to Add Swap Space in an Oracle Solaris ZFS Root Environment” in *Managing File Systems in Oracle Solaris 11.3*.

The following examples show how to adjust existing swap and dump devices under different circumstances.

EXAMPLE 24 Resetting the Dump Device `volsize` Property

Note that resizing a large dump device can be a time-consuming process.

```
# zfs set volsize=2G rpool/dump
# zfs get volsize rpool/dump
NAME          PROPERTY  VALUE   SOURCE
rpool/dump    volsize   2G      -
```

EXAMPLE 25 Resizing the Swap Volume for Immediate Use

This example shows two ways of adjusting the swap size.

The first method enables you to adjust the swap volume without having to reboot the system.

```
# swap -l
swapfile          dev      swaplo    blocks    free
/dev/zvol/dsk/rpool/swap  303,1      8    2097144    2097144
# zfs get volsize rpool/swap
NAME          PROPERTY  VALUE   SOURCE
rpool/swap    volsize   1G      local
# zfs set volsize=2g rpool/swap
# swap -l
swapfile          dev      swaplo    blocks    free
/dev/zvol/dsk/rpool/swap  303,1      8    2097144    2097144
/dev/zvol/dsk/rpool/swap  303,1    2097160    2097144    2097144
```

This second method to adjust swap size requires you to reboot the system for the new size to be displayed.

```
# swap -d /dev/zvol/dsk/rpool/swap
# zfs set volsize=2G rpool/swap
# swap -a /dev/zvol/dsk/rpool/swap
# init 6
```

Troubleshooting ZFS Dump Device Issues

This section describes certain issues and possible resolutions related to dump devices.

- The size of the dump device is too small.

When you reset the dump device, the output includes a message similar to the following:

```
dumpadm: dump device dump-path is too small to hold a system dump
```

To resolve this error, increase the size of the dump device. See [“Adjusting the Sizes of ZFS Swap and Dump Devices” on page 89](#).

For more information about swap and dump sizes, see [“Planning for Swap Space” in *Managing File Systems in Oracle Solaris 11.3*](#).

- The dump device is disabled.

If necessary, create a new dump device and enable it by using the `dumpadm -d` command. See [“How to Create a Dump Volume” on page 88](#).

- A dump device cannot be added to the non-root pool.

When you reset the dump device, the output includes a message similar to the following:

```
dump is not supported on device 'dump-path':
'pool' has multiple top level vdevs
```

Adding a dump device to pools with multiple top-level devices is not supported. Add the dump device to the root pool instead. Root pools support only a single top-level configuration.

- A crash dump was not created automatically.

In this case, use the `savecore` command to save the crash dump.

Booting From a ZFS Root File System

Both SPARC based and x86 based systems boot with a boot archive, which is a file system image that contains the files required for booting. The root file system that is selected for booting contains the path names of both the boot archive and the kernel file.

In the case of a ZFS boot, a device specifier identifies a storage pool, not a single root file system. A storage pool can contain multiple bootable ZFS root file systems. Thus, you must

specify a boot device and a root file system within the pool that was identified by the boot device.

By default, a ZFS boot process uses the file system that is defined in the pool's `bootfs` property. However, you can override the default file system. On SPARC systems, you can use the `boot -Z` command and specify an alternate bootable file system. On x86 systems, you can select an alternate boot device from the BIOS.

If you replace a root pool disk with the `zpool replace` command, you must install the boot information on the replacement disk. However, installing the boot information is not required if you merely attach additional disks to the root pool.

To install the boot information, use the `bootadm` command in one of the following ways:

- To install the boot information on the existing root pool's disk, use the following command:

```
# bootadm install-bootloader
```

- To install the boot information on an alternate pool, use the following command:

```
# bootadm install-bootloader -P alt-root-pool
```

- On x86 systems only, to install the GRUB legacy boot loader, use the following command:

```
x86# installgrub /boot/grub/stage1 /boot/grub/stage2 /dev/rdisk/c0t1d0s0
```

Booting From an Alternate Root Pool Disk

Note - The information in this section applies only to mirrored root pools.

When booting from an alternate root pool disk, ensure that all the root pool's disks are attached and online so you can boot from any of the disks, if necessary. On most systems, you cannot boot directly from a disk that has been detached, or boot from an active root pool disk that is currently offline.

Alternate Boot Disks on SPARC Systems

The primary disk in a mirrored root pool is typically the default boot device. To boot from a different device, you must specify that disk when issuing the command to boot.

If you want to change the default boot device, first display the pool's configuration to select the device you want. Then, at the OK prompt, update the system's PROM with the selected device. Boot the system and confirm that your selected device is the active boot device.

The following example assigns `c1t1d0` as the default boot device.

```
# zpool status
```

```
pool: rpool
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
rpool	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c1t0d0	ONLINE	0	0	0
c1t1d0	ONLINE	0	0	0
...				

```
ok boot /pci@7c0/pci@0/pci@1/pci@0,2/LSILogic,sas@2/disk@1
```

After the system is rebooted, you would confirm which active boot device is in the system.

```
# prtconf -vp | grep bootpath
bootpath: '/pci@7c0/pci@0/pci@1/pci@0,2/LSILogic,sas@2/disk@1,0:a'
```

Alternate Boot Disks on x86 Systems

On x86 based systems with a modern BIOS where the boot disk order is properly set, the system boots automatically from the second device if the primary root pool disk is detached, offline, or unavailable. In such systems, you merely need to confirm which active boot device is in the system, as shown in the following example.

```
# prtconf -v|sed -n '/bootpath/,/value/p'
name='bootpath' type=string items=1
value='/pci@0,0/pci8086,25f8@4/pci108e,286@0/disk@0,0:a'
```

Booting From a ZFS Root File System on a SPARC Based System

On a system with multiple ZFS BEs, you can boot from any BE by using the `beadm activate` command. Both the installation process and the `beadm` activation process automatically set the `bootfs` property.

By default, the `bootfs` property identifies the bootable file system entry in the `/pool-name/boot/menu.lst` file. However, a `menu.lst` entry can contain a `bootfs` command that specifies an alternate file system in the pool. Thus, the `menu.lst` file can contain entries for multiple root file systems within the pool.

When a ZFS root file system is installed, an entry similar to the following example is added to the `menu.lst` file:

```
title release-version SPARC
```

```
bootfs rpool/ROOT/solaris
```

When a new BE is created, the `menu.lst` file is updated automatically.

```
title release-version SPARC
bootfs rpool/ROOT/solaris
title solaris
bootfs rpool/ROOT/solaris2
```

▼ SPARC: How to Select the Boot Environment for Booting

1. **After a ZFS BE is activated, display a list of bootable file systems within a ZFS pool.**

```
# boot -L
```

2. **Select one of the bootable file systems in the list.**

Detailed instructions for booting that file system are displayed.

3. **Boot the selected file system by following the instructions.**

4. **Use the `boot -z file system` command to boot a specific ZFS file system.**

5. **(Optional) To make the selected BE persistent across reboots, activate the BE.**

Example 26 Booting From a Specific ZFS Boot Environment

If you have multiple ZFS BEs in a ZFS storage pool on your system's boot device, use the `beadm activate` command to specify a default BE.

In this example, the `beadm` lists following available ZFS BEs:

```
# beadm list
BE      Active Mountpoint Space Policy Created
--      -
solaris NR      /          3.80G static 2012-07-20 10:25
solaris-2 -      -          7.68M static 2012-07-19 13:44
```

To select a specific BE, you would use the `boot -L` command. For example:

```
ok boot -L
Boot device: /pci@7c0/pci@0/pci@1/pci@0,2/LSILogic,sas@2/disk@0,0:a File and args: -L
1 release-version SPARC
2 solaris
Select environment to boot: [ 1 - 2 ]: 1
```

To boot the selected entry, invoke:

```
boot [<root-device>] -Z rpool/ROOT/solaris-2
```

```
Program terminated
```

```
ok boot -Z rpool/ROOT/solaris-2
```

To boot automatically from the selected BE, activate that BE.

Booting From a ZFS Root File System on an x86 Based System

Starting in Oracle Solaris 11.1, x86 based systems are installed with GRUB2. The `menu.lst` file is replaced by the `/rpool/boot/grub/grub.cfg` file. Do not edit this file manually. Instead, use the `bootadm` sub commands to add, change, and remove menu entries.

Note - If your system's Oracle Solaris version still uses legacy GRUB, see [Booting From a ZFS Root File System on an x86 Based System](#), which describes ZFS root file system entries in the `menu.lst` file.

For more information about modifying the GRUB menu items, see [Booting and Shutting Down Oracle Solaris 11.3 Systems](#).

x86: Displaying the Root File System

When booting from a ZFS root file system on a GRUB2 system, the root device is specified as follows:

```
# bootadm list-menu
the location of the boot loader configuration files is: /rpool/boot/grub
default 0
console text
timeout 30
0 release-version
```

x86: Fast Rebooting a ZFS Root File System

The fast reboot feature provides the ability to reboot within seconds on x86 based systems. With the fast reboot feature, you can reboot to a new kernel without experiencing the long delays that can be imposed by the BIOS and boot loader.

You must still use the `init 6` command when transitioning between BEs with the `beadm activate` command. For other system operations, use the `reboot` command as appropriate.

Booting for Recovery Purposes in a ZFS Root Environment

You might need to boot the system to resolve a corrupt bootloader problem, a root password problem, or a bad shell. For the recovery procedures for each of these cases, see [“Shutting Down and Booting a System for Recovery Purposes” in *Booting and Shutting Down Oracle Solaris 11.3 Systems*](#).

If you need to replace a disk in root pool, see [“Replacing Disks in a ZFS Root Pool” on page 82](#). If you need to perform complete system (bare metal) recovery, see [Using Unified Archives for System Recovery and Cloning in Oracle Solaris 11.3](#).

Managing Oracle Solaris ZFS File Systems

This chapter provides detailed information about managing Oracle Solaris ZFS file systems. Concepts such as the hierarchical file system layout, property inheritance, and automatic mount point management and share interactions are included.

This chapter discusses the following topics:

- [“Introduction to ZFS File Systems” on page 97](#)
- [“Creating, Destroying, and Renaming ZFS File Systems” on page 98](#)
- [“Introducing ZFS Properties” on page 101](#)
- [“Querying ZFS File System Information” on page 116](#)
- [“Managing ZFS Properties” on page 119](#)
- [“Mounting ZFS File Systems” on page 124](#)
- [“Sharing and Unsharing ZFS File Systems” on page 129](#)
- [“Setting ZFS Quotas and Reservations” on page 140](#)
- [“Compressing ZFS File Systems” on page 146](#)
- [“Encrypting ZFS File Systems” on page 147](#)
- [“Migrating ZFS File Systems” on page 154](#)
- [“Upgrading ZFS File Systems” on page 158](#)

Note - The term *dataset* is used in this chapter as a generic term to refer to a file system, snapshot, clone, or volume.

Introduction to ZFS File Systems

You build a ZFS file system on top of a storage pool. ZFS file systems can be dynamically created and destroyed without requiring you to allocate or format any underlying disk space. Because these file systems are so lightweight and because they are the central point of administration in ZFS, you are likely to create many of them.

You can administer ZFS file systems by using the `zfs` command. The `zfs` command provides a set of subcommands that perform specific operations on file systems. This chapter describes

these subcommands in detail. Snapshots, clones, and volumes are also managed by using this command, but these features are only covered briefly in this chapter. For detailed information about snapshots and clones, see [Chapter 8, “Working With Oracle Solaris ZFS Snapshots and Clones”](#). For detailed information about ZFS volumes, see [“ZFS Volumes” on page 221](#).

All invocations of the `zfs` command require the name of the file system. The file system name is specified as a path name starting from the name of the pool as follows:

pool-name/[dataset-path]/filesystem-name

The pool name and the dataset path identify the location of the file system in the hierarchy. The last part in the name identifies the file system name. The file system name must satisfy the naming requirements in [“Naming ZFS Components” on page 22](#). For example, the `tank/home/jeff` file system name would refer to a ZFS file system named `jeff`, in the `/home` dataset path, in the `tank` pool,

Creating, Destroying, and Renaming ZFS File Systems

The following topics are covered in this section:

- [“How to Create a ZFS File System” on page 98](#)
- [“How to Destroy a ZFS File System” on page 99](#)
- [“How to Rename a ZFS File System” on page 100](#)

▼ How to Create a ZFS File System

The `zfs create` command automatically mounts the newly created file system, if it is created successfully. By default, file systems are mounted as `/dataset`, using the dataset name provided with the `create` subcommand. In this example, the newly created `jeff` file system is mounted at `/tank/home/jeff`. For more information about automatically managed mount points, see [“Managing ZFS Mount Points” on page 124](#).

Note - Encrypting a ZFS file system must be enabled when the file system is created. For information about encrypting a ZFS file system, see [“Encrypting ZFS File Systems” on page 147](#).

For more information about the `zfs create` command, see the [`zfs\(1M\)`](#) man page.

1. **Become root.**
2. **Create the ZFS file system.**

Example 27 Creating a Simple ZFS File System

In the following example, a file system named `jeff` is created in the `tank/home` file system.

```
# zfs create tank/home/jeff
```

Example 28 Creating a ZFS File System Using File System Properties

You can set file system properties when a file system is created. In the following example, a mount point of `/export/zfs` is created for the `tank/home` file system:

```
# zfs create -o mountpoint=/export/zfs tank/home
```

For more information about file system properties, see [“Introducing ZFS Properties” on page 101](#).

▼ How to Destroy a ZFS File System

To destroy a ZFS file system, use the `zfs destroy` command. By default, all of the snapshots for the dataset will be destroyed. The destroyed file system is automatically unmounted and unshared. For more information about automatically managed mounts or automatically managed shares, see [“Automatic Mount Points” on page 125](#).

1. **Become root.**
2. **Destroy the ZFS file system.**

```
# zfs destroy tank/home/mark
```



Caution - No confirmation prompt appears with the `destroy` subcommand. Use it with extreme caution.



Caution - No confirmation prompt appears with the `-f`, `-r`, or `-R` options to the `zfs destroy` command, so use these options carefully.

Example 29 Destroying an Active ZFS File System

If the file system to be destroyed is busy and cannot be unmounted, the `zfs destroy` command fails. To destroy an active file system, use the `-f` option. Use this option with caution as it can unmount, unshare, and destroy active file systems, causing unexpected application behavior.

```
# zfs destroy -f tank/home/matt
```

Example 30 Destroying a ZFS File System with Descendents

The `zfs destroy` command also fails if a file system has descendents. To recursively destroy a file system and all its descendents, use the `-r` option.

```
# zfs destroy tank/ws
cannot destroy 'tank/ws': filesystem has children
use '-r' to destroy the following datasets:
tank/ws/jeff
tank/ws/bill
tank/ws/mark
# zfs destroy -r tank/ws
```

Example 31 Destroying a ZFS File System with Dependents

If the file system to be destroyed has indirect dependents, even the recursive destroy command fails. To force the destruction of *all* dependents, including cloned file systems outside the target hierarchy, the `-R` option must be used. Use extreme caution with this option.

```
# zfs destroy -r tank/home/eric
cannot destroy 'tank/home/eric': filesystem has dependent clones
use '-R' to destroy the following datasets:
tank//home/eric-clone
# zfs destroy -R tank/home/eric
```

▼ How to Rename a ZFS File System

File systems can be renamed by using the `zfs rename` command. With the `rename` subcommand, you can perform the following operations:

- Change the name of a file system.
- Relocate the file system within the ZFS hierarchy.
- Change the name of a file system and relocate it within the ZFS hierarchy.

The `rename` operation attempts an unmount/remount sequence for the file system and any descendent file systems. The `rename` command fails if the operation is unable to unmount an active file system. If this problem occurs, you must forcibly unmount the file system. XREF!!!!

1. **Become root.**
2. **Rename the ZFS file system.**

The following example uses the `rename` subcommand to rename of a file system from `eric` to `eric_old`:

Example 32 Relocating a ZFS File System

The following example shows how to use `zfs rename` to relocate a file system:

```
# zfs rename tank/home/mark tank/ws/mark
```

In this example, the `mark` file system is relocated from `tank/home` to `tank/ws`. When you relocate a file system through `rename`, the new location must be within the same pool and it must have enough disk space to hold this new file system. If the new location does not have enough disk space, possibly because it has reached its quota, the `rename` operation fails.

For more information about quotas, see [“Setting ZFS Quotas and Reservations” on page 140](#).

Introducing ZFS Properties

Properties are the main mechanism that you use to control the behavior of file systems, volumes, snapshots, and clones. Unless stated otherwise, the properties defined in this section apply to all the dataset types.

- [“ZFS Read-Only Native Properties” on page 109](#)
- [“Settable ZFS Native Properties” on page 110](#)
- [“ZFS User Properties” on page 115](#)

Properties are divided into two types, native properties and user-defined properties. Native properties either provide internal statistics or control ZFS file system behavior. In addition, native properties are either settable or read-only. User properties have no effect on ZFS file system behavior, but you can use them to annotate datasets in a way that is meaningful in your environment. For more information about user properties, see [“ZFS User Properties” on page 115](#).

Most settable properties are also inheritable. An inheritable property is a property that, when set on a parent file system, is propagated down to all of its descendents.

All inheritable properties have an associated source that indicates how a property was obtained. The source of a property can have the following values:

<code>local</code>	Indicates that the property was explicitly set on the dataset by using the <code>zfs set</code> command as described in “Setting ZFS Properties” on page 119 .
<code>inherited from dataset-name</code>	Indicates that the property was inherited from the named ancestor.
<code>default</code>	Indicates that the property value was not inherited or set locally. This source is a result of no ancestor having the property set as source <code>local</code> .

The following table identifies both read-only and settable native ZFS file system properties. Read-only native properties are identified as such. All other native properties listed in this table are settable. For information about user properties, see [“ZFS User Properties” on page 115](#).

TABLE 3 ZFS Native Property Descriptions

Property Name	Type	Default Value	Description
aclinherit	String	secure	Controls how ACL entries are inherited when files and directories are created. The values are discard, noallow, secure, and passthrough. For a description of these values, see “ACL Properties” on page 190 .
aclmode	String	groupmask	Controls how an ACL entry is modified during a chmod operation. The values are discard, groupmask, and passthrough. For a description of these values, see “ACL Properties” on page 190 .
atime	Boolean	on	Controls whether the access time for files is updated when they are read. Turning this property off avoids producing write traffic when reading files and can result in significant performance gains, though it might confuse mailers and similar utilities.
available	Number	N/A	Read-only property that identifies the amount of disk space available to a file system and all its children, assuming no other activity in the pool. Because disk space is shared within a pool, available space can be limited by various factors including physical pool size, quotas, reservations, and other datasets within the pool. The property abbreviation is avail.
canmount	Boolean	on	Controls whether a file system can be mounted with the <code>zfs mount</code> command. This property can be set on any file system, and the property itself is not inheritable. However, when this property is set to off, a mount point can be inherited to descendent file systems, but the file system itself is never mounted. When the <code>noauto</code> option is set, a file system can only be mounted and unmounted explicitly. The file system is not mounted automatically when the file system is created or imported, nor is it mounted by the <code>zfs mount-a</code> command or unmounted by the <code>zfs unmount-a</code> command. For more information, see “The canmount Property” on page 110 .
casesensitivity	String	mixed	This property indicates whether the file name matching algorithm used by the file system should be <code>casesensitive</code> , <code>caseinsensitive</code> , or allow a combination of both styles of matching (<code>mixed</code>). Traditionally, UNIX and POSIX file systems have case-sensitive file names. The <code>mixed</code> value for this property indicates the file system can support requests for both case-sensitive and case-insensitive matching behavior. Currently, case-insensitive matching behavior on a file system that supports mixed behavior is limited to the Oracle Solaris SMB server product. For more information about using the <code>mixed</code> value, see “The casesensitivity Property” on page 111 .

Property Name	Type	Default Value	Description
			Regardless of the <code>casesensitivity</code> property setting, the file system preserves the case of the name specified to create a file. This property cannot be changed after the file system is created.
<code>checksum</code>	String	<code>on</code>	Controls the checksum used to verify data integrity. The default value is <code>on</code> , which automatically selects an appropriate algorithm, currently <code>fletcher4</code> . The values are <code>on</code> , <code>off</code> , <code>fletcher2</code> , <code>fletcher4</code> , <code>sha256</code> , and <code>sha256+mac</code> . A value of <code>off</code> disables integrity checking on user data. A value of <code>off</code> is not recommended.
<code>compression</code>	String	<code>off</code>	Enables or disables compression for a dataset. The values are <code>on</code> , <code>off</code> , <code>lzjb</code> , <code>lz4</code> , <code>gzip</code> , and <code>gzip-N</code> . Currently, setting this property to <code>lzjb</code> , <code>gzip</code> , or <code>gzip-N</code> has the same effect as setting this property to <code>on</code> . Enabling compression on a file system with existing data only compresses new data. Existing data remains uncompressed. The property abbreviation is <code>compress</code> .
<code>compressratio</code>	Number	N/A	Read-only property that identifies the compression ratio achieved for a dataset, expressed as a multiplier. Compression can be enabled by the <code>zfs set compression=on dataset</code> command. The value is calculated from the logical size of all files and the amount of referenced physical data. It includes explicit savings through the use of the <code>compression</code> property.
<code>copies</code>	Number	<code>1</code>	Sets the number of copies of user data per file system. Available values are 1, 2, or 3. These copies are in addition to any pool-level redundancy. Disk space used by multiple copies of user data is charged to the corresponding file and dataset, and counts against quotas and reservations. In addition, the <code>used</code> property is updated when multiple copies are enabled. Consider setting this property when the file system is created because changing this property on an existing file system only affects newly written data.
<code>creation</code>	String	N/A	Read-only property that identifies the date and time that a dataset was created.
<code>dedup</code>	String	<code>off</code>	Controls the ability to remove duplicate data in a ZFS file system. Possible values are <code>on</code> , <code>off</code> , <code>verify</code> , and <code>sha256[,verify]</code> . The default checksum for deduplication is <code>sha256</code> . For more information, see “The dedup Property” on page 112 .
<code>defaultgroupquota</code>	String	None	Sets a default group quota. The value applies to all groups who do not have an explicit group quota specified. The default value is none. For more information, see “Setting Default User and Group Quotas” on page 144 .
<code>defaultuserquota</code>	String	None	Sets a default user quota. The value applies to all users who do not have an explicit user quota specified. The default value is none. For more information, see “Setting Default User and Group Quotas” on page 144 .
<code>devices</code>	Boolean	<code>on</code>	Controls whether device files in a file system can be opened.
<code>encryption</code>	Boolean	<code>off</code>	Controls whether a file system is encrypted. An encrypted file system means that data is encoded and a key is needed by the file system owner to access the data.

Property Name	Type	Default Value	Description
<code>exec</code>	Boolean	<code>on</code>	Controls whether programs in a file system are allowed to be executed. Also, when set to <code>off</code> , <code>mmap(2)</code> calls with <code>PROT_EXEC</code> are disallowed.
<code>keychangedate</code>	String	<code>none</code>	Read-only property that identifies the date of the last wrapping key change from a <code>zfs key -c</code> operation for the specified file system. If no key change operation has occurred, the value of this property is the same as the file system's creation date.
<code>keysource</code>	String	<code>none</code>	Identifies the format and location of the key that wraps the file system keys. The valid property values are <code>raw</code> , <code>hex</code> , <code>passphrase</code> , <code>prompt</code> , or <code>file</code> . The key must be present when the file system is created, mounted, or loaded by using the <code>zfs key -l</code> command. If encryption is enabled for a new file system, the default <code>keysource</code> is <code>passphrase,prompt</code> .
<code>keystatus</code>	String	<code>none</code>	Read-only property that identifies the file system's encryption key status. The availability of a file system's key is indicated by <code>available</code> or <code>unavailable</code> . For file systems that do not have encryption enabled, <code>none</code> is displayed.
<code>logbias</code>	String	<code>latency</code>	Controls how ZFS optimizes synchronous requests for this file system. If <code>logbias</code> is set to <code>latency</code> , ZFS uses the pool's separate log devices, if any, to handle the requests at low latency. If <code>logbias</code> is set to <code>throughput</code> , ZFS does not use the pool's separate log devices. Instead, ZFS optimizes synchronous operations for global pool throughput and efficient use of resources. The default value is <code>latency</code> .
<code>mlslabel</code>	String	<code>None</code>	See the <code>multilevel</code> property for a description of the behavior of the <code>mlslabel</code> property on multilevel file systems. The following <code>mlslabel</code> description applies to non-multilevel file systems. Provides a sensitivity label that determines if a file system can be mounted in a zone. If the labeled file system matches the labeled zone, the file system can be mounted and accessed from the labeled zone. The default value is <code>none</code> . This property can only be modified with the appropriate privilege.
<code>mounted</code>	Boolean	<code>N/A</code>	Read-only property that indicates whether a file system, clone, or snapshot is currently mounted. This property does not apply to volumes. The value can be either <code>yes</code> or <code>no</code> .
<code>mountpoint</code>	String	<code>N/A</code>	Controls the mount point used for this file system. When the <code>mountpoint</code> property is changed for a file system, the file system and any descendents that inherit the mount point are unmounted. If the new value is <code>legacy</code> , then they remain unmounted. Otherwise, they are automatically remounted in the new location if the property was previously <code>legacy</code> or <code>none</code> , or if they were mounted before the property was changed. In addition, any shared file systems are unshared and shared in the new location. For more information about using this property, see “Managing ZFS Mount Points” on page 124 .
<code>multilevel</code>	Boolean	<code>off</code>	Objects in a multilevel file system are individually labeled with an explicit sensitivity label attribute that is automatically generated. Objects can be relabeled in place by changing this label attribute,

Property Name	Type	Default Value	Description
			<p>by using the <code>setlabel</code> or <code>setflabel</code> interfaces. The default value is <code>off</code>.</p> <p>A root file system, an Oracle Solaris Zone file system, or a file system that contains packaged Solaris code should not be multilevel.</p> <p>There are differences in the <code>mlslabel</code> property on a multilevel file system. The <code>mlslabel</code> value defines the highest possible label for objects in the file system. An attempt to create a file at (or relabel a file to) a label higher than the <code>mlslabel</code> value is not allowed. Mount policy based on the <code>mlslabel</code> value does not apply to a multilevel file system.</p> <p>For a multilevel file system, the <code>mlslabel</code> property can be set explicitly when the file system is created. Otherwise, a default <code>mlslabel</code> property of <code>ADMIN_HIGH</code> is automatically created. After creating a multilevel file system, the <code>mlslabel</code> property can be changed, but it cannot be set to a lower label, set to <code>none</code>, nor can it be removed.</p>
<code>nbmand</code>	Boolean	<code>off</code>	Controls whether the file system should be mounted with <code>nbmand</code> (Non-blocking mandatory) locks. This property is for SMB clients only. Changes to this property only take effect when the file system is unmounted and remounted.
<code>normalization</code>	String	<code>None</code>	This property indicates whether a file system should perform a unicode normalization of file names whenever two file names are compared, and which normalization algorithm should be used. File names are always stored unmodified, names are normalized as part of any comparison process. If this property is set to a legal value other than <code>none</code> , and the <code>utf8only</code> property was left unspecified, the <code>utf8only</code> property is automatically set to <code>on</code> . The default value of the <code>normalization</code> property is <code>none</code> . This property cannot be changed after the file system is created.
<code>origin</code>	String	<code>N/A</code>	<p>Read-only property for cloned file systems or volumes that identifies the snapshot from which the clone was created. The origin cannot be destroyed (even with the <code>-r</code> or <code>-f</code> option) as long as a clone exists.</p> <p>Non-cloned file systems have an origin of <code>none</code>.</p>
<code>primarycache</code>	String	<code>all</code>	Controls what is cached in the primary cache (ARC). Possible values are <code>all</code> , <code>none</code> , and <code>metadata</code> . If set to <code>all</code> , both user data and metadata are cached. If set to <code>none</code> , neither user data nor metadata is cached. If set to <code>metadata</code> , only metadata is cached. When these properties are set on existing file systems, only new I/O is cache based on the values of these properties. Some database environments might benefit from not caching user data. You must determine if setting cache properties is appropriate for your environment.
<code>quota</code>	Number (or none)	<code>none</code>	Limits the amount of disk space a file system and its descendents can consume. This property enforces a hard limit on the amount of disk space used, including all space consumed by descendents, such as file systems and snapshots. Setting a quota on a descendent of a file system that already has a quota does not override the ancestor's

Property Name	Type	Default Value	Description
			<p>quota, but rather imposes an additional limit. Quotas cannot be set on volumes, as the <code>volsize</code> property acts as an implicit quota.</p> <p>For information about setting quotas, see “Setting Quotas on ZFS File Systems” on page 141.</p>
<code>readonly</code>	Boolean	<code>off</code>	<p>Controls whether a dataset can be modified. When set to on, no modifications can be made.</p> <p>The property abbreviation is <code>rdonly</code>.</p>
<code>recordsize</code>	Number	128K	<p>Specifies a suggested block size for files in a file system.</p> <p>The property abbreviation is <code>recsize</code>. For a detailed description, see “The <code>recordsize</code> Property” on page 114.</p>
<code>referenced</code>	Number	N/A	<p>Read-only property that identifies the amount of data accessible by a dataset, which might or might not be shared with other datasets in the pool.</p> <p>When a snapshot or clone is created, it initially references the same amount of disk space as the file system or snapshot it was created from, because its contents are identical.</p> <p>The property abbreviation is <code>refer</code>.</p>
<code>refquota</code>	Number (or none)	<code>none</code>	<p>Sets the amount of disk space that a dataset can consume. This property enforces a hard limit on the amount of space used. This hard limit does not include disk space used by descendants, such as snapshots and clones.</p>
<code>refreservation</code>	Number (or none)	<code>none</code>	<p>Sets the minimum amount of disk space that is guaranteed to a dataset, not including descendants, such as snapshots and clones. When the amount of disk space used is below this value, the dataset is treated as if it were taking up the amount of space specified by <code>refreservation</code>. The <code>refreservation</code> reservation is accounted for in the parent dataset's disk space used, and counts against the parent dataset's quotas and reservations.</p> <p>If <code>refreservation</code> is set, a snapshot is only allowed if enough free pool space is available outside of this reservation to accommodate the current number of <i>referenced</i> bytes in the dataset.</p> <p>The property abbreviation is <code>refreserv</code>.</p>
<code>rekeydate</code>	String	N/A	<p>Read-only property that indicates the date of the last data encryption key change from a <code>zfs key -K</code> or <code>zfs clone -K</code> operation on this file system. If no rekey operation has been performed, the value of this property is the same as the <code>creation</code> date.</p>
<code>reservation</code>	Number (or none)	<code>none</code>	<p>Sets the minimum amount of disk space guaranteed to a file system and its descendants. When the amount of disk space used is below this value, the file system is treated as if it were using the amount of space specified by its reservation. Reservations are accounted for in the parent file system's disk space used, and count against the parent file system's quotas and reservations.</p> <p>The property abbreviation is <code>reserv</code>.</p>

Property Name	Type	Default Value	Description
			For more information, see “Setting Reservations on ZFS File Systems” on page 145 .
rstchown	Boolean	on	Indicates whether the file system owner can grant file ownership changes. The default is to restrict chown operations. When rstchown is set to off, the user has the PRIV_FILE_CHOWN_SELF privilege for chown operations.
secondarycache	String	all	Controls what is cached in the secondary cache (L2ARC). Possible values are all, none, and metadata. If set to all, both user data and metadata are cached. If set to none, neither user data nor metadata is cached. If set to metadata, only metadata is cached.
setuid	Boolean	on	Controls whether the setuid bit is honored in a file system.
shadow	String	None	Identifies a ZFS file system as a <i>shadow</i> of the file system described by the <i>URI</i> . Data is migrated to a shadow file system with this property set from the file system identified by the URI. The file system to be migrated must be read-only for a complete migration.
share.nfs	String	off	Controls whether an NFS share of a ZFS file system is created and published and what options are used. You can also publish and unpublish an NFS share by using the zfs share and zfs unshare commands. Using the zfs share command to publish an NFS share requires that an NFS share property is also set. For information about setting NFS share properties, see “Sharing and Unsharing ZFS File Systems” on page 129 . For more information about sharing ZFS file systems, see “Sharing and Unsharing ZFS File Systems” on page 129 .
share.smb	String	off	Controls whether a SMB share of a ZFS file system is created and published and what options are used. You can also publish and unpublish an SMB share by using the zfs share and zfs unshare commands. Using the zfs share command to publish an SMB share requires that an SMB share property is also set. For information about setting SMB share properties, see “Sharing and Unsharing ZFS File Systems” on page 129 .
snapdir	String	hidden	Controls whether the .zfs directory is hidden or visible in the root of the file system. For more information about using snapshots, see “Overview of ZFS Snapshots” on page 159 .
sync	String	standard	Determines the synchronous behavior of a file system's transactions. Possible values are: <ul style="list-style-type: none"> ■ standard, the default value, which means synchronous file system transactions, such as fsync, O_DSYNC, O_SYNC, and so on, are written to the intent log. ■ always, ensures that every file system transaction is written and flushed to stable storage by a returning system call. This value has a significant performance penalty. ■ disabled, means that synchronous requests are disabled. File system transactions are only committed to stable storage on the next transaction group commit, which might be after many seconds. This value gives the best performance, with no risk of corrupting the pool.

Property Name	Type	Default Value	Description
			Caution - This disabled value is very dangerous because ZFS is ignoring the synchronous transaction demands of applications, such as databases or NFS operations. Setting this value on the currently active root or /var file system might result in unexpected behavior, application data loss, or increased vulnerability to replay attacks. You should only use this value if you fully understand all the associated risks.
type	String	N/A	Read-only property that identifies the dataset type as filesystem (file system or clone), volume, or snapshot.
used	Number	N/A	Read-only property that identifies the amount of disk space consumed by a dataset and all its descendants. For a detailed description, see “The used Property” on page 109 .
usedbychildren	Number	off	Read-only property that identifies the amount of disk space that is used by children of this dataset, which would be freed if all the dataset's children were destroyed. The property abbreviation is usedchild.
usedbydataset	Number	off	Read-only property that identifies the amount of disk space that is used by a dataset itself, which would be freed if the dataset was destroyed, after first destroying any snapshots and removing any refreservation reservations. The property abbreviation is usedds.
usedbyrefreservation	Number	off	Read-only property that identifies the amount of disk space that is used by a refreservation set on a dataset, which would be freed if the refreservation was removed. The property abbreviation is usedrefreserv.
usedbysnapshots	Number	off	Read-only property that identifies the amount of disk space that is consumed by snapshots of a dataset. In particular, it is the amount of disk space that would be freed if all of this dataset's snapshots were destroyed. Note that this value is not simply the sum of the snapshots' used properties, because space can be shared by multiple snapshots. The property abbreviation is usedsnap.
utf8only	Boolean	Off	This property indicates whether a file system should reject file names that include characters that are not present in the UTF-8 character code set. If this property is explicitly set to off, the normalization property must either not be explicitly set or be set to none. The default value for the utf8only property is off. This property cannot be changed after the file system is created.
version	Number	N/A	Identifies the on-disk version of a file system, which is independent of the pool version. This property can only be set to a later version that is available from the supported software release. For more information, see the <code>zfs upgrade</code> command.
volblocksize	Number	8 KB	For volumes, specifies the block size of the volume. The block size cannot be changed after the volume has been written, so set the block size at volume creation time. The default block size for volumes is 8 KB. Any power of 2 from 512 bytes to 128 KB is valid. The property abbreviation is volblock.
volsize	Number	N/A	For volumes, specifies the logical size of the volume.

Property Name	Type	Default Value	Description
			For a detailed description, see “The volsize Property” on page 114.
vscan	Boolean	off	Controls whether regular files should be scanned for viruses when a file is opened and closed. In addition to enabling this property, a virus scanning service must also be enabled for virus scanning to occur if you have third-party virus scanning software. The default value is off.
xattr	Boolean	on	Indicates whether extended attributes are enabled (on) or disabled (off) for this file system.
zoned	Boolean	N/A	Indicates whether a file system has been added to a non-global zone. If this property is set, then the mount point is not honored in the global zone, and ZFS cannot mount such a file system when requested. When a zone is first installed, this property is set for any added file systems. For more information about using ZFS with zones installed, see “Using ZFS on an Oracle Solaris System With Zones Installed” on page 224.

ZFS Read-Only Native Properties

Read-only native properties can be retrieved but not set. Read-only native properties are not inherited. Some native properties are specific to a particular type of dataset. In such cases, the dataset type is mentioned in the description in [Table 3, “ZFS Native Property Descriptions,” on page 102.](#)

The used Property

The used property is a read-only property that identifies the amount of disk space consumed by this dataset and all its descendents. This value is checked against the dataset's quota and reservation. The disk space used does not include the dataset's reservation, but does consider the reservation of any descendent datasets. The amount of disk space that a dataset consumes from its parent, as well as the amount of disk space that is freed if the dataset is recursively destroyed, is the greater of its space used and its reservation.

When snapshots are created, their disk space is initially shared between the snapshot and the file system, and possibly with previous snapshots. As the file system changes, disk space that was previously shared becomes unique to the snapshot and is counted in the snapshot's space used. The disk space that is used by a snapshot accounts for its unique data. Additionally, deleting snapshots can increase the amount of disk space unique to (and used by) other snapshots.

The amount of disk space used, available, and referenced does not include pending changes. Pending changes are generally accounted for within a few seconds. Committing a change to a

disk using the `fsync(3c)` or `O_SYNC` function does not necessarily guarantee that the disk space usage information will be updated immediately.

The `usedbychildren`, `usedbydataset`, `usedbyrefreservation`, and `usedbysnapshots` property information can be displayed with the `zfs list -o space` command. These properties identify the used property into disk space that is consumed by descendents. For more information, see [Table 3, “ZFS Native Property Descriptions,” on page 102](#).

Settable ZFS Native Properties

Settable native properties are properties whose values can be both retrieved and set. Settable native properties are set by using the `zfs set` command, as described in [“Setting ZFS Properties” on page 119](#) or by using the `zfs create` command as described in [“How to Create a ZFS File System” on page 98](#). With the exceptions of quotas and reservations, settable native properties are inherited. For more information about quotas and reservations, see [“Setting ZFS Quotas and Reservations” on page 140](#).

Some settable native properties are specific to a particular type of dataset. In such cases, the dataset type is mentioned in the description in [Table 3, “ZFS Native Property Descriptions,” on page 102](#). If not specifically mentioned, a property applies to all dataset types: file systems, volumes, clones, and snapshots.

The `canmount` Property

If the `canmount` property is set to `off`, the file system cannot be mounted by using the `zfs mount` or `zfs mount -a` commands. Setting this property to `off` is similar to setting the `mountpoint` property to `none`, except that the file system still has a normal `mountpoint` property that can be inherited. For example, you can set this property to `off`, establish inheritable properties for descendent file systems, but the parent file system itself is never mounted nor is it accessible to users. In this case, the parent file system is serving as a *container* so that you can set properties on the container, but the container itself is never accessible.

In the following example, `userpool` is created, and its `canmount` property is set to `off`. Mount points for descendent user file systems are set to one common mount point, `/export/home`. Properties that are set on the parent file system are inherited by descendent file systems, but the parent file system itself is never mounted.

```
# zpool create userpool mirror c0t5d0 c1t6d0
# zfs set canmount=off userpool
# zfs set mountpoint=/export/home userpool
# zfs set compression=on userpool
# zfs create userpool/user1
```

```
# zfs create userpool/user2
# zfs mount
userpool/user1          /export/home/user1
userpool/user2          /export/home/user2
```

Setting the `canmount` property to `noauto` means that the file system can only be mounted explicitly, not automatically.

The `casesensitivity` Property

This property indicates whether the file name matching algorithm used by the file system should be `casesensitive`, `caseinsensitive`, or allow a combination of both styles of matching (`mixed`).

When a case-insensitive matching request is made of a *mixed* sensitivity file system, the behavior is generally the same as would be expected of a purely case-insensitive file system. The difference is that a mixed sensitivity file system might contain directories with multiple names that are unique from a case-sensitive perspective, but not unique from the case-insensitive perspective.

For example, a directory might contain files `foo`, `Foo`, and `F00`. If a request is made to case-insensitively match any of the possible forms of `foo`, (for example `foo`, `F00`, `Fo0`, `f0o`, and so on) one of the three existing files is chosen as the match by the matching algorithm. Exactly which file the algorithm chooses as a match is not guaranteed, but what is guaranteed is that the same file is chosen as a match for any of the forms of `foo`. The file chosen as a case-insensitive match for `foo`, `F00`, `fo0`, `Foo`, and so on, is always the same, so long as the directory remains unchanged.

The `utf8only`, `normalization`, and `casesensitivity` properties also provide new permissions that can be assigned to non-privileged users by using ZFS delegated administration. For more information, see [“Delegating ZFS Permissions” on page 208](#).

The `copies` Property

As a reliability feature, ZFS file system metadata is automatically stored multiple times across different disks, if possible. This feature is known as *ditto blocks*.

In this release, you can also store multiple copies of user data is also stored per file system by using the `zfs set copies` command. For example:

```
# zfs set copies=2 users/home
# zfs get copies users/home
NAME          PROPERTY  VALUE  SOURCE
users/home    copies    2      local
```

Available values are 1, 2, or 3. The default value is 1. These copies are in addition to any pool-level redundancy, such as in a mirrored or RAID-Z configuration.

The benefits of storing multiple copies of ZFS user data are as follows:

- Improves data retention by enabling recovery from unrecoverable block read faults, such as media faults (commonly known as *bit rot*) for all ZFS configurations.
- Provides data protection, even when only a single disk is available.
- Enables you to select data protection policies on a per-file system basis, beyond the capabilities of the storage pool.

Note - Depending on the allocation of the ditto blocks in the storage pool, multiple copies might be placed on a single disk. A subsequent full disk failure might cause all ditto blocks to be unavailable.

You might consider using ditto blocks when you accidentally create a non-redundant pool and when you need to set data retention policies.

The dedup Property

The dedup property controls whether duplicate data is removed from a file system. If a file system has the dedup property enabled, duplicate data blocks are removed synchronously. The result is that only unique data is stored and common components are shared between files.

Do not enable the dedup property on file systems that reside on production systems until you review the following considerations:

1. Determine if your data would benefit from deduplication space savings. You can run the `zdb -S` command to simulate the potential space savings of enabling dedup on your pool. This command must be run on a quiet pool. If your data is not dedup-able, then there's not point in enabling dedup. For example:

```
# zdb -S tank
```

```
Simulated DDT histogram:
```

bucket	allocated				referenced			
refcnt	blocks	LSIZE	PSIZE	DSIZE	blocks	LSIZE	PSIZE	DSIZE
1	2.27M	239G	188G	194G	2.27M	239G	188G	194G
2	327K	34.3G	27.8G	28.1G	698K	73.3G	59.2G	59.9G
4	30.1K	2.91G	2.10G	2.11G	152K	14.9G	10.6G	10.6G
8	7.73K	691M	529M	529M	74.5K	6.25G	4.79G	4.80G
16	673	43.7M	25.8M	25.9M	13.1K	822M	492M	494M
32	197	12.3M	7.02M	7.03M	7.66K	480M	269M	270M

64	47	1.27M	626K	626K	3.86K	103M	51.2M	51.2M
128	22	908K	250K	251K	3.71K	150M	40.3M	40.3M
256	7	302K	48K	53.7K	2.27K	88.6M	17.3M	19.5M
512	4	131K	7.50K	7.75K	2.74K	102M	5.62M	5.79M
2K	1	2K	2K	2K	3.23K	6.47M	6.47M	6.47M
8K	1	128K	5K	5K	13.9K	1.74G	69.5M	69.5M
Total		2.63M	277G	218G	225G	3.22M	337G	263G

$\text{dedup} = 1.20, \text{compress} = 1.28, \text{copies} = 1.03, \text{dedup} * \text{compress} / \text{copies} = 1.50$

If the estimated dedup ratio is greater than 2, then you might see dedup space savings.

In the above example, the dedup ratio is less than 2, so enabling dedup is not recommended.

2. Make sure your system has enough memory to support dedup.

- Each in-core dedup table entry is approximately 320 bytes
- Multiply the number of allocated blocks times 320. For example:

$\text{in-core DDT size} = 2.63\text{M} \times 320 = 841.60\text{M}$

3. Dedup performance is best when the deduplication table fits into memory. If the dedup table has to be written to disk, then performance will decrease. For example, removing a large file system with dedup enabled will severely decrease system performance if the system doesn't meet the memory requirements described above.

When dedup is enabled, the dedup checksum algorithm overrides the checksum property. Setting the property value to `verify` is equivalent to specifying `sha256,verify`. If the property is set to `verify` and two blocks have the same signature, ZFS does a byte-by-byte comparison with the existing block to ensure that the contents are identical.

This property can be enabled per file system. For example:

```
# zfs set dedup=on tank/home
```

You can use the `zfs get` command to determine if the dedup property is set.

Although deduplication is set as a file system property, the scope is pool-wide. For example, you can identify the deduplication ratio. For example:

```
# zpool list tank
NAME    SIZE  ALLOC  FREE   CAP  DEDUP  HEALTH  ALROOT
rpool  136G  55.2G  80.8G  40%  2.30x  ONLINE  -
```

The DEDUP column indicates how much deduplication has occurred. If the dedup property is not enabled on any file system or if the dedup property was just enabled on the file system, the DEDUP ratio is 1.00x.

You can use the `zpool get` command to determine the value of the `dedupratio` property. For example:

```
# zpool get dedupratio export
```

NAME	PROPERTY	VALUE	SOURCE
rpool	dedupratio	3.00x	-

This pool property illustrates how much data deduplication this pool has achieved.

The encryption Property

You can use the encryption property to encrypt ZFS file systems. For more information, see [“Encrypting ZFS File Systems” on page 147](#).

The recordsize Property

The recordsize property specifies a suggested block size for files in the file system.

This property is designed solely for use with database workloads that access files in fixed-size records. ZFS automatically adjust block sizes according to internal algorithms optimized for typical access patterns. For databases that create very large files but access the files in small random chunks, these algorithms might be suboptimal. Specifying a recordsize value greater than or equal to the record size of the database can result in significant performance gains. Use of this property for general purpose file systems is strongly discouraged and might adversely affect performance. The size specified must be a power of 2 greater than or equal to 512 bytes and less than or equal to 1 MB. Changing the file system's recordsize value only affects files created afterward. Existing files are unaffected.

The property abbreviation is recsize.

The share.smb Property

This property enables sharing of ZFS file systems with the Oracle Solaris SMB service, and identifies options to be used.

When the property is changed from off to on, any shares that inherit the property are re-shared with their current options. When the property is set to off, the shares that inherit the property are unshared. For examples of using the share.smb property, see [“Sharing and Unsharing ZFS File Systems” on page 129](#).

The volsize Property

The volsize property specifies the logical size of the volume. By default, creating a volume establishes a reservation for the same amount. Any changes to volsize are reflected in an

equivalent change to the reservation. These checks are used to prevent unexpected behavior for users. A volume that contains less space than it claims is available can result in undefined behavior or data corruption, depending on how the volume is used. These effects can also occur when the volume size is changed while the volume is in use, particularly when you shrink the size. Use extreme care when adjusting the volume size.

For more information about using volumes, see [“ZFS Volumes” on page 221](#).

ZFS User Properties

In addition to the native properties, ZFS supports arbitrary user properties. User properties have no effect on ZFS behavior, but you can use them to annotate datasets with information that is meaningful in your environment.

User property names must conform to the following conventions:

- They must contain a colon (':') character to distinguish them from native properties.
- They must contain lowercase letters, numbers, or the following punctuation characters: ':', '+', ',', '_', and '.'.
- The maximum length of a user property name is 256 characters.

The expected convention is that the property name is divided into the following two components but this namespace is not enforced by ZFS:

module:property

When making programmatic use of user properties, use a reversed DNS domain name for the *module* component of property names to reduce the chance that two independently developed packages will use the same property name for different purposes. Property names that begin with `com.oracle.` are reserved for use by Oracle Corporation.

The values of user properties must conform to the following conventions:

- They must consist of arbitrary strings that are always inherited and are never validated.
- The maximum length of the user property value is 1024 characters.

For example:

```
# zfs set dept:users=finance userpool/user1
# zfs set dept:users=general userpool/user2
# zfs set dept:users=itops userpool/user3
```

All of the commands that operate on properties, such as `zfs list`, `zfs get`, `zfs set`, and so on, can be used to manipulate both native properties and user properties.

For example:

```
zfs get -r dept:users userpool
NAME          PROPERTY  VALUE      SOURCE
userpool      dept:users all         local
userpool/user1 dept:users finance    local
userpool/user2 dept:users general   local
userpool/user3 dept:users itops      local
```

To clear a user property, use the `zfs inherit` command. For example:

```
# zfs inherit -r dept:users userpool
```

If the property is not defined in any parent dataset, it is removed entirely.

Querying ZFS File System Information

The `zfs list` command provides an extensible mechanism for viewing and querying dataset information. Both basic and complex queries are explained in this section.

Listing Basic ZFS Information

You can list basic dataset information by using the `zfs list` command with no options. This command displays the names of all datasets on the system and the values of their used, available, referenced, and mountpoint properties. For more information about these properties, see [“Introducing ZFS Properties” on page 101](#).

For example:

```
# zfs list
users          2.00G 64.9G 32K /users
users/home     2.00G 64.9G 35K /users/home
users/home/cindy 548K 64.9G 548K /users/home/cindy
users/home/mark 1.00G 64.9G 1.00G /users/home/mark
users/home/neil 1.00G 64.9G 1.00G /users/home/neil
```

You can also use this command to display specific datasets by providing the dataset name on the command line. Additionally, use the `-r` option to recursively display all descendents of that dataset. For example:

```
# zfs list -t all -r users/home/mark
NAME          USED  AVAIL  REFER  MOUNTPOINT
users/home/mark 1.00G 64.9G 1.00G /users/home/mark
users/home/mark@yesterday 0 - 1.00G -
users/home/mark@today 0 - 1.00G -
```

You can use the `zfs list` command with the mount point of a file system. For example:

```
# zfs list /user/home/mark
NAME                USED  AVAIL  REFER  MOUNTPOINT
users/home/mark     1.00G  64.9G  1.00G  /users/home/mark
```

The following example shows how to display basic information about tank/home/gina and all of its descendent file systems:

```
# zfs list -r users/home/gina
NAME                USED  AVAIL  REFER  MOUNTPOINT
users/home/gina     2.00G  62.9G   32K   /users/home/gina
users/home/gina/projects 2.00G  62.9G   33K   /users/home/gina/projects
users/home/gina/projects/fs1 1.00G  62.9G  1.00G   /users/home/gina/projects/fs1
users/home/gina/projects/fs2 1.00G  62.9G  1.00G   /users/home/gina/projects/fs2
```

For additional information about the `zfs list` command, see the [zfs\(1M\)](#) man page.

Creating Complex ZFS Queries

The `zfs list` output can be customized by using the `-o`, `-t`, and `-H` options.

You can customize property value output by using the `-o` option and a comma-separated list of desired properties. You can supply any dataset property as a valid argument. For a list of all supported dataset properties, see [“Introducing ZFS Properties” on page 101](#). In addition to the properties defined, the `-o` option list can also contain the literal name to indicate that the output should include the name of the dataset.

The following example uses `zfs list` to display the dataset name, along with the `share.nfs` and `mountpoint` property values.

```
# zfs list -r -o name,share.nfs,mountpoint users/home
NAME                NFS      MOUNTPOINT
users/home          on       /users/home
users/home/cindy    on       /users/home/cindy
users/home/gina     on       /users/home/gina
users/home/gina/projects on      /users/home/gina/projects
users/home/gina/projects/fs1 on      /users/home/gina/projects/fs1
users/home/gina/projects/fs2 on      /users/home/gina/projects/fs2
users/home/mark     on       /users/home/mark
users/home/neil     on       /users/home/neil
```

You can use the `-t` option to specify the types of datasets to display. The valid types are described in the following table.

TABLE 4 Types of ZFS Objects

Type	Description
filesystem	File systems and clones
volume	Volumes

Type	Description
share	File system share
snapshot	Snapshots

The `-t` options takes a comma-separated list of the types of datasets to be displayed. The following example uses the `-t` and `-o` options simultaneously to show the name and used property for all file systems:

```
# zfs list -r -t filesystem -o name,used users/home
NAME                                USED
users/home                         4.00G
users/home/cindy                    548K
users/home/gina                     2.00G
users/home/gina/projects            2.00G
users/home/gina/projects/fs1        1.00G
users/home/gina/projects/fs2        1.00G
users/home/mark                     1.00G
users/home/neil                     1.00G
```

You can use the `-H` option to omit the `zfs list` header from the generated output. With the `-H` option, all white space is replaced by the Tab character. This option can be useful when you need parseable output, for example, when scripting. The following example shows the output generated from using the `zfs list` command with the `-H` option:

```
# zfs list -r -H -o name users/home
users/home
users/home/cindy
users/home/gina
users/home/gina/projects
users/home/gina/projects/fs1
users/home/gina/projects/fs2
users/home/mark
users/home/neil
```

Listing Incomplete ZFS Datasets

An incomplete dataset is created when a dataset transfer started by running `zfs receive` is interrupted. The `zfs list -I` command can be used to show which datasets are incomplete. The state for each dataset can be either receiving or resumable. Arguments with the `-I` option are all, resumable or receiving.

```
# zfs list -I all
NAME                USED  AVAIL  REFER  TYPE  STATE
users/home/dst      189M  910G   189M   volume  resumable
```

You can use the following command to show just the names of the resumable datasets,

```
# # zfs list -HI resumable
users/home/dst
```

Creating Parsable Output with `zfs list`

The `zfs list -o` command can be used with the `-p` option to create exact machine-parsable numeric output. For example:

```
# zfs list -o guid users/home
GUID
3.30E
# zfs list -po guid users/home
GUID
3807001345661527925
```

Managing ZFS Properties

Dataset properties are managed through the `zfs` command's `set`, `inherit`, and `get` subcommands.

- [“Setting ZFS Properties” on page 119](#)
- [“Inheriting ZFS Properties” on page 120](#)
- [“Querying ZFS Properties” on page 121](#)

Setting ZFS Properties

You can use the `zfs set` command to modify any settable dataset property. Or, you can use the `zfs create` command to set properties when a dataset is created. For a list of settable dataset properties, see [“Settable ZFS Native Properties” on page 110](#).

The `zfs set` command takes a property/value sequence in the format of *property=value* followed by a dataset name. Only one property can be set or modified during each `zfs set` invocation.

The following example sets the `atime` property to `off` for `tank/home`.

```
# zfs set atime=off tank/home
```

In addition, any file system property can be set when a file system is created. For example:

```
# zfs create -o atime=off tank/home
```

You can specify numeric property values by using the following easy-to-understand suffixes (in increasing sizes): BKMGTPEZ. Any of these suffixes can be followed by an optional b, indicating bytes, with the exception of the B suffix, which already indicates bytes. The following four invocations of `zfs set` are equivalent numeric expressions that set the quota property to be set to the value of 20 GB on the `users/home/mark` file system:

```
# zfs set quota=20G users/home/mark
# zfs set quota=20g users/home/mark
# zfs set quota=20GB users/home/mark
# zfs set quota=20gb users/home/mark
```

If you attempt to set a property on a file system that is 100% full, you will see a message similar to the following:

```
# zfs set quota=20gb users/home/mark
cannot set property for '/users/home/mark': out of space
```

The values of non-numeric properties are case-sensitive and must be in lowercase letters, with the exception of mountpoint. The values of this property can have mixed upper and lower case letters.

For more information about the `zfs set` command, see the [zfs\(1M\)](#) man page.

Inheriting ZFS Properties

All settable properties, with the exception of quotas and reservations, inherit their value from the parent file system, unless a quota or reservation is explicitly set on the descendent file system. If no ancestor has an explicit value set for an inherited property, the default value for the property is used. You can use the `zfs inherit` command to clear a property value, thus causing the value to be inherited from the parent file system.

The following example uses the `zfs set` command to turn on compression for the `tank/home/jeff` file system. Then, `zfs inherit` is used to clear the compression property, thus causing the property to inherit the default value of `off`. Because neither `home` nor `tank` has the compression property set locally, the default value is used. If both had compression enabled, the value set in the most immediate ancestor would be used (`home` in this example).

```
# zfs set compression=on tank/home/jeff
# zfs get -r compression tank/home
```

NAME	PROPERTY	VALUE	SOURCE
tank/home	compression	off	default
tank/home/eric	compression	off	default
tank/home/eric@today	compression	-	-
tank/home/jeff	compression	on	local

```
# zfs inherit compression tank/home/jeff
# zfs get -r compression tank/home
```


NAME	PROPERTY	VALUE	SOURCE
tank/home	compression	off	default
tank/home/eric	compression	off	default
tank/home/eric@today	compression	-	-
tank/home/jeff	compression	off	default

The `inherit` subcommand is applied recursively when the `-r` option is specified. In the following example, the command causes the value for the `compression` property to be inherited by `tank/home` and any descendents it might have:

```
# zfs inherit -r compression tank/home
```

Note - Be aware that the use of the `-r` option clears the current property setting for all descendent file systems.

For more information about the `zfs inherit` command, see the [zfs\(1M\)](#) man page.

Querying ZFS Properties

The simplest way to query property values is by using the `zfs list` command. For more information, see [“Listing Basic ZFS Information” on page 116](#). However, for complicated queries and for scripting, use the `zfs get` command to provide more detailed information in a customized format.

You can use the `zfs get` command to retrieve any dataset property. The following example shows how to retrieve a single property value on a dataset:

```
# zfs get checksum tank/ws
```

NAME	PROPERTY	VALUE	SOURCE
tank/ws	checksum	on	default

The fourth column, `SOURCE`, indicates the origin of this property value. The following table defines the possible source values.

TABLE 5 Possible `SOURCE` Values (`zfs get` Command)

Source Value	Description
default	This property value was never explicitly set for this dataset or any of its ancestors. The default value for this property is being used.
inherited from <i>dataset-name</i>	This property value is inherited from the parent dataset specified in <i>dataset-name</i> .
local	This property value was explicitly set for this dataset by using <code>zfs set</code> .
temporary	This property value was set by using the <code>zfs mount -o</code> option and is only valid for the duration of the mount. For more information about temporary mount point properties, see “Using Temporary Mount Properties” on page 128 .

Source Value	Description
- (none)	This property is read-only. Its value is generated by ZFS.

You can use the special keyword `all` to retrieve all dataset property values. The following examples use the `all` keyword:

```
# zfs get all tank/home
```

NAME	PROPERTY	VALUE	SOURCE
tank/home	aclinherit	restricted	default
tank/home	aclmode	discard	default
tank/home	atime	on	default
tank/home	available	274G	-
tank/home	canmount	on	default
tank/home	casesensitivity	mixed	-
tank/home	checksum	on	default
tank/home	compression	off	default
tank/home	compressratio	1.00x	-
tank/home	copies	1	default
tank/home	creation	Tue Jul 30 10:08 2013	-
tank/home	dedup	off	default
tank/home	defaultgroupquota	none	-
tank/home	defaultuserquota	none	-
tank/home	devices	on	default
tank/home	encryption	off	-
tank/home	exec	on	default
tank/home	keychangedate	-	default
tank/home	keysource	none	default
tank/home	keystatus	none	-
tank/home	logbias	latency	default
tank/home	mlslabel	none	-
tank/home	mounted	yes	-
tank/home	mountpoint	/tank/home	default
tank/home	multilevel	off	-
tank/home	nbmand	off	default
tank/home	normalization	none	-
tank/home	primarycache	all	default
tank/home	quota	none	default
tank/home	readonly	off	default
tank/home	recordsize	128K	default
tank/home	referenced	31K	-
tank/home	refquota	none	default
tank/home	refreservation	none	default
tank/home	rekeydate	-	default
tank/home	reservation	none	default
tank/home	rstchown	on	default
tank/home	secondarycache	all	default
tank/home	setuid	on	default
tank/home	shadow	none	-
tank/home	share.*	...	default
tank/home	snapdir	hidden	default

tank/home	sync	standard	default
tank/home	type	filesystem	-
tank/home	used	31K	-
tank/home	usedbychildren	0	-
tank/home	usedbydataset	31K	-
tank/home	usedbyreservation	0	-
tank/home	usedbysnapshots	0	-
tank/home	utf8only	off	-
tank/home	version	6	-
tank/home	vscan	off	default
tank/home	xattr	on	default
tank/home	zoned	off	default

The `-s` option to `zfs get` enables you to specify, by source type, the properties to display. This option takes a comma-separated list indicating the desired source types. Only properties with the specified source type are displayed. The valid source types are `local`, `default`, `inherited`, `temporary`, and `none`. The following example shows all properties that have been locally set on `tank/ws`.

```
# zfs get -s local all tank/ws
```

NAME	PROPERTY	VALUE	SOURCE
tank/ws	compression	on	local

Any of the above options can be combined with the `-r` option to recursively display the specified properties on all children of the specified file system. In the following example, all temporary properties on all file systems within `tank/home` are recursively displayed:

```
# zfs get -r -s temporary all tank/home
```

NAME	PROPERTY	VALUE	SOURCE
tank/home	atime	off	temporary
tank/home/jeff	atime	off	temporary
tank/home/mark	quota	20G	temporary

You can query property values by using the `zfs get` command without specifying a target file system, which means the command operates on all pools or file systems. For example:

```
# zfs get -s local all
```

tank/home	atime	off	local
tank/home/jeff	atime	off	local
tank/home/mark	quota	20G	local

For more information about the `zfs get` command, see the [zfs\(1M\)](#) man page.

Querying ZFS Properties for Scripting

The `zfs get` command supports the `-H` and `-o` options, which are designed for scripting. You can use the `-H` option to omit header information and to replace white space with the Tab

character. Uniform white space allows for easily parsable data. You can use the `-o` option to customize the output in the following ways:

- The literal name can be used with a comma-separated list of properties as defined in the [“Introducing ZFS Properties” on page 101](#) section.
- A comma-separated list of literal fields, name, value, property, and source, to be output followed by a space and an argument, which is a comma-separated list of properties.

The following example shows how to retrieve a single value by using the `-H` and `-o` options of `zfs get`:

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

The `-p` option reports numeric values as their exact values. For example, 1 MB would be reported as 1000000. This option can be used as follows:

```
# zfs get -H -o value -p used tank/home
182983742
```

You can use the `-r` option, along with any of the preceding options, to recursively retrieve the requested values for all descendents. The following example uses the `-H`, `-o`, and `-r` options to retrieve the file system name and the value of the used property for `export/home` and its descendents, while omitting the header output:

```
# zfs get -H -o name,value -r used export/home
```

Mounting ZFS File Systems

This section describes how ZFS mounts file systems.

- [“Managing ZFS Mount Points” on page 124](#)
- [“Mounting ZFS File Systems” on page 126](#)
- [“Using Temporary Mount Properties” on page 128](#)
- [“Unmounting ZFS File Systems” on page 128](#)

Managing ZFS Mount Points

By default, a ZFS file system is automatically mounted when it is created. You can determine specific mount-point behavior for a file system as described in this section.

You can also set the default mount point for a pool's file system at creation time by using `zpool create's -m` option. For more information about creating pools, see [“Creating ZFS Storage Pools” on page 25](#).

All ZFS file systems are mounted by ZFS at boot time by using the Service Management Facility's (SMF) `svc://system/filesystem/local` service. File systems are mounted under `/path`, where *path* is the name of the file system.

You can override the default mount point by using the `zfs set` command to set the `mountpoint` property to a specific path. ZFS automatically creates the specified mount point, if needed, and automatically mounts the associated file system.

ZFS file systems are automatically mounted at boot time without requiring you to edit the `/etc/vfstab` file.

The `mountpoint` property is inherited. For example, if `pool/home` has the `mountpoint` property set to `/export/stuff`, then `pool/home/user` inherits `/export/stuff/user` for its `mountpoint` property value.

To prevent a file system from being mounted, set the `mountpoint` property to `none`. In addition, the `canmount` property can be used to control whether a file system can be mounted. For more information about the `canmount` property, see [“The `canmount` Property” on page 110](#).

File systems can also be explicitly managed through legacy mount interfaces by using `zfs set` to set the `mountpoint` property to `legacy`. Doing so prevents ZFS from automatically mounting and managing a file system. Legacy tools including the `mount` and `umount` commands, and the `/etc/vfstab` file must be used instead. For more information about legacy mounts, see [“Legacy Mount Points” on page 126](#).

Automatic Mount Points

- When you change the `mountpoint` property from `legacy` or `none` to a specific path, ZFS automatically mounts the file system.
- If ZFS is managing a file system but it is currently unmounted, and the `mountpoint` property is changed, the file system remains unmounted.

Any file system whose `mountpoint` property is not `legacy` is managed by ZFS. In the following example, a file system is created whose mount point is automatically managed by ZFS:

```
# zfs create pool/filesystem
# zfs get mountpoint pool/filesystem
NAME                PROPERTY    VALUE                SOURCE
pool/filesystem      mountpoint  /pool/filesystem     default
# zfs get mounted pool/filesystem
NAME                PROPERTY    VALUE                SOURCE
pool/filesystem      mounted     yes                  -
```

You can also explicitly set the `mountpoint` property as shown in the following example:

```
# zfs set mountpoint=/mnt pool/filesystem
```

```
# zfs get mountpoint pool/filesystem
NAME          PROPERTY      VALUE          SOURCE
pool/filesystem mountpoint    /mnt          local
# zfs get mounted pool/filesystem
NAME          PROPERTY      VALUE          SOURCE
pool/filesystem mounted      yes           -
```

When the mountpoint property is changed, the file system is automatically unmounted from the old mount point and remounted to the new mount point. Mount-point directories are created as needed. If ZFS is unable to unmount a file system due to it being active, an error is reported, and a forced manual unmount is necessary.

Legacy Mount Points

You can manage ZFS file systems with legacy tools by setting the mountpoint property to legacy. Legacy file systems must be managed through the mount and umount commands and the /etc/vfstab file. ZFS does not automatically mount legacy file systems at boot time, and the ZFS mount and umount commands do not operate on file systems of this type. The following examples show how to set up and manage a ZFS file system in legacy mode:

```
# zfs set mountpoint=legacy tank/home/eric
# mount -F zfs tank/home/eschrock /mnt
```

To automatically mount a legacy file system at boot time, you must add an entry to the /etc/vfstab file. The following example shows what the entry in the /etc/vfstab file might look like:

```
#device      device      mount      FS      fsck      mount      mount
#to mount    to fsck     point      type     pass     at boot   options
#
tank/home/eric - /mnt      zfs - yes -
```

The device to fsck and fsck pass entries are set to - because the fsck command is not applicable to ZFS file systems.

Mounting ZFS File Systems

ZFS automatically mounts file systems when file systems are created or when the system boots. Use of the zfs mount command is necessary only when you need to change mount options, or explicitly mount or unmount file systems.

The zfs mount command with no arguments shows all currently mounted file systems that are managed by ZFS. Legacy managed mount points are not displayed. For example:

```
# zfs mount | grep tank/home
```

```
zfs mount | grep tank/home
tank/home                /tank/home
tank/home/jeff           /tank/home/jeff
```

You can use the `-a` option to mount all ZFS managed file systems. Legacy managed file systems are not mounted. For example:

```
# zfs mount -a
```

By default, ZFS does not allow mounting on top of a nonempty directory. For example:

```
# zfs mount tank/home/lori
cannot mount 'tank/home/lori': filesystem already mounted
```

Legacy mount points must be managed through legacy tools. An attempt to use ZFS tools results in an error. For example:

```
# zfs mount tank/home/bill
cannot mount 'tank/home/bill': legacy mountpoint
use mount(8) to mount this filesystem
# mount -F zfs tank/home/billm
```

When a file system is mounted, it uses a set of mount options based on the property values associated with the file system. The correlation between properties and mount options is as follows:

TABLE 6 ZFS Mount-Related Properties and Mount Options

Property	Mount Option
atime	atime/noatime
devices	devices/nodevices
exec	exec/noexec
nbmand	nbmand/nonbmand
readonly	ro/rw
setuid	setuid/nosetuid
xattr	xattr/noxattr

The mount option `nosuid` is an alias for `nodevices,nosetuid`.

You can use the NFSv4 mirror mount features to help you better manage NFS-mounted ZFS home directories.

When file systems are created on the NFS server, the NFS client can automatically discover these newly created file systems within their existing mount of a parent file system.

For example, if the server `neo` already shares the `tank` file system and client `zee` has it mounted, `/tank/baz` is automatically visible on the client after it is created on the server.

```
zee# mount neo:/tank /mnt
zee# ls /mnt
baa    bar

neo# zfs create tank/baz

zee% ls /mnt
baa    bar    baz
zee% ls /mnt/baz
file1  file2
```

Using Temporary Mount Properties

If any of the mount options described in the preceding section are set explicitly by using the `-o` option with the `zfs mount` command, the associated property value is temporarily overridden. These property values are reported as `temporary` by the `zfs get` command and revert back to their original values when the file system is unmounted. If a property value is changed while the file system is mounted, the change takes effect immediately, overriding any temporary setting.

In the following example, the read-only mount option is temporarily set on the `tank/home/neil` file system. The file system is assumed to be unmounted.

```
# zfs mount -o ro users/home/neil
```

To temporarily change a property value on a file system that is currently mounted, you must use the special `remount` option. In the following example, the `atime` property is temporarily changed to `off` for a file system that is currently mounted:

```
# zfs mount -o remount,noatime users/home/neil
NAME                PROPERTY  VALUE  SOURCE
users/home/neil     atime    off    temporary
# zfs get atime users/home/perrin
```

For more information about the `zfs mount` command, see the [zfs\(1M\)](#) man page.

Unmounting ZFS File Systems

You can unmount ZFS file systems by using the `zfs unmount` subcommand. The `unmount` command can take either the mount point or the file system name as an argument.

In the following example, a file system is unmounted by its file system name:

```
# zfs unmount users/home/mark
```

In the following example, the file system is unmounted by its mount point:


```
# zfs unmount /users/home/mark
```

The unmount command fails if the file system is busy. To forcibly unmount a file system, you can use the `-f` option. Be cautious when forcibly unmounting a file system if its contents are actively being used. Unpredictable application behavior can result.

```
# zfs unmount tank/home/eric
cannot unmount '/tank/home/eric': Device busy
# zfs unmount -f tank/home/eric
```

To provide for backward compatibility, the legacy `umount` command can be used to unmount ZFS file systems. For example:

```
# umount /tank/home/bob
```

For more information about the `zfs umount` command, see the [zfs\(1M\)](#) man page.

Sharing and Unsharing ZFS File Systems

The Oracle Solaris11.1 release simplifies ZFS share administration by leveraging ZFS property inheritance. The new share syntax is enabled on pools running pool version 34.

The following are the file system packages for NFS and SMB:

- NFS client and server packages
 - `service/file-system/nfs` (server)
 - `service/file-system/nfs` (client)

For additional NFS configuration information, see [Managing Network File Systems in Oracle Solaris 11.3](#).

- SMB client and server packages
 - `service/file-system/smb` (server)
 - `service/file-system/smb` (client)

For additional SMB configuration information including SMB password management, see [“Managing SMB Mounts in Your Local Environment” in Managing SMB File Sharing and Windows Interoperability in Oracle Solaris 11.3](#).

Multiple shares can be defined per file system. A share name uniquely identifies each share. You can define the properties that are used to share a particular path in a file system. By default, all file systems are unshared. In general, the NFS server services are not started until a share is created. If you create a valid share, the NFS services are started automatically. If a ZFS file system's mountpoint property is set to legacy, the file system can only be shared by using the legacy share command.

- The `share.nfs` property replaces the `sharenfs` property in previous releases to define and publish an NFS share.
- The `share.smb` property replaces the `sharesmb` property in previous releases to define and publish an SMB share.
- Both the `sharenfs` property and `sharesmb` property are aliases to the `share.nfs` property and the `sharenfs` property.
- The `/etc/dfs/dfstab` file is no longer used to share file systems at boot time. Setting these properties share file systems automatically. SMF manages ZFS or UFS share information so that file systems are shared automatically when the system is rebooted. This feature means that all file systems whose `sharenfs` or `sharesmb` property are not set to off are shared at boot time.
- The `sharemgr` interface is no longer available. The legacy `share` command is still available to create a legacy share. See the examples below.
- The `share -a` command is like the previous `share -ap` command so that sharing a file system is persistent. The `share -p` option is no longer available.

For example, if you want to share the `tank/home` file system, use syntax similar to the following:

```
# zfs set share.nfs=on tank/home
```

In preceding example, where the `share.nfs` property is set on the `tank/home` file system, the `share.nfs` property value is inherited to any descendent file systems. For example:

```
# zfs create tank/home/userA
# zfs create tank/home/userB
```

You can also specify additional property values or modify existing property values on existing file system shares. For example:

```
# zfs set share.nfs.nosuid=on tank/home/userA
# zfs set share.nfs=on tank/home/userA
```

Legacy ZFS Sharing Syntax

Oracle Solaris 11 syntax is still supported so that you can share file systems in two steps. This syntax is supported in all pool versions.

- First, use the `zfs set share` command to create an NFS or SMB share of ZFS file system.

```
# zfs create rpool/fs1
# zfs set share=name=fs1,path=/rpool/fs1,prot=nfs rpool/fs1
name=fs1,path=/rpool/fs1,prot=nfs
```
- Then, set the `sharenfs` or `sharesmb` property to on to publish the share. For example:

```
# zfs set sharenfs=on rpool/fs1
# grep fs1 /etc/dfs/sharetab
/rpool/fs1      fs1      nfs      sec=sys,rw
```

File system shares can be displayed with the legacy `zfs get share` command.

```
# zfs get share rpool/fs1
NAME      PROPERTY  VALUE  SOURCE
rpool/fs1 share     name=fs1,path=/rpool/fs1,prot=nfs  local
```

In addition, the `share` command to share a file system, similar to the syntax in the Oracle Solaris 10 release, is still supported to share any directory within a file system. For example, to share a ZFS file system:

```
# share -F nfs /tank/zfsfs
# grep zfsfs /etc/dfs/sharetab
/tank/zfsfs    tank_zfsfs    nfs      sec=sys,rw
```

The above syntax is identical to sharing a UFS file system:

```
# share -F nfs /ufsfs
# grep ufsfs /etc/dfs/sharetab
/ufsfs         -             nfs      rw
/tank/zfsfs    tank_zfsfs    nfs      rw
```

New ZFS Sharing Syntax

The `zfs set` command is used to share and publish a ZFS file system over the NFS or SMB protocols. Or, you can set the `share.nfs` or `share.smb` property when the file system is created.

For example, the `tank/sales` file system is created and shared. The default share permissions are read-write for everyone. The descendent `tank/sales/logs` file system is also shared automatically because the `share.nfs` property is inherited to descendent file systems and the `tank/sales/log` file system is set to read-only access.

```
# zfs create -o share.nfs=on tank/sales
# zfs create -o share.nfs.ro=* tank/sales/logs
# zfs get -r share.nfs tank/sales
NAME      PROPERTY  VALUE  SOURCE
tank/sales share.nfs  on     local
tank/sales% share.nfs  on     inherited from tank/sales
tank/sales/log share.nfs  on     inherited from tank/sales
tank/sales/log% share.nfs  on     inherited from tank/sales
```

You can provide root access to a specific system for a shared file system as follows:

```
# zfs set share.nfs=on tank/home/data
```

```
# zfs set share.nfs.sec.default.root=neo.daleks.com tank/home/data
```

ZFS Sharing with Per-Property Inheritance

In pools that have been upgraded to the latest pool version 34, new sharing syntax is available that makes use of ZFS property inheritance to ease share maintenance. Each sharing characteristic becomes a separate share property. The share properties are identified by names that start with the `share.` prefix. Examples of share properties include `share.desc`, `share.nfs.nosuid`, and `share.smb.guestok`.

The `share.nfs` property controls whether NFS sharing is enabled. The `share.smb` property controls whether SMB sharing is enabled. The legacy `sharenfs` and `sharesmb` property names can still be used, because in new pools, `sharenfs` is an alias for `share.nfs` and `sharesmb` is an alias for `share.smb`. If you want to share the `tank/home` file system, use syntax similar to the following:

```
# zfs set share.nfs=on tank/home
```

In this example, the `share.nfs` property value is inherited to any descendent file systems. For example:

```
# zfs create tank/home/userA
# zfs create tank/home/userB
# grep tank/home /etc/dfs/sharetab
/tank/home      tank_home      nfs      sec=sys,rw
/tank/home/userA  tank_home_userA  nfs      sec=sys,rw
/tank/home/userB  tank_home_userB  nfs      sec=sys,rw
```

ZFS Sharing Inheritance in Older Pools

In older pools, only the `sharenfs` and `sharesmb` properties are inherited by descendent file systems. Other sharing characteristics are stored in the `.zfs/shares` file for each share and are not inherited.

A special rule is that whenever a new file system is created that inherits `sharenfs` or `sharesmb` from its parent, a default share is created for that file system from the `sharenfs` or `sharesmb` value. Note that when `sharenfs` is simply on, the default share that is created in a descendent file system has only the default NFS characteristics. For example:

```
# zpool get version tank
NAME  PROPERTY  VALUE  SOURCE
tank  version   33     default
# zfs create -o sharenfs=on tank/home
# zfs create tank/home/userA
# grep tank/home /etc/dfs/sharetab
/tank/home      tank_home      nfs      sec=sys,rw
/tank/home/userA  tank_home_userA  nfs      sec=sys,r
```

ZFS Named Shares

You can also create a *named* share, which provides more flexibility in setting permissions and properties in an SMB environment. For example:

```
# zfs share -o share.smb=on tank/workspace%myshare
```

In the preceding example, the `zfs share` command creates an SMB share called `myshare` of the `tank/workspace` file system. You can access the SMB share and display or set specific permissions or ACLs through the `.zfs/shares` directory of the file system. Each SMB share is represented by a separate `.zfs/shares` file. For example:

```
# ls -lv /tank/workspace/.zfs/shares
-rwxrwxrwx+ 1 root    root          0 May 15 10:31 myshare
0:everyone@:read_data/write_data/append_data/read_xattr/write_xattr
/execute/delete_child/read_attributes/write_attributes/delete
/read_acl/write_acl/write_owner/synchronize:allow
```

Named shares inherit sharing properties from the parent file system. If you add the `share.smb.guestok` property to the parent file system in the previous example, this property is inherited to the named share. For example:

```
# zfs get -r share.smb.guestok tank/workspace
NAME                PROPERTY            VALUE    SOURCE
tank/workspace       share.smb.guestok   on       inherited from tank
tank/workspace%myshare share.smb.guestok   on       inherited from tank
```

Named shares can be helpful in the NFS environment when defining shares for a subdirectory of the file system. For example:

```
# zfs create -o share.nfs=on -o share.nfs.anon=99 -o share.auto=off tank/home
# mkdir /tank/home/userA
# mkdir /tank/home/userB
# zfs share -o share.path=/tank/home/userA tank/home%userA
# zfs share -o share.path=/tank/home/userB tank/home%userB
# grep tank/home /etc/dfs/sharetab
/tank/home/userA      userA   nfs     anon=99,sec=sys,rw
/tank/home/userB      userB   nfs     anon=99,sec=sys,rw
```

The above example also illustrates that setting the `share.auto` to off for a file system turns off the auto share for that file system while leaving all other property inheritance intact. Unlike most other sharing properties, the `share.auto` property is not inheritable.

Named shares are also used when creating a public NFS share. A public share can only be created on a named NFS share. For example:

```
# zfs create -o mountpoint=/pub tank/public
# zfs share -o share.nfs=on -o share.nfs.public=on tank/public%pubshare
# grep pub /etc/dfs/sharetab
/pub      pubshare      nfs     public,sec=sys,rw
```

See the [share_nfs\(1M\)](#) and [share_smb\(1M\)](#) man pages for a detailed description of NFS and SMB share properties.

ZFS Automatic Shares

When an automatic (auto) share is created, a unique resource name is constructed from the file system name. The constructed name is a copy of the file system name except that the characters in the file system name that would be illegal in the resource name, are replaced with underscore (_) characters. For example, the resource name of `data/home/john` is `data_home_john`.

Setting a `share.autoname` property name allows you to replace the file system name with a specific name when creating the auto share. The specific name is also used to replace the prefix file system name in the case of inheritance. For example:

```
# zfs create -o share.smb=on -o share.autoname=john data/home/john
# zfs create data/home/john/backups
# grep john /etc/dfs/sharetab
/data/home/john john      smb
/data/home/john/backups john_backups  smb
```

If a legacy share command or the `zfs set share` command is used on a file system that has not yet been shared, its `share.auto` value is automatically set to `off`. The legacy commands always create named shares. This special rule prevents the auto share from interfering with the named share that is being created.

Displaying ZFS Share Information

Display the value of the file sharing properties by using `zfs get` command. The following example shows how to display the `share.nfs` property for a single file system:

```
# zfs get share.nfs tank/sales
NAME          PROPERTY  VALUE  SOURCE
tank/sales    share.nfs on      local
```

The following example shows how to display the `share.nfs` property for descendent file systems:

```
# zfs get -r share.nfs tank/sales
NAME          PROPERTY  VALUE  SOURCE
tank/sales    share.nfs on      local
tank/sales%   share.nfs on      inherited from tank/sales
tank/sales/log share.nfs on      inherited from tank/sales
tank/sales/log% share.nfs on      inherited from tank/sales
```

The extended share property information is not available in the `zfs get all` command syntax.

You can display specific details about NFS or SMB share information by using the following syntax:

```
# zfs get share.nfs.all tank/sales
NAME          PROPERTY          VALUE  SOURCE
tank/sales    share.nfs.aclok    off    default
tank/sales    share.nfs.anon     default
tank/sales    share.nfs.charset.* ...    default
tank/sales    share.nfs.cksum    default
tank/sales    share.nfs.index    default
tank/sales    share.nfs.log      default
tank/sales    share.nfs.noaclfab off    default
tank/sales    share.nfs.nosub    off    default
tank/sales    share.nfs.nosuid   off    default
tank/sales    share.nfs.public   -      -
tank/sales    share.nfs.sec      default
tank/sales    share.nfs.sec.*    ...    default
```

Because there are many share properties, consider displaying the properties with a non-default value. For example:

```
# zfs get -e -s local,received,inherited share.all tank/home
NAME          PROPERTY          VALUE  SOURCE
tank/home     share.auto        off    local
tank/home     share.nfs         on     local
tank/home     share.nfs.anon    99     local
tank/home     share.protocols   nfs    local
tank/home     share.smb.guestok on      inherited from tank
```

Changing ZFS Share Property Values

You can change share property values by specifying new or modified properties on a file system share. For example, if the read-only property is set when the file system is created, the property can be set to off.

```
# zfs create -o share.nfs.ro=* tank/data
# zfs get share.nfs.ro tank/data
NAME          PROPERTY          VALUE  SOURCE
tank/data     share.nfs.sec.sys.ro *      local
# zfs set share.nfs.ro=none tank/data
# zfs get share.nfs.ro tank/data
NAME          PROPERTY          VALUE  SOURCE
tank/data     share.nfs.sec.sys.ro off    local
```

If you create an SMB share, you can also add the NFS share protocol. For example:

```
# zfs set share.smb=on tank/multifs
# zfs set share.nfs=on tank/multifs
# grep multifs /etc/dfs/sharetab
/tank/multifs  tank_multifs  nfs      sec=sys,rw
```

```
/tank/multifs    tank_multifs    smb    -
```

Remove the SMB protocol:

```
# zfs set share.smb=off tank/multifs
# grep multifs /etc/dfs/sharetab
/tank/multifs    tank_multifs    nfs    sec=sys,rw
```

You can rename a named share. For example:

```
# zfs share -o share.smb=on tank/home/abc%abcshare
# grep abc /etc/dfs/sharetab
/tank/home/abc  abcshare    smb    -
# zfs rename tank/home/abc%abcshare tank/home/abc%alshare
# grep abc /etc/dfs/sharetab
/tank/home/abc  alshare    smb    -
```

Publishing and Unpublishing ZFS Shares

You can temporarily unshare a named share without destroying it by using the `zfs unshare` command. For example:

```
# zfs unshare tank/home/abc%alshare
# grep abc /etc/dfs/sharetab
#
# zfs share tank/home/abc%alshare
# grep abc /etc/dfs/sharetab
/tank/home/abc  alshare smb    -
```

When the `zfs unshare` command is issued, all file system shares are unshared. These shares remain unshared until the `zfs share` command is issued for the file system or the `share.nfs` or `share.smb` property is set for the file system.

Defined shares are not removed when the `zfs unshare` command is issued, and they are re-shared the next time the `zfs share` command is issued for the file system or the `share.nfs` or `share.smb` property is set for the file system.

Removing a ZFS Share

You can unshare a file system share by setting the `share.nfs` or `share.smb` property to `off`. For example:

```
# zfs set share.nfs=off tank/multifs
# grep multifs /etc/dfs/sharetab
#
```

You can permanently remove a named share by using the `zfs destroy` command. For example:

```
# zfs destroy tank/home/abc%alshare
```


ZFS File Sharing Within a Non-Global Zone

Starting with Oracle Solaris 11 you can create and publish NFS shares in a non-global zone.

- If a ZFS file system is mounted and available in a non-global zone, it can be shared in that zone.
- A file system can be shared in the global zone if it is not delegated to a non-global zone and is not mounted in a non-global zone. If a file system is added to a non-global zone, it can only be shared by using the legacy share command.

For example, the `/export/home/data` and `/export/home/data1` file systems are available in the `zfszone`.

```
zfszone# share -F nfs /export/home/data
zfszone# cat /etc/dfs/sharetab

zfszone# zfs set share.nfs=on tank/zones/export/home/data1
zfszone# cat /etc/dfs/sharetab
```

ZFS Sharing Migration/Transition Issues

Review the following transition issues:

- **Importing file systems with older sharing properties** - When importing a pool or receiving a file system stream that was created before Oracle Solaris 11, the `share nfs` and `share smb` properties include all the share properties directly in the property value. In most cases, these legacy share properties are converted to an equivalent set of named shares as soon as each file system is shared. Since import operations trigger mounting and sharing in most cases, the conversion to named shares happens directly during the import process.
- **Upgrading from Oracle Solaris 11** - The first file system sharing after a pool upgrade to version 34 can take a long time because the named shares are converted to the new format. The named shares created by the upgrade process are correct but cannot take advantage of share property inheritance.

- Display share property values:

```
# zfs get share.nfs filesystem
# zfs get share.smb filesystem
```

- If you boot back to an older BE, reset the `share nfs` and `share smb` properties to their original values.
- **Upgrading from Oracle Solaris 11** - In Oracle Solaris 11 and 11.1, the `share nfs` and `share smb` properties can have only `off` and `on` values. These properties are no longer used to define share characteristics.

The `/etc/dfs/dfstab` file is no longer used to share file systems at boot time. At boot time, all mounted ZFS file systems that include enabled file system shares are automatically shared. A share is enabled when its `share nfs` or `share smb` is set to `on`.

The sharemgr interface is no longer available. The legacy share command is still available to create a legacy share. The share -a command is like the previous share -ap command so that sharing a file system is persistent. The share -p option is no longer available.

- **Upgrading your system** – ZFS shares are incorrect if you boot back to an Oracle Solaris 11 BE due to property changes in this release. Non-ZFS shares are unaffected. If you plan to boot back to an older BE, first save a copy of the existing share configuration prior to the pkg update operation to be able to restore the ZFS share configuration.

In the older BE, use the sharemgr show -vp command to list all shares and their configuration.

Use the following commands to display share property values:

```
# zfs get sharenfs filesystem
# zfs get sharesmb filesystem
```

If you back to an older BE, reset the sharenfs and sharesmb properties and any shares defined with sharemgr to their original values.

- **Legacy unsharing behavior** – Using the unshare -a command or unshareall command unshares a file system, but does not update the SMF shares repository. If you try to re-share the existing share, the shares repository is checked for conflicts, and an error is displayed.

Troubleshooting ZFS File System Sharing Problems

Review the following share error conditions:

- **New shares or previous shares are not shared**
 - **Confirm the pool and the file system versions are current** - If new shares are not shared by setting the share.nfs or share.smb property, then confirm that the pool version is 34 and the file system version is 6.
 - **Share must exist before NFS services start**- NFS server services do not run until a file system is shared. Create the NFS share first and then attempt to access the share remotely.
 - **System with existing shares was upgraded but shares are not available** - A system with existing shares is upgraded but attempts to reshare the shares fail. The shares might not be shared because the share.auto property is disabled. If share.auto is set to off, only named shares are available, which enforces compatibility with earlier sharing syntax. The existing shares might look like this:

```
# zfs get share
NAME                                PROPERTY  VALUE  SOURCE
tank/data                          share     name=data,path=/tank/data,prot=nfs  local
```

1. Ensure that the share.auto property is enabled. If not, enable it.

- ```
zfs get -r share.auto tank/data
zfs set share.auto=on tank/data
```
- Reshare the file system.

```
zfs set -r share.nfs=on tank/data
```
  - You might also need to remove named shares and recreate them before the preceding command is successful.

```
zfs list -t share -Ho name -r tank/data | xargs -n1 zfs destroy
```
  - If necessary, recreate the named shares.

```
zfs create -o share.nfs=on tank/data%share
```

- **Sharing properties including named shares are not included in snapshots** - Share properties and `.zfs/shares` files are treated differently in `zfs clone` and `zfs send` operations. The `.zfs/shares` files are included in snapshots and are preserved in `zfs clone` and `zfs send` operations. For a description of the behavior of properties during `zfs send` and `zfs receive` operations, see [“Applying Different Property Values to a ZFS Snapshot Stream” on page 176](#). After a clone operation, all files are from the pre-clone snapshot, whereas the properties are inherited from the clone's new position in the ZFS file system hierarchy.
- **Named share request fails** - If a request to create a named share fails because the share would conflict with the auto share, you may have to disable the `auto.share` property.
- **Pool with shares was previously exported** - When a pool is imported read-only, neither its properties nor its files can be modified so creating a new share fails. If the shares existed before the pool was exported, the existing sharing characteristics are used, if possible.

The following table identifies known share states and how to resolve them, if necessary.

| Share State | Description                                                                                                                                                                                                                                                    | Resolution                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| INVALID     | The share is invalid because it is internally inconsistent or because it conflicts with another share.                                                                                                                                                         | Attempt to re-share the invalid share by using the following command:<br><br><b># zfs share FS%share</b><br><br>Using this command displays an error message about which aspect of the share is failing validation. Correct this, then retry the share.                                                                                                                                              |
| SHARED      | The share is shared.                                                                                                                                                                                                                                           | None needed.                                                                                                                                                                                                                                                                                                                                                                                         |
| UNSHARED    | The share is valid but is unshared.                                                                                                                                                                                                                            | Use the <code>zfs share</code> command to re-share either the individual share or the parent file system.                                                                                                                                                                                                                                                                                            |
| UNVALIDATED | The share is not yet validated. The file system that contains the share might not be in a shareable state. For example, it is not mounted or it is delegated to a zone other than the current zone. Alternatively, the ZFS properties representing the desired | Use the <code>zfs share</code> command to re-share the individual share or the parent file system. If the file system itself is shareable, an attempt to re-share will either succeed in sharing (and transition the state to shared) or fail to share (and transition the state to invalid). Or, you can use the <code>share -A</code> command to list all shares in all mounted file systems. This |

| Share State | Description                                                                | Resolution                                                                                                             |
|-------------|----------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
|             | share have been created, but have not yet been validated as a legal share. | will cause all shares in mounted file systems to be resolved as either unshared (valid but not yet shared) or invalid. |

## Setting ZFS Quotas and Reservations

You can use the `quota` property to set a limit on the amount of disk space a file system can use. In addition, you can use the `reservation` property to guarantee that a specified amount of disk space is available to a file system. Both properties apply to the file system on which they are set and all descendents of that file system.

That is, if a quota is set on the `tank/home` file system, the total amount of disk space used by `tank/home` *and all of its descendents* cannot exceed the quota. Similarly, if `tank/home` is given a reservation, `tank/home` *and all of its descendents* draw from that reservation. The amount of disk space used by a file system and all of its descendents is reported by the `used` property.

The `refquota` and `refreservation` properties are used to manage file system space without accounting for disk space consumed by descendents, such as snapshots and clones.

In this Oracle Solaris release, you can set a *user* or a *group* quota on the amount of disk space consumed by files that are owned by a particular user or group. The user and group quota properties cannot be set on a volume, on a file system before file system version 4, or on a pool before pool version 15.

Consider the following points to determine which quota and reservation features might best help you manage your file systems:

- The quota and reservation properties are convenient for managing disk space consumed by file systems and their descendents.
- The `refquota` and `refreservation` properties are appropriate for managing disk space consumed by file systems.
- Setting the `refquota` or `refreservation` property higher than the quota or reservation property has no effect. If you set the quota or `refquota` property, operations that try to exceed either value fail. It is possible to exceed a quota that is greater than the `refquota`. For example, if some snapshot blocks are modified, you might actually exceed the quota before you exceed the `refquota`.
- User and group quotas provide a way to more easily manage disk space with many user accounts, such as in a university environment.
- A convenient way to set a quota on a large file system for many different users is to set a default user or group quota.

For more information about setting quotas and reservations, see [“Setting Quotas on ZFS File Systems” on page 141](#) and [“Setting Reservations on ZFS File Systems” on page 145](#).

## Setting Quotas on ZFS File Systems

Quotas on ZFS file systems can be set and displayed by using the `zfs set` and `zfs get` commands. In the following example, a quota of 10 GB is set on `tank/home/jeff`:

```
zfs set quota=10G tank/home/jeff
zfs get quota tank/home/jeff
NAME PROPERTY VALUE SOURCE
tank/home/jeff quota 10G local
```

Quotas also affect the output of the `zfs list` and `df` commands. For example:

```
zfs list -r tank/home
NAME USED AVAIL REFER MOUNTPOINT
tank/home 1.45M 66.9G 36K /tank/home
tank/home/eric 547K 66.9G 547K /tank/home/eric
tank/home/jeff 322K 10.0G 291K /tank/home/jeff
tank/home/jeff/ws 31K 10.0G 31K /tank/home/jeff/ws
tank/home/lori 547K 66.9G 547K /tank/home/lori
tank/home/mark 31K 66.9G 31K /tank/home/mark

df -h /tank/home/jeff
Filesystem Size Used Avail Use% Mounted on
tank/home/jeff 10G 306K 10G 1% /tank/home/jeff
```

Note that although `tank/home` has 66.9 GB of disk space available, `tank/home/jeff` and `tank/home/jeff/ws` each have only 10 GB of disk space available, due to the quota on `tank/home/jeff`.

You can set a `refquota` on a file system that limits the amount of disk space that the file system can consume. This limit does not include disk space that is consumed by descendents. For example, `studentA`'s 10 GB quota is not impacted by space that is consumed by snapshots.

```
zfs set refquota=10g students/studentA
zfs list -t all -r students
NAME USED AVAIL REFER MOUNTPOINT
students 150M 66.8G 32K /students
students/studentA 150M 9.85G 150M /students/studentA
students/studentA@yesterday 0 - 150M -

zfs snapshot students/studentA@today
zfs list -t all -r students
students 150M 66.8G 32K /students
students/studentA 150M 9.90G 100M /students/studentA
students/studentA@yesterday 50.0M - 150M -
students/studentA@today 0 - 100M -
```

For additional convenience, you can set another quota on a file system to help manage the disk space that is consumed by snapshots. For example:

```
zfs set quota=20g students/studentA
```

```
zfs list -t all -r students
NAME USED AVAIL REFER MOUNTPOINT
students 150M 66.8G 32K /students
students/studentA 150M 9.90G 100M /students/studentA
students/studentA@yesterday 50.0M - 150M -
students/studentA@today 0 - 100M -
```

In this scenario, studentA might reach the refquota (10 GB) hard limit, but studentA can remove files to recover, even if snapshots exist.

In the preceding example, the smaller of the two quotas (10 GB as compared to 20 GB) is displayed in the `zfs list` output. To view the value of both quotas, use the `zfs get` command. For example:

```
zfs get refquota,quota students/studentA
NAME PROPERTY VALUE SOURCE
students/studentA refquota 10G local
students/studentA quota 20G local
```

Enforcement of a file system quota might be delayed by several seconds. This delay means that a user might exceed the file system quota before the system notices that the file system is over quota and refuses additional writes with the `EDQUOT` error message.

## Setting User and Group Quotas on a ZFS File System

You can set a user quota or a group quota by using the `zfs userquota` or `zfs groupquota` commands, respectively. For example:

```
zfs create students/compsci
zfs set userquota@student1=10G students/compsci
zfs create students/labstaff
zfs set groupquota@labstaff=20GB students/labstaff
```

Display the current user quota or group quota as follows:

```
zfs get userquota@student1 students/compsci
NAME PROPERTY VALUE SOURCE
students/compsci userquota@student1 10G local
zfs get groupquota@labstaff students/labstaff
NAME PROPERTY VALUE SOURCE
students/labstaff groupquota@labstaff 20G local
```

You can display general user or group disk space usage by querying the following properties:

```
zfs userspace students/compsci
TYPE NAME USED QUOTA
POSIX User root 350M none
POSIX User student1 426M 10G
zfs groupspace students/labstaff
```

| TYPE        | NAME     | USED | QUOTA |
|-------------|----------|------|-------|
| POSIX Group | labstaff | 250M | 20G   |
| POSIX Group | root     | 350M | none  |

To identify individual user or group disk space usage, query the following properties:

```
zfs get userused@student1 students/compsci
NAME PROPERTY VALUE SOURCE
students/compsci userused@student1 550M local
zfs get groupused@labstaff students/labstaff
NAME PROPERTY VALUE SOURCE
students/labstaff groupused@labstaff 250 local
```

The user and group quota properties are not displayed by using the `zfs get all dataset` command, which displays a list of all of the other file system properties.

You can remove a user quota or group quota as follows:

```
zfs set userquota@student1=none students/compsci
zfs set groupquota@labstaff=none students/labstaff
```

User and group quotas on ZFS file systems provide the following features:

- A user quota or group quota that is set on a parent file system is not automatically inherited by a descendent file system.
- However, the user or group quota is applied when a clone or a snapshot is created from a file system that has a user or group quota. Likewise, a user or group quota is included with the file system when a stream is created by using the `zfs send` command, even without the `-R` option.
- Unprivileged users can only access their own disk space usage. The root user or a user who has been granted the `userused` or `groupused` privilege, can access everyone's user or group disk space accounting information.
- The `userquota` and `groupquota` properties cannot be set on ZFS volumes, on a file system prior to file system version 4, or on a pool prior to pool version 15.

Enforcement of user and group quotas might be delayed by several seconds. This delay means that a user might exceed the user quota before the system notices that the user is over quota and refuses additional writes with the `EDQUOT` error message.

You can use the legacy `quota` command to review user quotas in an NFS environment, for example, where a ZFS file system is mounted. Without any options, the `quota` command only displays output if the user's quota is exceeded. For example:

```
zfs set userquota@student1=10m students/compsci
zfs userspace students/compsci
TYPE NAME USED QUOTA
POSIX User root 350M none
POSIX User student1 550M 10M
quota student1
```

Block limit reached on /students/compsci

If you reset the user quota and the quota limit is no longer exceeded, you can use the `quota -v` command to review the user's quota. For example:

```
zfs set userquota@student1=10GB students/compsci
zfs userspace students/compsci
TYPE NAME USED QUOTA
POSIX User root 350M none
POSIX User student1 550M 10G
quota student1
quota -v student1
Disk quotas for student1 (uid 102):
Filesystem usage quota limit timeleft files quota limit timeleft
/students/compsci
563287 10485760 10485760 - - - -
```

## Setting Default User and Group Quotas

Starting in the Oracle Solaris 11.3 release, you can set a default user quota or a default group quota that is applied automatically for anyone who does not have a specific quota defined. Similar to specific user and group quotas, default user and group quotas are not inheritable to descendent file systems. In addition, if a default user or group quota is set on a top-level file system, space consumed in descendent file systems is not charged to the top-level file system's default quota.

You can set a default user quota on a large shared file system. For example:

```
zfs set defaultuserquota=30gb students/labstaff/admindata
```

Using a default user quota on a large shared file system allows you to restrict growth without specifying individual user quotas. You can also monitor who is using the top-level file system.

```
zfs userspace students/labstaff/admindata
TYPE NAME USED QUOTA SOURCE
POSIX User admin1 2.00G 30G default
POSIX User admin2 4.00G 30G default
POSIX User root 3K 30G default
```

In the above example, each user that doesn't have an existing quota is allowed 30 GB of disk space in `students/labstaff/admindata`. Contrasting this behavior to setting a 30 GB file system quota on `students/labstaff/admindata`, means that a cumulative quota of 30 GB would apply to all users who didn't have an existing quota.

You can set a default group quota in a similar way. For example, the following syntax sets a 120 GB quota on the `students/math` file system. You can use the `zfs groupquota` command to track usage of a top-level file system with a default group quota.

```
zfs set defaultgroupquota=120g students/math
```



```
zfs groupspace students/math
TYPE NAME USED QUOTA SOURCE
POSIX Group root 6K 120G default
POSIX Group students 40.0G 120G default
```

## Setting Reservations on ZFS File Systems

A ZFS *reservation* is an allocation of disk space from the pool that is guaranteed to be available to a dataset. As such, you cannot reserve disk space for a dataset if that space is not currently available in the pool. The total amount of all outstanding, unconsumed reservations cannot exceed the amount of unused disk space in the pool. ZFS reservations can be set and displayed by using the `zfs set` and `zfs get` commands. For example:

```
zfs set reservation=5G tank/home/bill
zfs get reservation tank/home/bill
NAME PROPERTY VALUE SOURCE
tank/home/bill reservation 5G local
```

Reservations can affect the output of the `zfs list` command. For example:

```
zfs list -r tank/home
NAME USED AVAIL REFER MOUNTPOINT
tank/home 5.00G 61.9G 37K /tank/home
tank/home/bill 31K 66.9G 31K /tank/home/bill
tank/home/jeff 337K 10.0G 306K /tank/home/jeff
tank/home/lori 547K 61.9G 547K /tank/home/lori
tank/home/mark 31K 61.9G 31K /tank/home/mark
```

Note that `tank/home` is using 5 GB of disk space, although the total amount of space referred to by `tank/home` and its descendents is much less than 5 GB. The used space reflects the space reserved for `tank/home/bill`. Reservations are considered in the used disk space calculation of the parent file system and do count against its quota, reservation, or both.

```
zfs set quota=5G pool/filesystem
zfs set reservation=10G pool/filesystem/user1
cannot set reservation for 'pool/filesystem/user1': size is greater than
available space
```

A dataset can use more disk space than its reservation, as long as unreserved space is available in the pool, and the dataset's current usage is below its quota. A dataset cannot consume disk space that has been reserved for another dataset.

Reservations are not cumulative. That is, a second invocation of `zfs set` to set a reservation does not add its reservation to the existing reservation. Rather, the second reservation replaces the first reservation. For example:

```
zfs set reservation=10G tank/home/bill
```

```
zfs set reservation=5G tank/home/bill
zfs get reservation tank/home/bill
NAME PROPERTY VALUE SOURCE
tank/home/bill reservation 5G local
```

You can set a `refreservation` reservation to guarantee disk space for a dataset that does not include disk space consumed by snapshots and clones. This reservation is accounted for in the parent dataset's space used calculation, and counts against the parent dataset's quotas and reservations. For example:

```
zfs set refreservation=10g profs/prof1
zfs list
NAME USED AVAIL REFER MOUNTPOINT
profs 10.0G 23.2G 19K /profs
profs/prof1 10G 33.2G 18K /profs/prof1
```

You can also set a reservation on the same dataset to guarantee dataset space and snapshot space. For example:

```
zfs set reservation=20g profs/prof1
zfs list
NAME USED AVAIL REFER MOUNTPOINT
profs 20.0G 13.2G 19K /profs
profs/prof1 10G 33.2G 18K /profs/prof1
```

Regular reservations are accounted for in the parent's used space calculation.

In the preceding example, the smaller of the two quotas (10 GB as compared to 20 GB) is displayed in the `zfs list` output. To view the value of both quotas, use the `zfs get` command. For example:

```
zfs get reservation,refreserv profs/prof1
NAME PROPERTY VALUE SOURCE
profs/prof1 reservation 20G local
profs/prof1 refreservation 10G local
```

If `refreservation` is set, a snapshot is only allowed if sufficient unreserved pool space exists outside of this reservation to accommodate the current number of *referenced* bytes in the dataset.

## Compressing ZFS File Systems

Compression is the process where data is stored using less disk space. The following compression algorithms are available:

- `gzip` - standard UNIX compression.

- `gzip-N` - selects a specific gzip level. `gzip-1` provides the fastest gzip compression. `gzip-9` provides the best data compression. `gzip-6` is the default.
- `lz4` - provides better compression with lower CPU overhead
- `lzjb` - optimized for performance while providing decent compression
- `zle` - zero length encoding is useful for datasets with large blocks of zeros

---

**Note** - Currently, neither `lz4` or `gzip` compression is supported on root pools.

---

You can choose a specific compression algorithm by setting the compression ZFS property. To use the LZ4 algorithm, use a command like the following:

```
zfs set compress=lz4 pool/fs
```

## Encrypting ZFS File Systems

Encryption is the process where data is encoded for privacy and a key is needed by the data owner to access the encoded data. The benefits of using ZFS encryption are as follows:

- ZFS encryption is integrated with the ZFS command set. Like other ZFS operations, encryption operations such as key changes and rekey are performed online.
- You can use your existing storage pools as long as they are upgraded. You have the flexibility of encrypting specific file systems.
- Data is encrypted using AES (Advanced Encryption Standard) with key lengths of 128, 192, and 256 in the CCM and GCM operation modes.
- ZFS encryption uses the Oracle Solaris Cryptographic Framework, which gives it access to any available hardware acceleration or optimized software implementations of the encryption algorithms automatically.
- Currently, you cannot encrypt the ZFS root file system or other OS components, such as the `/var` directory, even if it is a separate file system.
- ZFS encryption is inheritable to descendent file systems.
- A regular user can create an encrypted file system and manage key operations if `create`, `mount`, `keysource`, `checksum`, and `encryption` permissions are assigned to him.

You can set an encryption policy when a ZFS file system is created, but the policy cannot be changed. For example, the `tank/home/darren` file system is created with the encryption property enabled. The default encryption policy is to prompt for a passphrase, which must be a minimum of 8 characters in length.

```
zfs create -o encryption=on tank/home/darren
Enter passphrase for 'tank/home/darren': xxxxxxxx
Enter again: xxxxxxxx
```

Confirm that the file system has encryption enabled. For example:

```
zfs get encryption tank/home/darren
NAME PROPERTY VALUE SOURCE
tank/home/darren encryption on local
```

The default encryption algorithm is `aes-128-ccm` when a file system's encryption value is on.

A *wrapping key* is used to encrypt the actual data encryption keys. The wrapping key is passed from the `zfs` command, as in the above example when the encrypted file system is created, to the kernel. A wrapping key is either in a file (in raw or hex format) or it is derived from a passphrase.

The format and location of the wrapping key are specified in the `keysource` property as follows:

`keysource=format,location`

- Format is one of the following:
  - `raw` – The raw key bytes
  - `hex` – A hexadecimal key string
  - `passphrase` – A character string that generates a key
- Location is one of the following:
  - `prompt` – You are prompted for a key or a passphrase when the file system is created or mounted
  - `file:///filename` – The key or a passphrase file location in a file system
  - `pkcs11` – A URI describing the location of a key or a passphrase in a PKCS#11 token
  - `https://location` – The key or a passphrase file location on a secure server. Transporting key information in the clear using this method is not recommended. A GET on the URL returns just the key value or the passphrase, according to what was requested in the format part of the `keysource` property.

When using an `https://` locator for the `keysource`, the certificate that the ZFS server presents must be one that is trusted by `libcurl` and `OpenSSL`. Add your own trust anchor or self signed certificate to the certificate store in `/etc/openssl/certs`. Place the PEM format certificate into the `/etc/certs/CA` directory and run the following command:

```
svcadm refresh ca-certificates
```

If the `keysource` format is *passphrase*, then the wrapping key is derived from the passphrase. Otherwise, the `keysource` property value points to the actual wrapping key, as raw bytes or in hexadecimal format. You can specify that the passphrase is stored in a file or stored in a raw stream of bytes that are prompted for, which is likely only suitable for scripting.

When a file system's `keysource` property values identifies *passphrase*, then the wrapping key is derived from the passphrase using PKCS#5 PBKDF2 and a per file system randomly generated

salt. This means that the same passphrase generates a different wrapping key if used on descendent file systems.

A file system's encryption policy is inherited by descendent file systems and cannot be removed. For example:

```
zfs snapshot tank/home/darren@now
zfs clone tank/home/darren@now tank/home/darren-new
Enter passphrase for 'tank/home/darren-new': xxxxxxxx
Enter again: xxxxxxxx
zfs set encryption=off tank/home/darren-new
cannot set property for 'tank/home/darren-new': 'encryption' is readonly
```

If you need to copy or migrate encrypted or unencrypted ZFS file systems, then consider the following points:

- Currently, you cannot send an unencrypted dataset stream and receive it as an encrypted stream even if the receiving pool's dataset has encryption enabled.
- You can use the following commands to migrate unencrypted data to a pool/file system with encryption enabled:
  - `cp -r`
  - `find | cpio`
  - `tar`
  - `rsync`
- A replicated encrypted file system stream can be received into a encrypted file system and the data remains encrypted. For more information, see [Example 36, “Sending and Receiving an Encrypted ZFS File System,”](#) on page 154.

## Changing an Encrypted ZFS File System's Keys

You can change an encrypted file system's wrapping key by using the `zfs key -c` command. The existing wrapping key must have been loaded first, either at boot time or by explicitly loading the file system key (`zfs key -l`) or by mounting the file system (`zfs mount filesystem`). For example:

```
zfs key -c tank/home/darren
Enter new passphrase for 'tank/home/darren': xxxxxxxx
Enter again: xxxxxxxx
```

In the following example, the wrapping key is changed and the `keysource` property value is changed to specify that the wrapping key comes from a file.

```
zfs key -c -o keysource=raw,file:///media/stick/key tank/home/darren
```

The data encryption key for an encrypted file system can be changed by using the `zfs key -K` command, but the new encryption key is only used for newly written data. This feature can be

used to provide compliance with NIST 800-57 guidelines on a data encryption key's time limit. For example:

```
zfs key -K tank/home/darren
```

In the above example, the data encryption key is not visible nor is it directly managed by you. In addition, you need the keychange delegation to perform a key change operation.

The following encryption algorithms are available:

- aes-128-ccm, aes-192-ccm, aes-256-ccm
- aes-128-gcm, aes-192-gcm, aes-256-gcm

The ZFS keysource property identifies the format and location of the key that wraps the file system's data encryption keys. For example:

```
zfs get keysource tank/home/darren
NAME PROPERTY VALUE SOURCE
tank/home/darren keysource passphrase,prompt local
```

The ZFS rekeydate property identifies the date of the last `zfs key -K` operation. For example:

```
zfs get rekeydate tank/home/darren
NAME PROPERTY VALUE SOURCE
tank/home/darren rekeydate Wed Jul 25 16:54 2012 local
```

If an encrypted file system's creation and rekeydate properties have the same value, the file system has never been rekeyed by an `zfs key -K` operation.

## Managing ZFS Encryption Keys

ZFS encryption keys can be managed in different ways, depending on your needs, either on the local system or remotely, if a centralized location is needed.

- **Locally** – The above examples illustrate that the wrapping key can be either a passphrase prompt or a raw key that is stored in a file on the local system.
- **Remotely** – Key information can be stored remotely by using a centralized key management system like Oracle Key Manager or by using a web service that supports a simple GET request on an http or https URI. Oracle Key Manager key information is accessible to an Oracle Solaris system by using a PKCS#11 token.

For more information about managing ZFS encryption keys, see <http://www.oracle.com/technetwork/articles/servers-storage-admin/manage-zfs-encryption-1715034.html>

For information about using Oracle Key Manager to manage key information, see:

[http://docs.oracle.com/cd/E24472\\_02/](http://docs.oracle.com/cd/E24472_02/)

## Delegating ZFS Key Operation Permissions

Review the following permission descriptions for delegating key operations:

- Loading or unloading a file system key by using the `zfs key -l` and `zfs key -u` commands require the key permission. In most cases, you will need the mount permission as well.
- Changing a file system key by using the `zfs key -c` and `zfs key -K` commands require the keychange permission.

Consider delegating separate permissions for key use (load or unload) and key change, which allows you to have a two-person key operation model. For example, determine which users can use the keys versus which users can change them. Or, both users need to be present for a key change. This model also allows you to build a key escrow system.

## Mounting an Encrypted ZFS File System

Review the following considerations when attempting to mount an encrypted ZFS file system:

- If an encrypted file system key is not available during boot time, the file system is not mounted automatically. For example, a file system with an encryption policy set to `passphrase,prompt` will not mount during boot time because the boot process is not interrupted to prompt for a passphrase.
- If you want to mount a file system with an encryption policy set to `passphrase,prompt` at boot time, you will need to either explicitly mount it with the `zfs mount` command and specify the passphrase or use the `zfs key -l` command to be prompted for the key after the system is booted.

For example:

```
zfs mount -a
Enter passphrase for 'tank/home/darren': xxxxxxxx
Enter passphrase for 'tank/home/ws': xxxxxxxx
Enter passphrase for 'tank/home/mark': xxxxxxxx
```

- If an encrypted file system's `keysource` property points to a file in another file system, the mount order of the file systems can impact whether the encrypted file system is mounted at boot, particularly if the file is on removable media.

## Upgrading Encrypted ZFS File Systems

Before you upgrade an Oracle Solaris 11 system to 11.1, ensure that your encrypted file systems are mounted. Mount the encrypted file systems and provide the passphrases, if prompted.

```
zfs mount -a
Enter passphrase for 'pond/amy': xxxxxxxx
```

```
Enter passphrase for 'pond/rory': xxxxxxxx
zfs mount | grep pond
pond /pond
pond/amy /pond/amy
pond/rory /pond/rory
```

Then, upgrade the encrypted file systems.

```
zfs upgrade -a
```

If you attempt to upgrade encrypted ZFS file systems that are unmounted, a message similar to the following is displayed:

```
zfs upgrade -a
cannot set property for 'pond/amy': key not present
```

In addition, the `zpool status` output might show corrupted data.

```
zpool status -v pond
.
.
.
pond/amy:<0x1>
pond/rory:<0x1>
```

If the above errors occur, remount the encrypted file systems as directed above. Then, scrub and clear the pool errors.

```
zpool scrub pond
zpool clear pond
```

For more information about upgrading file systems, see [“Upgrading ZFS File Systems” on page 158](#).

## Interactions Between ZFS Compression, Deduplication, and Encryption Properties

Review the following considerations when using the ZFS compression, deduplication, and encryption properties:

- When a file is written, the data is compressed, encrypted, and the checksum is verified. Then, the data is deduplicated, if possible.
- When a file is read, the checksum is verified and the data is decrypted. Then, the data is decompressed, if required.
- If the `dedup` property is enabled on an encrypted file system that is also cloned and the `zfs key -K` or `zfs clone -K` commands have not been used on the clones, data from all the clones will be deduplicated, if possible.



## Examples of Encrypting ZFS File Systems

### EXAMPLE 33 Encrypting a ZFS File System by Using a Raw Key

In the following example, an aes-256-ccm encryption key is generated by using the `pktool` command and is written to a file, `/cindykey.file`.

```
pktool genkey keystore=file outkey=/cindykey.file keytype=aes keylen=256
```

Then, the `/cindykey.file` is specified when the `tank/home/cindy` file system is created.

```
zfs create -o encryption=aes-256-ccm -o keysource=raw,file:///cindykey.file
tank/home/cindy
```

### EXAMPLE 34 Encrypting a ZFS File System With a Different Encryption Algorithm

You can create a ZFS storage pool and have all the file systems in the storage pool inherit an encryption algorithm. In this example, the `users` pool is created and the `users/home` file system is created and encrypted by using a passphrase. The default encryption algorithm is aes-128-ccm.

Then, the `users/home/mark` file system is created and encrypted by using the aes-256-ccm encryption algorithm.

```
zpool create -O encryption=on users mirror c0t1d0 c1t1d0 mirror c2t1d0 c3t1d0
Enter passphrase for 'users': xxxxxxxx
Enter again: xxxxxxxx
zfs create users/home
zfs get encryption users/home
NAME PROPERTY VALUE SOURCE
users/home encryption on inherited from users
zfs create -o encryption=aes-256-ccm users/home/mark
zfs get encryption users/home/mark
NAME PROPERTY VALUE SOURCE
users/home/mark encryption aes-256-ccm local
```

### EXAMPLE 35 Cloning an Encrypted ZFS File System

If the clone file system inherits the `keysource` property from the same file system as its origin snapshot, then a new `keysource` is not necessary, and you are not prompted for a new passphrase if `keysource=passphrase`, prompt. The same `keysource` is used for the clone. For example:

By default, you are not prompted for a key when cloning a descendent of an encrypted file system.

```
zfs create -o encryption=on tank/ws
```

```
Enter passphrase for 'tank/ws': xxxxxxxx
Enter again: xxxxxxxx
zfs create tank/ws/fs1
zfs snapshot tank/ws/fs1@snap1
zfs clone tank/ws/fs1@snap1 tank/ws/fs1clone
```

If you want to create a new key for the clone file system, use the `zfs clone -K` command.

If you clone an encrypted file system rather than a descendent encrypted file system, you are prompted to provide a new key. For example:

```
zfs create -o encryption=on tank/ws
Enter passphrase for 'tank/ws': xxxxxxxx
Enter again: xxxxxxxx
zfs snapshot tank/ws@1
zfs clone tank/ws@1 tank/ws1clone
Enter passphrase for 'tank/ws1clone': xxxxxxxx
Enter again: xxxxxxxx
```

#### **EXAMPLE 36** Sending and Receiving an Encrypted ZFS File System

In the following example, the `tank/home/darren@snap1` snapshot is created from the encrypted `/tank/home/darren` file system. Then, the snapshot is sent to `bpool/snaps`, with the encryption property enabled so the resulting received data is encrypted. However, the `tank/home/darren@snap1` stream is not encrypted during the send process.

```
zfs get encryption tank/home/darren
NAME PROPERTY VALUE SOURCE
tank/home/darren encryption on local
zfs snapshot tank/home/darren@snap1
zfs get encryption bpool/snaps
NAME PROPERTY VALUE SOURCE
bpool/snaps encryption on inherited from bpool
zfs send tank/home/darren@snap1 | zfs receive bpool/snaps/darren1012
zfs get encryption bpool/snaps/darren1012
NAME PROPERTY VALUE SOURCE
bpool/snaps/darren1012 encryption on inherited from bpool
```

In this case, a new key is automatically generated for the received encrypted file system.

## Migrating ZFS File Systems

To migrate a local or remote ZFS or UFS file system to a target ZFS file system, use shadow migration. The target file system is also called the shadow file system.

Use the following commands to manage shadow migration:

- The `shadowadm` command stops, resumes, or cancels shadow migration.
- The `shadowstat` command with its options monitors migrations running on the system. Use the `shadowstat` command without options to monitor the progress of migrations. The displayed information is continuously updated until you type `Ctrl-C`.  
The command's `-E` and `-e` options are particularly useful.
  - To list all currently running migrations and identify those that could not be completed because of errors, use the `-E` option.
  - To list a specific migration and check if errors are causing the migration to fail, use the `-e` option.

For an example that shows how these commands are used, see [Example 37, “Starting and Monitoring File System Migrations,”](#) on page 156.



---

**Caution** - When migrating file systems, observe the following rules:

- Do not add or remove data from the file system while it is being migrated. Otherwise, those changes are excluded from migration.
  - Do not change the mountpoint property of the shadow file system while migration is in progress.
- 

## ▼ How to Migrate a File System to a ZFS File System

1. **If you are migrating data from a remote NFS server, confirm that the name service information is accessible on both remote and local systems.**

For a large migration using NFS, you might consider doing a test migration of a subset of the data to ensure that the UID, GUID, and ACL information migrates correctly.

2. **If necessary, install the shadow-migration package on the target system.**

```
pkg install shadow-migration
```

3. **Enable the shadowd service.**

```
svcadm enable shadowd
```

4. **Set the local or remote file system to be migrated to read-only.**

- If you are migrating a local ZFS file system, set it to read-only. For example:

```
zfs set share.nfs=on /export/home/ufsdata
```

- If you are migrating a remote file system, share it as read-only. For example:

```
share set share.nfs.ro='*/export/home/ufldata'
```

**5. Create the target file system while setting the shadow file to the file system to be migrated.**

---

**Note** - The new target ZFS file system must be completely empty. Otherwise, migration will not start.

---

Specify the shadow setting depending on the type of system you are migrating.

- If you are migrating a local file system, specify the source path.

For example:

```
zfs create -o shadow=file:///west/home/data users/home/shadow
```

- If you are migrating a NFS file system. specify the source host name and path.

For example:

```
zfs create -o shadow=nfs://neo/export/home/ufldata users/home/shadow2
```

**6. (Optional) To check the progress of the migration, issue the shadowstat command.**

For examples of how to monitor migrations, see [Example 37, “Starting and Monitoring File System Migrations,” on page 156](#).

---

**Note** - Migrating file system data over NFS can be slow, depending on your network bandwidth. If the system is rebooted during migration, the migration continues after the system boot completes.

---

**Example 37** Starting and Monitoring File System Migrations

In this example, multiple migrations are initiated. The shadowadm command lists ongoing migrations while the shadowstat command monitors their progress.

```
zfs create -o shadow=nfs://system2/pool/data/jsmith/archive rpool/data/copyarchive
shadowadm list
PATH STATE
/rpool/data/copyarchive ACTIVE

zfs create -o shadow=nfs://system2/pool/data/jsmith/datlogs rpool/data/logcopy
shadowadm list
PATH STATE
/rpool/data/copyarchive ACTIVE
/rpool/data/logcopy ACTIVE

shadowstat
```

| DATASET                | BYTES<br>XFRD | EST   |        | ELAPSED<br>TIME |                      |
|------------------------|---------------|-------|--------|-----------------|----------------------|
|                        |               | LEFT  | ERRORS |                 |                      |
| rpool/data/copyarchive | 34.4M         | 3.37G | -      | 00:00:36        |                      |
| rpool/data/logcopy     | 1.12K         | 155K  | 1      | (completed)     | Errors are detected. |
| rpool/data/copyarchive | 34.5M         | 3.37G | -      | 00:00:37        |                      |
| rpool/data/logcopy     | 1.12K         | 155K  | 1      | (completed)     |                      |
| rpool/data/copyarchive | 35.0M         | 3.37G | -      | 00:00:38        |                      |
| rpool/data/logcopy     | 1.12K         | 155K  | 1      | (completed)     |                      |
| rpool/data/copyarchive | 35.2M         | 3.37G | -      | 00:00:39        |                      |
| rpool/data/logcopy     | 1.12K         | 155K  | 1      | (completed)     |                      |

^C

The previous shadowstat output indicates errors in the migration to rpool/data/logcopy.

The following output from the shadowstat -E and -e command options shows that migration to rpool/data/logcopy could not be completed because socket migration is not supported. The shadowadm command cancels the migration.

```
shadowstat -E
rpool/data/copyarchive:
No errors encountered.
rpool/data/logcopy:
PATH ERROR
errdir/cups-socket Operation not supported

shadowstat -e /rpool/data/logcopy
rpool/data/logcopy:
PATH ERROR
errdir/cups-socket Operation not supported

shadowadm cancel /rpool/data/logcopy
```

The following output shows information about the migration to rpool/data/copyarchive, which continues toward completion.

```
shadowadm list
PATH STATE
/rpool/data/copyarchive ACTIVE shadowstat

DATASET EST
BYTES BYTES ELAPSED
XFRD LEFT ERRORS TIME
rpool/data/copyarchive 251M 3.16G - 00:01:27
rpool/data/copyarchive 251M 3.16G - 00:01:28
rpool/data/copyarchive 252M 3.16G - 00:01:29

^C

shadowstat
No migrations in progress.
shadowadm list
exit
```

## Upgrading ZFS File Systems

If you have ZFS file systems from a previous Oracle Solaris release, you can upgrade your file systems with the `zfs upgrade` command to take advantage of the file system features in the current release. In addition, this command notifies you when your file systems are running older versions.

For example, this file system is at the current version 5.

```
zfs upgrade
```

This system is currently running ZFS filesystem version 5.

All filesystems are formatted with the current version.

Use this command to identify the features that are available with each file system version.

```
zfs upgrade -v
```

The following filesystem versions are supported:

| VER | DESCRIPTION                                               |
|-----|-----------------------------------------------------------|
| --- | -----                                                     |
| 1   | Initial ZFS filesystem version                            |
| 2   | Enhanced directory entries                                |
| 3   | Case insensitive and File system unique identifier (FUID) |
| 4   | userquota, groupquota properties                          |
| 5   | System attributes                                         |
| 6   | Multilevel file system support                            |

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

For information about upgrading encrypted file systems, see [“Upgrading Encrypted ZFS File Systems” on page 151](#).

## Working With Oracle Solaris ZFS Snapshots and Clones

---

This chapter describes how to create and manage Oracle Solaris ZFS snapshots and clones. It also provides information about saving snapshots.

The chapter covers the following topics:

- “Overview of ZFS Snapshots” on page 159.
- “Creating and Destroying ZFS Snapshots” on page 160.
- “Displaying and Accessing ZFS Snapshots” on page 163.
- “Rolling Back a ZFS Snapshot” on page 165.
- “Overview of ZFS Clones” on page 167.
- “Creating a ZFS Clone” on page 168.
- “Destroying a ZFS Clone” on page 168.
- “Replacing a ZFS File System With a ZFS Clone” on page 169.
- “Saving, Sending, and Receiving ZFS Data” on page 170.
- “Monitoring ZFS Pool Operations” on page 181

### Overview of ZFS Snapshots

A *snapshot* is a read-only copy of a file system or volume. Snapshots can be created almost instantly, and they initially consume no additional disk space within the pool. However, as data within the active dataset changes, the snapshot consumes disk space by continuing to reference the old data, thus preventing the disk space from being freed.

A *clone* is a writable volume or file system whose initial contents are the same as the dataset from which it was created. Clones can be created only from a snapshot.

ZFS snapshots include the following features:

- They persist across system reboots.
- The theoretical maximum number of snapshots is  $2^{64}$ .

- Snapshots use no separate backing store. Snapshots consume disk space directly from the same storage pool as the file system or volume from which they were created.
- Recursive snapshots are created quickly as one atomic operation. The snapshots are created together (all at once) or not created at all. The benefit of atomic snapshot operations is that the snapshot data is always taken at one consistent time, even across descendant file systems.

Snapshots of volumes cannot be accessed directly, but they can be cloned, backed up, rolled back to, and so on. For information about backing up a ZFS snapshot, see [“Saving, Sending, and Receiving ZFS Data” on page 170](#).

This section covers the following topics:

- [“Creating and Destroying ZFS Snapshots” on page 160](#)
- [“Displaying and Accessing ZFS Snapshots” on page 163](#)
- [“Rolling Back a ZFS Snapshot” on page 165](#)

## Creating and Destroying ZFS Snapshots

You create snapshots by using the `zfs snapshot` or the `zfs snap` command, which takes as its only argument the name of the snapshot to create. The snapshot name uses one of the following conventions:

- *filesystem@snapname*
- *volume@snapname*

The snapshot name must satisfy the naming requirements in [“Naming ZFS Components” on page 22](#).

The following example creates a snapshot of `system1/home/cindy` that is named `friday`.

```
zfs snapshot system1/home/cindy@friday
```

To create snapshots for all descendant file systems, use the `-r` option. For example:

```
zfs snapshot -r system1/home@snap1
zfs list -t snapshot -r system1/home
```

| NAME                     | USED | AVAIL | REFER | MOUNTPPOINT |
|--------------------------|------|-------|-------|-------------|
| system1/home@snap1       | 0    | -     | 2.11G | -           |
| system1/home/cindy@snap1 | 0    | -     | 115M  | -           |
| system1/home/lori@snap1  | 0    | -     | 2.00G | -           |
| system1/home/mark@snap1  | 0    | -     | 2.00G | -           |
| system1/home/tim@snap1   | 0    | -     | 57.3M | -           |

Snapshots have no modifiable properties, nor can dataset properties be applied to a snapshot. For example:



```
zfs set compression=on system1/home/cindy@friday
cannot set property for 'system1/home/cindy@friday':
this property can not be modified for snapshots
```

To destroy snapshots, use the `zfs destroy` command. For example:

```
zfs destroy system1/home/cindy@friday
```

You cannot destroy a dataset if snapshots of the dataset exist. For example:

```
zfs destroy system1/home/cindy
cannot destroy 'system1/home/cindy': filesystem has children
use '-r' to destroy the following datasets:
system1/home/cindy@tuesday
system1/home/cindy@wednesday
system1/home/cindy@thursday
```

In addition, if clones have been created from a snapshot, then you must destroy them before you can destroy the snapshot.

For more information about the `destroy` subcommand, see [“How to Destroy a ZFS File System” on page 99](#).

## Holding ZFS Snapshots

Older snapshots are sometimes inadvertently destroyed due to different automatic snapshot or data retention policies. If a removed snapshot is part of an ongoing ZFS send and receive operation, then the operation might fail. To avoid this scenario, consider placing a hold on a snapshot.

*Holding* a snapshot prevents it from being destroyed. In addition, this feature allows a snapshot with clones to be deleted pending the removal of the last clone by using the `zfs destroy -d` command. Each snapshot has an associated *user-reference count*, which is initialized to zero. This count increases by 1 whenever a hold is put on a snapshot and decreases by 1 whenever a hold is released.

In the previous Oracle Solaris release, you could destroy a snapshot only by using the `zfs destroy` command if it had no clones. In this Oracle Solaris release, the snapshot must also have a zero user-reference count.

You can hold a snapshot or set of snapshots. For example, the following syntax puts a hold tag, `keep`, on `system1/home/cindy/snap@1`:

```
zfs hold keep system1/home/cindy@snap1
```

To recursively hold the snapshots of all descendant file systems, use the `-r` option. For example:

```
zfs snapshot -r system1/home@now
```

```
zfs hold -r keep system1/home@now
```

This syntax adds a single reference, `keep`, to the given snapshot or set of snapshots. Each snapshot has its own tag namespace and hold tags must be unique within that space. If a hold exists on a snapshot, attempts to destroy that held snapshot by using the `zfs destroy` command will fail. For example:

```
zfs destroy system1/home/cindy@snap1
cannot destroy 'system1/home/cindy@snap1': dataset is busy
```

To destroy a held snapshot, use the `-d` option. For example:

```
zfs destroy -d system1/home/cindy@snap1
```

To display a list of held snapshots, use the `zfs holds` command. For example:

```
zfs holds system1/home@now
NAME TAG TIMESTAMP
system1/home@now keep Fri Aug 3 15:15:53 2012

zfs holds -r system1/home@now
NAME TAG TIMESTAMP
system1/home/cindy@now keep Fri Aug 3 15:15:53 2012
system1/home/lori@now keep Fri Aug 3 15:15:53 2012
system1/home/mark@now keep Fri Aug 3 15:15:53 2012
system1/home/tim@now keep Fri Aug 3 15:15:53 2012
system1/home@now keep Fri Aug 3 15:15:53 2012
```

To release a hold on a snapshot or set of snapshots, use the `zfs release` command. For example:

```
zfs release -r keep system1/home@now
```

If the snapshot is released, the snapshot can be destroyed by using the `zfs destroy` command. For example:

```
zfs destroy -r system1/home@now
```

Two properties identify snapshot hold information.

- The `defer_destroy` property is set to `on` if the snapshot has been marked for deferred destruction by using the `zfs destroy -d` command. Otherwise, the property is set to `off`.
- The `userrefs` property is set to the number of holds on this snapshot.

## Renaming ZFS Snapshots

You can rename snapshots, but only within the same pool and dataset from which they were created. For example:

```
zfs rename system1/home/cindy@snap1 system1/home/cindy@today
```

You can also use the following shortcut syntax:

```
zfs rename system1/home/cindy@snap1 today
```

The following snapshot rename operation is not supported because the target pool and file system name are different from the pool and file system where the snapshot was created:

```
zfs rename system1/home/cindy@today pool/home/cindy@aturday
cannot rename to 'pool/home/cindy@today': snapshots must be part of same
dataset
```

You can recursively rename snapshots by using the `zfs rename -r` command. For example:

```
zfs list -t snapshot -r users/home
NAME USED AVAIL REFER MOUNTPOINT
users/home@now 23.5K - 35.5K -
users/home@yesterday 0 - 38K -
users/home/lori@yesterday 0 - 2.00G -
users/home/mark@yesterday 0 - 1.00G -
users/home/neil@yesterday 0 - 2.00G -
zfs rename -r users/home@yesterday @2daysago
zfs list -t snapshot -r users/home
NAME USED AVAIL REFER MOUNTPOINT
users/home@now 23.5K - 35.5K -
users/home@2daysago 0 - 38K -
users/home/lori@2daysago 0 - 2.00G -
users/home/mark@2daysago 0 - 1.00G -
users/home/neil@2daysago 0 - 2.00G -
```

## Displaying and Accessing ZFS Snapshots

By default, snapshots no longer appear in the `zfs list` output. You must use the `zfs list -t snapshot` command to display snapshot information. You can also enable the `listsnapshots` pool property instead. For example:

```
zpool get listsnapshots system1
NAME PROPERTY VALUE SOURCE
system1 listsnapshots off default
zpool set listsnapshots=on system1
zpool get listsnapshots system1
NAME PROPERTY VALUE SOURCE
system1 listsnapshots on local
```

Snapshots of file systems are placed in the `.zfs/snapshot` directory within the root of the file system. For example, if `system1/home/cindy` is mounted on `/home/cindy`, then the `system1/`

home/cindy@thursday snapshot data is accessible in the /home/cindy/.zfs/snapshot/thursday directory.

```
ls /system1/home/cindy/.zfs/snapshot
thursday tuesday wednesday
```

The following examples shows how to list snapshots.

```
zfs list -t snapshot -r system1/home
NAME USED AVAIL REFER MOUNTPOINT
system1/home/cindy@tuesday 45K - 2.11G -
system1/home/cindy@wednesday 45K - 2.11G -
system1/home/cindy@thursday 0 - 2.17G -
```

The following example shows how to list snapshots that were created for a particular file system.

```
zfs list -r -t snapshot -o name,creation system1/home
NAME CREATION
system1/home/cindy@tuesday Fri Aug 3 15:18 2012
system1/home/cindy@wednesday Fri Aug 3 15:19 2012
system1/home/cindy@thursday Fri Aug 3 15:19 2012
system1/home/lori@today Fri Aug 3 15:24 2012
system1/home/mark@today Fri Aug 3 15:24 2012
```

## Disk Space Accounting for ZFS Snapshots

When you create a snapshot, its disk space is initially shared between the snapshot and the file system, and possibly with previous snapshots. As the file system changes, disk space that was previously shared becomes unique to the snapshot, and thus is counted in the snapshot's used property. Additionally, deleting snapshots can increase the amount of disk space unique to (and thus *used* by) other snapshots.

A snapshot's space referenced property value is the same as the file system's value was when the snapshot was created.

You can identify additional information about how the values of the used property are consumed. New read-only file system properties describe disk space usage for clones, file systems, and volumes. For example:

```
$ zfs list -o space -r rpool
NAME AVAIL USED USED SNAP USED DDS USED REF RESERV USED CHILD
rpool 124G 9.57G 0 302K 0 9.57G
rpool/ROOT 124G 3.38G 0 31K 0 3.38G
rpool/ROOT/solaris 124G 20.5K 0 0 0 20.5K
rpool/ROOT/solaris/var 124G 20.5K 0 20.5K 0 0
rpool/ROOT/solaris-1 124G 3.38G 66.3M 3.14G 0 184M
rpool/ROOT/solaris-1/var 124G 184M 49.9M 134M 0 0
rpool/VARSHARE 124G 39.5K 0 39.5K 0 0
```

|                   |      |       |   |       |       |     |
|-------------------|------|-------|---|-------|-------|-----|
| rpool/dump        | 124G | 4.12G | 0 | 4.00G | 129M  | 0   |
| rpool/export      | 124G | 63K   | 0 | 32K   | 0     | 31K |
| rpool/export/home | 124G | 31K   | 0 | 31K   | 0     | 0   |
| rpool/swap        | 124G | 2.06G | 0 | 2.00G | 64.7M | 0   |

For a description of these properties, see the used properties in the [zfs\(1M\)](#) man page.

## Rolling Back a ZFS Snapshot

You can to discard all changes made to a file system since a specific snapshot was created by using the `zfs rollback` command. The file system reverts to its state at the time the snapshot was taken. By default, the command cannot roll back to a snapshot other than the most recent snapshot.

To roll back to an earlier snapshot, you must destroy all intermediate snapshots by specifying the `-r` option.

If clones of any intermediate snapshots exist, use the `-R` option to destroy the clones as well.

---

**Note** - The file system that you want to roll back must be unmounted and remounted, if it is currently mounted. If the file system cannot be unmounted, the rollback fails. Use the `-f` option to force the file system to be unmounted, if necessary.

---

In the following example, the `system1/home/cindy` file system is rolled back to the `tuesday` snapshot.

```
zfs rollback system1/home/cindy@tuesday
cannot rollback to 'system1/home/cindy@tuesday': more recent snapshots exist
use '-r' to force deletion of the following snapshots:
system1/home/cindy@wednesday
system1/home/cindy@thursday
zfs rollback -r system1/home/cindy@tuesday
```

In the following example, the `wednesday` and `thursday` snapshots are destroyed because the file system is rolled back to the earlier `tuesday` snapshot.

```
zfs list -r -t snapshot -o name,creation system1/home/cindy
NAME CREATION
system1/home/cindy@tuesday Fri Aug 3 15:18 2012
```

## Identifying ZFS Snapshot Differences (`zfs diff`)

You can determine ZFS snapshot differences by using the `zfs diff` command.

For example, assume that the following two snapshots are created:

```
$ ls /system1/home/tim
fileA
$ zfs snapshot system1/home/tim@snap1
$ ls /system1/home/tim
fileA fileB
$ zfs snapshot system1/home/tim@snap2
```

To identify the differences between the two snapshots, you would use syntax similar to the following example:

```
$ zfs diff system1/home/tim@snap1 system1/home/tim@snap2
M /system1/home/tim/
+ /system1/home/tim/fileB
```

In the output, the M indicates that the directory has been modified. The + indicates that fileB exists in the later snapshot.

The R in the following output indicates that a file in a snapshot has been renamed.

```
$ mv /system1/cindy/fileB /system1/cindy/fileC
$ zfs snapshot system1/cindy@snap2
$ zfs diff system1/cindy@snap1 system1/cindy@snap2
M /system1/cindy/
R /system1/cindy/fileB -> /system1/cindy/fileC
```

The following table summarizes the file or directory changes that are identified by the `zfs diff` command.

| File or Directory Change                                                               | Identifier |
|----------------------------------------------------------------------------------------|------------|
| File or directory has been modified or file or directory link has changed              | M          |
| File or directory is present in the older snapshot but not in the more recent snapshot | -          |
| File or directory is present in the more recent snapshot but not in the older snapshot | +          |
| File or directory has been renamed                                                     | R          |

For more information, see the [zfs\(1M\)](#) man page.

## Displaying Recursive ZFS Snapshot Differences

If you compare different snapshots by using the `zfs diff` command, the high level differences are displayed such as a new file system or directory. For example, the `sales` file system has 2 descendant file systems, `data` and `logs` with files within each descendant file system.

```
zfs list -r sales
```

| NAME       | USED  | AVAIL | REFER | MOUNTPOINT  |
|------------|-------|-------|-------|-------------|
| sales      | 1.75M | 66.9G | 33K   | /sales      |
| sales/data | 806K  | 66.9G | 806K  | /sales/data |
| sales/logs | 806K  | 66.9G | 806K  | /sales/logs |

The high-level differences can be displayed between `sales@snap1` and `sales@snap2`, where the primary difference is addition of the `sales/logs` file system.

```
zfs diff sales@snap1 sales@snap2
M /sales/
+ /sales/logs
```

You can recursively identify snapshot differences including file names by using syntax similar to the following:

```
zfs diff -r -E sales@snap1
D /sales/ (sales)
+ /sales/data
D /sales/data/ (sales/data)
+ /sales/data/dfile.1
+ /sales/data/dfile.2
+ /sales/data/dfile.3
zfs diff -r -E sales@snap2
D /sales/ (sales)
+ /sales/data
+ /sales/logs
D /sales/logs/ (sales/logs)
+ /sales/logs/lfile.1
+ /sales/logs/lfile.2
+ /sales/logs/lfile.3
D /sales/data/ (sales/data)
+ /sales/data/dfile.1
+ /sales/data/dfile.2
+ /sales/data/dfile.3
```

In the output, the lines that begin with `D` and end with *(name)* indicate a file system (dataset) and mount point.

## Overview of ZFS Clones

A *clone* is a writable volume or file system whose initial contents are the same as the dataset from which it was created. As with snapshots, creating a clone is nearly instantaneous and initially consumes no additional disk space. In addition, you can snapshot a clone.

Clones can be created only from a snapshot. When a snapshot is cloned, an implicit dependency is created between the clone and snapshot. Even though the clone is created somewhere else in the file system hierarchy, the original snapshot cannot be destroyed as long as the clone exists.

The `origin` property exposes this dependency, and the `zfs destroy` command lists any such dependencies, if they exist.

Clones do not inherit the properties of the dataset from which they were created. Use the `zfs get` and `zfs set` commands to view and change the properties of a cloned dataset. For more information, see [“Setting ZFS Properties” on page 119](#).

Because a clone initially shares all its disk space with the original snapshot, its `used` property value is initially zero. As changes are made to the clone, it uses more disk space. The `used` property of the original snapshot does not include the disk space consumed by the clone.

This section covers the following topics:

- [“Creating a ZFS Clone” on page 168](#).
- [“Destroying a ZFS Clone” on page 168](#).
- [“Replacing a ZFS File System With a ZFS Clone” on page 169](#).

## Creating a ZFS Clone

To create a clone, use the `zfs clone` command, specifying the snapshot from which to create the clone and the name of the new file system or volume, which can be located anywhere in the ZFS hierarchy. The new dataset is the same type (for example, file system or volume) as the snapshot from which the clone was created. You cannot create a clone of a file system in a pool that is different from where the original file system snapshot resides.

The following example creates a new clone named `system1/home/matt/bug123` with the same initial contents as the snapshot `system1/ws/gate@yesterday`.

```
zfs snapshot system1/ws/gate@yesterday
zfs clone system1/ws/gate@yesterday system1/home/matt/bug123
```

The following example creates a cloned workspace from the `projects/newproject@today` snapshot for a temporary user as `projects/teamA/tempuser`. Properties then are set on the cloned workspace.

```
zfs snapshot projects/newproject@today
zfs clone projects/newproject@today projects/teamA/tempuser
zfs set share.nfs=on projects/teamA/tempuser
zfs set quota=5G projects/teamA/tempuser
```

## Destroying a ZFS Clone

You destroy ZFS clones by using the `zfs destroy` command. For example:



```
zfs destroy system1/home/matt/bug123
```

You must destroy clones before the parent snapshot can be destroyed.

## Replacing a ZFS File System With a ZFS Clone

To replace an active ZFS file system with a clone of that file system, use the `zfs promote` command. This feature enables you to clone and replace file systems so that the *original* file system becomes the clone of the specified file system. In addition, you can destroy the file system from which the clone was originally created. Without clone promotion, you cannot destroy an original file system of active clones. For more information, see [“Destroying a ZFS Clone” on page 168](#).

In the following example, the `system1/test/productA` file system is cloned and then the clone file system, `system1/test/productAbeta`, becomes the original `system1/test/productA` file system.

```
zfs create system1/test
zfs create system1/test/productA
zfs snapshot system1/test/productA@today
zfs clone system1/test/productA@today system1/test/productAbeta
zfs list -r system1/test
```

| NAME                        | USED | AVAIL | REFER | MOUNTPOINT                 |
|-----------------------------|------|-------|-------|----------------------------|
| system1/test                | 104M | 66.2G | 23K   | /system1/test              |
| system1/test/productA       | 104M | 66.2G | 104M  | /system1/test/productA     |
| system1/test/productA@today | 0    | -     | 104M  | -                          |
| system1/test/productAbeta   | 0    | 66.2G | 104M  | /system1/test/productAbeta |

```
zfs promote system1/test/productAbeta
zfs list -r system1/test
```

| NAME                            | USED | AVAIL | REFER | MOUNTPOINT                 |
|---------------------------------|------|-------|-------|----------------------------|
| system1/test                    | 104M | 66.2G | 24K   | /system1/test              |
| system1/test/productA           | 0    | 66.2G | 104M  | /system1/test/productA     |
| system1/test/productAbeta       | 104M | 66.2G | 104M  | /system1/test/productAbeta |
| system1/test/productAbeta@today | 0    | -     | 104M  | -                          |

In this `zfs list` output, note that the disk space accounting information for the original `productA` file system has been replaced with the `productAbeta` file system.

You can complete the clone replacement process by renaming the file systems. For example:

```
zfs rename system1/test/productA system1/test/productAlegacy
zfs rename system1/test/productAbeta system1/test/productA
zfs list -r system1/test
```

Optionally, you can remove the legacy file system. For example:

```
zfs destroy system1/test/productAlegacy
```

## Saving, Sending, and Receiving ZFS Data

The `zfs send` command creates a stream representation of a snapshot that is written to standard output. By default, a full stream is generated. You can redirect the output to a file or to a different system. The `zfs receive` command creates a snapshot whose contents are specified in the stream that is provided on standard input. If a full stream is received, a new file system is created as well. You can also send ZFS snapshot data and receive ZFS snapshot data and file systems.

In addition, this release includes the ability to resume transferring ZFS data. In particular, the transfer of large amounts of ZFS data can be interrupted due to network outages or system failure. To prevent having to resend the whole thing again, the `zfs send` and `zfs receive` commands can be run with the `-C` option to resume sending the ZFS data. For more information, see [“Using Resumable Replication” on page 174](#).

This section covers the following topics:

- [“Saving ZFS Data With Other Backup Products” on page 171](#).
- [“Types of ZFS Snapshot Streams” on page 171](#).
- [“Sending a ZFS Snapshot” on page 173](#).
- [“Receiving a ZFS Snapshot” on page 175](#).
- [“Applying Different Property Values to a ZFS Snapshot Stream” on page 176](#).
- [“Sending and Receiving Complex ZFS Snapshot Streams” on page 178](#).
- [“Remote Replication of ZFS Data” on page 181](#).

Note the following backup solutions for saving ZFS data:

- **Enterprise backup products** – These products provide the following features:
  - Per-file restoration
  - Backup media verification
  - Media management
- **File system snapshots and rolling back snapshots** – Create a copy of a file system and revert to a previous file system version, if necessary.

For more information about creating and rolling back to a snapshot, see [“Overview of ZFS Snapshots” on page 159](#).
- **Saving snapshots** – Using the `zfs send` and `zfs receive` commands, you can save incremental changes between snapshots but you cannot restore files individually. You must restore the entire file system snapshot..
- **Remote replication** – Copy a file system from one system to another system. This process is different from a traditional volume management product that might mirror devices across a WAN. No special configuration or hardware is required. Using the `zfs send` and `zfs receive` commands to replicate a ZFS file system enables you to re-create a file system on a storage pool on another system, and specify different levels of configuration for the newly created pool, such as RAID-Z, but with identical file system data.

- **Archive utilities** – Save ZFS data with archive utilities such as `tar`, `cpio`, and `pax` or third-party backup products. Currently, both `tar` and `cpio` translate NFSv4-style ACLs correctly, but `pax` does not.

## Saving ZFS Data With Other Backup Products

In addition to the `zfs send` and `zfs receive` commands, you can also use archive utilities such as the `tar` and `cpio` commands, to save ZFS files. These utilities save and restore ZFS file attributes and ACLs. Check the appropriate options for both the `tar` and `cpio` commands.

## Types of ZFS Snapshot Streams

The `zfs send` command can be used to create a stream of one or more snapshots. Then, you can use the snapshot stream to re-create a ZFS file system or volume by using the `zfs receive` command.

The `zfs send` options used to create the snapshot stream determine the stream format type that is generated.

- **Full stream** – Consists of all dataset content from the time that the dataset was created up to the specified snapshot.

The default stream generated by the `zfs send` command is a full stream. It contains one file system or volume, up to and including the specified snapshot. The stream does not contain snapshots other than the snapshot specified in the command.

- **Incremental stream** – Consists of the differences between one snapshot and another snapshot.

A *stream package* is a stream type that contains one or more full or incremental streams. The types of stream packages are:

- **Replication stream package** – Consists of the specified dataset and its descendants. It includes all intermediate snapshots. If the origin of a cloned dataset is not a descendant of the snapshot specified on the command line, that origin dataset is not included in the stream package. To receive the stream, the origin dataset must exist in the destination storage pool.

Assume that the following list of datasets and their origins were created in the order in which they appear.

| NAME           | ORIGIN |
|----------------|--------|
| pool/a         | -      |
| pool/a/1       | -      |
| pool/a/1@clone | -      |
| pool/b         | -      |

|                   |                 |
|-------------------|-----------------|
| pool/b/1          | pool/a/1@clone  |
| pool/b/1@clone2   | -               |
| pool/b/2          | pool/b/1@clone2 |
| pool/b@pre-send   | -               |
| pool/b/1@pre-send | -               |
| pool/b/2@pre-send | -               |
| pool/b@send       | -               |
| pool/b/1@send     | -               |
| pool/b/2@send     | -               |

Suppose you have a replication stream package created with the following syntax:

```
zfs send -R pool/b@send
```

This package would consist of the following full and incremental streams:

| TYPE | SNAPSHOT          | INCREMENTAL FROM  |
|------|-------------------|-------------------|
| full | pool/b@pre-send   | -                 |
| incr | pool/b@send       | pool/b@pre-send   |
| incr | pool/b/1@clone2   | pool/a/1@clone    |
| incr | pool/b/1@pre-send | pool/b/1@clone2   |
| incr | pool/b/1@send     | pool/b/1@send     |
| incr | pool/b/2@pre-send | pool/b/1@clone2   |
| incr | pool/b/2@send     | pool/b/2@pre-send |

In the output, the pool/a/1@clone snapshot is not included in the replication stream package. Therefore, this replication stream package can only be received in a pool that already has pool/a/1@clone snapshot.

- **Recursive stream package** – Consists of the specified dataset and its descendants. Unlike replication stream packages, intermediate snapshots are not included unless they are the origin of a cloned dataset that is included in the stream. By default, if the origin of a dataset is not a descendant of the snapshot specified in the command, the behavior is similar to replication streams. Note that a self-contained recursive stream does not have external dependencies.

You create a recursive stream package with syntax similar to the following example:

```
zfs send -r pool/b@send ...
```

This example package would consist of the following full and incremental streams:

| TYPE | SNAPSHOT        | INCREMENTAL FROM |
|------|-----------------|------------------|
| full | pool/b@send     | -                |
| incr | pool/b/1@clone2 | pool/a/1@clone   |
| incr | pool/b/1@send   | pool/b/1@clone2  |
| incr | pool/b/2@send   | pool/b/1@clone2  |

In the output, the pool/a/1@clone snapshot is not included in the recursive stream package. Therefore, similar to the replication stream package, this recursive stream package can only

be received in a pool that already has pool/a/1@clone snapshot. This behavior is similar to the replication stream package scenario described above.

- **Self-contained recursive stream package** - This type of package is not dependent on any datasets that are not included in the stream package. You create a recursive stream package with syntax similar to the following example:

```
zfs send -rc pool/b@send ...
```

This example package would consist of the following full and incremental streams:

| TYPE | SNAPSHOT        | INCREMENTAL FROM |
|------|-----------------|------------------|
| full | pool/b@send     | -                |
| full | pool/b/1@clone2 |                  |
| incr | pool/b/1@send   | pool/b/1@clone2  |
| incr | pool/b/2@send   | pool/b/1@clone2  |

Notice that this self-contained recursive stream has a full stream of the pool/b/1@clone2 snapshot, making it possible receive the pool/b/1 snapshot with no external dependencies.

## Sending a ZFS Snapshot

You can use the `zfs send` command to send a copy of a snapshot stream and receive the snapshot stream in another pool on the same system or in another pool on a different system that is used to store backup data. For example, to send the snapshot stream on a different pool to the same system, use a command similar to the following example:

```
zfs send ool/dana@snap1 | zfs recv spool/ds01
```

---

**Tip** - You can use `zfs recv` as an alias for the `zfs receive` command.

---

If you are sending the snapshot stream to a different system, pipe the `zfs send` output through the `ssh` command. For example:

```
sys1# zfs send ool/dana@snap1 | ssh sys2 zfs recv pool/dana
```

When you send a full stream, the destination file system must not exist.

If you need to store many copies, consider compressing a ZFS snapshot stream representation with the `gzip` command. For example:

```
zfs send pool/fs@snap | gzip > backupfile.gz
```

### EXAMPLE 38 Sending Incremental ZFS Data

You can send incremental data by using the `zfs send -i` option. For example:

```
sys1# zfs send -i ool/dana@snap1 system1/dana@snap2 | ssh system2 zfs recv ool/dana
```

The first argument (*snap1*) is the earlier snapshot and the second argument (*snap2*) is the later snapshot. In this case, the *ool/dana* file system must already exist for the incremental receive to be successful.

You can specify the incremental *snap1* source as the last component of the snapshot name. You would then have to specify only the name after the @ sign for *snap1*, which is assumed to be from the same file system as *snap2*. For example:

```
sys1# zfs send -i snap1 ool/dana@snap2 | ssh system2 zfs recv ool/dana
```

This shortcut syntax is equivalent to the incremental syntax.

The following message is displayed if you attempt to generate an incremental stream from a different file system *snapshot1*:

```
cannot send 'pool/fs@name': not an earlier snapshot from the same fs
```

Accessing file information in the original received file system can cause the incremental snapshot receive operation to fail with a message similar to this one:

```
cannot receive incremental stream of pool/dana@snap2 into pool/dana:
most recent snapshot of pool/dana@snap2 does not match incremental source
```

Consider setting the *atime* property to *off* if you need to access file information in the original received file system and if you also need to receive incremental snapshots into the received file system.

If you need to store many copies, consider compressing a ZFS snapshot stream representation with the *gzip* command. For example:

```
zfs send pool/fs@snap | gzip > backupfile.gz
```

#### **EXAMPLE 39** Sending ZFS Data From a Oracle Solaris 12.0 Dataset

The ability to use per record checksums in the output data stream is enabled by default. To transfer data to older systems, you must disable this feature using the *nocheck* argument.

```
zfs send -s nocheck pool/dana@snap1 | zfs recv pool/ds01
```

## Using Resumable Replication

If the transfer of data using *zfs receive* is interrupted, you can restart the process. For instance, if you started the transfer with this command.

```
system1# zfs send pool/dana@snap1 | ssh system2 zfs recv pool/dana
```

If the transfer is interrupted, you will have an incomplete dataset. You can resume the transfer using this series of commands:

```
system1# ssh system2 zfs receive -C pool/dana | zfs send -C pool/dana@snap1 | \
ssh system2 zfs receive pool/dana
```

To see which datasets are incomplete use the `zfs list -I` command, see [“Listing Incomplete ZFS Datasets” on page 118](#).

## Receiving a ZFS Snapshot

Keep the following key points in mind when you receive a file system snapshot:

- Both the snapshot and the file system are received.
- The file system and all descendant file systems are unmounted.
- The file systems are inaccessible while they are being received.
- A file system with the same name as the source file system to be received must not exist on the target system. If the file system name exists on the target system, rename the file system.

For example:

```
zfs send system1/gozer@0830 > /bkups/gozer.083006
zfs receive system1/gozer2@today < /bkups/gozer.083006
zfs rename system1/gozer system1/gozer.old
zfs rename system1/gozer2 system1/gozer
```

If you make a change to the destination file system and you want to perform another incremental send of a snapshot, you must first roll back the receiving file system.

Consider the following example. First, make a change to the file system as follows:

```
sys2# rm newsys/dana/file.1
```

Then, perform an incremental send of `system1/dana@snap3`. Note that either you must first roll back the receiving file system to receive the new incremental snapshot or eliminate the rollback step by using the `-F` option. For example:

```
sys1# zfs send -i system1/dana@snap2 system1/dana@snap3 | ssh sys2 zfs recv -F newsys/
dana
```

When you receive an incremental snapshot, the destination file system must already exist.

If you make changes to the file system and you do not roll back the receiving file system to receive the new incremental snapshot or you use the `-F` option, a message similar to the following example is displayed:

```
sys1# zfs send -i system1/dana@snap4 system1/dana@snap5 | ssh sys2 zfs recv newsys/dana
```

```
cannot receive: destination has been modified since most recent snapshot
```

The following checks are performed before the `-F` option is successful:

- If the most recent snapshot doesn't match the incremental source, neither the rollback nor the receive is completed, and an error message is returned.
- If you accidentally provide the name of different file system that doesn't match the incremental source specified in the `zfs receive` command, neither the rollback nor the receive is completed, and the following error message is returned:

```
cannot send 'pool/fs@name': not an earlier snapshot from the same fs
```

## Applying Different Property Values to a ZFS Snapshot Stream

You can send a ZFS snapshot stream with a certain file system property value, and then either specify a different local property value when the snapshot stream is received or specify that the original property value be used when the snapshot stream is received to re-create the original file system. In addition, you can disable a file system property when the snapshot stream is received.

- To revert a local property value to the received value, if any, use the `zfs inherit -S` command. If a property does not have a received value, the behavior of the `-S` option is the same as if you did not include the option. If the property does have a received value, the `zfs inherit` command masks the received value with the inherited value until issuing a `zfs inherit -S` command reverts it to the received value.
- You can determine the columns displayed by the `zfs get`. Use the `-o` option to include the new non-default `RECEIVED` column. Use the `-o all` option to include all columns, including `RECEIVED`.
- To include properties in the send stream without the `-R` option, use the `-p` option.
- To use the last element of the sent snapshot name to determine the new snapshot name, use the `-e` option.

The following example sends the `poolA/bee/cee@1` snapshot to the `poolD/eee` file system and uses only the last element (`cee@1`) of the snapshot name to create the received file system and snapshot.

```
zfs list -rt all poolA
NAME USED AVAIL REFER MOUNTPOINT
poolA 134K 134G 23K /poolA
poolA/bee 44K 134G 23K /poolA/bee
poolA/bee/cee 21K 134G 21K /poolA/bee/cee
poolA/bee/cee@1 0 - 21K -
zfs send -R poolA/bee/cee@1 | zfs receive -e poolD/eee
```



```
zfs list -rt all poolD
```

| NAME            | USED | AVAIL | REFER | MOUNTPOINT     |
|-----------------|------|-------|-------|----------------|
| poolD           | 134K | 134G  | 23K   | /poolD         |
| poolD/eee       | 44K  | 134G  | 23K   | /poolD/eee     |
| poolD/eee/cee   | 21K  | 134G  | 21K   | /poolD/eee/cee |
| poolD/eee/cee@1 | 0    | -     | 21K   | -              |

## Retaining Original Property Values

In some cases, file system properties in a send stream might not apply to the receiving file system or local file system properties, such as the mountpoint property value, might interfere with a restore.

For example, suppose that the `system1/data` file system has the `compression` property disabled. A snapshot of the `system1/data` file system is sent with properties (`-p` option) to a backup pool and is received with the `compression` property enabled.

```
zfs get compression system1/data
```

| NAME         | PROPERTY    | VALUE | SOURCE  |
|--------------|-------------|-------|---------|
| system1/data | compression | off   | default |

```
zfs snapshot system1/data@snap1
zfs send -p system1/data@snap1 | zfs recv -o compression=on -d bpool
zfs get -o all compression bpool/data
```

| NAME       | PROPERTY    | VALUE | RECEIVED | SOURCE |
|------------|-------------|-------|----------|--------|
| bpool/data | compression | on    | off      | local  |

In the example, the `compression` property is enabled when the snapshot is received into `bpool`. So, for `bpool/data`, the `compression` value is `on`.

If this snapshot stream is sent to a new pool, `restorepool`, for recovery purposes, you might want to keep all the original snapshot properties. In this case, you would use the `zfs send -b` option to restore the original snapshot properties. For example:

```
zfs send -b bpool/data@snap1 | zfs recv -d restorepool
zfs get -o all compression restorepool/data
```

| NAME             | PROPERTY    | VALUE | RECEIVED | SOURCE   |
|------------------|-------------|-------|----------|----------|
| restorepool/data | compression | off   | off      | received |

In the example, the `compression` value is `off`, which represents the snapshot `compression` value from the original `system1/data` file system.

## Disabling Original Property Values

If you have a local file system property value in a snapshot stream and you want to disable the property when it is received, use the `zfs receive -x` option. For example, the following

command sends a recursive snapshot stream of home directory file systems with all file system properties reserved to a backup pool but without the quota property values:

```
zfs send -R system1/home@snap1 | zfs recv -x quota bpool/home
zfs get -r quota bpool/home
```

| NAME                  | PROPERTY | VALUE | SOURCE  |
|-----------------------|----------|-------|---------|
| bpool/home            | quota    | none  | local   |
| bpool/home@snap1      | quota    | -     | -       |
| bpool/home/lori       | quota    | none  | default |
| bpool/home/lori@snap1 | quota    | -     | -       |
| bpool/home/mark       | quota    | none  | default |
| bpool/home/mark@snap1 | quota    | -     | -       |

If the recursive snapshot was not received with the -x option, the quota property would be set in the received file systems.

```
zfs send -R system1/home@snap1 | zfs recv bpool/home
zfs get -r quota bpool/home
```

| NAME                  | PROPERTY | VALUE | SOURCE   |
|-----------------------|----------|-------|----------|
| bpool/home            | quota    | none  | received |
| bpool/home@snap1      | quota    | -     | -        |
| bpool/home/lori       | quota    | 10G   | received |
| bpool/home/lori@snap1 | quota    | -     | -        |
| bpool/home/mark       | quota    | 10G   | received |
| bpool/home/mark@snap1 | quota    | -     | -        |

## Sending and Receiving Complex ZFS Snapshot Streams

This section describes how to use the -I and -R options with the `zfs send` command to send and receive more complex snapshot streams.

Keep the following points in mind when sending and receiving complex ZFS snapshot streams:

- To send all incremental streams from one snapshot to a cumulative snapshot, use the -I option. You can also use this option to send an incremental stream from the original snapshot to create a clone. The original snapshot must already exist on the receiving side to accept the incremental stream.
- To send a replication stream of all descendant file systems, use the -R option. When the replication stream is received, all properties, snapshots, descendant file systems, and clones are preserved.
- Using the `zfs send -r` command or the `zfs send -R` command to send package streams without the -c option will omit the origin of clones in some circumstances. For more information, see [“Types of ZFS Snapshot Streams” on page 171](#).
- Use both options to send an incremental replication stream.
  - Changes to properties are preserved, as are snapshot and file system rename and destroy operations.

- If the `-F` option is not specified when receiving the replication stream, dataset `destroy` operations are ignored. Thus, if necessary, you can undo the receive operation and restore the file system to its previous state.
- When sending incremental streams, if `-I` is used, all snapshots between `snapA` and `snapD` are sent. If `-i` is used, only `snapD` (for all descendants) snapshots are sent.
- To receive any of these types of `zfs send` streams, the receiving system must be running a software version capable of sending them. The stream version is incremented.

**EXAMPLE 40** Sending and Receiving Complex ZFS Snapshot Streams

You can combine a group of incremental snapshots into one snapshot by using the `-I` option. For example:

```
zfs send -I pool/fs@snapA pool/fs@snapD > /snaps/fs@all-I
```

You would then remove the incremental `snapB`, `snapC`, and `snapD` snapshots.

```
zfs destroy pool/fs@snapB
zfs destroy pool/fs@snapC
zfs destroy pool/fs@snapD
```

To receive the combined snapshot, you would use the following command.

```
zfs receive -d -F pool/fs < /snaps/fs@all-I
zfs list
```

| NAME          | USED | AVAIL | REFER | MOUNTPOINT |
|---------------|------|-------|-------|------------|
| pool          | 428K | 16.5G | 20K   | /pool      |
| pool/fs       | 71K  | 16.5G | 21K   | /pool/fs   |
| pool/fs@snapA | 16K  | -     | 18.5K | -          |
| pool/fs@snapB | 17K  | -     | 20K   | -          |
| pool/fs@snapC | 17K  | -     | 20.5K | -          |
| pool/fs@snapD | 0    | -     | 21K   | -          |

You can also use the `-I` command to combine a snapshot and a clone snapshot to create a combined dataset. For example:

```
zfs create pool/fs
zfs snapshot pool/fs@snap1
zfs clone pool/fs@snap1 pool/clone
zfs snapshot pool/clone@snapA
zfs send -I pool/fs@snap1 pool/clone@snapA > /snaps/fsclonesnap-I
zfs destroy pool/clone@snapA
zfs destroy pool/clone
zfs receive -F pool/clone < /snaps/fsclonesnap-I
```

To replicate a ZFS file system and all descendant file systems up to the named snapshot, use the `-R` option. When this stream is received, all properties, snapshots, descendant file systems, and clones are preserved.

The following example creates snapshots for user file systems. One replication stream is created for all user snapshots. Next, the original file systems and snapshots are destroyed and then recovered.

```
zfs snapshot -r users@today
zfs list
NAME USED AVAIL REFER MOUNTPOINT
users 187K 33.2G 22K /users
users@today 0 - 22K -
users/user1 18K 33.2G 18K /users/user1
users/user1@today 0 - 18K -
users/user2 18K 33.2G 18K /users/user2
users/user2@today 0 - 18K -
users/user3 18K 33.2G 18K /users/user3
users/user3@today 0 - 18K -
zfs send -R users@today > /snaps/users-R
zfs destroy -r users
zfs receive -F -d users < /snaps/users-R
zfs list
NAME USED AVAIL REFER MOUNTPOINT
users 196K 33.2G 22K /users
users@today 0 - 22K -
users/user1 18K 33.2G 18K /users/user1
users/user1@today 0 - 18K -
users/user2 18K 33.2G 18K /users/user2
users/user2@today 0 - 18K -
users/user3 18K 33.2G 18K /users/user3
users/user3@today 0 - 18K -
```

The following example uses the -R command to replicate the users file system and its descendants, and to send the replicated stream to another pool, users2.

```
zfs create users2 mirror c0t1d0 c1t1d0
zfs receive -F -d users2 < /snaps/users-R
zfs list
NAME USED AVAIL REFER MOUNTPOINT
users 224K 33.2G 22K /users
users@today 0 - 22K -
users/user1 33K 33.2G 18K /users/user1
users/user1@today 15K - 18K -
users/user2 18K 33.2G 18K /users/user2
users/user2@today 0 - 18K -
users/user3 18K 33.2G 18K /users/user3
users/user3@today 0 - 18K -
users2 188K 16.5G 22K /users2
users2@today 0 - 22K -
users2/user1 18K 16.5G 18K /users2/user1
users2/user1@today 0 - 18K -
users2/user2 18K 16.5G 18K /users2/user2
users2/user2@today 0 - 18K -
users2/user3 18K 16.5G 18K /users2/user3
```

```
users2/user3@today 0 - 18K -
```

## Remote Replication of ZFS Data

You can use the `zfs send` and `zfs recv` commands to remotely copy a snapshot stream representation from one system to another system. For example:

```
zfs send system1/cindy@today | ssh newsys zfs recv sandbox/restfs@today
```

This command sends the `system1/cindy@today` snapshot data and receives it into the `sandbox/restfs` file system. The command also creates a `restfs@today` snapshot on the `newsys` system. In this example, the user has been configured to use `ssh` on the remote system.

## Monitoring ZFS Pool Operations

When you issue ZFS commands that initiate background tasks to run on the data such as sending, receiving, scrubbing, or resilvering data, you can monitor the status and progress of these tasks in real time. You can specify the frequency rate in which information is displayed. You can also determine for how long the monitoring should run.

To monitor pool operations, you use the `zpool monitor` command. Depending on which options you use, the command provides the following information about the task. The information is provided for each individual pool.

- Start time.
- Current amount of data.
- Timestamp, if appropriate on a per feature basis.
- Amount of data at start of the task, if appropriate.

You can display information about tasks on an individual pool or on all existing pools on the system.

Use the `zpool monitor` command as follows:

```
zpool monitor -t provider [-T d|u] [pool] [interval [count]]
```

**-t *provider*** Specifies one of the following providers about which the task information is displayed. For *provider*, you can specify one of the following:

- `send`
- `receive` or `recv`
- `scrub`

---

■ `resilver`

---

**Note** - For an updated list of providers, type the `zpool help monitor` command.

---

|                     |                                                                                                                                                                                                                                                                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-T d u</code> | Specifies the time stamp and its display format. To display in standard date format, specify <code>d</code> . To display a printed representation of the internal representation of time, specify <code>u</code> . For more information about these display formats, see the <a href="#">date(1)</a> and the <a href="#">time(2)</a> man pages.  |
| <i>interval</i>     | Rate, in seconds, at which the displayed information is updated.                                                                                                                                                                                                                                                                                 |
| <i>count</i>        | Number of times the command displays task-related information within the specified interval.<br><br>If <i>count</i> is specified, then the command updates the information for as many times as specified by <i>count</i> before exiting. If <i>count</i> is not specified, then the information is updated continuously until you press Ctrl-C. |

---

**Note** - You can also customize the information to display. With the `-o` option, you can filter the display fields to be included in the command output. With the `-p` option, you can display information in machine-parsable format. For more information, see the [zpool\(1M\)](#) man page.

---

The command output arranges the information according to specific fields:

|          |                                                                                                                                                                                                                                                          |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DONE     | Amount of data that has been processed so far since the <code>zpool monitor</code> command was issued.                                                                                                                                                   |
| OTHER    | Additional information depending on the specified provider, such as the current item being processed, or the current state of the task.                                                                                                                  |
| PCTDONE  | Percentage of data that has been processed.                                                                                                                                                                                                              |
| POOL     | Pool from which the information is retrieved.                                                                                                                                                                                                            |
| PROVIDER | Task that is providing the information.                                                                                                                                                                                                                  |
| SPEED    | Units per second, usually bytes but dependent on the unit that the provider uses.                                                                                                                                                                        |
| STRTTIME | Time the provider started on the displayed task.                                                                                                                                                                                                         |
| TAG      | Distinguishes whole operations. The TAG value is unique at any one time but values can be reused in subsequent operations. For example, two simultaneous send operations would have different TAG values even if both tasks operate on the same dataset. |

|           |                                                       |
|-----------|-------------------------------------------------------|
| TIMELEFT  | Relative time that a specific task will be completed. |
| TIMESTAMP | Time the monitored data snapshot is taken.            |
| TOTAL     | Estimate of total amount of data to be processed.     |

**EXAMPLE 41** Monitoring a ZFS Snapshot Stream Representation

The `zfs send` command creates a stream representation of a snapshot. The following example shows how to obtain information about this task. The information would be updated two times in a 5-second interval.

```
zpool monitor -t send 5 2
```

| POOL  | PROVIDER | PCTDONE | TOTAL | SPEED | TIMELEFT | OTHER                |
|-------|----------|---------|-------|-------|----------|----------------------|
| poolA | send     | 41.7    | 10.0G | 5.93M | 16m49s   | poolA/fs:1/team3@all |
| poolB | send     | 53.9    | 10.0G | 7.71M | 10m13s   | poolB/fs1/team3@all  |
| poolC | send     | 97.9    | 10.0G | 14.4M | 14s      | poolC/fs1/team1@all  |
| poolA | send     | 43.5    | 10.0G | 6.17M | 15m40s   | poolA/fs:1/team3@all |
| poolB | send     | 55.8    | 10.0G | 7.95M | 9m30s    | poolB/fs1/team3@all  |
| poolC | send     | 99.2    | 10.0G | 14.5M | 5s       | poolC/fs1/team1@all  |

**EXAMPLE 42** Monitoring the Reception of a Stream

This example shows how to monitor the status and progress of a receive operation. Without a designated count, the information is continuously updated every 5 seconds. The monitoring ends when the administrator presses Ctrl-C.

```
zpool monitor -t recv 5
```

| pool  | provider | pctdone | total | speed | timeleft | other                |
|-------|----------|---------|-------|-------|----------|----------------------|
| poolA | receive  | 34.0    | 12.0G | 6.01M | 22m31s   | poolA/backup_all/fs2 |
| poolA | receive  | 68.0    | 6.01G | 6.01M | 5m28s    | poolA/backup_fs:1    |
| poolB | receive  | 20.6    | 10.0G | 3.04M | 44m39s   | poolB/backup_all/fs2 |
| poolB | receive  | 100.0   | 6.01G | 9.48M | 1s       | poolB/backup_fs1     |
| poolB | receive  | 16.7    | 12.0G | 4.16M | 41m04s   | poolB/pA-bkup/fs2    |
| poolC | receive  | 26.2    | 10.0G | 3.98M | 31m41s   | poolC/backup_all/fs2 |

^C

**EXAMPLE 43** Monitoring a Resilvering Operation

This example shows how to check the status of a resilvering operation on all the three ZFS pools on the system.

```
zpool monitor -t resilver 5 4
```

| pool  | provider | pctdone | total | speed | timeleft | other |
|-------|----------|---------|-------|-------|----------|-------|
| poolA | resilver | 10.7    | 12.0G | 37.7M | 4m50s    | (2/2) |
| poolB | resilver | 7.1     | 10.0G | 31.5M | 5m02s    | (2/2) |
| poolC | resilver | 1.8     | 10.0G | 42.9M | 3m54s    | (2/2) |

|       |          |      |       |       |       |       |
|-------|----------|------|-------|-------|-------|-------|
| poolA | resilver | 13.9 | 12.0G | 38.0M | 4m38s | (2/2) |
| poolB | resilver | 9.0  | 10.0G | 30.4M | 5m07s | (2/2) |
| poolC | resilver | 5.3  | 10.0G | 41.7M | 3m52s | (2/2) |
| poolA | resilver | 14.7 | 12.0G | 36.1M | 4m50s | (2/2) |
| poolB | resilver | 10.8 | 10.0G | 29.2M | 5m12s | (2/2) |
| poolC | resilver | 7.2  | 10.0G | 41.0M | 3m51s | (2/2) |
| poolA | resilver | 14.7 | 12.0G | 32.8M | 5m19s | (2/2) |
| poolB | resilver | 10.8 | 10.0G | 29.6M | 5m08s | (2/2) |
| poolC | resilver | 7.2  | 10.0G | 40.7M | 3m53s | (2/2) |

**EXAMPLE 44** Monitoring a Scrubbing Operation

This example shows how to monitor the progress of a scrubbing operation on poolB.

```
zpool monitor -t scrub 5 poolB
pool provider pctdone total speed timeleft
poolB scrub 16.3 14.0G 35.3M 5m39s
poolB scrub 16.4 14.0G 33.2M 6m00s
poolB scrub 16.5 14.0G 31.1M 6m25s
poolB scrub 16.5 14.0G 29.3M 6m48s
^C
```

## Copying ZFS Files

You can quickly copy large files using the `cp -z` command. The `-z` option copies the metadata associated with each record, rather copying the metadata and all of the data. For files with record sizes of 4K or more, this method can be much faster than using the standard `cp` command. It is also very useful if you need to make many copies of one large file, like an OS image. See the `reflink(3c)` man page for more information.



## Using ACLs and Attributes to Protect Oracle Solaris ZFS Files

---

This chapter provides information about using access control lists (ACLs) to protect your ZFS files by providing more granular permissions than the standard UNIX permissions.

This chapter covers the following topics:

- [“Oracle Solaris ACL Model” on page 185.](#)
- [“Setting ACLs on ZFS Files” on page 191.](#)
- [“Command Syntax for Setting ACLs” on page 191.](#)
- [“Setting ACL Inheritance on ZFS Files” on page 197.](#)
- [“Applying Special Attributes to ZFS Files” on page 204.](#)

### Oracle Solaris ACL Model

The Oracle Solaris ACL model fully supports the interoperability that NFSv4 offers between UNIX and non-UNIX clients. It is similar to Windows NT-style ACLs, and provides more fine-grained access control than is available with standard file permissions. ACLs are set and displayed with the `chmod` and `ls` commands.

The ACL model has two types of Access Control Entries (ACEs) that affect access checking: `ALLOW` and `DENY`. Therefore, you cannot infer from any single ACE that defines a set of permissions whether the permissions that are not defined in that ACE are allowed or denied.

For information about ACLs and backup products, see [“Saving ZFS Data With Other Backup Products” on page 171.](#)

### ACL Formats

ACLs have two basic formats:

- **Trivial ACL** – Contains only entries for traditional UNIX user categories that are represented as owner@, group@, and everyone@.

For a newly created file, the default ACL has the following entries:

```
0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow
```

For a newly created directory, the default ACL has the following entries:

```
0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
2:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

- **Non-Trivial ACL** – Contains entries for added user categories. The entries might also include inheritance flags, or are ordered in a non-traditional way.

A non-trivial entry might look like the following example, where permissions are specifically granted to user Bob.

```
0:user:bob:read_data/write_data:file_inherit:allow
```

## ACL Entry Descriptions

Use the following sample entry as a reference to understand the elements that comprise an ACL entry. These elements apply to both trivial and non-trivial ACLs.

```
0:user:bob:read_data/write_data:file_inherit:allow
```

|                   |                                                                                                                                                                                                           |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Index             | A number at the beginning of the entry, such as the number zero (0) in the example. The index identifies a specific entry and distinguishes the entry from others in the ACL.                             |
| ACL entry type    | The user category. In trivial ACLs, only entries for owner@, group@, and everyone@ are set. In non-trivial ACLs, user:username and group:groupname are added. In the example, the entry type is user:bob. |
| Access privileges | Permissions that are granted or denied to the entry type. In the example, user Bob's permissions are read_data and write_data.                                                                            |

|                    |                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Inheritance flags  | An optional list of ACL flags that control how permissions are propagated in a directory structure. In the sample entry, <code>file_inherit</code> is also granted to user Bob. |
| Permission control | Determines whether the permissions in an entry are allowed or denied. In the example, the permissions for Bob are allowed.                                                      |

The following table describes each ACL entry type.

**TABLE 7** ACL Entry Types

| ACL Entry Type         | Description                                                                                                                                                                                                                                         |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>owner@</code>    | Specifies the access granted to the owner of the object.                                                                                                                                                                                            |
| <code>group@</code>    | Specifies the access granted to the owning group of the object.                                                                                                                                                                                     |
| <code>everyone@</code> | Specifies the access granted to any user or group that does not match any other ACL entry.                                                                                                                                                          |
| <code>user</code>      | With a user name, specifies the access granted to an additional user of the object. Must include the ACL entry ID, which contains a user name or user ID. If the value is not a valid numeric UID or user name, the ACL entry type is invalid.      |
| <code>group</code>     | With a group name, specifies the access granted to an additional group of the object. Must include the ACL entry ID, which contains a group name or group ID. If the value is not a valid numeric GID or group name, the ACL entry type is invalid. |

The following table describes ACL access privileges.

**TABLE 8** ACL Access Privileges

| Access Privilege              | Compact Access Privilege | Description                                                                                                                                                                                                                                   |
|-------------------------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>add_file</code>         | <code>w</code>           | Permission to add a new file to a directory.                                                                                                                                                                                                  |
| <code>add_subdirectory</code> | <code>p</code>           | On a directory, permission to create a subdirectory.                                                                                                                                                                                          |
| <code>append_data</code>      | <code>p</code>           | Permission to modify a file but only beginning from the end-of-file (EOF).                                                                                                                                                                    |
| <code>delete</code>           | <code>d</code>           | Permission to delete a file. For more information about specific <code>delete</code> permission behavior, see <a href="#">Table 9, “ACL delete and delete_child Permission Behavior,” on page 188</a> .                                       |
| <code>delete_child</code>     | <code>D</code>           | Permission to delete a file or directory within a directory. For more information about specific <code>delete_child</code> permission behavior, see <a href="#">Table 9, “ACL delete and delete_child Permission Behavior,” on page 188</a> . |
| <code>execute</code>          | <code>x</code>           | Permission to execute a file or search the contents of a directory.                                                                                                                                                                           |
| <code>list_directory</code>   | <code>r</code>           | Permission to list the contents of a directory.                                                                                                                                                                                               |
| <code>read_acl</code>         | <code>c</code>           | Permission to read the ACL (ls).                                                                                                                                                                                                              |
| <code>read_attributes</code>  | <code>a</code>           | Permission to read basic attributes (non-ACLs) of a file, which are equivalent to <code>stat</code> level attributes. Allowing this access mask bit means the entity can execute <code>ls(1)</code> and <code>stat(2)</code> .                |

| Access Privilege | Compact Access Privilege | Description                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| read_data        | r                        | Permission to read the contents of the file.                                                                                                                                                                                                                                                                                                                                                                                                |
| read_xattr       | R                        | Permission to read the extended attributes of a file or perform a lookup in the file's extended attributes directory.                                                                                                                                                                                                                                                                                                                       |
| synchronize      | s                        | Permission to access a file locally at the ZFS server with synchronized read and write operations.                                                                                                                                                                                                                                                                                                                                          |
| write_xattr      | W                        | Permission to create extended attributes or write to the extended attributes directory.<br><br>Granting this permission to a user means that the user can create an extended attribute directory for a file. The attribute file's permissions control the user's access to the attribute.                                                                                                                                                   |
| write_data       | w                        | Permission to modify or replace the contents of a file.                                                                                                                                                                                                                                                                                                                                                                                     |
| write_attributes | A                        | Permission to change the times associated with a file or directory to an arbitrary value.                                                                                                                                                                                                                                                                                                                                                   |
| write_acl        | C                        | Permission to write the ACL or the ability to modify the ACL by using the <code>chmod</code> command.                                                                                                                                                                                                                                                                                                                                       |
| write_owner      | o                        | Permission to change the file's owner or group. Or, the ability to execute the <code>chown</code> or <code>chgrp</code> commands on the file.<br><br>Permission to take ownership of a file or permission to change the group ownership of the file to a group of which the user is a member. If you want to change the file or group ownership to an arbitrary user or group, then the <code>PRIV_FILE_CHOWN</code> privilege is required. |

The following table provides additional details about ACL `delete` and `delete_child` behavior.

**TABLE 9** ACL delete and delete\_child Permission Behavior

| Parent Directory Permissions      | Target Object Permissions |                   |                               |
|-----------------------------------|---------------------------|-------------------|-------------------------------|
| " " (empty)                       | ACL allows delete         | ACL denies delete | Delete permission unspecified |
| ACL allows delete_child           | Permit                    | Permit            | Permit                        |
| ACL denies delete_child           | Permit                    | Deny              | Deny                          |
| ACL allows only write and execute | Permit                    | Permit            | Permit                        |
| ACL denies write and execute      | Permit                    | Deny              | Deny                          |

## ZFS ACL Sets

An ACL set consists of a combination of ACL permissions. These ACL sets are predefined and cannot be modified.

| ACL Set Name | Included ACL Permissions |
|--------------|--------------------------|
| full_set     | All permissions          |

| ACL Set Name | Included ACL Permissions                                   |
|--------------|------------------------------------------------------------|
| modify_set   | All permissions except write_acl and write_owner           |
| read_set     | read_data, read_attributes, read_xattr, and read_acl       |
| write_set    | write_data, append_data, write_attributes, and write_xattr |

You can apply an ACL set rather than having to set individual permissions separately. For example, granting Bob the read\_set ACL set gives him permissions to read ACLs as well as file contents and their basic and extended attributes.

```
chmod A+user:bob:read_set:allow file.1
ls -v file.1
-r--r--r--+ 1 root root 206695 Jul 20 13:43 file.1
0:user:bob:read_data/read_xattr/read_attributes/read_acl:allow
...
```

## ACL Inheritance

ACL inheritance means that a newly created file or directory can inherit the ACLs they are intended to inherit without disregarding the existing permission bits on the parent directory.

By default, ACLs are not propagated. If you set a non-trivial ACL on a directory, it is not inherited to any subsequent directory. You must specify the inheritance of an ACL on a file or directory.

**Note** - Currently, the successful\_access, failed\_access, and inherited flags apply only to the SMB server.

The following table describes the optional inheritance flags.

**TABLE 10** ACL Inheritance Flags

| Inheritance Flag | Compact Inheritance Flag | Description                                                                                                                                                                                                                                     |
|------------------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file_inherit     | f                        | Only inherit the ACL from the parent directory to the directory's files.                                                                                                                                                                        |
| dir_inherit      | d                        | Only inherit the ACL from the parent directory to the directory's subdirectories.                                                                                                                                                               |
| inherit_only     | i                        | Inherit the ACL from the parent directory. Applies only to newly created files or subdirectories and not the directory itself. This flag requires the file_inherit flag, the dir_inherit flag, or both, to indicate what to inherit.            |
| no_propagate     | n                        | Only inherit the ACL from the parent directory to the first-level contents of the directory, not the second-level or subsequent contents. This flag requires the file_inherit flag, the dir_inherit flag, or both, to indicate what to inherit. |

| Inheritance Flag  | Compact Inheritance Flag | Description                                                                                                                               |
|-------------------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| -                 | N/A                      | No permission granted.                                                                                                                    |
| successful_access | S                        | Indicates whether an alarm or audit record should be initiated upon a successful access. This flag is used with audit or alarm ACE types. |
| failed_access     | F                        | Indicates whether an alarm or audit record should be initiated when an access fails. This flag is used with audit or alarm ACE types.     |
| inherited         | I                        | Indicates that an ACE was inherited.                                                                                                      |

In addition, you can set a default ACL inheritance policy on the file system that is more strict or less strict by using the `aclinherit` file system property. For more information about this property, see [“ACL Properties” on page 190](#).

For more information about setting ACL inheritance on ZFS files, see [“Setting ACL Inheritance on ZFS Files” on page 197](#).

## ACL Properties

The ZFS file system includes the ACL properties to determine the specific behavior of ACL inheritance and ACL interaction with `chmod` operations. These properties are:

- `aclinherit` – Determine the behavior of ACL inheritance. Values include the following:
  - `discard` – For new objects, no ACL entries are inherited when a file or directory is created. The ACL on the file or directory is equal to the permission mode of the file or directory.
  - `noallow` – For new objects, only inheritable ACL entries that have an access type of `deny` are inherited.
  - `restricted` – For new objects, the `write_owner` and `write_acl` permissions are removed when an ACL entry is inherited.
  - `passthrough` – When a property value is set to `passthrough`, files are created with a mode determined by the inheritable ACEs. If no inheritable ACEs exist that affect the mode, then the mode is set in accordance to the requested mode from the application.
  - `passthrough-x` – Has the same semantics as `passthrough` except that when `passthrough-x` is enabled, files are created with the execute (`x`) permission only if the execute permission is set in file creation mode and in an inheritable ACE that affects the mode.

The default mode for the `aclinherit` is `restricted`.

For more information about the `aclinherit` modes, see [“Modifying ACL Inheritance With the ACL Inherit Mode” on page 200](#).

- `aclmode` – Modifies ACL behavior when a file is initially created or controls how an ACL is modified during a `chmod` operation. Values include the following:
  - `discard` – Deletes all ACL entries that do not represent the mode of the file. This is the default value.
  - `mask` – Reduces user or group permissions. The permissions are reduced such that they are no greater than the group permission bits unless it is a user entry that has the same UID as the owner of the file or directory. In this case, the ACL permissions are reduced so that they are no greater than owner permission bits. The mask value also preserves the ACL across mode changes, provided that an explicit ACL set operation has not been performed.
  - `passthrough` – Indicates that no changes are made to the ACL other than generating the necessary ACL entries to represent the new mode of the file or directory.

The default mode for the `aclmode` is `discard`.

For more information about using the `aclmode` property, see [Example 46, “ACL Properties and Modified ACL Permissions,” on page 196](#).

## Setting ACLs on ZFS Files

The primary rules of ACL access on a ZFS file are as follows:

- ZFS processes ACL entries in the order they are listed in the ACL, from the top down.
- Only ACL entries whose specified user matches the requester of the access are processed.
- Once an allow permission has been granted, it cannot be denied by a subsequent ACL deny entry in the same ACL permission set.
- The owner of the file is granted the `write_acl` permission unconditionally even if the permission is explicitly denied. Otherwise, any permission left unspecified is denied.

In the cases of deny permissions or when an access permission is missing, the privilege subsystem determines the access request that is granted for the owner of the file or for superuser. This mechanism prevents owners of files from getting locked out of their files and enables superuser to modify files for recovery purposes.

## Command Syntax for Setting ACLs

To set or modify ACLs, use the `chmod` command. The command syntax resembles the syntax for setting permission bits on files, except that you specify *A* before typing the operator (+, =, or -).

- Command syntax for trivial ACLs

```
chmod [options] A[index]{+|=}owner@ |group@ |everyone@: \
```

```
access-permissions/...[:inheritance-flags]:deny | allow file
```

```
chmod [options] A-owner@, group@, everyone@: \
access-permissions/...[:inheritance-flags]:deny | allow file ...
```

```
chmod [options] A[index]- file
```

- Command syntax for non-trivial ACLs

```
chmod [options] A[index]{+|=}user|group:name: \
access-permissions/...[:inheritance-flags]:deny | allow file
```

```
chmod [options] A-user|group:name: \
access-permissions/...[:inheritance-flags]:deny | allow file ...
```

```
chmod [options] A[index]- file
```

The `chmod` command uses the following operators:

- `A+` adds an ACL entry.
- `A=` replaces an ACL entry.

To replace an entire ACL for a file, use this operator without specifying an index ID. In the following example, ACL entries for `file.1` are removed and replaced with the single entry for `everyone@`.

```
% chmod A=everyone@:read_data:allow file.1
```

- `A-` removes an ACL entry.

To universally remove all non-trivial ACL entries for a file, use this operator and specify the file name without listing each entry to be removed.

```
% chmod A- filename
```

Use this command syntax to restore a trivial ACL to the file. After you issue the command, only the entries for `owner@`, `group@`, and `everyone@` that comprise a trivial ACL remain.



---

**Caution** - Be careful with modifying existing ACLs. Using the operators without an index has a different effect from using them with an index. For example, `chmod A=` replaces an entire ACL, while `chmod A3=` replaces only the existing entry that has index number 3.

---

Permissions and inheritance flags are represented by unique letters listed in [Table 8, “ACL Access Privileges,” on page 187](#) and [Table 10, “ACL Inheritance Flags,” on page 189](#).

When you set ZFS ACLs, you can either use the letters that correspond to those permissions (compact mode) or type the permissions in full (verbose mode).

In this example, both commands grant read and execute permissions to user Joe on `file.1`:

- `chmod A+user:joe:rx:allow file.1`
- `chmod A+user:joe:read_data/execute:allow file.1`



Likewise, to grant user John inheritable read, write, and execute permissions for the newly created `dir.2` and its files, you can use either one of the following commands:

- `chmod A+user:john:rxw:fd:allow dir.2`
- `chmod A+user:john:read_data/write_data/execute:file_inherit/dir_inherit:allow dir.2`

## Displaying ACL Information

With the `ls` command, you can display ACL information in one of two formats. The `-v` option displays the permissions in full or verbose form. The `-V` option generates compact output by using letters that represent the permissions and flags.

The following example shows how the same ACL information is displayed in both verbose and compact format:

```
ls -v file.1
-rw-r--r-- 1 root root 206695 Jul 20 14:27 file.1
0:owner@:read_data/write_data/append_data/read_attributes
/write_xattr/read_xattr/write_attributes/read_acl/write_acl
/write_owner/synchronize:allow
1:group@:read_data/read_attributes/read_xattr/read_acl
/synchronize:allow
2:everyone@:read_data/append_data/read_xattr/read_acl
/synchronize:allow

ls -V file.1
-rw-r--r-- 1 root root 206695 Jul 20 14:27 file.1
owner@:rw-p--aARWcCos:-----:allow
group@:r-----a-R-c--s:-----:allow
everyone@:r-----a-R-c--s:-----:allow
```

For an explanation of the permissions for each user category, see [Table 8, “ACL Access Privileges,”](#) on page 187.

## Modifying ACLs on ZFS Files

This section provides sample commands for setting and displaying ACLs.

In the following example, `write_data` permissions are granted for `group@`. The index of `group@` is 1.

```
chmod A1=group@:read_data/write_data:allow file.1
ls -v file.1
```

```
-rw-rw-r-- 1 root root 206695 Jul 20 13:43 file.1
0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:read_data/write_data:allow
2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow
```

In the following example, `read_data/execute` permissions are added for the user Joe on the `test.dir` directory.

```
chmod A0+user:joe:read_data/execute:allow test.dir
ls -dv test.dir
drwxr-xr-x+ 2 root root 2 Jul 20 14:23 test.dir
0:user:joe:list_directory/read_data/execute:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

In the following example, access permissions are removed for user Joe.

```
chmod A0- test.dir
ls -dv test.dir
drwxr-xr-x 2 root root 2 Jul 20 14:23 test.dir
0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
2:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

## ACL Interaction With Permission Bits

In ZFS files, the UNIX permission bits correspond to the ACL entries but are stored in cache. When you change a file's permission bits, the file's ACL is updated accordingly. Likewise, modifying a file's ACL causes changes in the permission bits.

For more information about permission bits, see [chmod\(1\)](#).

The following examples show the relationship between a file or directory's ACLs and the permission bits and how permission changes in one affect the other.

**EXAMPLE 45** ACLs and Permission Bits

The first example begins with the following ACL for file.2, whose permission bits are set to 644.

```
ls -v file.2
-rw-r--r-- 1 root root 2693 Jul 20 14:26 file.2 Permission bits are 644.
0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow
```

The chmod command removes the ACL entry for everyone@. Accordingly, the read permission bits for everyone are also removed and are changed to 640.

```
chmod A2- file.2 Access is removed for everyone@
ls -v file.2
-rw-r----- 1 root root 2693 Jul 20 14:26 file.2 Permission bits become 640.
0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
Entry for everyone@ is removed.
```

Next, the ACL is replaced with just read\_data/write\_data permissions for everyone@. No owner@ or group@ ACL entry exists to override the permissions for owner and group. Consequently, the permission bits become 666.

```
chmod A=everyone@:rw:allow file.2
ls -v file.2
-rw-rw-rw- 1 root root 2440 Jul 20 14:28 file.3 Permission bits become 666.
0:everyone@:read_data/write_data:allow
All other ACL entries removed.
```

Next, the ACL is replaced with read permissions just for user Joe. The command, however, leaves no trivial ACL entries. Consequently, the permission bits are set to 000, which denies Joe access to file.2. The file effectively becomes inaccessible.

```
chmod A=user:joe:r:allow file.2
ls -v file.2
-----+ 1 root root 2440 Jul 20 14:28 file.3 Permission bits become 000.
0:user:joe:read_data:allow
No trivial ACL entries exist.
```

The example ends with showing how setting permission bits also update the ACL. The bits for file.2 are set to 655. Automatically, default trivial ACL permissions are set.

```
chmod 655 file.2
ls -v file.3
```

```
-rw-r-xr-x 1 root root 2440 Jul 20 14:28 file.3 Permission bits set to 655.
0:user:joe:read_data:allow
1:owner@:execute:deny Based on 655 bits, ACL execute permission is denied.
2:owner@:read_data/write_data/append_data/read_xattr/write_xattr Default ACL entries
restored.
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
3:group@:read_data/read_xattr/execute/read_attributes/read_acl
/synchronize:allow
4:everyone@:read_data/read_xattr/execute/read_attributes/read_acl
/synchronize:allow
```

**EXAMPLE 46** ACL Properties and Modified ACL Permissions

The following examples illustrate how specific `aclmode` and `aclinherit` property values affect ACL behavior. If these properties are set, ACL permissions for a file or directory are either reduced or expanded to be consistent with the owning group.

Suppose that the `aclmode` property is set to `mask` and the `aclinherit` property is set to `restricted` in the pool, and that the original file and group ownership and ACL permissions are as follows:

```
zfs set aclmode=mask system1/data
zfs set aclinherit=restricted system1/data

ls -lV file.1
-rwxrwx---+ 1 root root 206695 Aug 30 16:03 file.1
user:amy:r-----a-R-c---:-----:allow
user:rory:r-----a-R-c---:-----:allow
group:sysadmin:rw-p--aARWc---:-----:allow
group:staff:rw-p--aARWc---:-----:allow
owner@:rwxp--aARWcCos:-----:allow
group@:rwxp--aARWc--s:-----:allow
everyone@:-----a-R-c--s:-----:allow
```

To understand the meaning of the values set for the two properties, see [“ACL Properties” on page 190](#).

A `chown` operation changes `file.1`'s ownership to Amy and the group Staff.

```
chown amy:staff file.1
```

Amy then changes `file.1`'s permission bits to 640. Because of the ACL properties that were previously set, the permissions for the groups in the ACL are reduced in order to not exceed the permissions of the owning Staff.

```
su - amy
$ chmod 640 file.1
$ ls -lV file.1
-rw-r-----+ 1 amy staff 206695 Aug 30 16:03 file.1
```

```

user:amy:r-----a-R-c---:-----:allow
user:rory:r-----a-R-c---:-----:allow
group:sysadmin:r-----a-R-c---:-----:allow
group:staff:r-----a-R-c---:-----:allow
owner@:rw-p--aARWcCos:-----:allow
group@:r-----a-R-c--s:-----:allow
everyone@:-----a-R-c--s:-----:allow

```

Amy then changes the permission bits to 770. Consequently, the permissions of the groups in the ACL are also changed to match the permission of the owning group Staff.

```

$ chmod 770 file.1
$ ls -lv file.1
-rwxrwx---+ 1 amy staff 206695 Aug 30 16:03 file.1
user:amy:r-----a-R-c---:-----:allow
user:rory:r-----a-R-c---:-----:allow
group:sysadmin:rw-p--aARWc---:-----:allow
group:staff:rw-p--aARWc---:-----:allow
owner@:rwxp--aARWcCos:-----:allow
group@:rwxp--aARWc--s:-----:allow
everyone@:-----a-R-c--s:-----:allow

```

## Setting ACL Inheritance on ZFS Files

You can determine how ACLs are inherited on files and directories.

The `aclinherit` property can be set globally on a file system. By default, `aclinherit` is set to `restricted`.

For more information, see [“ACL Inheritance” on page 189](#).

## Granting ACLs That Are Inherited by Files

This section identifies the file ACEs that are applied when the `file_inherit` flag is set.

In the following example, `read_data/write_data` permissions are added for files in the `test2.dir` directory for user `joe` so that he has read access on any newly created files.

```

chmod A+user:joe:read_data/write_data:file_inherit:allow test2.dir
ls -dv test2.dir
drwxr-xr-x+ 2 root root 2 Jul 20 14:55 test2.dir
0:user:joe:read_data/write_data:file_inherit:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow

```

```
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

In the following example, user joe's permissions are applied on the newly created test2.dir/file.2 file. The ACL inheritance granted, read\_data:file\_inherit:allow, means user joe can read the contents of any newly created file.

```
touch test2.dir/file.2
ls -lv test2.dir/file.2
-rw-r--r--+ 1 root root 0 Jul 20 14:56 test2.dir/file.2
0:user:joe:read_data:inherited:allow
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow
```

Because the aclinherit property for this file system is set to the default mode, restricted, user joe does not have write\_data permission on file.2 because the group permission of the file does not allow it.

The inherit\_only permission, which is applied when the file\_inherit or dir\_inherit flags are set, is used to propagate the ACL through the directory structure. As such, user joe is granted or denied permission from everyone@ permissions only if he is the file owner or is a member of the file's group owner. For example:

```
mkdir test2.dir/subdir.2
ls -dv test2.dir/subdir.2
drwxr-xr-x+ 2 root root 2 Jul 20 14:57 test2.dir/subdir.2
0:user:joe:list_directory/read_data/add_file/write_data:file_inherit
/inherit_only/inherited:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

## Granting ACLs That Are Inherited by Both Files and Directories

This section provides examples that identify the file and directory ACLs that are applied when both the file\_inherit and dir\_inherit flags are set.

In the following example, user joe is granted read, write, and execute permissions that are inherited for newly created files and directories.

```
chmod A+user:joe:read_data/write_data/execute:file_inherit/dir_inherit:allow
test3.dir
ls -dv test3.dir
drwxr-xr-x+ 2 root root 2 Jul 20 15:00 test3.dir
0:user:joe:list_directory/read_data/add_file/write_data/execute
:file_inherit/dir_inherit:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

The inherited text in the following output is an informational message that indicates that the ACE is inherited.

```
touch test3.dir/file.3
ls -v test3.dir/file.3
-rw-r--r--+ 1 root root 0 Jul 20 15:01 test3.dir/file.3
0:user:joe:read_data:inherited:allow
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow
```

In these examples, because the permission bits of the parent directory for group@ and everyone@ deny write and execute permissions, user joe is denied write and execute permissions. The default aclinherit property is restricted, which means that write\_data and execute permissions are not inherited.

In the following example, user joe is granted read, write, and execute permissions that are inherited for newly created files, but are not propagated to subsequent contents of the directory.

```
chmod A+user:joe:read_data/write_data/execute:file_inherit/no_propagate:allow
test4.dir
ls -dv test4.dir
drwxr--r--+ 2 root root 2 Mar 1 12:11 test4.dir
0:user:joe:list_directory/read_data/add_file/write_data/execute
:file_inherit/no_propagate:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:list_directory/read_data/read_xattr/read_attributes/read_acl
```

```
/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/read_attributes/read_acl
/synchronize:allow
```

As the following example illustrates, joe's read\_data/write\_data/execute permissions are reduced based on the owning group's permissions.

```
touch test4.dir/file.4
ls -v test4.dir/file.4
-rw-r--r--+ 1 root root 0 Jul 20 15:09 test4.dir/file.4
0:user:joe:read_data:inherited:allow
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow
```

## Modifying ACL Inheritance With the ACL Inherit Mode

This section describes the `aclinherit` property values.

**EXAMPLE 47** ACL Inheritance With the ACL Inherit Mode Set to discard

If the `aclinherit` property on a file system is set to `discard`, then ACLs can potentially be discarded when the permission bits on a directory change. For example:

```
zfs set aclinherit=discard system1/cindy
chmod A+user:joe:read_data/write_data/execute:dir_inherit:allow test5.dir
ls -dv test5.dir
drwxr-xr-x+ 2 root root 2 Jul 20 14:18 test5.dir
0:user:joe:list_directory/read_data/add_file/write_data/execute
:dir_inherit:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

If, at a later time, you decide to tighten the permission bits on a directory, the non-trivial ACL is discarded. For example:

```
chmod 744 test5.dir
```



```
ls -dv test5.dir
drwxr--r-- 2 root root 2 Jul 20 14:18 test5.dir
0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:list_directory/read_data/read_xattr/read_attributes/read_acl
/synchronize:allow
2:everyone@:list_directory/read_data/read_xattr/read_attributes/read_acl
/synchronize:allow
```

**EXAMPLE 48** ACL Inheritance With the ACL Inherit Mode Set to noallow

In the following example, two non-trivial ACLs with file inheritance are set. One ACL allows read\_data permission, and one ACL denies read\_data permission. This example also illustrates how you can specify two ACEs in the same chmod command.

```
zfs set aclinherit=noallow system1/cindy
chmod A+user:joe:read_data:file_inherit:deny,user:lp:read_data:file_inherit:allow
test6.dir
ls -dv test6.dir
drwxr-xr-x+ 2 root root 2 Jul 20 14:22 test6.dir
0:user:joe:read_data:file_inherit:deny
1:user:lp:read_data:file_inherit:allow
2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/delete_child
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
3:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
4:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

As the following example shows, when a new file is created, the ACL that allows read\_data permission is discarded.

```
touch test6.dir/file.6
ls -v test6.dir/file.6
-rw-r--r--+ 1 root root 0 Jul 20 14:23 test6.dir/file.6
0:user:joe:read_data:inherited:deny
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow
```

## ACL passthrough Inherit Mode

A file system that has the `aclinherit` property set to `passthrough` inherits all inheritable ACL entries without any modifications made to the ACL entries when they are inherited. Files are created with a permission mode that is determined by the inheritable ACEs. If no inheritable ACEs exist that affect the permission mode, then the permission mode is set in accordance to the requested mode from the application.

### EXAMPLE 49 ACL Inheritance With ACL Inherit Mode Set to `passthrough` in Verbose Mode

If the `aclinherit` property on the `system1/cindy` file system is set to `passthrough`, then user `joe` would inherit the ACL applied on `test4.dir` for the newly created file `.5` as follows:

```
zfs set aclinherit=passthrough system1/cindy
touch test4.dir/file.5
ls -v test4.dir/file.5
-rw-r--r--+ 1 root root 0 Jul 20 14:16 test4.dir/file.5
0:user:joe:read_data/write_data/execute:inherited:allow
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow
```

### EXAMPLE 50 ACL Inheritance With ACL Inherit Mode Set to `passthrough` in Compact Mode

The following examples use compact ACL syntax to show how to inherit permission bits by setting `aclinherit` mode to `passthrough`.

In this example, an ACL is set on `test1.dir` to force inheritance. The syntax creates an `owner@`, `group@`, and `everyone@` ACL entry for newly created files. Newly created directories inherit an `@owner`, `@group`, and `@everyone` ACL entry.

```
zfs set aclinherit=passthrough system1/cindy
pwd
/system1/cindy
mkdir test1.dir

chmod A=owner@:rwxpcCosRrWaAdD:fd:allow,group@:rwxp:fd:allow,
everyone@:fd:allow test1.dir
ls -Vd test1.dir
drwxrwx---+ 2 root root 2 Jul 20 14:42 test1.dir
owner@:rwxpdDaARWcCos:fd----:allow
group@:rwxp-----:fd----:allow
everyone@:-----:fd----:allow
```

In this example, a newly created file inherits the ACL that was specified to be inherited to newly created files.

```
cd test1.dir
touch file.1
ls -V file.1
-rwxrwx---+ 1 root root 0 Jul 20 14:44 file.1
owner@: rwxpdDaARWcCos:-----I:allow
group@: rwxp-----:-----I:allow
everyone@:-----:-----I:allow
```

In this example, a newly created directory inherits both ACEs that control access to this directory as well as ACEs for future propagation to children of the newly created directory.

```
mkdir subdir.1
ls -dV subdir.1
drwxrwx---+ 2 root root 2 Jul 20 14:45 subdir.1
owner@: rwxpdDaARWcCos:fd---I:allow
group@: rwxp-----:fd---I:allow
everyone@:-----:fd---I:allow
```

The fd---I entries are for propagating inheritance and are not considered during access control.

In the following example, a file is created with a trivial ACL in another directory where inherited ACEs are not present.

```
cd /system1/cindy
mkdir test2.dir
cd test2.dir
touch file.2
ls -V file.2
-rw-r--r-- 1 root root 0 Jul 20 14:48 file.2
owner@: rw-p--aARWcCos:-----:allow
group@: r-----a-R-c--s:-----:allow
everyone@: r-----a-R-c--s:-----:allow
```

## ACL Inherit passthrough-x Mode

When `aclinherit=passthrough-x` is enabled, files are created with the execute (x) permission for owner@, group@, or everyone@, but only if execute permission is set in the file creation mode and in an inheritable ACE that affects the mode.

The following example shows how to inherit the execute permission by setting the `aclinherit` mode to `passthrough-x`.

```
zfs set aclinherit=passthrough-x system1/cindy
```

The following ACL is set on `/system1/cindy/test1.dir` to provide executable ACL inheritance for files for owner@.

```
chmod A=owner@: rwxpcCosRrWaAdD:fd:allow,group@: rwxp:fd:allow,
```

```
everyone@::fd:allow test1.dir
ls -Vd test1.dir
drwxrwx---+ 2 root root 2 Jul 20 14:50 test1.dir
owner@:rwxpdDaARWcCos:fd-----:allow
group@:rwxp-----:fd-----:allow
everyone@:-----:fd-----:allow
```

A file (file1) is created with requested permissions 0666. The resulting permissions are 0660. The execution permission was not inherited because the creation mode did not request it.

```
touch test1.dir/file1
ls -V test1.dir/file1
-rw-rw----+ 1 root root 0 Jul 20 14:52 test1.dir/file1
owner@:rw-pdDaARWcCos:-----I:allow
group@:rw-p-----:-----I:allow
everyone@:-----:-----I:allow
```

Next, an executable called t is generated by using the cc compiler in the testdir directory.

```
cc -o t t.c
ls -V t
-rwxrwx---+ 1 root root 7396 Dec 3 15:19 t
owner@:rwxpdDaARWcCos:-----I:allow
group@:rwxp-----:-----I:allow
everyone@:-----:-----I:allow
```

The resulting permissions are 0770 because cc requested permissions 0777, which caused the execute permission to be inherited from the owner@, group@, and everyone@ entries.

## Applying Special Attributes to ZFS Files

This section shows how to apply special attributes to ZFS files and how to display them. For more information about displaying and applying special attributes, see the [ls\(1\)](#) and [chmod\(1\)](#) man pages.

### Applying Immutability to a ZFS File

Use the following syntax to make a file immutable:

```
chmod S+ci file.1
echo this >>file.1
-bash: file.1: Not owner
rm file.1
rm: cannot remove `file.1': Not owner
```

You can display special attributes on ZFS files by using the following syntax:

```
ls -l/c file.1
-rw-r--r--+ 1 root root 206695 Jul 20 14:27 file.1
{A-----im----}
```

Use the following syntax to remove file immutability:

```
chmod S-ci file.1
ls -l/c file.1
-rw-r--r--+ 1 root root 206695 Jul 20 14:27 file.1
{A-----m----}
rm file.1
```

## Preventing Accidental Deletions With the `nounlink` Attribute

The `nounlink` attribute complements the immutability of files or directories in ZFS by securing them from being accidentally removed. However, unlike the `immutable` attribute, `nounlink` only prevents a file from being deleted or renamed. The file can still be changed by applications or by users.

For some examples, see the following [blog entry](#).

## Applying Read-Only Access to a ZFS File

The following example shows how to apply read-only access to a ZFS file.

```
chmod S+cR file.2
echo this >>file.2
-bash: file.2: Not owner
```

## Displaying and Changing ZFS File Attributes

You can display and set special attributes with the following syntax:

```
ls -l/v file.3
-r--r--r-- 1 root root 206695 Jul 20 14:59 file.3
{archive,nohidden,noreadonly,nosystem,noappendonly,nonodump,
noimmutable,av_modified,noav_quarantined,nonounlink,nooffline,nospase}
chmod S+cR file.3
ls -l/v file.3
-r--r--r-- 1 root root 206695 Jul 20 14:59 file.3
{archive,nohidden,readonly,nosystem,noappendonly,nonodump,noimmutable,
av_modified,noav_quarantined,nonounlink,nooffline,nospase}
```

Some of these attributes apply only in an Oracle Solaris SMB environment.

You can clear all attributes on a file. For example:

```
chmod S-a file.3
ls -l/v file.3
-r--r--r-- 1 root root 206695 Jul 20 14:59 file.3
{noarchive,nohidden,noreadonly,nosystem,noappendonly,nonodump,
noimmutable,noav_modified,noav_quarantined,nonounlink,nooffline,nospase}
```

## Oracle Solaris ZFS Delegated Administration

---

This chapter describes how to use delegated administration to allow nonprivileged users to perform ZFS administration tasks.

The following sections are provided in this chapter:

- [“Overview of ZFS Delegated Administration” on page 207.](#)
- [“Delegating ZFS Permissions” on page 208.](#)
- [“Displaying ZFS Delegated Permissions Examples” on page 216.](#)
- [“Delegating ZFS Permissions Examples” on page 212.](#)
- [“Removing ZFS Delegated Permissions Examples” on page 218.](#)

### Overview of ZFS Delegated Administration

ZFS delegated administration enables you to distribute refined permissions to specific users, groups, or everyone. Two types of delegated permissions are supported:

- Individual permissions can be explicitly delegated such as `create`, `destroy`, `mount`, `snapshot`, and so on.
- Groups of permissions called *permission sets* can be defined. A permission set can later be updated, and all of the consumers of the set automatically get the change. Permission sets begin with the `@` symbol and are limited to 64 characters in length. After the `@` symbol, the remaining characters in the set name have the same restrictions as normal ZFS file system names.

ZFS delegated administration provides features similar to the RBAC security model. ZFS delegation provides the following advantages for administering ZFS storage pools and file systems:

- Permissions follow the ZFS storage pool whenever a pool is migrated.
- Provides dynamic inheritance where you can control how the permissions propagate through the file systems.
- Can be configured so that only the creator of a file system can destroy the file system.

- You can delegate permissions to specific file systems. Newly created file systems can automatically pick up permissions.
- Provides simple NFS administration. For example, a user with explicit permissions can create a snapshot over NFS in the appropriate `.zfs/snapshot` directory.

Consider using delegated administration for distributing ZFS tasks. For information about using RBAC to manage general Oracle Solaris administration tasks, see [Chapter 1, “About Using Rights to Control Users and Processes”](#) in *Securing Users and Processes in Oracle Solaris 11.3*.

## Disabling ZFS Delegated Permissions

You control the delegated administration features by using a pool's delegation property. For example:

```
zpool get delegation users
NAME PROPERTY VALUE SOURCE
users delegation on default
zpool set delegation=off users
zpool get delegation users
NAME PROPERTY VALUE SOURCE
users delegation off local
```

By default, the delegation property is enabled.

## Delegating ZFS Permissions

You can use the `zfs allow` command to delegate permissions on ZFS file systems to non-root users in the following ways:

- Individual permissions can be delegated to a user, group, or everyone.
- Groups of individual permissions can be delegated as a *permission set* to a user, group, or everyone.
- Permissions can be delegated either locally to the current file system only or to all descendants of the current file system.

The following table describes the operations that can be delegated and any dependent permissions that are required to perform the delegated operations.

| Permission<br>(Subcommand) | Description                                                        | Dependencies                                         |
|----------------------------|--------------------------------------------------------------------|------------------------------------------------------|
| allow                      | The permission to grant permissions that you have to another user. | Must also have the permission that is being allowed. |



| Permission<br>(Subcommand) | Description                                                                                    | Dependencies                                                                                                                                                                     |
|----------------------------|------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| clone                      | The permission to clone any of the dataset's snapshots.                                        | Must also have the create permission and the mount permission in the original file system.                                                                                       |
| create                     | The permission to create descendant datasets.                                                  | Must also have the mount permission.                                                                                                                                             |
| destroy                    | The permission to destroy a dataset.                                                           | Must also have the mount permission.                                                                                                                                             |
| diff                       | The permission to identify paths within a dataset.                                             | Non-root users need this permission to use the <code>zfs diff</code> command.                                                                                                    |
| hold                       | The permission to hold a snapshot.                                                             |                                                                                                                                                                                  |
| mount                      | The permission to mount and unmount a file system, and create and destroy volume device links. |                                                                                                                                                                                  |
| promote                    | The permission to promote a clone to a dataset.                                                | Must also have the mount permission and the promote permission in the original file system.                                                                                      |
| receive                    | The permission to create descendant file systems with the <code>zfs receive</code> command.    | Must also have the mount permission and the create permission.                                                                                                                   |
| release                    | The permission to release a snapshot hold, which might destroy the snapshot.                   |                                                                                                                                                                                  |
| rename                     | The permission to rename a dataset.                                                            | Must also have the create permission and the mount permission in the new parent.                                                                                                 |
| rollback                   | The permission to roll back a snapshot.                                                        |                                                                                                                                                                                  |
| send                       | The permission to send a snapshot stream.                                                      |                                                                                                                                                                                  |
| share                      | The permission to share and unshare a file system.                                             | Must have both <code>share</code> and <code>share.nfs</code> to create an NFS share.<br><br>Must have both <code>share</code> and <code>share.smb</code> to create an SMB share. |
| snapshot                   | The permission to create a snapshot of a dataset.                                              |                                                                                                                                                                                  |

You can delegate the following set of permissions but a permission might be limited to access, read, or change permission:

- `groupquota`
- `groupused`
- `key`
- `keychange`
- `userprop`
- `userquota`
- `userused`

In addition, you can delegate administration of the following ZFS properties to non-root users:

- aclinherit
- aclmode
- atime
- canmount
- casesensitivity
- checksum
- compression
- copies
- dedup
- defaultgroupquota
- defaultuserquota
- devices
- encryption
- exec
- keysource
- logbias
- mountpoint
- nbmand
- normalization
- primarycache
- quota
- readonly
- recordsize
- refquota
- refreservation
- reservation
- rstchown
- secondarycache
- setuid
- shadow
- share.nfs
- share.smb
- snapdir
- sync
- utf8only
- version
- volblocksize

- volsize
- vscan
- xattr
- zoned

Some of these properties can be set only at dataset creation time. For a description of these properties, see [zfs\(1M\)](#).

## Delegating ZFS Permissions (zfs allow)

The `zfs allow` syntax follows:

```
zfs allow [-ldugecs] everyone|user|group[,...] perm|@setname,...] filesystem| volume
```

The following `zfs allow` syntax (in bold) identifies to whom the permissions are delegated:

```
zfs allow [-uge]|user|group|everyone [,...] filesystem | volume
```

Multiple entities can be specified as a comma-separated list. If no `-uge` options are specified, then the argument is interpreted preferentially as the keyword `everyone`, then as a user name, and lastly, as a group name. To specify a user or group named “everyone,” use the `-u` or `-g` option. To specify a group with the same name as a user, use the `-g` option. The `-c` option delegates create-time permissions.

The following `zfs allow` syntax (in bold) identifies how permissions and permission sets are specified:

```
zfs allow [-s] ... perm|@setname [,...] filesystem | volume
```

Multiple permissions can be specified as a comma-separated list. Permission names are the same as ZFS subcommands and properties. For more information, see the preceding section.

Permissions can be aggregated into *permission sets* and are identified by the `-s` option.

Permission sets can be used by other `zfs allow` commands for the specified file system and its descendants. Permission sets are evaluated dynamically, so changes to a set are immediately updated. Permission sets follow the same naming requirements as ZFS file systems, but the name must begin with an at sign (@) and can be no more than 64 characters in length.

The following `zfs allow` syntax (in bold) identifies how the permissions are delegated:

```
zfs allow [-ld] filesystem | volume
```

The `-l` option indicates that the permissions are allowed for the specified file system and not its descendants, unless the `-d` option is also specified. The `-d` option indicates that the permissions are allowed for the descendant file systems and not for this file system, unless the `-l` option is

also specified. If neither option is specified, then the permissions are allowed for the file system or volume and all of its descendants.

## Removing ZFS Delegated Permissions (**zfs unallow**)

You can remove previously delegated permissions with the `zfs unallow` command.

For example, assume that you delegated create, destroy, mount, and snapshot permissions as follows:

```
zfs allow cindy create,destroy,mount,snapshot system1/home/cindy
zfs allow system1/home/cindy
---- Permissions on system1/home/cindy -----
Local+descendant permissions:
user cindy create,destroy,mount,snapshot
```

To remove these permissions, you would use the following syntax:

```
zfs unallow cindy system1/home/cindy
zfs allow system1/home/cindy
```

## Delegating ZFS Permissions Examples

### EXAMPLE 51 Delegating Permissions to an Individual User

When you delegate create and mount permissions to an individual user, you must ensure that the user has permissions on the underlying mount point.

For example, to delegate user mark create and mount permissions on the `system1` file system, set the permissions first:

```
chmod A+user:mark:add_subdirectory:fd:allow /system1/home
```

Then, use the `zfs allow` command to delegate create, destroy, and mount permissions. For example:

```
zfs allow mark create,destroy,mount system1/home
```

Now, user mark can create his own file systems in the `system1/home` file system. For example:

```
su mark
```

```
mark$ zfs create system1/home/mark
mark$ ^D
su lp
$ zfs create system1/home/lp
cannot create 'system1/home/lp': permission denied
```

**EXAMPLE 52** Delegating create and destroy Permissions to a Group

The following example shows how to set up a file system so that anyone in the staff group can create and mount file systems in the system1/home file system, as well as destroy their own file systems. However, staff group members cannot destroy anyone else's file systems.

```
zfs allow staff create,mount system1/home
zfs allow -c create,destroy system1/home
zfs allow system1/home
---- Permissions on system1/home -----
Create time permissions:
create,destroy
Local+descendant permissions:
group staff create,mount
su cindy
cindy% zfs create system1/home/cindy/files
cindy% exit
su mark
mark% zfs create system1/home/mark/data
mark% exit
cindy% zfs destroy system1/home/mark/data
cannot destroy 'system1/home/mark/data': permission denied
```

**EXAMPLE 53** Delegating Permissions at the Correct File System Level

Ensure that you delegate users permission at the correct file system level. For example, user mark is delegated create, destroy, and mount permissions for the local and descendant file systems. User mark is delegated local permission to snapshot the system1/home file system, but he is not allowed to snapshot his own file system. So, he has not been delegated the snapshot permission at the correct file system level.

```
zfs allow -l mark snapshot system1/home
zfs allow system1/home
---- Permissions on system1/home -----
Create time permissions:
create,destroy
Local permissions:
user mark snapshot
Local+descendant permissions:
group staff create,mount
su mark
mark$ zfs snapshot system1/home@snap1
```

```
mark$ zfs snapshot system1/home/mark@snap1
cannot create snapshot 'system1/home/mark@snap1': permission denied
```

To delegate user mark permission at the descendant file system level, use the `zfs allow -d` option. For example:

```
zfs unallow -l mark snapshot system1/home
zfs allow -d mark snapshot system1/home
zfs allow system1/home
---- Permissions on system1/home -----
Create time permissions:
create,destroy
descendant permissions:
user mark snapshot
Local+descendant permissions:
group staff create,mount
su mark
$ zfs snapshot system1/home@snap2
cannot create snapshot 'system1/home@snap2': permission denied
$ zfs snapshot system1/home/mark@snappy
```

Now, user mark can only create a snapshot below the `system1/home` file system level.

#### **EXAMPLE 54** Defining and Using Complex Delegated Permissions

You can delegate specific permissions to users or groups. For example, the following `zfs allow` command delegates specific permissions to the `staff` group. In addition, `destroy` and `snapshot` permissions are delegated after `system1/home` file systems are created.

```
zfs allow staff create,mount system1/home
zfs allow -c destroy,snapshot system1/home
zfs allow system1/home
---- Permissions on system1/home -----
Create time permissions:
create,destroy,snapshot
Local+descendant permissions:
group staff create,mount
```

Because user mark is a member of the `staff` group, he can create file systems in `system1/home`. In addition, user mark can create a snapshot of `system1/home/mark2` because he has specific permissions to do so. For example:

```
su mark
$ zfs create system1/home/mark2
$ zfs allow system1/home/mark2
---- Permissions on system1/home/mark2 -----
Local permissions:
user mark create,destroy,snapshot
---- Permissions on system1/home -----
Create time permissions:
```

```
create,destroy,snapshot
Local+descendant permissions:
group staff create,mount
```

But, user mark cannot create a snapshot in `system1/home/mark` because he doesn't have specific permissions to do so. For example:

```
$ zfs snapshot system1/home/mark@snap1
cannot create snapshot 'system1/home/mark@snap1': permission denied
```

In this example, user mark has create permission in his home directory, which means he can create snapshots. This scenario is helpful when your file system is NFS mounted.

```
$ cd /system1/home/mark2
$ ls
$ cd .zfs
$ ls
shares snapshot
$ cd snapshot
$ ls -l
total 3
drwxr-xr-x 2 mark staff 2 Sep 27 15:55 snap1
$ pwd
/system1/home/mark2/.zfs/snapshot
$ mkdir snap2
$ zfs list
zfs list -r system1/home
NAME USED AVAIL REFER MOUNTPOINT
system1/home/mark 63K 62.3G 32K /system1/home/mark
system1/home/mark2 49K 62.3G 31K /system1/home/mark2
system1/home/mark2@snap1 18K - 31K -
system1/home/mark2@snap2 0 - 31K -
$ ls
snap1 snap2
$ rmdir snap2
$ ls
snap1
```

#### EXAMPLE 55 Defining and Using a ZFS Delegated Permission Set

The following example shows how to create the permission set `@myset` and delegates the permission set and the rename permission to the group `staff` for the `system1` file system. User `cindy`, a `staff` group member, has the permission to create a file system in `system1`. However, user `lp` does not have permission to create a file system in `system1`.

```
zfs allow -s @myset create,destroy,mount,snapshot,promote,clone,readonly system1
zfs allow system1
---- Permissions on system1 -----
Permission sets:
@myset clone,create,destroy,mount,promote,readonly,snapshot
```

```
zfs allow staff @myset, rename system1
zfs allow system1
---- Permissions on system1 -----
Permission sets:
@myset clone, create, destroy, mount, promote, readonly, snapshot
Local+descendant permissions:
group staff @myset, rename
chmod A+group:staff:add_subdirectory:fd:allow system1
su cindy
cindy% zfs create system1/data
cindy% zfs allow system1
---- Permissions on system1 -----
Permission sets:
@myset clone, create, destroy, mount, promote, readonly, snapshot
Local+descendant permissions:
group staff @myset, rename
cindy% ls -l /system1
total 15
drwxr-xr-x 2 cindy staff 2 Jun 24 10:55 data
cindy% exit
su lp
$ zfs create system1/lp
cannot create 'system1/lp': permission denied
```

## Displaying ZFS Delegated Permissions Examples

You can use the following command to display permissions:

```
zfs allow dataset
```

This command displays permissions that are set or allowed on the specified dataset. The output contains the following components:

- Permission sets
- Individual permissions or create-time permissions
- Local dataset
- Local and descendant datasets
- Descendant datasets only

### EXAMPLE 56 Displaying Basic Delegated Administration Permissions

The following output indicates that user cindy has create, destroy, mount, snapshot permissions on the system1/cindy file system.

```
zfs allow system1/cindy

```



```
Local+descendant permissions on (system1/cindy)
user cindy create,destroy,mount,snapshot
```

#### EXAMPLE 57 Displaying Complex Delegated Administration Permissions

The output in this example indicates the following permissions on the pool/fred and pool file systems.

For the pool/fred file system:

- Two permission sets are defined:
  - @eng (create, destroy, snapshot, mount, clone, promote, rename)
  - @simple (create, mount)
- Create-time permissions are set for the @eng permission set and the mountpoint property. Create-time means that after a file system set is created, the @eng permission set and the permission to set the mountpoint property are delegated.
- User tom is delegated the @eng permission set, and user joe is granted create, destroy, and mount permissions for local file systems.
- User fred is delegated the @basic permission set, and share and rename permissions for the local and descendant file systems.
- User barney and the staff group are delegated the @basic permission set for descendant file systems only.

For the pool file system:

- The permission set @simple (create, destroy, mount) is defined.
- The group staff is granted the @simple permission set on the local file system.

Here is the output for this example:

```
$ zfs allow pool/fred
---- Permissions on pool/fred -----
Permission sets:
@eng create,destroy,snapshot,mount,clone,promote,rename
@simple create,mount
Create time permissions:
@eng,mountpoint
Local permissions:
user tom @eng
user joe create,destroy,mount
Local+descendant permissions:
user fred @basic,share,rename
user barney @basic
group staff @basic
---- Permissions on pool -----
Permission sets:
@simple create,destroy,mount
```

```
Local permissions:
group staff @simple
```

## Removing ZFS Delegated Permissions Examples

You can use the `zfs unallow` command to remove delegated permissions. For example, user cindy has create, destroy, mount, and snapshot permissions on the `system1/cindy` file system.

```
zfs allow cindy create,destroy,mount,snapshot system1/home/cindy
zfs allow system1/home/cindy
---- Permissions on system1/home/cindy -----
Local+descendant permissions:
user cindy create,destroy,mount,snapshot
```

The following `zfs unallow` syntax removes user cindy's snapshot permission from the `system1/home/cindy` file system:

```
zfs unallow cindy snapshot system1/home/cindy
zfs allow system1/home/cindy
---- Permissions on system1/home/cindy -----
Local+descendant permissions:
user cindy create,destroy,mount
cindy% zfs create system1/home/cindy/data
cindy% zfs snapshot system1/home/cindy@today
cannot create snapshot 'system1/home/cindy@today': permission denied
```

As another example, user mark has the following permissions on the `system1/home/mark` file system:

```
zfs allow system1/home/mark
---- Permissions on system1/home/mark -----
Local+descendant permissions:
user mark create,destroy,mount

```

The following `zfs unallow` syntax removes all permissions for user mark from the `system1/home/mark` file system:

```
zfs unallow mark system1/home/mark
```

The following `zfs unallow` syntax removes a permission set on the `system1` file system.

```
zfs allow system1
---- Permissions on system1 -----
Permission sets:
@myset clone,create,destroy,mount,promote,readonly,snapshot
Create time permissions:
```

```
create,destroy,mount
Local+descendant permissions:
group staff create,mount
zfs unallow -s @myset system1
zfs allow system1
---- Permissions on system1 -----
Create time permissions:
create,destroy,mount
Local+descendant permissions:
group staff create,mount
```



# ◆ ◆ ◆ 11 CHAPTER 11

## Oracle Solaris ZFS Advanced Topics

---

This chapter describes ZFS volumes, using ZFS on an Oracle Solaris system with zones installed, ZFS alternate root pools, and ZFS rights profiles.

The chapter covers the following topics:

- [“ZFS Volumes” on page 221.](#)
- [“Using ZFS on an Oracle Solaris System With Zones Installed” on page 224.](#)
- [“Using a ZFS Pool With an Alternate Root Location” on page 230.](#)

## ZFS Volumes

A ZFS volume is a dataset that represents a block device. ZFS volumes are identified as devices in the `/dev/zvol/{dsk,rdsk}/pool` directory.

In the following example, a 5-GB ZFS volume, `system1/vol`, is created:

```
zfs create -V 5gb system1/vol
```

Be careful when changing the size of the volume. For example, if the size of the volume shrinks, data corruption might occur. In addition, if you create a snapshot of a volume that changes in size, you might introduce inconsistencies if you attempt to roll back the snapshot or create a clone from the snapshot. Thus, when you create a volume, a reservation is automatically set to the initial size of the volume to ensure data integrity.

You can display a ZFS volume's property information by using the `zfs get` or `zfs get all` command. For example:

```
zfs get all system1/vol
```

A question mark (?) displayed for `volsize` in the `zfs get` output indicates an unknown value because an I/O error occurred. For example:

```
zfs get -H volsize system1/vol
system1/vol volsize ? local
```

An I/O error generally indicates a problem with a pool device. For information about resolving pool device problems, see [“Identifying Problems With ZFS Storage Pools” on page 236](#).

If you are using an Oracle Solaris system with zones installed, you cannot create or clone a ZFS volume in a native zone.

## Using a ZFS Volume as a Swap or Dump Device

During an installation of a ZFS root file system or a migration from a UFS root file system, a swap device and a dump device are created on a ZFS volume in the ZFS root pool. The following examples show how to display information about the swap device and the dump device.

```
swap -l
swapfile dev swaplo blocks free
/dev/zvol/dsk/rpool/swap 253,3 16 8257520 8257520
```

```
dumpadm
Dump content: kernel pages
Dump device: /dev/zvol/dsk/rpool/dump (dedicated)
Savecore directory: /var/crash/
Savecore enabled: yes
```

If you need to change your swap area or dump device after the system is installed, use the swap and dumpadm commands as in previous Oracle Solaris releases. If you need to create an additional swap volume, create a ZFS volume of a specific size and then enable swap on that device. Then, add an entry for the new swap device in the /etc/vfstab file. For example:

```
zfs create -V 2G rpool/swap2
swap -a /dev/zvol/dsk/rpool/swap2
swap -l
swapfile dev swaplo blocks free
/dev/zvol/dsk/rpool/swap 256,1 16 2097136 2097136
/dev/zvol/dsk/rpool/swap2 256,5 16 4194288 4194288
```

Do not swap to a file on a ZFS file system. A ZFS swap file configuration is not supported.

For information about adjusting the size of the swap and dump volumes, see [“Adjusting the Sizes of ZFS Swap and Dump Devices” on page 89](#).

## Using a ZFS Volume as an iSCSI LUN

A ZFS volume as an iSCSI target is managed just like any other ZFS dataset, except that you cannot rename the dataset, roll back a volume snapshot, or export the pool while the ZFS volumes are shared as iSCSI LUNs. If you attempt to perform those operations, messages similar to the following are displayed:

```
zfs rename system1/volumes/v2 system1/volumes/v1
cannot rename 'system1/volumes/v2': dataset is busy
zpool export system1
cannot export 'system1': pool is busy
```

All iSCSI target configuration information is stored within the dataset. Like an NFS shared file system, an iSCSI target that is imported on a different system is shared appropriately.

The Common Multiprotocol SCSI Target (COMSTAR) software framework enables you to convert any Oracle Solaris stem into a SCSI target device that can be accessed over a storage network by initiator hosts. You can create and configure a ZFS volume to be shared as an iSCSI logical unit (LUN).

## ▼ How to Use a ZFS Volume as an iSCSI LUN

1. **First, install the COMSTAR package.**

```
pkg install group/feature/storage-server
```

2. **Create a ZFS volume to be used as an iSCSI target.**

For example:

```
zfs create -V 2g system1/volumes/v2
```

3. **Create the SCSI-block-device-based LUN.**

For example:

```
sbdadm create-lu /dev/zvol/rdisk/system1/volumes/v2
```

Created the following LU:

| GUID                             | DATA SIZE  | SOURCE                             |
|----------------------------------|------------|------------------------------------|
| 600144f000144f1dafaa4c0faff20001 | 2147483648 | /dev/zvol/rdisk/system1/volumes/v2 |

```
sbdadm list-lu
```

Found 1 LU(s)

| GUID                             | DATA SIZE  | SOURCE                             |
|----------------------------------|------------|------------------------------------|
| 600144f000144f1dafaa4c0faff20001 | 2147483648 | /dev/zvol/rdisk/system1/volumes/v2 |

4. **Share LUN views to all ZFS clients. or selected ZFS clients.**

You can expose the LUN views to all ZFS clients or to a selected list of ZFS clients. In the following example, the LUN view is shared to all ZFS clients.

- a. **Identify the LUN GUID.**

```
stmfadm list-lu
LU Name: 600144F000144F1DAFAA4C0FAFF20001
```

**b. Share the LUN view.**

```
stmfadm add-view 600144F000144F1DAFAA4C0FAFF20001
stmfadm list-view -l 600144F000144F1DAFAA4C0FAFF20001
View Entry: 0
Host group : All
Target group: All
LUN : 0
```

**5. Create the iSCSI targets.**

For information about creating the iSCSI targets, see [Chapter 8, “Configuring Storage Devices With COMSTAR”](#) in *Managing Devices in Oracle Solaris 11.3*.

## Using ZFS on an Oracle Solaris System With Zones Installed

The Oracle™ Solaris Zones feature in the Oracle Solaris operating system provides an isolated environment in which to run applications on your system. The following sections describe how to use ZFS on a system with Oracle Solaris zones:

- [“Adding ZFS File Systems to a Non-Global Zone”](#) on page 225.
- [“Delegating Datasets to a Non-Global Zone”](#) on page 226.
- [“Adding ZFS Volumes to a Non-Global Zone”](#) on page 226.
- [“Using ZFS Storage Pools Within a Zone”](#) on page 227.
- [“Managing ZFS Properties Within a Zone”](#) on page 227.
- [“Understanding the zoned Property”](#) on page 228.

Keep the following points in mind when associating ZFS datasets with zones:

- You can add a ZFS file system or a clone to a native zone with or without delegating administrative control.
- You can add a ZFS volume as a device to native zones.
- You cannot associate ZFS snapshots with zones at this time.

---

**Note** - Oracle Solaris kernel zones use storage differently from native Oracle Solaris zones. For more information about storage use in kernel zones, see the Storage Access section of the [solaris-kz\(5\)](#) man page.

For information about storage use on shared storage, see [Chapter 13, “Getting Started With Oracle Solaris Zones on Shared Storage”](#) in *Creating and Using Oracle Solaris Zones*.

---

Adding a ZFS filesystem by using an fs resource enables the native zone to share disk space with the global or kernel zone. However, the zone administrator cannot control properties or



create new file systems in the underlying file system hierarchy. This operation is identical to adding any other type of file system to a zone. You should add a file system to a native zone only for the sole purpose of sharing common disk space.

You can also delegate ZFS datasets to a native zone, which would give the zone administrator complete control over the dataset and all its children. The zone administrator can create and destroy file systems or clones within that dataset, as well as modify properties of the datasets. The zone administrator cannot affect datasets that have not been added to the zone, including exceeding any top-level quotas set on the delegated dataset.

When both a source zonepath and a target zonepath reside on a ZFS file system and are in the same pool, the `zoneadm clone` command, not `zfs clone`, becomes the command for cloning zones. The `zoneadm clone` command creates a ZFS snapshot of the source zonepath and sets up the target zonepath. For more information, see [Creating and Using Oracle Solaris Zones](#).

## Adding ZFS File Systems to a Non-Global Zone

A ZFS file system that is added to a native zone must have its mountpoint property set to `legacy`. For example, for the `system1/zone/zion` file system, you would type the following command on the global or kernel zone:

```
global# zfs set mountpoint=legacy system1/zone/zion
```

Then you would add that file system to the native zone by using the `add fs` subcommand of the `zonecfg` command.

---

**Note** - To add the files system, ensure that it is not previously mounted on another location.

---

```
global# zonecfg -z zion
zonecfg:zion> add fs
zonecfg:zion:fs> set type=zfs
zonecfg:zion:fs> set special=system1/zone/zion
zonecfg:zion:fs> set dir=/opt/data
zonecfg:zion:fs> end
```

This syntax adds the ZFS file system, `system1/zone/zion`, to the already configured `zion` zone, which is mounted at `/opt/data`. The zone administrator can create and destroy files within the file system. The file system cannot be remounted to a different location. Likewise, the zone administrator cannot change properties on the file system such as `atime`, `readonly`, `compression`, and so on.

The global zone administrator is responsible for setting and controlling properties of the file system.

For more information about the `zonecfg` command and about configuring resource types with `zonecfg`, see [Creating and Using Oracle Solaris Zones](#).

## Delegating Datasets to a Non-Global Zone

To meet the primary goal of delegating the administration of storage to a zone, ZFS supports adding datasets to a native zone through the use of the `zonecfg add dataset` command.

In the following example, a ZFS file system is delegated to a native zone by a global zone administrator from the global zone or kernel zone.

```
global# zonecfg -z zion
zonecfg:zion> add dataset
zonecfg:zion:dataset> set name=system1/zone/zion
zonecfg:zion:dataset> set alias=system1
zonecfg:zion:dataset> end
```

Unlike adding a file system, this syntax causes the ZFS file system `system1/zone/zion` to be visible within the already configured `zion` zone. Within the `zion` zone, this file system is not accessible as `system1/zone/zion`, but as a *virtual pool* named `system1`. The delegated file system alias provides a view of the original pool to the zone as a virtual pool. The alias property specifies the name of the virtual pool. If no alias is specified, a default alias matching the last component of the file system name is used. In the example, the default alias would be `zion`.

Within delegated datasets, the zone administrator can set file system properties, as well as create descendant file systems. In addition, the zone administrator can create snapshots and clones, and otherwise control the entire file system hierarchy. If ZFS volumes are created within delegated file systems, these volumes might conflict with ZFS volumes that are added as device resources.

## Adding ZFS Volumes to a Non-Global Zone

You can add or create a ZFS volume in a native zone or you can add access to a volume's data in a native zone in the following ways:

- In a native zone, a privileged zone administrator can create a ZFS volume as descendant of a previously delegated file system. For example, you can type the following command for the file system `system1/zone/zion` that was delegated in the previous example:

```
zfs create -V 2g system1/zone/zion/vol1
```

After the volume is created, the zone administrator can manage the volume's properties and data in the native zone as well as create snapshots.

- In a global or kernel zone, use the `zonecfg add device` command and specify a ZFS volume whose data can be accessed in a native zone. For example:

```
global# zonecfg -z zion
```

```
zonecfg:zion> add device
zonecfg:zion:device> set match=/dev/zvol/dsk/system1/volumes/vol2
zonecfg:zion:device> end
```

In this example, only the volume data can be accessed in the native zone.

## Using ZFS Storage Pools Within a Zone

ZFS storage pools cannot be created or modified within a native zone. The delegated administration model centralizes control of physical storage devices within the global or kernel zone and control of virtual storage to native zones. Although a pool-level dataset can be added to a native zone, any command that modifies the physical characteristics of the pool, such as creating, adding, or removing devices, is not allowed from within a native zone. Even if physical devices are added to a native zone by using the `zonecfg add device` command, or if files are used, the `zpool` command does not allow the creation of any new pools within the native zone.

Kernel zones are more powerful and more flexible in terms of data storage management. Devices and volumes can be delegated to a kernel zone, much like a global zone. Also, a ZFS storage pool can be created in a kernel zone.

## Managing ZFS Properties Within a Zone

After a dataset is delegated to a zone, the zone administrator can control specific dataset properties. After a dataset is delegated to a zone, all its ancestors are visible as read-only datasets, while the dataset itself is writable, as are all of its descendants. For example, consider the following configuration:

```
global# zfs list -Ho name
system1
system1/home
system1/data
system1/data/matrix
system1/data/zion
system1/data/zion/home
```

If `system1/data/zion` were added to a zone with the default `zion` alias, each dataset would have the following properties.

| Dataset      | Visible | Writable | Immutable Properties |
|--------------|---------|----------|----------------------|
| system1      | No      | -        | -                    |
| system1/home | No      | -        | -                    |

| Dataset                | Visible | Writable | Immutable Properties      |
|------------------------|---------|----------|---------------------------|
| system1/data           | No      | -        | -                         |
| system1/data/zion      | Yes     | Yes      | zoned, quota, reservation |
| system1/data/zion/home | Yes     | Yes      | zoned                     |

Note that every parent of `system1/zone/zion` is invisible and all descendants are writable. The zone administrator cannot change the `zoned` property because doing so would expose a security risk that described in the next section.

Privileged users in the zone can change any other settable property, except for `quota` and `reservation` properties. This behavior allows the global zone administrator to control the disk space consumption of all datasets used by the native zone.

In addition, the `share.nfs` and `mountpoint` properties cannot be changed by the global zone administrator after a dataset has been delegated to a native zone.

## Understanding the `zoned` Property

When a dataset is delegated to a native zone, the dataset must be specially marked so that certain properties are not interpreted within the context of the global or kernel zone. After a dataset has been delegated to a native zone and is under the control of a zone administrator, its contents can no longer be trusted. As with any file system, `setuid` binaries, symbolic links, or otherwise questionable contents might exist that might adversely affect the security of the global or kernel zone. In addition, the `mountpoint` property cannot be interpreted in the context of the global or kernel zone. Otherwise, the zone administrator could affect the global or kernel zone's namespace. To address the latter, ZFS uses the `zoned` property to indicate that a dataset has been delegated to a native zone at one point in time.

The `zoned` property is a boolean value that is automatically turned on when a zone containing a ZFS dataset is first booted. A zone administrator does not need to manually set this property. If the `zoned` property is set, the dataset cannot be mounted or shared in the global or kernel zone. In the following example, `system1/zone/zion` has been delegated to a zone, while `system1/zone/global` has not:

```
zfs list -o name,zoned,mountpoint -r system1/zone
NAME ZONED MOUNTPOINT MOUNTED
system1/zone/global off /system1/zone/global yes
system1/zone/zion on /system1/zone/zion yes
zfs mount
system1/zone/global /system1/zone/global
system1/zone/zion /export/zone/zion/root/system1/zone/zion
```

```

root@kzx-05:~# zonecfg -z sol info dataset

dataset:
 name: rpool/foo
 alias: foo
root@kzx-05:~# zfs list -o name,zoned,mountpoint,mounted -r rpool/foo

NAME ZONED MOUNTPOINT MOUNTED
rpool/foo on /system/zones/sol/root/foo yes
root@kzx-05:~# zfs mount | grep /foo

rpool/foo /system/zones/sol/root/foo

```

When a dataset is removed from a zone or a zone is destroyed, the zoned property is *not* automatically cleared. This behavior would avoid the inherent security risks associated with these tasks. Because an untrusted user has complete access to the dataset and its descendants, the mountpoint property might be set to bad values, or `setuid` binaries might exist on the file systems.

To prevent accidental security risks, the zoned property must be manually cleared by the global zone administrator if you want to reuse the dataset in any way. Before setting the zoned property to `off`, ensure that the mountpoint property for the dataset and all its descendants are set to reasonable values and that no `setuid` binaries exist, or turn off the `setuid` property.

After you have verified that no security vulnerabilities are left, the zoned property can be turned off by using the `zfs set` or `zfs inherit` command. If the zoned property is turned off while a dataset is in use within a zone, the system might behave in unpredictable ways. Only change the property if you are sure the dataset is no longer in use by a native zone.

## Copying Zones to Other Systems

When you need to migrate one or more zones needs to another system, use Oracle Solaris Unified Archives, which manage all cloning and recovery operations in the operating system and which operate on global, native, as well as kernel zones. For more information about Unified Archives, see [Using Unified Archives for System Recovery and Cloning in Oracle Solaris 11.3](#). For instructions about migrating zones, which include copying zones to other systems, see [Chapter 7, “Migrating and Converting Oracle Solaris Zones” in \*Creating and Using Oracle Solaris Zones\*](#).

If all zones on one system need to move to another ZFS pool on a different system, consider using a replication stream because it preserves snapshots and clones. Snapshots and clones are used extensively by `pkg update`, `beadm create`, and the `zoneadm clone` commands.

In the following example, the `sysA`'s zones are installed in the `rpool/zones` file system and they need to be copied to the `newpool/zones` file system on `sysB`. The following commands create a snapshot and copy the data to `sysB` by using a replication stream:

```
sysA# zfs snapshot -r rpool/zones@send-to-sysB
sysA# zfs send -R rpool/zones@send-to-sysB | ssh sysB zfs receive -d newpool
```

---

**Note** - The commands refer only to the ZFS aspect of the operation. You would need to perform other zones-related command to complete the task. For specific information, refer to [Chapter 7, “Migrating and Converting Oracle Solaris Zones”](#) in *Creating and Using Oracle Solaris Zones*.

---

## Using a ZFS Pool With an Alternate Root Location

A pool is intrinsically tied to the host system. The host system maintains information about the pool so that it can detect when the pool is unavailable. Although useful for normal operations, this information can prove a hindrance when you are booting from alternate media or creating a pool on removable media. To solve this problem, ZFS provides an *alternate root location* pool feature. An alternate root pool location does not persist across system reboots, and all mount points are modified to be relative to the root of the pool.

## Creating a ZFS Pool With an Alternate Root Location

The most common reason for creating a pool at an alternate location is for use with removable media. In these circumstances, users typically want a single file system, and they want it to be mounted wherever they choose on the target system. When a pool is created by using the `zpool create -R` option, the mount point of the root file system is automatically set to `/`, which is the equivalent of the alternate root value.

In the following example, a pool called `morpheus` is created with `/mnt` as the alternate root location:

```
zpool create -R /mnt morpheus c0t0d0
zfs list morpheus
```

| NAME     | USED  | AVAIL | REFER | MOUNTPOINT |
|----------|-------|-------|-------|------------|
| morpheus | 32.5K | 33.5G | 8K    | /mnt       |

Note the single file system, `morpheus`, whose mount point is the alternate root location of the pool, `/mnt`. The mount point that is stored on disk is `/` and the full path to `/mnt` is interpreted only in this initial context of the pool creation. This file system can then be exported and imported under an arbitrary alternate root location on a different system by using `-R alternate-root-value` syntax.

```
zpool export morpheus
zpool import morpheus
```

```
cannot mount '/': directory is not empty
zpool export morpheus
zpool import -R /mnt morpheus
zfs list morpheus
NAME USED AVAIL REFER MOUNTPOINT
morpheus 32.5K 33.5G 8K /mnt
```

## Importing a Pool With an Alternate Root Location

Pools can also be imported using an alternate root location. This feature allows for recovery situations, where the mount points should not be interpreted in context of the current root mount point, but under some temporary directory where repairs can be performed. This feature also can be used when you are mounting removable media as described in the preceding section.

In the following example, a pool called `morpheus` is imported with `/mnt` as the alternate root mount point. This example assumes that `morpheus` was previously exported.

```
zpool import -R /a pool
zpool list morpheus
NAME SIZE ALLOC FREE CAP HEALTH ALTROOT
pool 44.8G 78K 44.7G 0% ONLINE /a
zfs list pool
NAME USED AVAIL REFER MOUNTPOINT
pool 73.5K 44.1G 21K /a/pool
```

## Importing a Pool With a Temporary Name

In addition to importing a pool at an alternate root location, you can import a pool with a temporary name. In certain shared storage or recovery situations, this feature allows two pools with the same persistent name to be simultaneously imported. One of those pools must be imported with a temporary name.

In the following example, the `rpool` pool is imported at an alternate root location and with a temporary name. Because the persistent pool name conflicts with a pool that is already imported, it must be imported by pool ID or by specifying the devices.

```
zpool import
pool: rpool
id: 16760479674052375628
state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

rpool ONLINE
c8d1s0 ONLINE
```

```
zpool import -R /a -t altrpool 16760479674052375628
zpool list
NAME SIZE ALLOC FREE CAP DEDUP HEALTH ALTROOT
altrpool 97G 22.4G 74G 23% 1.00x ONLINE /a
rpool 465G 75.1G 390G 16% 1.00x ONLINE -
```

A pool can also be created with a temporary name by using the `zpool create -t` option.



## Oracle Solaris ZFS Troubleshooting and Pool Recovery

---

This chapter describes how to identify and recover from ZFS failures. Information for preventing failures is provided as well.

The following sections are provided in this chapter:

- “Identifying ZFS Problems” on page 233.
- “Resolving General Hardware Problems” on page 234.
- “Identifying Problems With ZFS Storage Pools” on page 236.
- “Resolving ZFS Storage Device Problems” on page 241.
- “Resolving Data Problems in a ZFS Storage Pool” on page 255.
- “Repairing a Damaged ZFS Configuration” on page 267.
- “Repairing an Unbootable System” on page 268.

For information about complete root pool recovery, see *Using Unified Archives for System Recovery and Cloning in Oracle Solaris 11.3*.

### Identifying ZFS Problems

As a combined file system and volume manager, ZFS can exhibit many different failures. This chapter outlines how to diagnose general hardware failures and then how to resolve pool device and file system problems. You can encounter the following types of problems:

- **General Hardware Problems** – Hardware problems can impact your pool performance and the availability of your pool data. Rule out general hardware problems, such as faulty components and memory, before determining problems at a higher level, such as your pools and file systems.
- **ZFS storage pool problems**
  - “Identifying Problems With ZFS Storage Pools” on page 236.
  - “Resolving ZFS Storage Device Problems” on page 241.
- **Data is corrupted** – “Resolving Data Problems in a ZFS Storage Pool” on page 255.

- **Configuration is damaged** – [“Repairing a Damaged ZFS Configuration” on page 267.](#)
- **System won't boot** – [“Repairing an Unbootable System” on page 268.](#)

Note that a single pool can experience all three errors, so a complete repair procedure involves finding and correcting one error, proceeding to the next error, and so on.

## Resolving General Hardware Problems

Review the following sections to determine whether pool problems or file system unavailability is related to a hardware problem, such as faulty system board, memory, device, HBA, or a misconfiguration.

For example, a failing or faulty disk on a busy ZFS pool can greatly degrade overall system performance.

If you start by diagnosing and identifying hardware problems first, which can be easier to detect and all your hardware checks out, you can then move on to diagnosing pool and file system problems as described in the rest of this chapter. If your hardware, pool, and file system configurations are healthy, consider diagnosing application problems, which are generally more complex to unravel and are not covered in this guide.

## Identifying Hardware and Device Faults

The Oracle Solaris Fault Manager tracks software, hardware and specific device problems by identifying error telemetry information that indicate a specific symptom in an error log and then reporting actual fault diagnosis when the error symptom results in an actual fault.

The following command identifies any software or hardware related fault.

```
fmadm faulty
```

Use the above command routinely to identify failed services or devices.

Use the following command routinely to identify hardware or device related errors.

```
fmdump -eV | more
```

Error messages in this log file that describe `vdev.open_failed`, `checksum`, or `io_failure` issues need your attention or they might evolve into actual faults that are displayed with the `fmadm faulty` command.

If the above indicates that a device is failing, then this is a good time to make sure you have a replacement device available.

You can also track additional device errors by using `iostat` command. Use the following syntax to identify a summary of error statistics.

```
iostat -en
---- errors ---
s/w h/w trn tot device
0 0 0 0 c0t5000C500335F95E3d0
0 0 0 0 c0t5000C500335FC3E7d0
0 0 0 0 c0t5000C500335BA8C3d0
0 12 0 12 c2t0d0
0 0 0 0 c0t5000C500335E106Bd0
0 0 0 0 c0t50015179594B6F11d0
0 0 0 0 c0t5000C500335DC60Fd0
0 0 0 0 c0t5000C500335F907Fd0
0 0 0 0 c0t5000C500335BD117d0
```

In the above output, errors are reported on an internal disk `c2t0d0`. Use the following syntax to display more detailed device errors.

## Resolving Persistent or Transient Transport Errors

Persistent SCSI transport errors that refer to retries or resets can be caused by down-rev firmware, a bad disk, a bad cable, or a faulty hardware connection. Some transient transport errors can be resolved by upgrading your HBA or device firmware. If transport errors persist after firmware is updated and all devices are deemed operational, then look for a bad cable or other faulty connection between hardware components.

## System Reporting of ZFS Error Messages

In addition to persistently tracking errors within the pool, ZFS also displays `syslog` messages when events of interest occur. The following scenarios generate notification events:

- **Device state transition** – If a device becomes `FAULTED`, ZFS logs a message indicating that the fault tolerance of the pool might be compromised. A similar message is sent if the device is later brought online, restoring the pool to health.
- **Data corruption** – If any data corruption is detected, ZFS logs a message describing when and where the corruption was detected. This message is only logged the first time it is detected. Subsequent accesses do not generate a message.
- **Pool failures and device failures** – If a pool failure or a device failure occurs, the fault manager daemon reports these errors through `syslog` messages as well as the `fmddump` command.

If ZFS detects a device error and automatically recovers from it, no notification occurs. Such errors do not constitute a failure in the pool redundancy or in data integrity. Moreover, such errors are typically the result of a driver problem accompanied by its own set of error messages.

## Identifying Problems With ZFS Storage Pools

The following sections describe how to identify and resolve problems with your ZFS file systems or storage pools:

- [“Determining If Problems Exist in a ZFS Storage Pool” on page 237.](#)
- [“Reviewing ZFS Storage Pool Status Information” on page 237.](#)
- [“System Reporting of ZFS Error Messages” on page 235.](#)

You can use the following features to identify problems with your ZFS configuration:

- Detailed ZFS storage pool information can be displayed by using the `zpool status` command.
- Pool and device failures are reported through ZFS/FMA diagnostic messages.
- Previous ZFS commands that modified pool state information can be displayed by using the `zpool history` command.
- A ZFS storage pool that is accidentally destroyed can be recovered by using the `zpool import -D` command, but it's important that the pool is recovered quickly so that the devices are not reused or accidentally overwritten. For more information, see [“Recovering Destroyed ZFS Storage Pools” on page 70](#). No similar feature exists to recover ZFS file systems or data. Always have good backups.

Most ZFS troubleshooting involves the `zpool status` command. This command analyzes the various failures in a system and identifies the most severe problem, presenting you with a suggested action and a link to a knowledge article for more information. Note that the command only identifies a single problem with a pool, though multiple problems can exist. For example, data corruption errors generally imply that one of the devices has failed, but replacing the failed device might not resolve all of the data corruption problems.

In addition, a ZFS diagnostic engine diagnoses and reports pool failures and device failures. Checksum, I/O, device, and pool errors associated with these failures are also reported. ZFS failures as reported by `fmd` are displayed on the console as well as the system messages file. In most cases, the `fmd` message directs you to the `zpool status` command for further recovery instructions.

The basic recovery process is as follows:

- If appropriate, use the `zpool history` command to identify the ZFS commands that preceded the error scenario. For example:

```
zpool history system1
History for 'system1':
2012-11-12.13:01:31 zpool create system1 mirror c0t1d0 c0t2d0 c0t3d0
2012-11-12.13:28:10 zfs create system1/eric
2012-11-12.13:37:48 zfs set checksum=off system1/eric
```

In this output, note that checksums are disabled for the `system1/eric` file system. This configuration is not recommended.

- Identify the errors through the `fmfd` messages that are displayed on the system console or in the `/var/adm/messages` file.
- Find further repair instructions by using the `zpool status -x` command.
- Repair the failures, which involves the following steps:
  - Replacing the unavailable or missing device and bring it online.
  - Restoring the faulted configuration or corrupted data from a backup.
  - Verifying the recovery by using the `zpool status -x` command.
  - Backing up your restored configuration, if applicable.

This section describes how to interpret `zpool status` output in order to diagnose the type of failures that can occur. Although most of the work is performed automatically by the command, it is important to understand exactly what problems are being identified in order to diagnose the failure. Subsequent sections describe how to repair the various problems that you might encounter.

## Determining If Problems Exist in a ZFS Storage Pool

The easiest way to determine if any known problems exist on a system is to use the `zpool status -x` command. This command describes only pools that are exhibiting problems. If no unhealthy pools exist on the system, then the command displays the following:

```
zpool status -x
all pools are healthy
```

Without the `-x` flag, the command displays the complete status for all pools (or the requested pool, if specified on the command line), even if the pools are otherwise healthy.

For more information about command-line options to the `zpool status` command, see [“Querying ZFS Storage Pool Status” on page 55](#).

## Reviewing ZFS Storage Pool Status Information

ZFS storage pool status information is displayed by using the `zpool status` command. For example:

```
zpool status pond
pool: pond
state: DEGRADED
```

```
status: One or more devices are unavailable in response to persistent errors.
Sufficient replicas exist for the pool to continue functioning in a
degraded state.
action: Determine if the device needs to be replaced, and clear the errors
using 'zpool clear' or 'fmadm repaired', or replace the device
with 'zpool replace'.
Run 'zpool status -v' to see device specific details.
scan: scrub repaired 0 in 0h0m with 0 errors on Wed Jun 20 13:16:09 2012
config:
```

| NAME                  | STATE    | READ | WRITE | CKSUM |
|-----------------------|----------|------|-------|-------|
| pond                  | DEGRADED | 0    | 0     | 0     |
| mirror-0              | ONLINE   | 0    | 0     | 0     |
| c0t5000C500335F95E3d0 | ONLINE   | 0    | 0     | 0     |
| c0t5000C500335F907Fd0 | ONLINE   | 0    | 0     | 0     |
| mirror-1              | DEGRADED | 0    | 0     | 0     |
| c0t5000C500335BD117d0 | ONLINE   | 0    | 0     | 0     |
| c0t5000C500335DC60Fd0 | UNAVAIL  | 0    | 0     | 0     |

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

This output is described in the following section.

## Overall Pool Status Information

This section in the `zpool status` output contains the following fields, some of which are only displayed for pools exhibiting problems:

|        |                                                                                                                                                                                                                                                                 |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| pool   | Identifies the name of the pool.                                                                                                                                                                                                                                |
| state  | Indicates the current health of the pool. This information refers only to the ability of the pool to provide the necessary replication level.                                                                                                                   |
| status | Describes what is wrong with the pool. This field is omitted if no errors are found.                                                                                                                                                                            |
| action | A recommended action for repairing the errors. This field is omitted if no errors are found.                                                                                                                                                                    |
| see    | Refers to a knowledge article containing detailed repair information. Online articles are updated more often than this guide can be updated. So, always reference them for the most up-to-date repair procedures. This field is omitted if no errors are found. |
| scrub  | Identifies the current status of a scrub operation, which might include the date and time that the last scrub was completed, a scrub is in progress, or if no scrub was requested.                                                                              |

errors                      Identifies known data errors or the absence of known data errors.

## ZFS Storage Pool Configuration Information

The `config` field in the `zpool status` output describes the configuration of the devices in the pool, as well as their state and any errors generated from the devices. The state can be one of the following: `ONLINE`, `FAULTED`, `DEGRADED`, or `SUSPENDED`. If the state is anything but `ONLINE`, the fault tolerance of the pool has been compromised.

The second section of the configuration output displays error statistics. These errors are divided into three categories:

- `READ` – I/O errors that occurred while issuing a read request
- `WRITE` – I/O errors that occurred while issuing a write request
- `CKSUM` – Checksum errors, meaning that the device returned corrupted data as the result of a read request

These errors can be used to determine if the damage is permanent. A small number of I/O errors might indicate a temporary outage, while a large number might indicate a permanent problem with the device. These errors do not necessarily correspond to data corruption as interpreted by applications. If the device is in a redundant configuration, the devices might show uncorrectable errors, while no errors appear at the mirror or RAID-Z device level. In such cases, ZFS successfully retrieved the good data and attempted to heal the damaged data from existing replicas.

For more information about interpreting these errors, see [“Determining the Type of Device Failure” on page 245](#).

Finally, additional auxiliary information is displayed in the last column of the `zpool status` output. This information expands on the state field, aiding in the diagnosis of failures. If a device is `UNAVAIL`, this field indicates whether the device is inaccessible or whether the data on the device is corrupted. If the device is undergoing resilvering, this field displays the current progress.

For information about monitoring resilvering progress, see [“Viewing Resilvering Status” on page 253](#).

## ZFS Storage Pool Scrubbing Status

The scrub section of the `zpool status` output describes the current status of any scrubbing operations. This information is distinct from whether any errors are detected on the system, though this information can be used to determine the accuracy of the data corruption error reporting. If the last scrub ended recently, most likely, any known data corruption has been discovered.

The following `zpool status scrub` status messages are provided:

- Scrub in-progress report. For example:

```
scan: scrub in progress since Wed Jun 20 14:56:52 2012
529M scanned out of 71.8G at 48.1M/s, 0h25m to go
0 repaired, 0.72% done
```

- Scrub completion message. For example:

```
scan: scrub repaired 0 in 0h11m with 0 errors on Wed Jun 20 15:08:23 2012
```

- Ongoing scrub cancellation message. For example:

```
scan: scrub canceled on Wed Jun 20 16:04:40 2012
```

Scrub completion messages persist across system reboots.

For more information about the data scrubbing and how to interpret this information, see [“Checking ZFS File System Integrity” on page 260](#).

## ZFS Data Corruption Errors

The `zpool status` command also shows whether any known errors are associated with the pool. These errors might have been found during data scrubbing or during normal operation. ZFS maintains a persistent log of all data errors associated with a pool. This log is rotated whenever a complete scrub of the system finishes.

Data corruption errors are always fatal. Their presence indicates that at least one application experienced an I/O error due to corrupt data within the pool. Device errors within a redundant pool do not result in data corruption and are not recorded as part of this log. By default, only the number of errors found is displayed. A complete list of errors and their specifics can be found by using the `zpool status -v` option. For example:

```
zpool status -v system1
pool: system1
state: ONLINE
status: One or more devices has experienced an error resulting in data
corruption. Applications may be affected.
action: Restore the file in question if possible. Otherwise restore the
entire pool from backup.
see: http://support.oracle.com/msg/ZFS-8000-8A
scan: scrub repaired 0 in 0h0m with 2 errors on Fri Jun 29 16:58:58 2012
config:
```

| NAME    | STATE  | READ | WRITE | CKSUM |
|---------|--------|------|-------|-------|
| system1 | ONLINE | 2    | 0     | 0     |
| c8t0d0  | ONLINE | 0    | 0     | 0     |
| c8t1d0  | ONLINE | 2    | 0     | 0     |



errors: Permanent errors have been detected in the following files:

/system1/file.1

A similar message is also displayed by `fmd` on the system console and the `/var/adm/messages` file. These messages can also be tracked by using the `fmdump` command.

For more information about interpreting data corruption errors, see [“Identifying the Type of Data Corruption” on page 263](#).

## Resolving ZFS Storage Device Problems

Review the following sections to resolve a missing, removed or faulted device.

### Resolving a Missing or Removed Device

If a device cannot be opened, it displays the `UNAVAIL` state in the `zpool status` output. This state means that ZFS was unable to open the device when the pool was first accessed, or the device has since become unavailable. If the device causes a top-level virtual device to be unavailable, then nothing in the pool can be accessed. Otherwise, the fault tolerance of the pool might be compromised. In either case, the device just needs to be reattached to the system to restore normal operations. If you need to replace a device that is `UNAVAIL` because it has failed, see [“Replacing a Device in a ZFS Storage Pool” on page 248](#).

If a device is `UNAVAIL` in a root pool or a mirrored root pool, see the following references:

- **Mirrored root pool disk failed** – [“Booting From an Alternate Root Pool Disk” on page 91](#)
- **Replacing a disk in a root pool**
  - [“How to Replace a Disk in a ZFS Root Pool” on page 83](#)
  - [“How to Replace a Disk in a ZFS Root Pool” on page 83](#)
- **Full root pool disaster recovery** – [Using Unified Archives for System Recovery and Cloning in Oracle Solaris 11.3](#)

For example, you might see a message similar to the following from `fmd` after a device failure:

```
SUNW-MSG-ID: ZFS-8000-QJ, TYPE: Fault, VER: 1, SEVERITY: Minor
EVENT-TIME: Wed Jun 20 13:09:55 MDT 2012
...
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: e13312e0-be0a-439b-d7d3-cddaefe717b0
DESC: Outstanding dtls on ZFS device 'id1,sd@n5000c500335dc60f/a' in pool 'pond'.
AUTO-RESPONSE: No automated response will occur.
```

IMPACT: None at this time.

REC-ACTION: Use 'fmadm faulty' to provide a more detailed view of this event. Run 'zpool status -lx' for more information. Please refer to the associated reference document at <http://support.oracle.com/msg/ZFS-8000-QJ> for the latest service procedures and policies regarding this diagnosis.

To view more detailed information about the device problem and the resolution, use the `zpool status -v` command. For example:

```
zpool status -v
pool: pond
state: DEGRADED
status: One or more devices are unavailable in response to persistent errors.
Sufficient replicas exist for the pool to continue functioning in a
degraded state.
action: Determine if the device needs to be replaced, and clear the errors
using 'zpool clear' or 'fmadm repaired', or replace the device
with 'zpool replace'.
scan: scrub repaired 0 in 0h0m with 0 errors on Wed Jun 20 13:16:09 2012
config:
```

| NAME                  | STATE    | READ | WRITE | CKSUM |
|-----------------------|----------|------|-------|-------|
| pond                  | DEGRADED | 0    | 0     | 0     |
| mirror-0              | ONLINE   | 0    | 0     | 0     |
| c0t5000C500335F95E3d0 | ONLINE   | 0    | 0     | 0     |
| c0t5000C500335F907Fd0 | ONLINE   | 0    | 0     | 0     |
| mirror-1              | DEGRADED | 0    | 0     | 0     |
| c0t5000C500335BD117d0 | ONLINE   | 0    | 0     | 0     |
| c0t5000C500335DC60Fd0 | UNAVAIL  | 0    | 0     | 0     |

device details:

```
c0t5000C500335DC60Fd0 UNAVAIL cannot open
status: ZFS detected errors on this device.
The device was missing.
see: http://support.oracle.com/msg/ZFS-8000-LR for recovery
```

You can see from this output that the `c0t5000C500335DC60Fd0` device is not functioning. If you determine that the device is faulty, replace it.

If necessary, use the `zpool online` command to bring the replaced device online. For example:

```
zpool online pond c0t5000C500335DC60Fd0
```

Let FMA know that the device has been replaced if the output of the `fmadm faulty` identifies the device error. For example:

```
fmadm faulty
```

| TIME            | EVENT-ID                             | MSG-ID      | SEVERITY |
|-----------------|--------------------------------------|-------------|----------|
| Jun 20 13:15:41 | 3745f745-371c-c2d3-d940-93acbb881bd8 | ZFS-8000-LR | Major    |

```

Problem Status : solved
Diag Engine : zfs-diagnosis / 1.0
System
Manufacturer : unknown
Name : ORCL,SPARC-T3-4
Part_Number : unknown
Serial_Number : 1120BDRCCD
Host_ID : 84a02d28

```

```

Suspect 1 of 1 :
Fault class : fault.fs.zfs.open_failed
Certainty : 100%
Affects : zfs://pool=86124fa573cad84e/
 vdev=25d36cd46e0a7f49/pool_name=pond/
 vdev_name=id1,sd@n5000c500335dc60f/a
Status : faulted and taken out of service

```

```

FRU
Name : "zfs://pool=86124fa573cad84e/
 vdev=25d36cd46e0a7f49/pool_name=pond/
 vdev_name=id1,sd@n5000c500335dc60f/a"
Status : faulty

```

Description : ZFS device 'id1,sd@n5000c500335dc60f/a'  
in pool 'pond' failed to open.

Response : An attempt will be made to activate a hot spare if available.

Impact : Fault tolerance of the pool may be compromised.

Action : Use 'fmadm faulty' to provide a more detailed view of this event.  
Run 'zpool status -lx' for more information. Please refer to the  
associated reference document at  
<http://support.oracle.com/msg/ZFS-8000-LR> for the latest service  
procedures and policies regarding this diagnosis.

Extract the string in the Affects: section of the `fmadm faulty` output and include it with the  
following command to let FMA know that the device is replaced:

```

fmadm repaired zfs://pool=86124fa573cad84e/ \
 vdev=25d36cd46e0a7f49/pool_name=pond/ \
 vdev_name=id1,sd@n5000c500335dc60f/a
fmadm: recorded repair to of zfs://pool=86124fa573cad84e/
 vdev=25d36cd46e0a7f49/pool_name=pond/vdev_
name=id1,sd@n5000c500335dc60f/a

```

As a last step, confirm that the pool with the replaced device is healthy. For example:

```

zpool status -x system1
pool 'system1' is healthy

```

## Resolving a Removed Device

If a device is completely removed from the system, ZFS detects that the device cannot be opened and places it in the REMOVED state. Depending on the data replication level of the pool, this removal might or might not result in the entire pool becoming unavailable. If one disk in a mirrored or RAID-Z device is removed, the pool continues to be accessible. A pool might become UNAVAIL, which means no data is accessible until the device is reattached, under the following conditions:

If a redundant storage pool device is accidentally removed and reinserted, then you can just clear the device error, in most cases. For example:

```
zpool clear system1 c1t1d0
```

## Physically Reattaching a Device

Exactly how a missing device is reattached depends on the device in question. If the device is a network-attached drive, connectivity to the network should be restored. If the device is a USB device or other removable media, it should be reattached to the system. If the device is a local disk, a controller might have failed such that the device is no longer visible to the system. In this case, the controller should be replaced, at which point the disks will again be available. Other problems can exist and depend on the type of hardware and its configuration. If a drive fails and it is no longer visible to the system, the device should be treated as a damaged device. Follow the procedures in [“Replacing or Repairing a Damaged Device” on page 245](#).

A pool might be SUSPENDED if device connectivity is compromised. A SUSPENDED pool remains in the wait state until the device issue is resolved. For example:

```
zpool status cybermen
pool: cybermen
state: SUSPENDED
status: One or more devices are unavailable in response to IO failures.
The pool is suspended.
action: Make sure the affected devices are connected, then run 'zpool clear' or
'fmadm repaired'.
Run 'zpool status -v' to see device specific details.
see: http://support.oracle.com/msg/ZFS-8000-HC
scan: none requested
config:
```

| NAME     | STATE   | READ | WRITE | CKSUM |
|----------|---------|------|-------|-------|
| cybermen | UNAVAIL | 0    | 16    | 0     |
| c8t3d0   | UNAVAIL | 0    | 0     | 0     |
| c8t1d0   | UNAVAIL | 0    | 0     | 0     |

After device connectivity is restored, clear the pool or device errors.

```
zpool clear cybermen
fmadm repaired zfs://pool=name/vdev=guid
```

## Notifying ZFS of Device Availability

After a device is reattached to the system, ZFS might or might not automatically detect its availability. If the pool was previously UNAVAIL or SUSPENDED, or the system was rebooted as part of the attach procedure, then ZFS automatically rescans all devices when it tries to open the pool. If the pool was degraded and the device was replaced while the system was running, you must notify ZFS that the device is now available and ready to be reopened by using the `zpool online` command. For example:

```
zpool online system1 c0t1d0
```

For more information about bringing devices online, see [“Taking Devices in a Storage Pool Offline or Returning Online” on page 44](#).

## Replacing or Repairing a Damaged Device

This section describes how to determine device failure types, clear transient errors, and replacing a device.

### Determining the Type of Device Failure

The term *damaged device* is rather vague and can describe a number of possible situations:

- **Bit rot** – Over time, random events such as magnetic influences and cosmic rays can cause bits stored on disk to flip. These events are relatively rare but common enough to cause potential data corruption in large or long-running systems.
- **Misdirected reads or writes** – Firmware bugs or hardware faults can cause reads or writes of entire blocks to reference the incorrect location on disk. These errors are typically transient, though a large number of them might indicate a faulty drive.
- **Administrator error** – Administrators can unknowingly overwrite portions of a disk with bad data (such as copying `/dev/zero` over portions of the disk) that cause permanent corruption on disk. These errors are always transient.
- **Temporary outage** – A disk might become unavailable for a period of time, causing I/Os to fail. This situation is typically associated with network-attached devices, though local disks can experience temporary outages as well. These errors might or might not be transient.
- **Bad or flaky hardware** – This situation is a catch-all for the various problems that faulty hardware exhibits, including consistent I/O errors, faulty transports causing random corruption, or any number of failures. These errors are typically permanent.

- **Offline device** – If a device is offline, it is assumed that the administrator placed the device in this state because it is faulty. The administrator who placed the device in this state can determine if this assumption is accurate.

Determining exactly what is wrong with a device can be a difficult process. The first step is to examine the error counts in the `zpool status` output. For example:

```
zpool status -v system1
pool: system1
state: ONLINE
status: One or more devices has experienced an error resulting in data
corruption. Applications may be affected.
action: Restore the file in question if possible. Otherwise restore the
entire pool from backup.
see: http://support.oracle.com/msg/ZFS-8000-8A
config:

NAME STATE READ WRITE CKSUM
system1 ONLINE 2 0 0
 c8t0d0 ONLINE 0 0 0
 c8t0d0 ONLINE 2 0 0
```

errors: Permanent errors have been detected in the following files:

```
/system1/file.1
```

The errors are divided into I/O errors and checksum errors, both of which might indicate the possible failure type. Typical operation predicts a very small number of errors (just a few over long periods of time). If you are seeing a large number of errors, then this situation probably indicates impending or complete device failure. However, an administrator error can also result in large error counts. The other source of information is the `syslog` system log. If the log shows a large number of SCSI or Fibre Channel driver messages, then this situation probably indicates serious hardware problems. If no `syslog` messages are generated, then the damage is likely transient.

The goal is to answer the following question:

*Is another error likely to occur on this device?*

Errors that happen only once are considered *transient* and do not indicate potential failure. Errors that are persistent or severe enough to indicate potential hardware failure are considered *fatal*. The act of determining the type of error is beyond the scope of any automated software currently available with ZFS, and so much must be done manually by you, the administrator. After determination is made, the appropriate action can be taken. Either clear the transient errors or replace the device due to fatal errors. These repair procedures are described in the next sections.

Even if the device errors are considered transient, they still might have caused uncorrectable data errors within the pool. These errors require special repair procedures, even if the

underlying device is deemed healthy or otherwise repaired. For more information about repairing data errors, see [“Repairing Corrupted ZFS Data” on page 262](#).

## Clearing Transient or Persistent Device Errors

If the device errors are deemed transient, in that they are unlikely to affect the future health of the device, they can be safely cleared to indicate that no fatal error occurred. To clear error counters for RAID-Z or mirrored devices, use the `zpool clear` command. For example:

```
zpool clear system1 c1t1d0
```

This syntax clears any device errors and clears any data error counts associated with the device.

To clear all errors associated with the virtual devices in a pool, and to clear any data error counts associated with the pool, use the following syntax:

```
zpool clear system1
```

For more information about clearing pool errors, see [“Clearing Storage Pool Device Errors” on page 46](#).

Transient device errors are most likely cleared by using the `zpool clear` command. If a device has failed, then see the next section about replacing a device. If a redundant device was accidentally overwritten or was UNAVAIL for a long period of time, then this error might need to be resolved by using the `fmadm repaired` command as directed in the `zpool status` output. For example:

```
zpool status -v pond
pool: pond
state: DEGRADED
status: One or more devices are unavailable in response to persistent errors.
Sufficient replicas exist for the pool to continue functioning in a
degraded state.
action: Determine if the device needs to be replaced, and clear the errors
using 'zpool clear' or 'fmadm repaired', or replace the device
with 'zpool replace'.
scan: scrub repaired 0 in 0h0m with 0 errors on Wed Jun 20 15:38:08 2012
config:
```

| NAME                  | STATE    | READ | WRITE | CKSUM |
|-----------------------|----------|------|-------|-------|
| pond                  | DEGRADED | 0    | 0     | 0     |
| mirror-0              | DEGRADED | 0    | 0     | 0     |
| c0t5000C500335F95E3d0 | ONLINE   | 0    | 0     | 0     |
| c0t5000C500335F907Fd0 | UNAVAIL  | 0    | 0     | 0     |
| mirror-1              | ONLINE   | 0    | 0     | 0     |
| c0t5000C500335BD117d0 | ONLINE   | 0    | 0     | 0     |
| c0t5000C500335DC60Fd0 | ONLINE   | 0    | 0     | 0     |

device details:

```
c0t5000C500335F907Fd0 UNAVAIL cannot open
status: ZFS detected errors on this device.
The device was missing.
see: http://support.oracle.com/msg/ZFS-8000-LR for recovery
```

errors: No known data errors

## Replacing a Device in a ZFS Storage Pool

If device damage is permanent or future permanent damage is likely, the device must be replaced. Whether the device can be replaced depends on the configuration.

- [“Determining If a Device Can Be Replaced” on page 248.](#)
- [“Devices That Cannot be Replaced” on page 249.](#)
- [“Replacing a Device in a ZFS Storage Pool” on page 249.](#)
- [“Viewing Resilvering Status” on page 253.](#)

### Determining If a Device Can Be Replaced

If the device to be replaced is part of a redundant configuration, sufficient replicas from which to retrieve good data must exist. For example, if two disks in a four-way mirror are UNAVAIL, then either disk can be replaced because healthy replicas are available. However, if two disks in a four-way RAID-Z (raidz1) virtual device are UNAVAIL, then neither disk can be replaced because insufficient replicas from which to retrieve data exist. If the device is damaged but otherwise online, it can be replaced as long as the pool is not in the UNAVAIL state. However, any corrupted data on the device is copied to the new device, unless sufficient replicas with good data exist.

In the following configuration, the `c1t1d0` disk can be replaced, and any data in the pool is copied from the healthy replica, `c1t0d0`:

|        |          |
|--------|----------|
| mirror | DEGRADED |
| c1t0d0 | ONLINE   |
| c1t1d0 | UNAVAIL  |

The `c1t0d0` disk can also be replaced, though no self-healing of data can take place because no good replica is available.

In the following configuration, neither UNAVAIL disk can be replaced. The ONLINE disks cannot be replaced either because the pool itself is UNAVAIL.

|        |         |
|--------|---------|
| raidz1 | UNAVAIL |
|--------|---------|



```

c1t0d0 ONLINE
c2t0d0 UNAVAIL
c3t0d0 UNAVAIL
c4t0d0 ONLINE

```

In the following configuration, either top-level disk can be replaced, though any bad data present on the disk is copied to the new disk.

```

c1t0d0 ONLINE
c1t1d0 ONLINE

```

If either disk is UNAVAIL, then no replacement can be performed because the pool itself is UNAVAIL.

## Devices That Cannot be Replaced

If the loss of a device causes the pool to become UNAVAIL or the device contains too many data errors in a non-redundant configuration, then the device cannot be safely replaced. Without sufficient redundancy, no good data with which to heal the damaged device exists. In this case, the only option is to destroy the pool and re-create the configuration, and then to restore your data from a backup copy.

For more information about restoring an entire pool, see [“Repairing ZFS Storage Pool-Wide Damage” on page 266](#).

## Replacing a Device in a ZFS Storage Pool

After you have determined that a device can be replaced, use the `zpool replace` command to replace the device. If you are replacing the damaged device with different device, use syntax similar to the following:

```
zpool replace system1 c1t1d0 c2t0d0
```

This command migrates data to the new device from the damaged device or from other devices in the pool if it is in a redundant configuration. When the command is finished, it detaches the damaged device from the configuration, at which point the device can be removed from the system. If you have already removed the device and replaced it with a new device in the same location, use the single device form of the command. For example:

```
zpool replace system1 c1t1d0
```

This command takes an unformatted disk, formats it appropriately, and then resilvers data from the rest of the configuration.

For more information about the `zpool replace` command, see [“Replacing Devices in a Storage Pool” on page 46](#).

**EXAMPLE 58** Replacing a SATA Disk in a ZFS Storage Pool

The following example shows how to replace a device (c1t3d0) in a mirrored storage pool system1 on a system with SATA devices. To replace the disk c1t3d0 with a new disk at the same location (c1t3d0), then you must unconfigure the disk before you attempt to replace it. If the disk to be replaced is not a SATA disk, then see [“Replacing Devices in a Storage Pool” on page 46](#).

The basic steps follow:

- Take offline the disk (c1t3d0) to be replaced. You cannot unconfigure a SATA disk that is currently being used.
- Use the `cfgadm` command to identify the SATA disk (c1t3d0) to be unconfigured and unconfigure it. The pool will be degraded with the offline disk in this mirrored configuration, but the pool will continue to be available.
- Physically replace the disk (c1t3d0). Ensure that the blue Ready to Remove LED is illuminated before you physically remove the UNAVAIL drive, if available.
- Reconfigure the SATA disk (c1t3d0).
- Bring the new disk (c1t3d0) online.
- Run the `zpool replace` command to replace the disk (c1t3d0).

---

**Note** - If you had previously set the pool property `autoreplace` to on, then any new device, found in the same physical location as a device that previously belonged to the pool is automatically formatted and replaced without using the `zpool replace` command. This feature might not be supported on all hardware.

---

- If a failed disk is automatically replaced with a hot spare, you might need to detach the hot spare after the failed disk is replaced. For example, if c2t4d0 is still an active hot spare after the failed disk is replaced, then detach it.

```
zpool detach system1 c2t4d0
```

- If FMA is reporting the failed device, then you should clear the device failure.

```
fmadm faulty
```

```
fmadm repaired zfs://pool=name/vdev=guid
```

The following example walks through the steps to replace a disk in a ZFS storage pool.

```
zpool offline system1 c1t3d0
cfgadm | grep c1t3d0
sata1/3::disk/c1t3d0 disk connected configured ok
cfgadm -c unconfigure sata1/3
Unconfigure the device at: /devices/pci@0,0/pci1022,7458@2/pci11ab,11ab@1:3
This operation will suspend activity on the SATA device
Continue (yes/no)? yes
```

```
cfgadm | grep sata1/3
sata1/3 disk connected unconfigured ok
<Physically replace the failed disk c1t3d0>
cfgadm -c configure sata1/3
cfgadm | grep sata1/3
sata1/3::disk/c1t3d0 disk connected configured ok
zpool online system1 c1t3d0
zpool replace system1 c1t3d0
zpool status system1
pool: system1
state: ONLINE
scrub: resilver completed after 0h00m with 0 errors on Tue Feb 2 13:17:32 2010
config:
```

| NAME     | STATE  | READ | WRITE | CKSUM |
|----------|--------|------|-------|-------|
| system1  | ONLINE | 0    | 0     | 0     |
| mirror-0 | ONLINE | 0    | 0     | 0     |
| c0t1d0   | ONLINE | 0    | 0     | 0     |
| c1t1d0   | ONLINE | 0    | 0     | 0     |
| mirror-1 | ONLINE | 0    | 0     | 0     |
| c0t2d0   | ONLINE | 0    | 0     | 0     |
| c1t2d0   | ONLINE | 0    | 0     | 0     |
| mirror-2 | ONLINE | 0    | 0     | 0     |
| c0t3d0   | ONLINE | 0    | 0     | 0     |
| c1t3d0   | ONLINE | 0    | 0     | 0     |

errors: No known data errors

Note that the preceding `zpool` output might show both the new and old disks under a *replacing* heading. For example:

|            |          |   |   |   |
|------------|----------|---|---|---|
| replacing  | DEGRADED | 0 | 0 | 0 |
| c1t3d0s0/o | FAULTED  | 0 | 0 | 0 |
| c1t3d0     | ONLINE   | 0 | 0 | 0 |

This text means that the replacement process is in progress and the new disk is being resilvered.

If you are going to replace a disk (`c1t3d0`) with another disk (`c4t3d0`), then you only need to run the `zpool replace` command. For example:

```
zpool replace system1 c1t3d0 c4t3d0
zpool status
pool: system1
state: DEGRADED
scrub: resilver completed after 0h00m with 0 errors on Tue Feb 2 13:35:41 2010
config:
```

| NAME     | STATE    | READ | WRITE | CKSUM |
|----------|----------|------|-------|-------|
| system1  | DEGRADED | 0    | 0     | 0     |
| mirror-0 | ONLINE   | 0    | 0     | 0     |
| c0t1d0   | ONLINE   | 0    | 0     | 0     |

|           |          |   |   |   |
|-----------|----------|---|---|---|
| c1t1d0    | ONLINE   | 0 | 0 | 0 |
| mirror-1  | ONLINE   | 0 | 0 | 0 |
| c0t2d0    | ONLINE   | 0 | 0 | 0 |
| c1t2d0    | ONLINE   | 0 | 0 | 0 |
| mirror-2  | DEGRADED | 0 | 0 | 0 |
| c0t3d0    | ONLINE   | 0 | 0 | 0 |
| replacing | DEGRADED | 0 | 0 | 0 |
| c1t3d0    | OFFLINE  | 0 | 0 | 0 |
| c4t3d0    | ONLINE   | 0 | 0 | 0 |

errors: No known data errors

You might need to run the `zpool status` command several times until the disk replacement is completed.

```
zpool status system1
```

```
pool: system1
```

```
state: ONLINE
```

```
scrub: resilver completed after 0h0m with 0 errors on Tue Feb 2 13:35:41 2010
```

```
config:
```

| NAME     | STATE  | READ | WRITE | CKSUM |
|----------|--------|------|-------|-------|
| system1  | ONLINE | 0    | 0     | 0     |
| mirror-0 | ONLINE | 0    | 0     | 0     |
| c0t1d0   | ONLINE | 0    | 0     | 0     |
| c1t1d0   | ONLINE | 0    | 0     | 0     |
| mirror-1 | ONLINE | 0    | 0     | 0     |
| c0t2d0   | ONLINE | 0    | 0     | 0     |
| c1t2d0   | ONLINE | 0    | 0     | 0     |
| mirror-2 | ONLINE | 0    | 0     | 0     |
| c0t3d0   | ONLINE | 0    | 0     | 0     |
| c4t3d0   | ONLINE | 0    | 0     | 0     |

#### EXAMPLE 59 Replacing a Failed Log Device

ZFS identifies intent log failures in the `zpool status` command output. Fault Management Architecture (FMA) reports these errors as well. Both ZFS and FMA describe how to recover from an intent log failure.

The following example shows how to recover from a failed log device (c0t5d0) in the storage pool (storpool). The basic steps follow:

- Review the `zpool status -x` output and FMA diagnostic message, described in *ZFS intent log read failure (Doc ID 1021625.1)* in <https://support.oracle.com/>.
- Physically replace the failed log device.
- Bring the new log device online.
- Clear the pool's error condition.
- Clear the FMA error.

For example, if the system shuts down abruptly before synchronous write operations are committed to a pool with a separate log device, you see messages similar to the following:

```
zpool status -x
pool: storpool
state: FAULTED
status: One or more of the intent logs could not be read.
Waiting for administrator intervention to fix the faulted pool.
action: Either restore the affected device(s) and run 'zpool online',
or ignore the intent log records by running 'zpool clear'.
scrub: none requested
config:

NAME STATE READ WRITE CKSUM
storpool FAULTED 0 0 0 bad intent log
 mirror-0 ONLINE 0 0 0
 c0t1d0 ONLINE 0 0 0
 c0t4d0 ONLINE 0 0 0
 logs FAULTED 0 0 0 bad intent log
 c0t5d0 UNAVAIL 0 0 0 cannot open
<Physically replace the failed log device>
zpool online storpool c0t5d0
zpool clear storpool
fmadm faulty
fmadm repair zfs://pool=name/vdev=guid
```

You can resolve the log device failure in the following ways:

- Replace or recover the log device. In this example, the log device is `c0t5d0`.
- Bring the log device back online.

```
zpool online storpool c0t5d0
```

- Reset the failed log device error condition.

```
zpool clear storpool
```

To recover from this error without replacing the failed log device, you can clear the error with the `zpool clear` command. In this scenario, the pool will operate in a degraded mode and the log records will be written to the main pool until the separate log device is replaced.

Consider using mirrored log devices to avoid the log device failure scenario.

## Viewing Resilvering Status

The process of replacing a device can take an extended period of time, depending on the size of the device and the amount of data in the pool. The process of moving data from one device to another device is known as *resilvering* and can be monitored by using the `zpool status` command.

The following `zpool status` resilver status messages are provided:

- Resilver in-progress report. For example:

```
scan: resilver in progress since Mon Jun 7 09:17:27 2010
13.3G scanned
13.3G resilvered at 18.5M/s, 82.34% done, 0h2m to go
```

- Resilver completion message. For example:

```
resilvered 16.2G in 0h16m with 0 errors on Mon Jun 7 09:34:21 2010
```

Resilver completion messages persist across system reboots.

Traditional file systems resilver data at the block level. Because ZFS eliminates the artificial layering of the volume manager, it can perform resilvering in a much more powerful and controlled manner. The two main advantages of this feature are as follows:

- ZFS only resilvers the minimum amount of necessary data. In the case of a short outage (as opposed to a complete device replacement), the entire disk can be resilvered in a matter of minutes or seconds. When an entire disk is replaced, the resilvering process takes time proportional to the amount of data used on disk. Replacing a 500-GB disk can take seconds if a pool has only a few gigabytes of used disk space.
- If the system loses power or is rebooted, the resilvering process resumes exactly where it left off, without any need for manual intervention.

To view the resilvering process, use the `zpool status` command. For example:

```
zpool status system1
pool: system1
state: ONLINE
status: One or more devices is currently being resilvered. The pool will
continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
scan: resilver in progress since Mon Jun 7 10:49:20 2010
54.6M scanned54.5M resilvered at 5.46M/s, 24.64% done, 0h0m to go

config:
```

| NAME        | STATE  | READ | WRITE | CKSUM           |
|-------------|--------|------|-------|-----------------|
| system1     | ONLINE | 0    | 0     | 0               |
| mirror-0    | ONLINE | 0    | 0     | 0               |
| replacing-0 | ONLINE | 0    | 0     | 0               |
| c1t0d0      | ONLINE | 0    | 0     | 0               |
| c2t0d0      | ONLINE | 0    | 0     | 0 (resilvering) |
| c1t1d0      | ONLINE | 0    | 0     | 0               |

In this example, the disk `c1t0d0` is being replaced by `c2t0d0`. This event is observed in the status output by the presence of the `replacing` virtual device in the configuration. This device is not real, nor is it possible for you to create a pool by using it. The purpose of this device is solely to display the resilvering progress and to identify which device is being replaced.

Note that any pool currently undergoing resilvering is placed in the **ONLINE** or **DEGRADED** state because the pool cannot provide the desired level of redundancy until the resilvering process is completed. Resilvering proceeds as fast as possible, though the I/O is always scheduled with a lower priority than user-requested I/O, to minimize impact on the system. After the resilvering is completed, the configuration reverts to the new, complete, configuration. For example:

```
zpool status system1
pool: system1
state: ONLINE
scrub: resilver completed after 0h1m with 0 errors on Tue Feb 2 13:54:30 2010
config:

NAME STATE READ WRITE CKSUM
system1 ONLINE 0 0 0
 mirror-0 ONLINE 0 0 0
 c2t0d0 ONLINE 0 0 0 377M resilvered
 c1t1d0 ONLINE 0 0 0

errors: No known data errors
```

The pool is once again **ONLINE**, and the original failed disk (**c1t0d0**) has been removed from the configuration.

## Changing Pool Devices

Do not try to change pool devices under an active pool.

Disks are identified both by their path and by their device ID, if available. On systems where device ID information is available, this identification method allows devices to be reconfigured without updating ZFS. Because device ID generation and management can vary by system, export the pool first before moving devices, such as moving a disk from one controller to another controller. A system event, such as a firmware update or other hardware change, might change the device IDs in your ZFS storage pool, which can cause the devices to become unavailable.

An additional problem is that if you attempt to change the devices underneath a pool and then you use the `zpool status` command as a non-root user, the previous device names could be displayed.

## Resolving Data Problems in a ZFS Storage Pool

Examples of data problems include the following:

- Pool or file system space is missing

- Transient I/O errors due to a bad disk or controller
- On-disk data corruption due to cosmic rays
- Driver bugs resulting in data being transferred to or from the wrong location
- A user overwriting portions of the physical device by accident

In some cases, these errors are transient, such as a random I/O error while the controller is having problems. In other cases, the damage is permanent, such as on-disk corruption. Even still, whether the damage is permanent does not necessarily indicate that the error is likely to occur again. For example, if you accidentally overwrite part of a disk, no type of hardware failure has occurred, and the device does not need to be replaced. Identifying the exact problem with a device is not an easy task and is covered in more detail in a later section.

## Resolving ZFS Space Issues

Review the following sections if you are unsure how ZFS reports file system and pool space accounting.

### ZFS File System Space Reporting

The `zpool list` and `zfs list` commands are better than the previous `df` and `du` commands for determining your available pool and file system space. With the legacy commands, you cannot easily discern between pool and file system space, nor do the legacy commands account for space that is consumed by descendant file systems or snapshots.

For example, the following root pool (`rpool`) has 5.46 GB allocated and 68.5 GB free.

```
zpool list rpool
NAME SIZE ALLOC FREE CAP DEDUP HEALTH ALTROOT
rpool 74G 5.46G 68.5G 7% 1.00x ONLINE -
```

If you compare the pool space accounting with the file system space accounting by reviewing the `USED` column of your individual file systems, you can see that the pool space that is reported in `ALLOC` is accounted for in the file systems' `USED` total. For example:

```
zfs list -r rpool
NAME USED AVAIL REFER MOUNTPOINT
rpool 5.41G 67.4G 74.5K /rpool
rpool/ROOT 3.37G 67.4G 31K legacy
rpool/ROOT/solaris 3.37G 67.4G 3.07G /
rpool/ROOT/solaris/var 302M 67.4G 214M /var
rpool/dump 1.01G 67.5G 1000M -
rpool/export 97.5K 67.4G 32K /rpool/export
rpool/export/home 65.5K 67.4G 32K /rpool/export/home
rpool/export/home/admin 33.5K 67.4G 33.5K /rpool/export/home/admin
rpool/swap 1.03G 67.5G 1.00G -
```



## ZFS Storage Pool Space Reporting

The `SIZE` value that is reported by the `zpool list` command is generally the amount of physical disk space in the pool, but varies depending on the pool's redundancy level. See the examples below. The `zfs list` command lists the usable space that is available to file systems, which is disk space minus ZFS pool redundancy metadata overhead, if any.

The following ZFS dataset configurations are tracked as allocated space by the `zfs list` command but they are not tracked as allocated space in the `zpool list` output:

- ZFS file system quota
- ZFS file system reservation
- ZFS logical volume size

The following items describe how using different pool configurations, ZFS volumes and ZFS reservations can impact your consumed and available disk space. Depending upon your configuration, monitoring pool space should be tracked by using the steps listed below.

- **Non-redundant storage pool** – When a pool is created with one 136-GB disk, the `zpool list` command reports `SIZE` and initial `FREE` values as 136 GB. The initial `AVAIL` space reported by the `zfs list` command is 134 GB, due to a small amount of pool metadata overhead. For example:

```
zpool create system1 c0t6d0
zpool list system1
NAME SIZE ALLOC FREE CAP DEDUP HEALTH ALTROOT
system1 136G 95.5K 136G 0% 1.00x ONLINE -
zfs list system1
NAME USED AVAIL REFER MOUNTPOINT
system1 72K 134G 21K /system1
```

- **Mirrored storage pool** – When a pool is created with two 136-GB disks, `zpool list` command reports `SIZE` as 136 GB and initial `FREE` value as 136 GB. This reporting is referred to as the *deflated* space value. The initial `AVAIL` space reported by the `zfs list` command is 134 GB, due to a small amount of pool metadata overhead. For example:

```
zpool create system1 mirror c0t6d0 c0t7d0
zpool list system1
NAME SIZE ALLOC FREE CAP DEDUP HEALTH ALTROOT
system1 136G 95.5K 136G 0% 1.00x ONLINE -
zfs list system1
NAME USED AVAIL REFER MOUNTPOINT
system1 72K 134G 21K /system1
```

- **RAID-Z storage pool** – When a `raidz2` pool is created with three 136-GB disks, the `zpool list` commands reports `SIZE` as 408 GB and initial `FREE` value as 408 GB. This reporting is referred to as the *inflated* disk space value, which includes redundancy overhead, such as parity information. The initial `AVAIL` space reported by the `zfs list` command is 133

GB, due to the pool redundancy overhead. The space discrepancy between the `zpool list` and the `zfs list` output for a RAID-Z pool is because `zpool list` reports the inflated pool space.

```
zpool create system1 raidz2 c0t6d0 c0t7d0 c0t8d0
zpool list system1
NAME SIZE ALLOC FREE CAP DEDUP HEALTH ALTROOT
system1 408G 286K 408G 0% 1.00x ONLINE -
zfs list system1
NAME USED AVAIL REFER MOUNTPOINT
system1 73.2K 133G 20.9K /system1
```

- **NFS mounted file system space** – Neither the `zpool list` or the `zfs list` account for NFS mounted file system space. However, local data files can be hidden under a mounted NFS file system. If you are missing file system space, ensure that you do not have data files hidden under an NFS file system.
- **Using ZFS Volumes** – When a ZFS file system is created and pool space is consumed, you can view the file system space consumption by using the `zpool list` command. For example:

```
zpool create nova mirror c1t1d0 c2t1d0
zfs create nova/fs1
mkfile 10g /nova/fs1/file1_10g
zpool list nova
NAME SIZE ALLOC FREE CAP DEDUP HEALTH ALTROOT
nova 68G 10.0G 58.0G 14% 1.00x ONLINE -
zfs list -r nova
NAME USED AVAIL REFER MOUNTPOINT
nova 10.0G 56.9G 32K /nova
nova/fs1 10.0G 56.9G 10.0G /nova/fs1
```

If you create a 10-GB ZFS volume, the space is not accounted for in the `zpool list` command. The space is accounted for in the `zfs list` command. If you are using ZFS volumes in your storage pools, monitor ZFS volume space consumption by using the `zfs list` command. For example:

```
zfs create -V 10g nova/vol1
zpool list nova
NAME SIZE ALLOC FREE CAP DEDUP HEALTH ALTROOT
nova 68G 10.0G 58.0G 14% 1.00x ONLINE -
zfs list -r nova
NAME USED AVAIL REFER MOUNTPOINT
nova 20.3G 46.6G 32K /nova
nova/fs1 10.0G 46.6G 10.0G /nova/fs1
nova/vol1 10.3G 56.9G 16K -
```

Note in the above output that ZFS volume space is not tracked in the `zpool list` output so use the `zfs list` or the `zfs list -o space` command to identify space that is consumed by ZFS volumes.

In addition, because ZFS volumes act like raw devices, some amount of space for metadata is automatically reserved through the refreservation property, which causes volumes to consume slightly more space than the amount specified when the volume was created. Do not remove the refreservation on ZFS volumes or you risk running out of volume space.

- **Using ZFS Reservations** – If you create a file system with a reservation or add a reservation to an existing file system, reservations or refreservations are not tracked by the `zpool list` command.

Identify space that is consumed by file system reservations by using the `zfs list -r` command to identify the increased USED space. For example:

```
zfs create -o reservation=10g nova/fs2
zpool list nova
NAME SIZE ALLOC FREE CAP DEDUP HEALTH ALTROOT
nova 68G 10.0G 58.0G 14% 1.00x ONLINE -
zfs list -r nova
NAME USED AVAIL REFER MOUNTPOINT
nova 30.3G 36.6G 33K /nova
nova/fs1 10.0G 36.6G 10.0G /nova/fs1
nova/fs2 31K 46.6G 31K /nova/fs2
nova/vol1 10.3G 46.9G 16K -
```

If you create a file system with a refreservation, it can be identified by using the `zfs list -r` command. For example:

```
zfs create -o refreservation=10g nova/fs3
zfs list -r nova
NAME USED AVAIL REFER MOUNTPOINT
nova 40.3G 26.6G 35K /nova
nova/fs1 10.0G 26.6G 10.0G /nova/fs1
nova/fs2 31K 36.6G 31K /nova/fs2
nova/fs3 10G 36.6G 31K /nova/fs3
nova/vol1 10.3G 36.9G 16K -
```

Use the following command to identify all existing reservations to account for total USED space.

```
zfs get -r reserv,refreserv nova
NAME PROPERTY VALUE SOURCE
nova reservation none default
nova refreservation none default
nova/fs1 reservation none default
nova/fs1 refreservation none default
nova/fs2 reservation 10G local
nova/fs2 refreservation none default
nova/fs3 reservation none default
nova/fs3 refreservation 10G local
```

```
nova/vol1 reservation none default
nova/vol1 refreservation 10.3G local
```

## Checking ZFS File System Integrity

No `fsck` utility equivalent exists for ZFS. This utility has traditionally served two purposes, those of file system repair and file system validation.

### File System Repair

With traditional file systems, the way in which data is written is inherently vulnerable to unexpected failure causing file system inconsistencies. Because a traditional file system is not transactional, unreferenced blocks, bad link counts, or other inconsistent file system structures are possible. The addition of journaling does solve some of these problems, but can introduce additional problems when the log cannot be rolled back. The only way for inconsistent data to exist on disk in a ZFS configuration is through hardware failure (in which case the pool should have been redundant) or when a bug exists in the ZFS software.

The `fsck` utility repairs known problems specific to UFS file systems. Most ZFS storage pool problems are generally related to failing hardware or power failures. Many problems can be avoided by using redundant pools. If your pool is damaged due to failing hardware or a power outage, see [“Repairing ZFS Storage Pool-Wide Damage” on page 266](#).

If your pool is not redundant, the risk that file system corruption can render some or all of your data inaccessible is always present.

### File System Validation

In addition to performing file system repair, the `fsck` utility validates that the data on disk has no problems. Traditionally, this task requires unmounting the file system and running the `fsck` utility, possibly taking the system to single-user mode in the process. This scenario results in downtime that is proportional to the size of the file system being checked. Instead of requiring an explicit utility to perform the necessary checking, ZFS provides a mechanism to perform routine checking of all inconsistencies. This feature, known as *scrubbing*, is commonly used in memory and other systems as a method of detecting and preventing errors before they result in a hardware or software failure.

## Controlling ZFS Data Scrubbing

Whenever ZFS encounters an error, either through scrubbing or when accessing a file on demand, the error is logged internally so that you can obtain a quick overview of all known errors within the pool.

### Explicit ZFS Data Scrubbing

The simplest way to check data integrity is to initiate an explicit scrubbing of all data within the pool. This operation traverses all the data in the pool once and verifies that all blocks can be read. Scrubbing proceeds as fast as the devices allow, though the priority of any I/O remains below that of normal operations. This operation might negatively impact performance, though the pool's data should remain usable and nearly as responsive while the scrubbing occurs. To initiate an explicit scrub, use the `zpool scrub` command. For example:

```
zpool scrub system1
```

The status of the current scrubbing operation can be displayed by using the `zpool status` command. For example:

```
zpool status -v system1
pool: system1
state: ONLINE
scan: scrub in progress since Mon Jun 7 12:07:52 2010
201M scanned out of 222M at 9.55M/s, 0h0m to go
0 repaired, 90.44% done
config:
```

| NAME     | STATE  | READ | WRITE | CKSUM |
|----------|--------|------|-------|-------|
| system1  | ONLINE | 0    | 0     | 0     |
| mirror-0 | ONLINE | 0    | 0     | 0     |
| c1t0d0   | ONLINE | 0    | 0     | 0     |
| c1t1d0   | ONLINE | 0    | 0     | 0     |

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

Only one active scrubbing operation per pool can occur at one time.

You can stop a scrubbing operation that is in progress by using the `-s` option. For example:

```
zpool scrub -s system1
```

In most cases, a scrubbing operation to ensure data integrity should continue to completion. Stop a scrubbing operation at your own discretion if system performance is impacted by the operation.

Performing routine scrubbing guarantees continuous I/O to all disks on the system. Routine scrubbing has the side effect of preventing power management from placing idle disks in low-

power mode. If the system is generally performing I/O all the time, or if power consumption is not a concern, then this issue can safely be ignored. If the system is largely idle, and you want to conserve power to the disks, you should consider using a cron scheduled explicit scrub rather than background scrubbing. This will still perform complete scrubs of data, though it will only generate a large amount of I/O until the scrubbing is finished, at which point the disks can be power managed as normal. The downside (besides increased I/O) is that there will be large periods of time when no scrubbing is being done at all, potentially increasing the risk of corruption during those periods.

For more information about interpreting `zpool status` output, see [“Querying ZFS Storage Pool Status” on page 55](#).

## ZFS Data Scrubbing and Resilvering

When a device is replaced, a resilvering operation is initiated to move data from the good copies to the new device. This action is a form of disk scrubbing. Therefore, only one such action can occur at a given time in the pool. If a scrubbing operation is in progress, a resilvering operation suspends the current scrubbing and restarts it after the resilvering is completed.

For more information about resilvering, see [“Viewing Resilvering Status” on page 253](#).

## Repairing Corrupted ZFS Data

Data corruption occurs when one or more device errors (indicating one or more missing or damaged devices) affects a top-level virtual device. For example, one half of a mirror can experience thousands of device errors without ever causing data corruption. If an error is encountered on the other side of the mirror in the exact same location, corrupted data is the result.

Data corruption is always permanent and requires special consideration during repair. Even if the underlying devices are repaired or replaced, the original data is lost forever. Most often, this scenario requires restoring data from backups. Data errors are recorded as they are encountered, and they can be controlled through routine pool scrubbing as explained in the following section. When a corrupted block is removed, the next scrubbing pass recognizes that the corruption is no longer present and removes any trace of the error from the system.

The following sections describe how to identify the type of data corruption and how to repair the data, if possible.

- [“Identifying the Type of Data Corruption” on page 263](#).
- [“Repairing a Corrupted File or Directory” on page 264](#).
- [“Repairing ZFS Storage Pool-Wide Damage” on page 266](#).

ZFS uses checksums, redundancy, and self-healing data to minimize the risk of data corruption. Nonetheless, data corruption can occur if a pool isn't redundant, if corruption occurred while a pool was degraded, or an unlikely series of events conspired to corrupt multiple copies of a piece of data. Regardless of the source, the result is the same: The data is corrupted and therefore no longer accessible. The action taken depends on the type of data being corrupted and its relative value. Two basic types of data can be corrupted:

- **Pool metadata** – ZFS requires a certain amount of data to be parsed to open a pool and access datasets. If this data is corrupted, the entire pool or portions of the dataset hierarchy will become unavailable.
- **Object data** – In this case, the corruption is within a specific file or directory. This problem might result in a portion of the file or directory being inaccessible, or this problem might cause the object to be broken altogether.

Data is verified during normal operations as well as through a scrubbing. For information about how to verify the integrity of pool data, see [“Checking ZFS File System Integrity” on page 260](#).

## Identifying the Type of Data Corruption

By default, the `zpool status` command shows only that corruption has occurred, but not where this corruption occurred. For example:

```
zpool status system1
pool: system1
state: ONLINE
status: One or more devices has experienced an error resulting in data
corruption. Applications may be affected.
action: Restore the file in question if possible. Otherwise restore the
entire pool from backup.
see: http://support.oracle.com/msg/ZFS-8000-8A
config:
```

| NAME                  | STATE  | READ | WRITE | CKSUM |
|-----------------------|--------|------|-------|-------|
| system1               | ONLINE | 4    | 0     | 0     |
| c0t5000C500335E106Bd0 | ONLINE | 0    | 0     | 0     |
| c0t5000C500335FC3E7d0 | ONLINE | 4    | 0     | 0     |

```
errors: 2 data errors, use '-v' for a list
```

Each error indicates only that an error occurred at a given point in time. Each error is not necessarily still present on the system. Under normal circumstances, this is the case. Certain temporary outages might result in data corruption that is automatically repaired after the outage ends. A complete scrub of the pool is guaranteed to examine every active block in the pool, so the error log is reset whenever a scrub finishes. If you determine that the errors are no longer

present, and you don't want to wait for a scrub to complete, reset all errors in the pool by using the `zpool online` command.

If the data corruption is in pool-wide metadata, the output is slightly different. For example:

```
zpool status -v morpheus
pool: morpheus
id: 13289416187275223932
state: UNAVAIL
status: The pool metadata is corrupted.
action: The pool cannot be imported due to damaged devices or data.
see: http://support.oracle.com/msg/ZFS-8000-72
config:

morpheus FAULTED corrupted data
c1t10d0 ONLINE
```

In the case of pool-wide corruption, the pool is placed into the `FAULTED` state because the pool cannot provide the required redundancy level.

## Repairing a Corrupted File or Directory

If a file or directory is corrupted, the system might still function, depending on the type of corruption. Any damage is effectively unrecoverable if no good copies of the data exist on the system. If the data is valuable, you must restore the affected data from backup. Even so, you might be able to recover from this corruption without restoring the entire pool.

If the damage is within a file data block, then the file can be safely removed, thereby clearing the error from the system. Use the `zpool status -v` command to display a list of file names with persistent errors. For example:

```
zpool status system1 -v
pool: system1
state: ONLINE
status: One or more devices has experienced an error resulting in data
corruption. Applications may be affected.
action: Restore the file in question if possible. Otherwise restore the
entire pool from backup.
see: http://support.oracle.com/msg/ZFS-8000-8A
config:

NAME STATE READ WRITE CKSUM
system1 ONLINE 4 0 0
 c0t5000C500335E106Bd0 ONLINE 0 0 0
 c0t5000C500335FC3E7d0 ONLINE 4 0 0

errors: Permanent errors have been detected in the following files:
/system1/file.1
```



```
/system1/file.2
```

The list of file names with persistent errors might be described as follows:

- If the full path to the file is found and the dataset is mounted, the full path to the file is displayed. For example:

```
/path1/a.txt
```

- If the full path to the file is found, but the dataset is not mounted, then the dataset name with no preceding slash (/), followed by the path within the dataset to the file, is displayed. For example:

```
path1/documents/e.txt
```

- If the object number to a file path cannot be successfully translated, either due to an error or because the object doesn't have a real file path associated with it, as is the case for a `dnode_t`, then the dataset name followed by the object's number is displayed. For example:

```
path1/dnode:<0x0>
```

- If an object in the metaobject set (MOS) is corrupted, then a special tag of `<metadata>`, followed by the object number, is displayed.

You can attempt to resolve more minor data corruption by using scrubbing the pool and clearing the pool errors in multiple iterations. If the first scrub and clear iteration does not resolve the corrupted files, run them again. For example:

```
zpool scrub system1
zpool clear system1
```

If the corruption is within a directory or a file's metadata, the only choice is to move the file elsewhere. You can safely move any file or directory to a less convenient location, allowing the original object to be restored in its place.

## Repairing Corrupted Data With Multiple Block References

If a damaged file system has corrupted data with multiple block references, such as snapshots, the `zpool status -v` command cannot display **all** corrupted data paths. The current `zpool status` reporting of corrupted data is limited by the amount of metadata corruption and if any blocks have been reused after the `zpool status` command is executed. Deduplicated blocks makes reporting all corrupted data even more complicated.

If you have corrupted data and the `zpool status -v` command identifies that snapshot data is impacted, then considering running the following command to identify additional corrupted paths:

```
find mount-point -inum $inode -print
find mount-point/.zfs/snapshot -inum $inode -print
```

The first command searches for the inode number of the reported corrupted data in the specified file system and all its snapshots. The second command searches for snapshots with the same inode number.

## Repairing ZFS Storage Pool-Wide Damage

If the damage is in pool metadata and that damage prevents the pool from being opened or imported, then the following options are available to you:

- You can attempt to recover the pool by using the `zpool clear -F` command or the `zpool import -F` command. These commands attempt to roll back the last few pool transactions to an operational state. You can use the `zpool status` command to review a damaged pool and the recommended recovery steps. For example:

```
zpool status
pool: storpool
state: UNAVAIL
status: The pool metadata is corrupted and the pool cannot be opened.
action: Recovery is possible, but will result in some data loss.
Returning the pool to its state as of Fri Jun 29 17:22:49 2012
should correct the problem. Approximately 5 seconds of data
must be discarded, irreversibly. Recovery can be attempted
by executing 'zpool clear -F tpool'. A scrub of the pool
is strongly recommended after recovery.
see: http://support.oracle.com/msg/ZFS-8000-72
scrub: none requested
config:
```

| NAME     | STATE   | READ | WRITE | CKSUM |                |
|----------|---------|------|-------|-------|----------------|
| storpool | UNAVAIL | 0    | 0     | 1     | corrupted data |
| c1t1d0   | ONLINE  | 0    | 0     | 2     |                |
| c1t3d0   | ONLINE  | 0    | 0     | 4     |                |

The recovery process as described in the preceding output is to use the following command:

```
zpool clear -F storpool
```

If you attempt to import a damaged storage pool, you will see messages similar to the following:

```
zpool import storpool
cannot import 'storpool': I/O error
Recovery is possible, but will result in some data loss.
Returning the pool to its state as of Fri Jun 29 17:22:49 2012
should correct the problem. Approximately 5 seconds of data
must be discarded, irreversibly. Recovery can be attempted
```

by executing 'zpool import -F storpool'. A scrub of the pool is strongly recommended after recovery.

The recovery process as described in the preceding output is to use the following command:

```
zpool import -F storpool
Pool storpool returned to its state as of Fri Jun 29 17:22:49 2012.
Discarded approximately 5 seconds of transactions
```

If the damaged pool is in the `zpool.cache` file, the problem is discovered when the system is booted, and the damaged pool is reported in the `zpool status` command. If the pool isn't in the `zpool.cache` file, it won't successfully import or open and you will see the damaged pool messages when you attempt to import the pool.

- You can import a damaged pool in read-only mode. This method enables you to import the pool so that you can access the data. For example:

```
zpool import -o readonly=on storpool
```

For more information about importing a pool read-only, see [“Importing a Pool in Read-Only Mode” on page 69](#).

- You can import a pool with a missing log device by using the `zpool import -m` command. For more information, see [“Importing a Pool With a Missing Log Device” on page 68](#).
- If the pool cannot be recovered by either pool recovery method, you must restore the pool and all its data from a backup copy. The mechanism you use varies widely depending on the pool configuration and backup strategy. First, save the configuration as displayed by the `zpool status` command so that you can re-create it after the pool is destroyed. Then, use the `zpool destroy -f` command to destroy the pool.

Also, keep a file describing the layout of the datasets and the various locally set properties somewhere safe, as this information will become inaccessible if the pool is ever rendered inaccessible. With the pool configuration and dataset layout, you can reconstruct your complete configuration after destroying the pool. The data can then be populated by using whatever backup or restoration strategy you use.

## Repairing a Damaged ZFS Configuration

ZFS maintains a cache of active pools and their configuration in the root file system. If this cache file is corrupted or somehow becomes out of sync with configuration information that is stored on disk, the pool can no longer be opened. ZFS tries to avoid this situation, though arbitrary corruption is always possible given the qualities of the underlying storage. This situation typically results in a pool disappearing from the system when it should otherwise be available. This situation can also manifest as a partial configuration that is missing an unknown number of top-level virtual devices. In either case, the configuration can be recovered by exporting the pool (if it is visible at all) and re-importing it.

For information about importing and exporting pools, see [“Migrating ZFS Storage Pools” on page 65](#).

## Repairing an Unbootable System

ZFS is designed to be robust and stable despite errors. Even so, software bugs or certain unexpected problems might cause the system to panic when a pool is accessed. As part of the boot process, each pool must be opened, which means that such failures will cause a system to enter into a panic-reboot loop. To recover from this situation, ZFS must be informed not to look for any pools on startup.

ZFS maintains an internal cache of available pools and their configurations in `/etc/zfs/zpool.cache`. The location and contents of this file are private and are subject to change. If the system becomes unbootable, boot to the milestone `none` by using the `-m milestone=none boot` option. After the system is up, remount your root file system as writable and then rename or move the `/etc/zfs/zpool.cache` file to another location. These actions cause ZFS to forget that any pools exist on the system, preventing it from trying to access the unhealthy pool causing the problem. You can then proceed to a normal system state by issuing the `svcadm milestone all` command. You can use a similar process when booting from an alternate root to perform repairs.

After the system is up, you can attempt to import the pool by using the `zpool import` command. However, doing so will likely cause the same error that occurred during boot, because the command uses the same mechanism to access pools. If multiple pools exist on the system, do the following:

- Rename or move the `zpool.cache` file to another location as discussed in the preceding text.
- Determine which pool might have problems by using the `fmddump -eV` command to display the pools with reported fatal errors.
- Import the pools one by one, skipping the pools that are having problems, as described in the `fmddump` output.

## Recommended Oracle Solaris ZFS Practices

---

This chapter describes recommended practices for creating, monitoring, and maintaining your ZFS storage pools and file systems.

The following sections are provided in this chapter:

- “Recommended Storage Pool Practices” on page 269.
- “Recommended File System Practices” on page 277.

For general ZFS tuning information that includes tuning for an Oracle database, see [Chapter 3](#), “Oracle Solaris ZFS Tunable Parameters” in *Oracle Solaris 11.3 Tunable Parameters Reference Manual*.

### Recommended Storage Pool Practices

The following sections provide recommended practices for creating and monitoring ZFS storage pools. For information about troubleshooting storage pool problems, see [Chapter 12](#), “Oracle Solaris ZFS Troubleshooting and Pool Recovery”.

### General System Practices

- Keep system up-to-date with latest Oracle Solaris updates and releases
- Confirm that your controller honors cache flush commands so that you know your data is safely written, which is important before changing the pool's devices or splitting a mirrored storage pool. This is generally not a problem on Oracle/Sun hardware, but it is good practice to confirm that your hardware's cache flushing setting is enabled.
- Size memory requirements to actual system workload
  - With a known application memory footprint, such as for a database application, you might cap the ARC size so that the application will not need to reclaim its necessary memory from the ZFS cache.
  - Consider deduplication memory requirements
  - Identify ZFS memory usage with the following command:

```
mdb -k
> ::memstat
Page Summary Pages MB %Tot

Kernel 388117 1516 19%
ZFS File Data 81321 317 4%
Anon 29928 116 1%
Exec and libs 1359 5 0%
Page cache 4890 19 0%
Free (cachelist) 6030 23 0%
Free (freelist) 1581183 6176 76%

Total 2092828 8175
Physical 2092827 8175
> $q
```

- See Document 1663862.1, *Memory Management Between ZFS and Applications in Oracle Solaris 11.x*, in [My Oracle Support \(MOS\)](#) for tips on tuning the ZFS ARC cache. This document includes a script which you can use to modify the `user_reserver_hint_pct` memory management parameter.
- Consider using ECC memory to protect against memory corruption. Silent memory corruption can potentially damage your data.
- Perform regular backups – Although a pool that is created with ZFS redundancy can help reduce down time due to hardware failures, it is not immune to hardware failures, power failures, or disconnected cables. Make sure you backup your data on a regular basis. If your data is important, it should be backed up. Different ways to provide copies of your data are:
  - Regular or daily ZFS snapshots
  - Weekly backups of ZFS pool data. You can use the `zpool split` command to create an exact duplicate of ZFS mirrored storage pool.
  - Monthly backups by using an enterprise-level backup product
- Hardware RAID
  - Consider using JBOD-mode for storage arrays rather than hardware RAID so that ZFS can manage the storage and the redundancy.
  - Use hardware RAID or ZFS redundancy or both
  - Using ZFS redundancy has many benefits – For production environments, configure ZFS so that it can repair data inconsistencies. Use ZFS redundancy, such as RAID-Z, RAID-Z-2, RAID-Z-3, mirror, regardless of the RAID level implemented on the underlying storage device. With such redundancy, faults in the underlying storage device or its connections to the host can be discovered and repaired by ZFS.
  - If you are confident in the redundancy of your hardware RAID solution, then consider using ZFS without ZFS redundancy with your hardware RAID array. However, follow these recommendations to help ensure data integrity.

- Assign the size of the LUNs and the ZFS storage pool according to your comfort level by considering that ZFS will not be able to resolve data inconsistencies if the hardware RAID array experiences a failure.
- Create RAID5 LUNs with global hot spares.
- Monitor both the ZFS storage pool by using `zpool status` and the underlying LUNs by using your hardware RAID monitoring tools.
- Promptly replace any failed devices.
- Scrub your ZFS storage pools routinely, such as monthly, if you are using datacenter quality services.
- Always have good, recent backups of your important data.

See also [“Pool Creation Practices on Local or Network Attached Storage Arrays” on page 274.](#)

- Crash dumps consume more disk space, generally in the 1/2-3/4 size of physical memory range.

## ZFS Storage Pool Creation Practices

The following sections provide general and more specific pool practices.

### General Storage Pool Practices

- Use whole disks to enable disk write cache and provide easier maintenance. Creating pools on slices adds complexity to disk management and recovery.
- Use ZFS redundancy so that ZFS can repair data inconsistencies.
  - The following message is displayed when a non-redundant pool is created:

```
zpool create system1 c4t1d0 c4t3d0
'system1' successfully created, but with no redundancy; failure
of one device will cause loss of the pool
```
  - For mirrored pools, use mirrored disk pairs
  - For RAID-Z pools, group 3-9 disks per VDEV
  - Do not mix RAID-Z and mirrored components within the same pool. These pools are harder to manage and performance might suffer.
- Use hot spares to reduce down time due to hardware failures
- Use similar size disks so that I/O is balanced across devices
  - Smaller LUNs can be expanded to large LUNs
  - Do not expand LUNs from extremely varied sizes, such as 128 MB to 2 TB, to keep optimal metaslab sizes
- Consider creating a small root pool and larger data pools to support faster system recovery

- Recommended minimum pool size is 8 GB. Although the minimum pool size is 64 MB, anything less than 8 GB makes allocating and reclaiming free pool space more difficult.
- Recommended maximum pool size should comfortably fit your workload or data size. Do not try to store more data than you can routinely back up on a regular basis. Otherwise, your data is at risk due to some unforeseen event.

See also [“Pool Creation Practices on Local or Network Attached Storage Arrays” on page 274](#).

## Root Pool Creation Practices

- **SPARC (SMI (VTOC)):** Create root pools with slices by using the `s*` identifier. Do not use the `p*` identifier. In general, a system's ZFS root pool is created when the system is installed. If you are creating a second root pool or re-creating a root pool, use syntax similar to the following on a SPARC system:

```
zpool create rpool c0t1d0s0
```

Or, create a mirrored root pool. For example:

```
zpool create rpool mirror c0t1d0s0 c0t2d0s0
```

- **Solaris 11.1 x86 (EFI (GPT)):** Create root pools with whole disks by using the `d*` identifier. Do not use the `p*` identifier. In general, a system's ZFS root pool is created when the system is installed. If you are creating a second root pool or re-creating a root pool, use syntax similar to the following:

```
zpool create rpool c0t1d0
```

Or, create a mirrored root pool. For example:

```
zpool create rpool mirror c0t1d0 c0t2d0
```

- The root pool must be created as a mirrored configuration or as a single-disk configuration. Neither a RAID-Z nor a striped configuration is supported. You cannot add additional disks to create multiple mirrored top-level virtual devices by using the `zpool add` command, but you can expand a mirrored virtual device by using the `zpool attach` command.
- The root pool cannot have a separate log device.
- Pool properties can be set during an AI installation, but the `gzip` compression algorithm is not supported on root pools.
- Do not rename the root pool after it is created by an initial installation. Renaming the root pool might cause an unbootable system.
- Do not create a root pool on a USB stick on a production system because root pool disks are critical for continuous operation, particularly in an enterprise environment. Consider using a system's internal disks for the root pool, or at least, use the same quality disks that you would use for your non-root data. In addition, a USB stick might not be large enough to support a dump volume size that is equivalent to at least 1/2 the size of physical memory.



- Rather than adding a hot spare to a root pool, consider creating a two- or a three-way mirror root pool. In addition, do not share a hot spare between a root pool and a data pool.
- Do not use a VMware thinly-provisioned device for a root pool device.

## Non-Root Pool Creation Practices

- Create non-root pools with whole disks by using the d\* identifier. Do not use the p\* identifier.
  - ZFS works best without any additional volume management software.
  - For better performance, use individual disks or at least LUNs made up of just a few disks. By providing ZFS with more visibility into the LUNs setup, ZFS is able to make better I/O scheduling decisions.
- Create redundant pool configurations across multiple controllers to reduce down time due to a controller failure.
  - **Mirrored storage pools** – Consume more disk space but generally perform better with small random reads.

```
zpool create system1 mirror c1d0 c2d0 mirror c3d0 c4d0
```

- **RAID-Z storage pools** – Can be created with 3 parity strategies, where parity equals 1 (raidz), 2 (raidz2), or 3 (raidz3). A RAID-Z configuration maximizes disk space and generally performs well when data is written and read in large chunks (128K or more).
  - Consider a single-parity RAID-Z (raidz) configuration with 2 VDEVs of 3 disks (2+1) each.

```
zpool create rzpool raidz1 c1t0d0 c2t0d0 c3t0d0 raidz1 c1t1d0 c2t1d0 c3t1d0
```

- A RAIDZ-2 configuration offers better data availability, and performs similarly to RAID-Z. RAIDZ-2 has significantly better mean time to data loss (MTTDL) than either RAID-Z or 2-way mirrors. Create a double-parity RAID-Z (raidz2) configuration at 6 disks (4+2).

```
zpool create rzpool raidz2 c0t1d0 c1t1d0 c4t1d0 c5t1d0 c6t1d0 c7t1d0
raidz2 c0t2d0 c1t2d0 c4t2d0 c5t2d0 c6t2d0 c7t2d0
```

- A RAIDZ-3 configuration maximizes disk space and offers excellent availability because it can withstand 3 disk failures. Create a triple-parity RAID-Z (raidz3) configuration at 9 disks (6+3).

```
zpool create rzpool raidz3 c0t0d0 c1t0d0 c2t0d0 c3t0d0 c4t0d0
c5t0d0 c6t0d0 c7t0d0 c8t0d0
```

## Pool Creation Practices on Local or Network Attached Storage Arrays

Consider the following storage pool practices when creating an a ZFS storage pool on a storage array that is connected locally or remotely.

- If you create an pool on SAN devices and the network connection is slow, the pool's devices might be `UNAVAIL` for a period of time. You need to assess whether the network connection is appropriate for providing your data in a continuous fashion. Also, consider that if you are using SAN devices for your root pool, they might not be available as soon as the system is booted and the root pool's devices might also be `UNAVAIL`.
- Confirm with your array vendor that the disk array is not flushing its cache after a flush write cache request is issued by ZFS.
- Use whole disks, not disk slices, as storage pool devices so that Oracle Solaris ZFS activates the local small disk caches, which get flushed at appropriate times.
- For best performance, create one LUN for each physical disk in the array. Using only one large LUN can cause ZFS to queue up too few read I/O operations to actually drive the storage to optimal performance. Conversely, using many small LUNs could have the effect of swamping the storage with a large number of pending read I/O operations.
- A storage array that uses dynamic (or thin) provisioning software to implement virtual space allocation is not recommended for Oracle Solaris ZFS. When Oracle Solaris ZFS writes the modified data to free space, it writes to the entire LUN. The Oracle Solaris ZFS write process allocates all the virtual space from the storage array's point of view, which negates the benefit of dynamic provisioning.

Consider that dynamic provisioning software might be unnecessary when using ZFS:

- You can expand a LUN in an existing ZFS storage pool and it will use the new space.
- Similar behavior works when a smaller LUN is replaced with a larger LUN.
- If you assess the storage needs for your pool and create the pool with smaller LUNs that equal the required storage needs, then you can always expand the LUNs to a larger size if you need more space.
- If the array can present individual devices (JBOD-mode), then consider creating redundant ZFS storage pools (mirror or RAID-Z) on this type of array so that ZFS can report and correct data inconsistencies.

## Pool Creation Practices for an Oracle Database

Consider the following storage pool practices when creating an Oracle database.

- Use a mirrored pool or hardware RAID for pools
- RAID-Z pools are generally not recommended for random read workloads
- Create a small separate pool with a separate log device for database redo logs
- Create a small separate pool for the archive log

For more information about tuning ZFS for an Oracle database, “[Tuning ZFS for an Oracle Database](#)” in *Oracle Solaris 11.3 Tunable Parameters Reference Manual*.

## Using ZFS Storage Pools in VirtualBox

- Virtual Box is configured to ignore cache flush commands from the underlying storage by default. This means that in the event of a system crash or a hardware failure, data could be lost.
- Enable cache flushing on Virtual Box by issuing the following command:

```
VBoxManage setextradata vm-name "VBoxInternal/Devices/type/0/LUN#n/Config/IgnoreFlush" 0
```

- *vm-name* – the name of the virtual machine
- *type* – the controller type, either `piix3ide` (if you're using the usual IDE virtual controller) or `ahci`, if you're using a SATA controller
- *n* – the disk number

## Storage Pool Practices for Performance

- In general, keep pool capacity below 90% for best performance. The percentage where performance might be impacted depends greatly on workload:
  - If data is mostly added (write once, remove never), then it's very easy for ZFS to find new blocks. In this case, the percentage can be higher than normal. Maybe up to 95%.
  - If data is made of large files or large blocks (such as 128K files or 1MB blocks) and the data is removed in bulk operations, the percentage can be higher than normal. Maybe up to 95%
  - If a large percentage (more than 50%) of the pool is made up of 8k chunks (DBfiles, iSCSI Luns, or many small files) and have constant rewrites, then the 90% rule should be followed strictly.
  - If all of the data is small blocks that have constant rewrites, then you should monitor your pool closely once the capacity gets over 80%. The sign to watch for is increased disk IOPS to achieve the same level of client IOPS
- Mirrored pools are recommended over RAID-Z pools for random read/write workloads
- Separate log devices
  - Recommended to improve synchronous write performance
  - With a high synchronous write load, prevents fragmentation of writing many log blocks in the main pool
- Separate cache devices are recommended to improve read performance
- Scrub/resilver - A very large RAID-Z pool with lots of devices will have longer scrub and resilver times

- Pool performance is slow – Use the `zpool status` command to rule out any hardware problems that are causing pool performance problems. If no problems show up in the `zpool status` command, use the `fmddump` command to display hardware faults or use the `fmddump -eV` command to review any hardware errors that have not yet resulted in a reported fault.

## ZFS Storage Pool Maintenance and Monitoring Practices

- Make sure that pool capacity is below 90% for best performance.

Pool performance can degrade when a pool is very full and file systems are updated frequently, such as on a busy mail server. Full pools might cause a performance penalty, but no other issues. If the primary workload is immutable files, then keep pool in the 95-96% utilization range. Even with mostly static content in the 95-96% range, write, read, and resilvering performance might suffer.

  - Monitor pool and file system space to make sure that they are not full.
  - Consider using ZFS quotas and reservations to make sure file system space does not exceed 90% pool capacity.
- Monitor pool health
  - Monitor a redundant pool with `zpool status` and `fmddump` at least once per week
  - Monitor a non-redundant pool with `zpool status` and `fmddump` at least twice per week
- Run `zpool scrub` on a regular basis to identify data integrity problems.
  - If you have consumer-quality drives, consider a weekly scrubbing schedule.
  - If you have datacenter-quality drives, consider a monthly scrubbing schedule.
  - You should also run a scrub prior to replacing devices or temporarily reducing a pool's redundancy to ensure that all devices are currently operational.
- Monitoring pool or device failures - Use `zpool status` as described below. Also use `fmddump` or `fmddump -eV` to see if any device faults or errors have occurred.
  - Redundant pools, monitor pool health with `zpool status` and `fmddump` on a weekly basis
  - Non-redundant pools, monitor pool health with `zpool status` and `fmddump` on a semiweekly basis
- Pool device is `UNAVAIL` or `OFFLINE` – If a pool device is not available, then check to see if the device is listed in the `format` command output. If the device is not listed in the `format` output, then it will not be visible to ZFS.

If a pool device has `UNAVAIL` or `OFFLINE`, then this generally means that the device has failed or cable has disconnected, or some other hardware problem, such as a bad cable or bad controller has caused the device to be inaccessible.
- Consider configuring the `smtp-notify` service to notify you when a hardware component is diagnosed as faulty. For more information, see the Notification Parameters section of [smf\(5\)](#) and [smtp-notify\(1M\)](#).

By default, some notifications are set up automatically to be sent to the root user. If you add an alias for your user account as root in the `/etc/aliases` file, you will receive electronic mail notifications, similar to the following:

```
From noaccess@tardis.space.com Fri Jun 29 16:58:59 2012
Date: Fri, 29 Jun 2012 16:58:58 -0600 (MDT)
From: No Access User <noaccess@tardis.space.com>
Message-Id: <201206292258.q5TMwwFL002753@tardis.space.com>
Subject: Fault Management Event: tardis:ZFS-8000-8A
To: root@tardis.central.com
Content-Length: 771
```

```
SUNW-MSG-ID: ZFS-8000-8A, TYPE: Fault, VER: 1, SEVERITY: Critical
EVENT-TIME: Fri Jun 29 16:58:58 MDT 2012
PLATFORM: ORCL,SPARC-T3-4, CSN: 1120BDRCCD, HOSTNAME: tardis
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: 76c2d1d1-4631-4220-dbbc-a3574b1ee807
DESC: A file or directory in pool 'pond' could not be read due to corrupt data.
AUTO-RESPONSE: No automated response will occur.
IMPACT: The file or directory is unavailable.
REC-ACTION: Use 'fmadm faulty' to provide a more detailed view of this event.
Run 'zpool status -xv' and examine the list of damaged files to determine what
has been affected. Please refer to the associated reference document at
http://support.oracle.com/msg/ZFS-8000-8A for the latest service procedures
and policies regarding this diagnosis.
```

- Monitor your storage pool space – Use the `zpool list` command and the `zfs list` command to identify how much disk is consumed by file system data. ZFS snapshots can consume disk space and if they are not listed by the `zfs list` command, they can silently consume disk space. Use the `zfs list -t snapshot` command to identify disk space that is consumed by snapshots.

## Recommended File System Practices

The following sections describe recommended file system practices.

### Root File System Practices

- Consider keeping the root file system small and isolated from other non-root related data so that root pool recovery is faster.
- Do not include file systems in `rpool/ROOT`, which is a special container that requires no administration and should not contain any additional components.

## File System Creation Practices

The following sections describe ZFS file system creation practices.

- Create one file system per user for home directories
- Consider using file system quotas and reservations to manage and reserve disk space for important file systems
- Consider using user and group quotas to manage disk space in an environment with many users
- Use ZFS property inheritance to apply properties to many descendant file systems

## File System Creation Practices for an Oracle Database

Consider the following file system practices when creating an Oracle database.

- Match the ZFS `recordsize` property to the Oracle `db_block_size`.
- Create database table and index file systems in main database pool, using an 8 KB `recordsize` and the default `primarycache` value.
- Create temp data and undo table space file systems in the main database pool, using default `recordsize` and `primarycache` values.
- Create archive log file system in the archive pool, enabling compression and default `recordsize` value and `primarycache` set to `metadata`.

For more information, see the following white paper:

[http://blogs.oracle.com/storage/entry/new\\_white\\_paper\\_configuring\\_oracle](http://blogs.oracle.com/storage/entry/new_white_paper_configuring_oracle)

## Monitoring ZFS File System Practices

You should monitor your ZFS file systems to ensure they are available and to identify space consumption issues.

- Weekly, monitor file system space availability with the `zpool list` and `zfs list` commands rather than the `du` and `df` commands because legacy commands do not account for space that is consumed by descendant file systems or snapshots.

For more information, see “Resolving ZFS Space Issues” on page 256.

- Display file system space consumption by using the `zfs list -o space` command.
- File system space can be unknowingly consumed by snapshots. You can display all dataset information by using the following syntax:

```
zfs list -t all
```

- A separate `/var` file system is created automatically when a system is installed, but you should set a quota and reservation on this file system to ensure that it does not unknowingly consume root pool space.
- In addition, you can use the `fsstat` command to display file operation activity of ZFS file systems. Activity can be reported by mount point or by file system type. The following example shows general ZFS file system activity:

```
fsstat /
new name name attr attr lookup rddir read read write write
file remov chng get set ops ops ops bytes ops bytes
832 589 286 837K 3.23K 2.62M 20.8K 1.15M 1.75G 62.5K 348M /
```

- Backups
  - Keep file system snapshots
  - Consider enterprise-level software for weekly and monthly backups
  - Store root pool snapshots on a remote system for bare metal recovery





# ◆ ◆ ◆ A P P E N D I X A

## Oracle Solaris ZFS Version Descriptions

---

This appendix describes available ZFS versions, features of each version, and the Oracle Solaris OS that provides the ZFS version and feature.

The following sections are provided in this appendix:

- [“Overview of ZFS Versions” on page 281.](#)
- [“ZFS Pool Versions” on page 281.](#)
- [“ZFS File System Versions” on page 283.](#)

### Overview of ZFS Versions

New ZFS pool and file system features are introduced and accessible by using a specific ZFS version that is available in Oracle Solaris releases. You can use the `zpool upgrade` or `zfs upgrade` to identify whether a pool or file system is at lower version than the currently running Oracle Solaris release provides. You can also use these commands to upgrade your pool and file system versions.

For information about using the `zpool upgrade` and `zfs upgrade` commands, see [“Upgrading ZFS File Systems” on page 158](#) and [“Upgrading ZFS Storage Pools” on page 72.](#)

### ZFS Pool Versions

The following table provides a list of ZFS pool versions that are available in the Oracle Solaris release. This list can be created using the `zpool upgrade -v` command.

| Version | Oracle Solaris Release | Description                         |
|---------|------------------------|-------------------------------------|
| 1       | S11 11/11              | Initial ZFS version                 |
| 2       | S11 11/11              | Ditto blocks (replicated metadata)  |
| 3       | S11 11/11              | Hot spares and double parity RAID-Z |
| 4       | S11 11/11              | <code>zpool history</code>          |

| Version | Oracle Solaris Release | Description                                      |
|---------|------------------------|--------------------------------------------------|
| 5       | S11 11/11              | gzip compression algorithm                       |
| 6       | S11 11/11              | bootfs pool property                             |
| 7       | S11 11/11              | Separate intent log devices                      |
| 8       | S11 11/11              | Delegated administration                         |
| 9       | S11 11/11              | refquota and reservation properties              |
| 10      | S11 11/11              | Cache devices                                    |
| 11      | S11 11/11              | Improved scrub performance                       |
| 12      | S11 11/11              | Snapshot properties                              |
| 13      | S11 11/11              | snapped property                                 |
| 14      | S11 11/11              | aclinherit passthrough-x property                |
| 15      | S11 11/11              | user and group space accounting                  |
| 16      | S11 11/11              | stmf property                                    |
| 17      | S11 11/11              | Triple-parity RAID-Z                             |
| 18      | S11 11/11              | Snapshot user holds                              |
| 19      | S11 11/11              | Log device removal                               |
| 20      | S11 11/11              | zle (zero-length encoding) compression algorithm |
| 21      | S11 11/11              | Deduplication                                    |
| 22      | S11 11/11              | Received properties                              |
| 23      | S11 11/11              | Slim ZIL                                         |
| 24      | S11 11/11              | System attributes                                |
| 25      | S11 11/11              | Improved scrub stats                             |
| 26      | S11 11/11              | Improved snapshot deletion performance           |
| 27      | S11 11/11              | Improved snapshot creation performance           |
| 28      | S11 11/11              | Multiple vdev replacements                       |
| 29      | S11 11/11              | RAID-Z/mirror hybrid allocator                   |
| 30      | S11 11/11              | Encryption                                       |
| 31      | S11 11/11              | Improved 'zfs list' performance                  |
| 32      | S11 11/11              | One MB blocksize                                 |
| 33      | S11 11/11              | Improved share support                           |
| 34      | S11.1                  | Sharing with inheritance                         |
| 35      | S11.2                  | Sequential resilver                              |
| 36      | S11.3                  | Efficient log block allocation                   |
| 37      | S11.3                  | lz4 compression                                  |

## ZFS File System Versions

The following table lists the ZFS file system versions that are available in the Oracle Solaris release. Keep in mind that a feature that is available in a specific file system version requires a specific pool version. This list can be created using the `zfs upgrade -v` command.

| Version | Oracle Solaris Release | Description                                                 |
|---------|------------------------|-------------------------------------------------------------|
| 1       | S11 11/11              | Initial ZFS file system version                             |
| 2       | S11 11/11              | Enhanced directory entries                                  |
| 3       | S11 11/11              | Case insensitivity and file system unique identifier (FUID) |
| 4       | S11 11/11              | userquota and groupquota properties                         |
| 5       | S11 11/11              | System attributes                                           |
| 6       | S11.1                  | Multilevel file system support                              |



# Glossary

---

## B

**boot environment** A bootable Oracle Solaris environment consisting of a ZFS root file system and, optionally, other file systems mounted underneath it. Exactly one boot environment can be active at a time.

## C

**checksum** A 256-bit hash of the data in a file system block. The checksum capability can range from the simple and fast fletcher4 (the default) to cryptographically strong hashes such as SHA256.

**clone** A file system whose initial contents are identical to the contents of a snapshot.  
For information about clones, see [“Overview of ZFS Clones” on page 167](#).

## D

**dataset** A generic name for the following ZFS components: clones, file systems, snapshots, and volumes. Each dataset is identified by a unique name in the ZFS namespace.  
For more information about datasets, see [Chapter 7, “Managing Oracle Solaris ZFS File Systems”](#).

**deduplication** The process of eliminating duplicate blocks of data in a ZFS file system. After removing duplicate blocks, the unique blocks are stored in the deduplication table.

## F

**file system** A ZFS dataset of type `filesystem` that is mounted within the standard system namespace and behaves like other file systems.  
For more information about file systems, see [Chapter 7, “Managing Oracle Solaris ZFS File Systems”](#).

## M

**mirror** A virtual device that stores identical copies of data on two or more disks. If any disk in a mirror fails, any other disk in that mirror can provide the same data.

## P

**pool** A logical group of devices describing the layout and physical characteristics of the available storage. Disk space for datasets is allocated from a pool.

For more information about storage pools, see [Chapter 5, “Managing Oracle Solaris ZFS Storage Pools”](#).

## R

**RAID-Z** A virtual device that stores data and parity on multiple disks. For more information about RAID-Z, see [“RAID-Z Storage Pool Configuration” on page 19](#).

**resilvering** The process of copying data from one device to another device. For example, if a mirror device is replaced or taken offline, the data from an up-to-date mirror device is copied to the newly restored mirror device. In traditional volume management products, this process is referred to as *mirror resynchronization*.

For more information about ZFS resilvering, see [“Viewing Resilvering Status” on page 253](#).

**root pool** A ZFS pool that contains the boot file system.

## S

**snapshot** A read-only copy of a file system or volume at a given point in time.

For more information about snapshots, see [“Overview of ZFS Snapshots” on page 159](#).

## V

**virtual device** A logical device in a pool, which can be a physical device, a file, or a collection of devices.

For more information about virtual devices, see [“Querying ZFS Storage Pool Status” on page 55](#).

**volume** A dataset that represents a block device. For example, you can create a ZFS volume as a swap device.

For more information about ZFS volumes, see [“ZFS Volumes” on page 221](#).





# Index

---

## A

access control lists *See* ACLs

accessing

    snapshot, 163

ACL entries

    aclinherit property, 102

    aclmode property, 102

aclinherit property, 102, 190

aclmode property, 102

ACLs

    access privileges, 187

    ACL inheritance, 189, 197

    aclinherit property, 190

    description, 185

    description of entries, 186

    entry types, 187

    formats, 185

    interaction with permission bits, 194

    setting on ZFS files

        compact output, 193

        description, 191

        verbose mode, 191

        verbose output, 193

    trivial ACLs on ZFS files

        modifying, 193

adding

    cache devices, example of, 39

    devices to a pool, 37

    disks to a RAID-Z configuration, example of, 38

    mirrored log device, 39

    ZFS file system to native zones, 225

    ZFS volumes to native zones, 226

adjusting swap and dump device sizes, 89

allocated property, 53

alternate root pools, 230

altroot property, 53

atime property, 102

attaching devices to a pool, 41

automatic mount points, 124

automatic naming

    of a ZFS file system, 134

autoreplace property, 54

available property, 102

## B

boot environment (BE), 81

bootblocks, installing, 90

bootfs property, 54

booting

    root file system, 90

    ZFS BE on SPARC systems, 93

## C

cache devices

    adding, example of, 39

    considerations for using, 32

    creating a ZFS storage pool with, 32

    removing, example of, 40

cachefile property, 54

canmount property

    description, 102

    detailed description, 110

capacity property, 54

casesensitivity property

    description, 103

    detailed description, 111

checking data integrity, 260

checksum property, 103

clearing

    device errors, 247

- devices in a pool , 46
  - clones
    - creating, 168
    - destroying, 168
    - features, 167
    - promoting, 169
  - command history, displaying, 58
  - compact output display of ACL information, 193
  - components
    - of ZFS storage pools *See* storage pools
    - ZFS naming requirements, 22
  - compressing a ZFS file system
    - overview, 146
  - compression algorithms
    - in ZFS, 146
  - compression property, 103
  - compressratio property, 103
  - copies property, 103
    - detailed description, 111
  - crash dumps, saving, 90
  - creating
    - alternate root pools, 230
    - clones, 168
    - double-parity RAID-Z storage pool
      - example of, 29
    - file system, 26, 26
    - hot spares, 48
    - mirrored ZFS storage pool, 30
    - new pool from a split mirrored pool, 42
    - single-parity RAID-Z storage pool
      - example of, 29
    - snapshots, 160
    - storage pools, 25, 26
      - cache devices, 32
      - log devices, 31
    - triple-parity RAID-Z storage pool
      - example of, 29
    - ZFS file system, 98
    - ZFS volumes, 221
  - creation property, 103
- D**
- data
    - corrupted, 262
    - duplication type, choosing, 23
  - identifying corruption, 240
    - repair, 260
    - saving, 173
    - scrubbing and resilvering, 260, 262
    - self-healing, 19
    - sending and receiving, 170
    - validation *See* data scrubbing and resilvering
  - dataset
    - description, 97
  - dataset types
    - description, 117
  - datasets
    - delegating to a native zone, 226
  - dedup property, 103
    - detailed description, 112
  - dedupditto property, 54
  - dedupratio property, 54
  - defaultgroupquota property, 103
  - defaultuserquota property, 103
  - delegated administration, 207
  - delegating
    - datasets to a native zone, 226
    - permissions
      - command description, 211
      - groups, 213
      - individual users, 212
  - delegation property
    - description, 54
    - disabling, 208
  - destroying
    - clones, 168
    - snapshots, 161
    - storage pools, 36
    - ZFS file system, 99
    - ZFS file system with dependents, 100
  - detecting
    - in-use devices, 34
    - mismatched redundancy levels, 35
  - device failures
    - determining replaceability, 248
    - types of, 245
  - devices
    - adding to a storage pool, 37
    - attaching to a pool, 41
    - detaching from ZFS storage pool, 41
    - detecting in-use devices, 34

- dump devices, enabling, 90
- log devices, 31
- removing from a storage pool, 40
- replaceability of, 248
- replacing, 46
- returning online, 45
- taking offline, 44

devices property, 103

disks in storage pools *See* storage pools

displaying

- ACL information in compact format, 193
- ACL information in verbose format, 193
- delegated permissions, 216
- pool
  - health status, 61, 63
  - I/O statistics, 59, 60
- syslog reporting of ZFS error messages, 235

dump devices *See* swap and dump devices

dynamic striping, 20

## E

EFI label

- interaction with ZFS, 18

enabling a dump device, 90

encrypting a ZFS file system, 147

- changing keys, 149
- examples of, 153
- overview, 147

encryption property, 103

errors, clearing, 46

exec property, 104

exporting storage pools, 66

## F

failmode property, 54

failures

- corrupted data, 262
- identifying, 233
- missing (UNAVAIL) devices, 244

fast reboot feature, x86, 94

file access time updated

- atime property, 102

file system

- adding to native zones, 225

booting

- root file system, 90
- ZFS BE on SPARC, 93

components, 22

converting to snapshot stream, 171

hierarchy, 23

managing

- properties within a zone, 227

migrating, 155

replacing with clones, 169

rights profiles, 21

setting

- ACL inheritance on ZFS files (verbose mode), 197
- ACLs on ZFS files, 191, 191, 193

snapshots

- accessing, 163
- renaming, 162

trivial ACL on ZFS files

- modifying, 193

using with zones installed, 225

files in storage pools *See* storage pools

free property, 55

## G

guid property, 55

gzip compression algorithm

- in ZFS, 146

## H

hardware and software requirements, 21

health property, 55

hot spares

- activating and deactivating, 49
- adding, 48
- detaching, 50

## I

identifying

- storage pool for import, 67
- storage requirements, 22
- type of data corruption, 263

- importing
  - alternate root pools, 231
  - storage pools, 68, 70
- in-use devices, 34
- inheriting
  - ZFS properties
    - description, 120
- installing
  - bootblocks, 90
  - replacement devices *See* replacing devices
  - root pool, automatic, 76

## K

- keychangedate property, 104
- keysource property, 104
- keystatus property, 104

## L

- legacy mount points, 125
- listing
  - descendents of ZFS file systems, 117
  - file systems, 28
  - pool information, 27, 55, 56
  - types of ZFS file systems, 118
  - ZFS file systems, 116
  - ZFS file systems without header information, 118
  - ZFS properties, 121
  - ZFS properties by source value, 123
  - ZFS properties for scripting, 123
- listshares property, 55
- listsnapshots property, 55
- log devices
  - removing, example of, 40
- log devices, creating a ZFS storage pool with, 31
- logbias property, 104
- lz4 compression algorithm
  - in ZFS, 146
- lzjb compression algorithm
  - in ZFS, 146

## M

- migrating

- description, 154
- file systems, 154, 155
- storage pools, 65
- mirrored configuration
  - for redundancy, 19
  - log devices
    - adding, 39
    - creating pool with, 31
  - splitting a mirrored pool to create a new pool, 42
  - storage pools, 30
- mlslabel property, 104
- monitoring
  - data scrubbing, 184
  - resilvering task, 183
  - send stream progress, 183
  - status of stream reception, 183
  - tasks running on pools, 181
- mount points
  - automatic, 124
  - default
    - storage pools, 35
  - legacy, 125
  - managing ZFS
    - description, 124
- mounted property, 104
- mounting
  - ZFS file systems, 127
- mountpoint
  - default for ZFS file system, 98
- mountpoint property, 104
- multilevel property, 104

## N

- named shares
  - on a ZFS file system, 133
- names
  - for ZFS file systems, 97
- naming requirements of ZFS components, 22
- native zones
  - adding ZFS file system, 225
  - delegating datasets to, 226
- nbmand property, 105
- NFSv4 ACLs
  - description, 186
  - inheritance, 189, 189

- model, 185
- property, 190
- normalization property, 105
- notifying ZFS of reattached device, 245

## O

- Oracle Solaris ACLs *See* ACLs
- origin property, 105

## P

- permission sets, defined, 207
- planning ZFS implementation, 22
- pool properties, list of, 53, 55
  - See also* ZFS properties
- primarycache property, 105
- properties of ZFS
  - description, 101
  - description of heritable properties, 101

## Q

- quota property, 105
- quotas and reservations
  - description, 140

## R

- RAID-Z configuration
  - adding disks to, 38
  - conceptual view, 19
  - double-parity, 19
  - example of, 29
  - redundancy feature, 19
  - single-parity, 19
- read-only properties of ZFS
  - description, 109
- readonly property, 106
- receiving file system data, 175
- recordsize property, 106
  - detailed description, 114
- recovering destroyed storage pools, 70
- recursive stream package, 172
- redundancy

- methods, 18
- mismatched levels, 35
- referenced property, 106
- refquota property, 106
- refreservation property, 106
- rekeydate property, 106
- removing
  - cache devices, 40
  - devices from a storage pool, 40
  - log devices, 40
  - permissions, 212
- renaming
  - snapshots, 162
  - storage pools, 68
  - ZFS file system, 100
- repairing
  - an unbootable system, 268
  - corrupted file or directory, 264
  - damaged ZFS configurations, 267
  - pool-wide damage, 267
- replacing
  - a missing device, 241
  - devices, 46, 249, 253
- replication
  - stream package, 171
- reservation property, 106
- rights profiles for ZFS management, 21
- rolling back snapshots, 165
- root pools
  - alternate location, 230
  - automatic installation, 76
  - considerations for configuration, 76
  - mirrored configuration in
    - SPARC or x86/EFI (GPT), 78
    - SPARC or x86/VTOC, 79
  - replacing disks, 82
  - space requirements, 75
- rstchown property, 107

## S

- saving
  - crash dumps, 90
  - file system data, 173
- scripting pool output
  - pool output, 56

- scrubbing and resilvering, 260, 262
- secondarycache property, 107
- sending and receiving file system data, 170
- separate log devices, considerations for using, 31
- settable properties of ZFS
  - description, 110
- setting
  - ACL inheritance, 197
  - ACLs on ZFS files
    - compact output, 193
    - description, 191
    - verbose mode, 191
    - verbose output, 193
  - compression property, 28
  - legacy mount points, 126
  - mountpoint property, 28
  - quota property, 28
  - share.nfs property, 28
  - ZFS atime property, 119
  - ZFS file system quota, 141
  - ZFS file system reservation, 145
  - ZFS mount points, 126
  - ZFS quota, 120
- setuid property, 107
- shadow migration, 154
- shadow property, 107
- share.nfs property
  - description, 107
  - example, 130
- share.smb property, 107
  - detailed description, 114
- sharenfs property
  - example, 130, 133
- sharesmb property
  - example, 130
- sharing
  - ZFS file systems, 129
    - named shares, 133
    - with automatic naming, 134
- sharing ZFS file systems
  - share.smb property, 114
- size property, 55
- snapdir property, 107
- snapshot
  - accessing, 163
  - applying property values, 176
  - copying, 173
  - creating, 160
  - destroying, 161
  - features, 159
  - monitoring streams
    - receiving, 183
    - sending, 183
  - renaming, 162
  - rolling back, 165
  - sending and receiving data streams, 173, 175
  - space accounting, 164
- Solaris ACLs *See* ACLs
- splitting a mirrored pool, 42
- storage pools
  - clearing device errors, 46, 247
  - components
    - disks, 17
    - files, 18
    - virtual devices, 25
  - creating
    - mirrored configuration, 30
    - performing a dry run, 33
    - RAID-Z configuration, 29
  - default mount point, 35
  - destroying, 36
  - device failure in, 245
  - devices
    - adding, 37
    - attaching and detaching, 41
    - configuring vdevs, 25
    - determining replaceability, 248
    - removing, 40
    - replacing, 46, 241, 249
    - taking offline and returning online, 44
  - displaying
    - health status, 61, 63
    - I/O statistics, 60
  - dynamic striping, 20
  - exporting, 66
  - files in, 18
  - importing
    - alternate source directories, 70
    - identifying available pools, 67
    - renaming pools while, 68
  - listing, 56

- migrating, 65
- mirrored configuration, 19
- notifying ZFS of device availability, 245
- problems in, 233, 236, 237
- RAID-Z configuration, 19
- recovering a destroyed pool, 70
- rights profiles, 21
- scripting storage pool output, 56, 57
- splitting a mirrored pool, 42
- status information for troubleshooting, 238
- system error messages, 235
- upgrading, 72
- using whole disks, 18
- viewing resilvering process, 253
- storage requirements, 22
- stream package
  - recursive, 172
  - replication, 171
- swap and dump devices
  - adjusting sizes, 89
  - description, 86
  - viewing, 87
- sync property, 107

## T

- tasks
  - creating a ZFS file system, 98
  - destroying a ZFS file system, 99, 100
- traditional file systems and ZFS, 15
- trivial ACLs, 193
- troubleshooting
  - clear device errors, 247
  - data corruption, 240, 263
  - determining if a device can be replaced, 248
  - device failure, 245
  - file system migration, 154
  - identifying problems, 236, 237
  - missing (UNAVAIL) devices, 244
  - notifying ZFS of reattached device, 245
  - pool status information, 238
  - repairing
    - corrupted file or directory, 264
    - damaged ZFS configuration, 267
    - pool-wide damage, 267
    - unbootable system, 268

- replacing
  - devices, 249, 253
  - missing device, 241
  - storage pool creation issues, 34
  - syslog reporting of ZFS error messages, 235
  - ZFS failures, 233
- type property, 108

## U

- unmounting
  - ZFS file systems, 128
- unsharing
  - ZFS file systems, 129
- upgrading
  - storage pool, 72
  - ZFS file systems
    - description, 158
- used property
  - description, 108
  - detailed description, 109
- usedbychildren property, 108
- usedbydataset property, 108
- usedbyrefreservation property, 108
- usedbysnapshots property, 108
- user properties of ZFS, 115
- utf8only property, 108

## V

- verbose output display of ACL information, 193
- version property, 55, 108
- virtual devices, 25
- volblocksize property, 108
- volsize property, 108
  - detailed description, 114
- vscan property, 109

## W

- whole disks as components *See* storage pools

## X

- xattr property, 109

**Z****ZFS**

- comparison with traditional file systems, 15
- features, 15
- on zones, 225
- planning deployment of, 22
- versions, 281
- zfs create command
  - description, 98
- zfs destroy -r command, 100
- zfs destroy command, 99
- ZFS file system
  - description, 97
  - names, 97
- ZFS file systems
  - compressing, 146
  - dataset types
    - description, 117
  - default mountpoint, 98
  - destroying, 99
  - destroying with dependents, 100
  - encrypting, 147
  - inheriting property, 120
  - listing
    - descendents, 117
    - description, 116
    - properties by source value, 123
    - properties for scripting, 123
    - properties of, 121
    - types of datasets, 118
    - without headers, 118
  - managing
    - automatic mount points, 124
    - legacy mount points, 125
    - mount points, 124
  - mounting, 127
  - renaming, 100
  - setting
    - a quota, 141
    - a reservation, 145
    - atime property, 119
    - legacy mount points, 126
    - mountpoint property, 126
    - quota property, 120
  - sharing, 129
  - unmounting, 128
  - unsharing, 129
  - upgrading
    - description, 158
  - zfs get command
    - description, 121
    - H and -o options, 123
    - s option (source type), 123
  - zfs inherit command, 120
  - ZFS intent log (ZIL), 31
  - zfs list command
    - description, 116
    - H option (without headers), 118
    - r option (recursive), 117
    - t option (dataset types), 118
  - zfs mount command, 127
  - ZFS properties
    - canmount property, 110
    - casesensitivity property, 111
    - copies property, 111
    - dedup property, 112
    - description, 101
    - description of inheritable, 101
    - list of, 101
    - managing in zones, 227
    - read-only, 109
    - recordsize property, 114
    - settable, 110
    - used property, 109
    - user properties, 115
    - volsize property, 114
  - zfs rename command, 100
  - zfs set command
    - atime property, 119
    - mountpoint property, 126
    - mountpoint=legacy property, 126
    - quota property, 120, 141
    - reservation property, 145
    - share property, 130
  - zfs unmount command, 128, 129
  - zfs upgrade command, 158
  - ZFS volumes
    - adding to native zones, 226
    - creating, 221
  - zle compression algorithm



- in ZFS, 146
- zoned property, 109, 228
- zones
  - adding ZFS file system, 225
  - adding ZFS volumes, 226
  - delegating datasets to a native zone, 226
  - managing ZFS properties, 227
  - using with ZFS, 225
  - zoned property, 228

