

Oracle® Solaris Administration: ZFS File Systems

Copyright © 2006, 2012, 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, the following notice is applicable:

U.S. GOVERNMENT RIGHTS. Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

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 on 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. 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.

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 disposition de votre contrat de licence ou de la loi, vous ne pouvez pas copier, reproduire, traduire, diffuser, modifier, breveter, 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 concédé sous licence au Gouvernement des Etats-Unis, ou à toute entité qui délivre la licence de ce logiciel ou l'utilise pour le compte du Gouvernement des Etats-Unis, la notice suivante s'applique:

U.S. GOVERNMENT RIGHTS. Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

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. 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.

Contents

Preface	11
1 Oracle Solaris ZFS File System (Introduction)	15
What's New in ZFS?	15
ZFS Manual Page Change (zfs.1m)	16
Improved aclmode Property	16
Identifying Pool Devices By Physical Location	16
ZFS Shadow Migration	18
ZFS File Sharing Improvements	18
ZFS File System Encryption	18
ZFS Send Stream Enhancements	19
ZFS Snapshot Differences (zfs diff)	19
ZFS Storage Pool Recovery and Performance Enhancements	19
Tuning ZFS Synchronous Behavior	20
Improved ZFS Pool Messages	20
ZFS ACL Interoperability Enhancements	21
Splitting a Mirrored ZFS Storage Pool (zpool split)	22
ZFS iSCSI Changes	23
New ZFS System Process	23
ZFS Deduplication Property	23
What Is Oracle Solaris ZFS?	24
ZFS Pooled Storage	24
Transactional Semantics	24
Checksums and Self-Healing Data	25
Unparalleled Scalability	25
ZFS Snapshots	26
Simplified Administration	26
ZFS Terminology	26

ZFS Component Naming Requirements	28
2 Getting Started With Oracle Solaris ZFS	29
ZFS Rights Profiles	29
ZFS Hardware and Software Requirements and Recommendations	30
Creating a Basic ZFS File System	30
Creating a Basic ZFS Storage Pool	31
▼ How to Identify Storage Requirements for Your ZFS Storage Pool	31
▼ How to Create a ZFS Storage Pool	32
Creating a ZFS File System Hierarchy	32
▼ How to Determine Your ZFS File System Hierarchy	33
▼ How to Create ZFS File Systems	33
3 Oracle Solaris ZFS and Traditional File System Differences	37
ZFS File System Granularity	37
ZFS Disk Space Accounting	38
Out of Space Behavior	39
Mounting ZFS File Systems	39
Traditional Volume Management	40
New Solaris ACL Model	40
4 Managing Oracle Solaris ZFS Storage Pools	41
Components of a ZFS Storage Pool	41
Using Disks in a ZFS Storage Pool	41
Using Slices in a ZFS Storage Pool	43
Using Files in a ZFS Storage Pool	44
Considerations for ZFS Storage Pools	44
Replication Features of a ZFS Storage Pool	45
Mirrored Storage Pool Configuration	45
RAID-Z Storage Pool Configuration	46
ZFS Hybrid Storage Pool	47
Self-Healing Data in a Redundant Configuration	47
Dynamic Striping in a Storage Pool	47
Creating and Destroying ZFS Storage Pools	48

Creating ZFS Storage Pools	48
Displaying Storage Pool Virtual Device Information	54
Handling ZFS Storage Pool Creation Errors	55
Destroying ZFS Storage Pools	57
Managing Devices in ZFS Storage Pools	58
Adding Devices to a Storage Pool	59
Attaching and Detaching Devices in a Storage Pool	63
Creating a New Pool By Splitting a Mirrored ZFS Storage Pool	65
Onlining and Offlining Devices in a Storage Pool	68
Clearing Storage Pool Device Errors	70
Replacing Devices in a Storage Pool	70
Designating Hot Spares in Your Storage Pool	73
Managing ZFS Storage Pool Properties	77
Querying ZFS Storage Pool Status	80
Displaying Information About ZFS Storage Pools	80
Viewing I/O Statistics for ZFS Storage Pools	85
Determining the Health Status of ZFS Storage Pools	87
Migrating ZFS Storage Pools	92
Preparing for ZFS Storage Pool Migration	92
Exporting a ZFS Storage Pool	92
Determining Available Storage Pools to Import	93
Importing ZFS Storage Pools From Alternate Directories	95
Importing ZFS Storage Pools	95
Recovering Destroyed ZFS Storage Pools	98
Upgrading ZFS Storage Pools	100
5 Managing ZFS Root Pool Components	103
Managing ZFS Root Pool Components (Overview)	103
ZFS Root Pool Requirements	104
Troubleshooting ZFS Root Pool Installation Problems	105
Managing Your ZFS Root Pool	106
Installing a ZFS Root Pool	106
▼ How to Update Your ZFS Boot Environment	107
▼ How to Mount an Alternate BE	108
▼ How to Configure a Mirrored Root Pool	109

▼ How to Replace a Disk in a ZFS Root Pool	110
▼ How to Create a BE in Another Root Pool	112
Managing Your ZFS Swap and Dump Devices	113
Adjusting the Sizes of Your ZFS Swap and Dump Devices	114
Troubleshooting ZFS Dump Device Issues	115
Booting From a ZFS Root File System	116
Booting From an Alternate Disk in a Mirrored ZFS Root Pool	116
Booting From a ZFS Root File System on a SPARC Based System	117
Booting From a ZFS Root File System on an x86 Based System	118
Booting For Recovery Purposes in a ZFS Root Environment	119
6 Managing Oracle Solaris ZFS File Systems	123
Managing ZFS File Systems (Overview)	123
Creating, Destroying, and Renaming ZFS File Systems	124
Creating a ZFS File System	124
Destroying a ZFS File System	125
Renaming a ZFS File System	126
Introducing ZFS Properties	127
ZFS Read-Only Native Properties	138
Settable ZFS Native Properties	139
ZFS User Properties	145
Querying ZFS File System Information	146
Listing Basic ZFS Information	147
Creating Complex ZFS Queries	147
Managing ZFS Properties	149
Setting ZFS Properties	149
Inheriting ZFS Properties	150
Querying ZFS Properties	151
Mounting ZFS File Systems	154
Managing ZFS Mount Points	154
Mounting ZFS File Systems	156
Using Temporary Mount Properties	157
Unmounting ZFS File Systems	158
Sharing and Unsharing ZFS File Systems	158
Legacy ZFS Sharing Syntax	160

New ZFS Sharing Syntax	160
ZFS Sharing Migration/Transition Issues	167
Setting ZFS Quotas and Reservations	167
Setting Quotas on ZFS File Systems	168
Setting Reservations on ZFS File Systems	171
Encrypting ZFS File Systems	173
Changing an Encrypted ZFS File System's Keys	175
Mounting an Encrypted ZFS File System	176
Interactions Between ZFS Compression, Deduplication, and Encryption Properties	176
Examples of Encrypting ZFS File Systems	177
Migrating ZFS File Systems	178
▼ How to Migrate a File System to a ZFS File System	180
Troubleshooting ZFS File System Migrations	181
Upgrading ZFS File Systems	181
7 Working With Oracle Solaris ZFS Snapshots and Clones	183
Overview of ZFS Snapshots	183
Creating and Destroying ZFS Snapshots	184
Displaying and Accessing ZFS Snapshots	187
Rolling Back a ZFS Snapshot	188
Identifying ZFS Snapshot Differences (<code>zfs diff</code>)	189
Overview of ZFS Clones	190
Creating a ZFS Clone	191
Destroying a ZFS Clone	191
Replacing a ZFS File System With a ZFS Clone	191
Sending and Receiving ZFS Data	192
Saving ZFS Data With Other Backup Products	193
Identifying ZFS Snapshot Streams	193
Sending a ZFS Snapshot	195
Receiving a ZFS Snapshot	196
Applying Different Property Values to a ZFS Snapshot Stream	197
Sending and Receiving Complex ZFS Snapshot Streams	199
Remote Replication of ZFS Data	202

8	Using ACLs and Attributes to Protect Oracle Solaris ZFS Files	203
	New Solaris ACL Model	203
	Syntax Descriptions for Setting ACLs	204
	ACL Inheritance	208
	ACL Properties	209
	Setting ACLs on ZFS Files	210
	Setting and Displaying ACLs on ZFS Files in Verbose Format	212
	Setting ACL Inheritance on ZFS Files in Verbose Format	217
	Setting and Displaying ACLs on ZFS Files in Compact Format	223
	Applying Special Attributes to ZFS Files	229
9	Oracle Solaris ZFS Delegated Administration	231
	Overview of ZFS Delegated Administration	231
	Disabling ZFS Delegated Permissions	232
	Delegating ZFS Permissions	232
	Delegating ZFS Permissions (<code>zfs allow</code>)	235
	Removing ZFS Delegated Permissions (<code>zfs unallow</code>)	235
	Delegating ZFS Permissions (Examples)	236
	Displaying ZFS Delegated Permissions (Examples)	240
	Removing ZFS Delegated Permissions (Examples)	241
10	Oracle Solaris ZFS Advanced Topics	243
	ZFS Volumes	243
	Using a ZFS Volume as a Swap or Dump Device	244
	Using a ZFS Volume as an iSCSI LUN	244
	Using ZFS on a Solaris System With Zones Installed	245
	Adding ZFS File Systems to a Non-Global Zone	246
	Delegating Datasets to a Non-Global Zone	247
	Adding ZFS Volumes to a Non-Global Zone	248
	Using ZFS Storage Pools Within a Zone	248
	Managing ZFS Properties Within a Zone	248
	Understanding the <code>zoned</code> Property	249
	Copying Zones to Other Systems	250
	Using ZFS Alternate Root Pools	251
	Creating ZFS Alternate Root Pools	251

Importing Alternate Root Pools	252
11 Oracle Solaris ZFS Troubleshooting and Pool Recovery	253
Identifying ZFS Failures	253
Missing Devices in a ZFS Storage Pool	254
Damaged Devices in a ZFS Storage Pool	254
Corrupted ZFS Data	254
Checking ZFS File System Integrity	255
File System Repair	255
File System Validation	255
Controlling ZFS Data Scrubbing	255
Resolving Problems With ZFS	257
Determining If Problems Exist in a ZFS Storage Pool	258
Reviewing <code>zpool status</code> Output	258
System Reporting of ZFS Error Messages	261
Repairing a Damaged ZFS Configuration	262
Resolving a Missing Device	262
Physically Reattaching a Device	264
Notifying ZFS of Device Availability	264
Replacing or Repairing a Damaged Device	265
Determining the Type of Device Failure	265
Clearing Transient Errors	266
Replacing a Device in a ZFS Storage Pool	267
Repairing Damaged Data	274
Identifying the Type of Data Corruption	274
Repairing a Corrupted File or Directory	275
Repairing ZFS Storage Pool-Wide Damage	277
Repairing an Unbootable System	278
12 Archiving Snapshots and Root Pool Recovery	281
Overview of ZFS Recovery Process	281
ZFS Pool Recovery Requirements	282
Creating a ZFS Snapshot Archive for Recovery	282
▼ How to Create a ZFS Snapshot Archive	282
Recreating Your Root Pool and Recovering Root Pool Snapshots	284

▼ How to Recreate the Root Pool on the Recovery System	284
13 Recommended Oracle Solaris ZFS Practices	289
Recommended Storage Pool Practices	289
General System Practices	289
ZFS Storage Pool Creation Practices	290
Storage Pool Practices for Performance	292
ZFS Storage Pool Maintenance and Monitoring Practices	293
Recommended File System Practices	294
File System Creation Practices	294
Monitoring ZFS File System Practices	295
A Oracle Solaris ZFS Version Descriptions	297
Overview of ZFS Versions	297
ZFS Pool Versions	297
ZFS File System Versions	299
Index	301

Preface

The *Oracle Solaris Administration: ZFS File Systems* provides information about setting up and managing Oracle Solaris ZFS file systems.

This guide contains information for both SPARC based and x86 based systems.

Note – This Oracle Solaris release supports systems that use the SPARC and x86 families of processor architectures. The supported systems appear in the *Oracle Solaris Hardware Compatibility List* at <http://www.oracle.com/webfolder/technetwork/hcl/index.html>. This document cites any implementation differences between the platform types.

Who Should Use This Book

This guide is intended for anyone who is interested in setting up and managing Oracle Solaris ZFS file systems. Experience using the Oracle Solaris operating system (OS) or another UNIX version is recommended.

How This Book Is Organized

The following table describes the chapters in this book.

Chapter	Description
Chapter 1, “Oracle Solaris ZFS File System (Introduction)”	Provides an overview of ZFS and its features and benefits. It also covers some basic concepts and terminology.
Chapter 2, “Getting Started With Oracle Solaris ZFS”	Provides step-by-step instructions on setting up basic ZFS configurations with basic pools and file systems. This chapter also provides the hardware and software required to create ZFS file systems.
Chapter 3, “Oracle Solaris ZFS and Traditional File System Differences”	Identifies important features that make ZFS significantly different from traditional file systems. Understanding these key differences will help reduce confusion when you use traditional tools to interact with ZFS.
Chapter 4, “Managing Oracle Solaris ZFS Storage Pools”	Provides a detailed description of how to create and administer ZFS storage pools.

Chapter	Description
Chapter 5, “Managing ZFS Root Pool Components”	Describes how to manage ZFS root pool components, such as configuring a mirrored root pool, upgrading your ZFS boot environments, and resizing swap and dump devices.
Chapter 6, “Managing Oracle Solaris ZFS File Systems”	Provides detailed information about managing ZFS file systems. Included are such concepts as the hierarchical file system layout, property inheritance, and automatic mount point management and share interactions.
Chapter 7, “Working With Oracle Solaris ZFS Snapshots and Clones”	Describes how to create and administer ZFS snapshots and clones.
Chapter 8, “Using ACLs and Attributes to Protect Oracle Solaris ZFS Files”	Describes how to use access control lists (ACLs) to protect your ZFS files by providing more granular permissions than the standard UNIX permissions.
Chapter 9, “Oracle Solaris ZFS Delegated Administration”	Describes how to use ZFS delegated administration to allow nonprivileged users to perform ZFS administration tasks.
Chapter 10, “Oracle Solaris ZFS Advanced Topics”	Provides information about using ZFS volumes, using ZFS on an Oracle Solaris system with zones installed, and using alternate root pools.
Chapter 11, “Oracle Solaris ZFS Troubleshooting and Pool Recovery”	Describes how to identify ZFS failures and how to recover from them. Steps for preventing failures are covered as well.
Chapter 12, “Archiving Snapshots and Root Pool Recovery”	Describes how to archive root pool snapshots and perform root pool recovery.
Chapter 13, “Recommended Oracle Solaris ZFS Practices”	Describes recommended practices for creating, monitoring, and maintaining your ZFS storage pools and file systems.
Appendix A, “Oracle Solaris ZFS Version Descriptions”	Describes available ZFS versions, features of each version, and the Solaris OS that provides the ZFS version and feature.

Related Books

Related information about general Oracle Solaris system administration topics can be found in the following books:

- *Oracle Solaris Administration: Common Tasks*
- *System Administration Guide: Advanced Administration*
- *Oracle Solaris Administration: Devices and File Systems*
- *Oracle Solaris Administration: Security Services*

Access to Oracle Support

Oracle customers 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.

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P-1 Typographic Conventions

Typeface	Meaning	Example
AaBbCc123	The names of commands, files, and directories, and onscreen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name% you have mail.</code>
AaBbCc123	What you type, contrasted with onscreen computer output	<code>machine_name% su</code> Password:
<i>aabbcc123</i>	Placeholder: replace with a real name or value	The command to remove a file is <i>rm filename</i> .
<i>AaBbCc123</i>	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . <i>A cache</i> is a copy that is stored locally. Do <i>not</i> save the file. Note: Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX system prompt and superuser prompt for shells that are included in the Oracle Solaris OS. Note that the default system prompt that is displayed in command examples varies, depending on the Oracle Solaris release.

TABLE P-2 Shell Prompts

Shell	Prompt
Bash shell, Korn shell, and Bourne shell	\$

TABLE P-2 Shell Prompts *(Continued)*

Shell	Prompt
Bash shell, Korn shell, and Bourne shell for superuser	#
C shell	machine_name%
C shell for superuser	machine_name#

Oracle Solaris ZFS File System (Introduction)

This chapter provides an overview of the Oracle Solaris ZFS file system and its features and benefits. This chapter also covers some basic terminology used throughout the rest of this book.

The following sections are provided in this chapter:

- “What's New in ZFS?” on page 15
- “What Is Oracle Solaris ZFS?” on page 24
- “ZFS Terminology” on page 26
- “ZFS Component Naming Requirements” on page 28

What's New in ZFS?

This section summarizes new features in the ZFS file system.

- “ZFS Manual Page Change (`zfs.1m`)” on page 16
- “Improved `ac_lmode` Property” on page 16
- “Identifying Pool Devices By Physical Location” on page 16
- “ZFS Shadow Migration” on page 18
- “ZFS File Sharing Improvements” on page 18
- “ZFS File System Encryption” on page 18
- “ZFS Send Stream Enhancements” on page 19
- “ZFS Snapshot Differences (`zfs diff`)” on page 19
- “ZFS Storage Pool Recovery and Performance Enhancements” on page 19
- “Tuning ZFS Synchronous Behavior” on page 20
- “Improved ZFS Pool Messages” on page 20
- “ZFS ACL Interoperability Enhancements” on page 21
- “Splitting a Mirrored ZFS Storage Pool (`zpool split`)” on page 22
- “ZFS iSCSI Changes” on page 23
- “New ZFS System Process” on page 23
- “ZFS Deduplication Property” on page 23

ZFS Manual Page Change (zfs.1m)

Oracle Solaris 11: The `zfs.1m` manual page has been revised so that core ZFS file system features remain in the `zfs.1m` page, but delegated administration, encryption, and share syntax and examples are covered in the following pages:

- `zfs_allow(1M)`
- `zfs_encrypt(1M)`
- `zfs_share(1M)`

Improved `aclmode` Property

Oracle Solaris 11: The `aclmode` property modifies Access Control List (ACL) behavior when a file is initially created or whenever controls how an ACL is modified during a `chmod` operation. The `aclmode` property has been reintroduced with the following property values:

- `discard` – A file system with an `aclmode` property of `discard` deletes all ACL entries that do not represent the mode of the file. This is the default value.
- `mask` – A file system with an `aclmode` property of `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 an explicit ACL set operation has not been performed.
- `passthrough` – A file system with an `aclmode` property of `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.

For more information, see [Example 8–14](#).

Identifying Pool Devices By Physical Location

Oracle Solaris 11: In this Solaris release, use the `zpool status -l` command to display physical disk location information for pool devices that is available from the `/dev/chassis` directory. This directory contains chassis, receptacle, and occupant values for the devices on your system.

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-J4400.0912QAJ001 SUN-Storage-J4400.rack22
```

For example:


```
% zpool status -l export
pool: export
state: ONLINE
scan: resilvered 379G in 8h31m with 0 errors on Thu Jan 27 23:10:20 2011
config:
```

NAME	STATE	READ	WRITE	CKSUM
export	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__2/disk	ONLINE	0	0	0
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__3/disk	ONLINE	0	0	0
mirror-1	ONLINE	0	0	0
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__4/disk	ONLINE	0	0	0
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__5/disk	ONLINE	0	0	0
mirror-2	ONLINE	0	0	0
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__6/disk	ONLINE	0	0	0
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__7/disk	ONLINE	0	0	0
mirror-3	ONLINE	0	0	0
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__8/disk	ONLINE	0	0	0
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__9/disk	ONLINE	0	0	0
mirror-4	ONLINE	0	0	0
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__10/disk	ONLINE	0	0	0
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__11/disk	ONLINE	0	0	0
spares				
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__0/disk	AVAIL			
/dev/chassis/SUN-Storage-J4400.rack22/SCSI_Device__1/disk	AVAIL			

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

The `zpool iostat` command has also been updated to provide physical location information for a pool's devices. For example:

```
# zpool iostat -lv
pool      capacity      operations      bandwidth
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/...rack22/SCSI_Device__2/disk  -  -    1     0   4.47K  60.3K
  /dev/chassis/...rack22/SCSI_Device__3/disk  -  -    1     0   4.45K  60.3K
mirror    490G   438G     2     5    8.62K  59.9K
  /dev/chassis/...rack22/SCSI_Device__4/disk  -  -    1     0   4.52K  59.9K
  /dev/chassis/...rack22/SCSI_Device__5/disk  -  -    1     0   4.48K  59.9K
mirror    490G   438G     2     5    8.60K  60.2K
  /dev/chassis/...rack22/SCSI_Device__6/disk  -  -    1     0   4.50K  60.2K
  /dev/chassis/...rack22/SCSI_Device__7/disk  -  -    1     0   4.49K  60.2K
mirror    490G   438G     2     5    8.47K  60.1K
.
.
.
```

In addition, the `crainfo`, `diskinfo`, `format`, and `prtconf` commands also provide physical disk location information. For more information, see [“Identifying Devices by Physical Locations”](#) in *Oracle Solaris Administration: Devices and File Systems*.

ZFS Shadow Migration

Oracle Solaris 11: In this release, you can migrate data from an old file system to a new file system while simultaneously allowing access and modification of the new file system during the migration process.

Setting the shadow property on a new ZFS file system triggers the migration of the older data. The shadow property can be set to migrate data from the local system or a remote system with either of the following values:

```
file:///path  
nfs://host:path
```

For more information, see [“Migrating ZFS File Systems”](#) on page 178.

ZFS File Sharing Improvements

Oracle Solaris 11: In this Solaris release, ZFS file systems are shared in a two-step process so that the share properties are set with one command and the NFS or SMB share is published in another step.

- You create the NFS or SMB share of a ZFS file system and identify the file sharing properties by using the `zfs set share` command.
- The NFS or SMB shares are published by setting the `sharenfs` or `sharesmb` property to on.

For more information, see [“Sharing and Unsharing ZFS File Systems”](#) on page 158.

ZFS File System Encryption

Oracle Solaris 11: In this release, you can encrypt a ZFS file system.

For example, the `tank/cindy` 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/cindy  
Enter passphrase for 'tank/cindy': xxx  
Enter again: xxx
```

An encryption policy is set when a ZFS file system is created. A file system's encryption policy is inherited by descendent file systems and cannot be removed.

For more information, see [“Encrypting ZFS File Systems”](#) on page 173.

ZFS Send Stream Enhancements

Oracle Solaris 11: In this release, you can set file system properties that are sent and received in a snapshot stream. These enhancements provide flexibility in applying file system properties in a send stream to the receiving file system or in determining whether the local file system properties, such as the `mountpoint` property value, should be ignored when received.

For more information, see [“Applying Different Property Values to a ZFS Snapshot Stream”](#) on page 197.

ZFS Snapshot Differences (`zfs diff`)

Oracle Solaris 11: In this release, you can determine ZFS snapshot differences by using the `zfs diff` command.

For example, assume that the following two snapshots are created:

```
$ ls /tank/cindy
fileA
$ zfs snapshot tank/cindy@0913
$ ls /tank/cindy
fileA fileB
$ zfs snapshot tank/cindy@0914
```

For example, to identify the differences between two snapshots, use syntax similar to the following:

```
$ zfs diff tank/cindy@0913 tank/cindy@0914
M      /tank/cindy/
+      /tank/cindy/fileB
```

In the output, the `M` indicates that the directory has been modified. The `+` indicates that `fileB` exists in the later snapshot.

For more information, see [“Identifying ZFS Snapshot Differences \(`zfs diff`\)”](#) on page 189.

ZFS Storage Pool Recovery and Performance Enhancements

Oracle Solaris 11: In this release, the following new ZFS storage pool features are provided:

- You can import a pool with a missing log by using the `zpool import -m` command. For more information, see [“Importing a Pool With a Missing Log Device”](#) on page 96.
- You can import a pool in read-only mode. This feature is primarily for pool recovery. If a damaged pool cannot be accessed because the underlying devices are damaged, you can import the pool read-only to recover the data. For more information, see [“Importing a Pool in Read-Only Mode”](#) on page 98.

- A RAID-Z (raidz1, raidz2, or raidz3) storage pool that is created in this release and upgraded to at least pool version 29 will have some latency-sensitive metadata automatically mirrored to improve read I/O throughput performance. For existing RAID-Z pools that are upgraded to at least pool version 29, some metadata will be mirrored for all newly written data.

Mirrored metadata in a RAID-Z pool does *not* provide additional protection against hardware failures, similar to what a mirrored storage pool provides. Additional space is consumed by mirrored metadata, but the RAID-Z protection remains the same as in previous releases. This enhancement is for performance purposes only.

Tuning ZFS Synchronous Behavior

Oracle Solaris 11: In this release, you can determine a ZFS file system's synchronous behavior by using the `sync` property.

The default synchronous behavior is to write all synchronous file system transactions to the intent log and to flush all devices to ensure that the data is stable. Disabling the default synchronous behavior is not recommended. Applications that depend on synchronous support might be affected, and data loss could occur.

The `sync` property can be set before or after the file system is created. In either case, the property value takes effect immediately. For example:

```
# zfs set sync=always tank/Neil
```

The `zil_disable` parameter is no longer available in Oracle Solaris releases that include the `sync` property.

For more information, see [Table 6-1](#).

Improved ZFS Pool Messages

Oracle Solaris 11: In this release, you can use the `-T` option to provide an interval and count value for the `zpool list` and `zpool status` commands to display additional information.

In addition, more pool scrub and resilver information is provided by the `zpool status` command as follows:

- Resilver in-progress report. For example:


```
scan: resilver in progress since Thu May 26 11:26:32 2011
1.26G scanned out of 2.40G at 6.15M/s, 0h3m to go
1.26G resilvered, 56.3% done
```
- Scrub in-progress report. For example:

```
scan: scrub in progress since Fri May 27 08:24:17 2011
18.0M scanned out of 2.35G at 8.99M/s, 0h4m to go
0 repaired, 0.75% done
```

- Resilver completion message. For example:

```
scan: resilvered 2.34G in 1h2m with 0 errors on Thu May 26 11:56:40 2011
```

- Scrub completion message. For example:

```
scan: scrub repaired 512B in 1h2m with 0 errors on Fri May 27 08:54:50 2011
```

- Ongoing scrub cancellation message. For example:

```
scan: scrub canceled on Wed Fri Jun 10 09:06:24 2011
```

- Scrub and resilver completion messages persist across system reboots

The following syntax uses the interval and count option to display ongoing pool resilvering information. You can use the `-T d` value to display the information in standard date format or `-T u` to display the information in an internal format.

```
# zpool status -T d tank 3 2
Wed Jun 22 14:35:40 GMT 2011
  pool: tank
  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 Wed Jun 22 14:33:29 2011
        3.42G scanned out of 7.75G at 28.2M/s, 0h2m to go
        3.39G resilvered, 44.13% done
  config:
    NAME        STATE      READ WRITE CKSUM
    tank        ONLINE    0     0     0
      mirror-0  ONLINE    0     0     0
        c2t3d0  ONLINE    0     0     0
        c2t4d0  ONLINE    0     0     0
      mirror-1  ONLINE    0     0     0
        c2t7d0  ONLINE    0     0     0
        c2t8d0  ONLINE    0     0     0 (resilvering)

errors: No known data errors
```

ZFS ACL Interoperability Enhancements

Oracle Solaris 11: In this release, the following ACL enhancements are provided:

- Trivial ACLs do not require deny Access control entries (ACEs) except for unusual permissions. For example, a mode of `0644`, `0755`, or `0664` does not require deny ACEs, but a mode, such as `0705`, `0060`, and so on, does require deny ACEs.

The old behavior includes deny ACEs in a trivial ACL like `644`. For example:

```
# ls -lv file.1
-rw-r--r--  1 root   root      206663 Jun 14 11:52 file.1
0:owner@:execute:deny
```

```

1:owner@:read_data/write_data/append_data/write_xattr/write_attributes
/write_acl/write_owner:allow
2:group@:write_data/append_data/execute:deny
3:group@:read_data:allow
4:everyone@:write_data/append_data/write_xattr/execute/write_attributes
/write_acl/write_owner:deny
5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow

```

The new behavior for a trivial ACL like 644 does not include the deny ACEs. For example:

```

# ls -v file.1
-rw-r--r--  1 root   root       206663 Jun 22 14:30 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/read_xattr/read_attributes/read_acl/synchronize:allow
 2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow

```

- ACLs are no longer split into multiple ACEs during inheritance to try to preserve the original unmodified permission. Instead, the permissions are modified as necessary to enforce the file creation mode.
- The `aclinherit` property behavior includes a reduction in permissions when the property is set to `restricted`, which means that ACLs are no longer split into multiple ACEs during inheritance.
- An existing ACL is discarded during `chmod(2)` operations by default. This change means that the ZFS `aclmode` property is no longer available.
- A new permission mode calculation rule specifies that if an ACL has a *user* ACE that is also the file owner, then those permissions are included in the permission mode computation. The same rule applies if a *group* ACE is the group owner of the file.

For more information, see [Chapter 8, “Using ACLs and Attributes to Protect Oracle Solaris ZFS Files.”](#)

Splitting a Mirrored ZFS Storage Pool (`zpool split`)

Oracle Solaris 11: In this release, you can use the `zpool split` command to split a mirrored storage pool, which detaches a disk or disks in the original mirrored pool to create another identical pool.

For more information, see “[Creating a New Pool By Splitting a Mirrored ZFS Storage Pool](#)” on page 65.

ZFS iSCSI Changes

Oracle Solaris 11: In this release, the iSCSI target daemon is replaced by using the Common Multiprotocol SCSI Target (COMSTAR) target daemon. This change also means that the `share_iscsi` property that was used to share a ZFS volume as an iSCSI LUN is no longer available. Use the `stmfadm` command to configure and share a ZFS volume as an iSCSI LUN.

For more information, see [“Using a ZFS Volume as an iSCSI LUN” on page 244](#).

New ZFS System Process

Oracle Solaris 11: In this release, each ZFS storage pool has an associated process, `zpool -poolname`. The threads in this process are the pool's I/O processing threads to handle I/O tasks, such as compression and checksum validation, that are associated with the pool. The purpose of this process is to provide visibility into each storage pool's CPU utilization.

Information about these running processes can be reviewed by using the `ps` and `prstat` commands. These processes are only available in the global zone. For more information, see [SDC\(7\)](#).

ZFS Deduplication Property

Oracle Solaris 11: In this release, you can use the deduplication (`dedup`) property to remove redundant data from your ZFS file systems. 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.

You can enable this property as follows:

```
# zfs set dedup=on tank/home
```

Although deduplication is set as a file system property, the scope is pool-wide. For example, you can identify the deduplication ratio as follows:

```
# zpool list tank
NAME      SIZE  ALLOC   FREE   CAP  DEDUP  HEALTH  ALTROOT
tank     136G  55.2G  80.8G   40%  2.30x  ONLINE  -
```

The `zpool list` output has been updated to support the deduplication property.

For more information about setting the deduplication property, see [“The dedup Property” on page 143](#).

Do not enable the `dedup` property on file systems that reside on production systems until you review the following considerations:

- Determine if your data would benefit from deduplication space savings
- Determine whether your system has enough physical memory to support deduplication
- Potential system performance impacts

For more information about these considerations, see [“The dedup Property”](#) on page 143.

What Is Oracle Solaris ZFS?

The Oracle Solaris ZFS file system is a file system that fundamentally changes the way file systems are administered, with features and benefits not found in other file systems available today. ZFS is robust, scalable, and easy to administer.

ZFS Pooled Storage

ZFS uses the concept of *storage pools* to manage physical storage. Historically, file systems were constructed on top of a single physical device. To address multiple devices and provide for data redundancy, the concept of a *volume manager* was introduced to provide a representation of a single device so that file systems would not need to be modified to take advantage of multiple devices. This design added another layer of complexity and ultimately prevented certain file system advances because the file system had no control over the physical placement of data on the virtualized volumes.

ZFS eliminates volume management altogether. Instead of forcing you to create virtualized volumes, ZFS aggregates devices into a storage pool. The storage pool describes the physical characteristics of the storage (device layout, data redundancy, and so on) and acts as an arbitrary data store from which file systems can be created. File systems are no longer constrained to individual devices, allowing them to share disk space with all file systems in the pool. You no longer need to predetermine the size of a file system, as file systems grow automatically within the disk space allocated to the storage pool. When new storage is added, all file systems within the pool can immediately use the additional disk space without additional work. In many ways, the storage pool works similarly to a virtual memory system: When a memory DIMM is added to a system, the operating system doesn't force you to run commands to configure the memory and assign it to individual processes. All processes on the system automatically use the additional memory.

Transactional Semantics

ZFS is a transactional file system, which means that the file system state is always consistent on disk. Traditional file systems overwrite data in place, which means that if the system loses power, for example, between the time a data block is allocated and when it is linked into a directory, the file system will be left in an inconsistent state. Historically, this problem was

solved through the use of the `fsck` command. This command was responsible for reviewing and verifying the file system state, and attempting to repair any inconsistencies during the process. This problem of inconsistent file systems caused great pain to administrators, and the `fsck` command was never guaranteed to fix all possible problems. More recently, file systems have introduced the concept of *journaling*. The journaling process records actions in a separate journal, which can then be *replayed* safely if a system crash occurs. This process introduces unnecessary overhead because the data needs to be written twice, often resulting in a new set of problems, such as when the journal cannot be replayed properly.

With a transactional file system, data is managed using *copy on write* semantics. Data is never overwritten, and any sequence of operations is either entirely committed or entirely ignored. Thus, the file system can never be corrupted through accidental loss of power or a system crash. Although the most recently written pieces of data might be lost, the file system itself will always be consistent. In addition, synchronous data (written using the `O_DSYNC` flag) is always guaranteed to be written before returning, so it is never lost.

Checksums and Self-Healing Data

With ZFS, all data and metadata is verified using a user-selectable checksum algorithm. Traditional file systems that do provide checksum verification have performed it on a per-block basis, out of necessity due to the volume management layer and traditional file system design. The traditional design means that certain failures, such as writing a complete block to an incorrect location, can result in data that is incorrect but has no checksum errors. ZFS checksums are stored in a way such that these failures are detected and can be recovered from gracefully. All checksum verification and data recovery are performed at the file system layer, and are transparent to applications.

In addition, ZFS provides for self-healing data. ZFS supports storage pools with varying levels of data redundancy. When a bad data block is detected, ZFS fetches the correct data from another redundant copy and repairs the bad data, replacing it with the correct data.

Unparalleled Scalability

A key design element of the ZFS file system is scalability. The file system itself is 128 bit, allowing for 256 quadrillion zettabytes of storage. All metadata is allocated dynamically, so no need exists to preallocate inodes or otherwise limit the scalability of the file system when it is first created. All the algorithms have been written with scalability in mind. Directories can have up to 2^{48} (256 trillion) entries, and no limit exists on the number of file systems or the number of files that can be contained within a file system.

ZFS Snapshots

A *snapshot* is a read-only copy of a file system or volume. Snapshots can be created quickly and easily. Initially, snapshots consume no additional disk space within the pool.

As data within the active dataset changes, the snapshot consumes disk space by continuing to reference the old data. As a result, the snapshot prevents the data from being freed back to the pool.

Simplified Administration

Most importantly, ZFS provides a greatly simplified administration model. Through the use of a hierarchical file system layout, property inheritance, and automatic management of mount points and NFS share semantics, ZFS makes it easy to create and manage file systems without requiring multiple commands or editing configuration files. You can easily set quotas or reservations, turn compression on or off, or manage mount points for numerous file systems with a single command. You can examine or replace devices without learning a separate set of volume manager commands. You can send and receive file system snapshot streams.

ZFS manages file systems through a hierarchy that allows for this simplified management of properties such as quotas, reservations, compression, and mount points. In this model, file systems are the central point of control. File systems themselves are very cheap (equivalent to creating a new directory), so you are encouraged to create a file system for each user, project, workspace, and so on. This design enables you to define fine-grained management points.

ZFS Terminology

This section describes the basic terminology used throughout this book:

boot environment	A boot environment is 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.
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 190 .
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. Datasets are identified using the following format:

pool/path[@snapshot]

pool Identifies the name of the storage pool that contains the dataset

path Is a slash-delimited path name for the dataset component

snapshot Is an optional component that identifies a snapshot of a dataset

For more information about datasets, see [Chapter 6, “Managing Oracle Solaris ZFS File Systems.”](#)

file system	A ZFS dataset of type <code>filesystem</code> that is mounted within the standard system namespace and behaves like other file systems. For more information about file systems, see Chapter 6, “Managing Oracle Solaris ZFS File Systems.”
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.
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 4, “Managing Oracle Solaris ZFS Storage Pools.”
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 46.
resilvering	The process of copying data from one device to another device is known as <i>resilvering</i> . 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. This process is referred to as <i>mirror resynchronization</i> in traditional volume management products. For more information about ZFS resilvering, see “Viewing Resilvering Status” on page 272.
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 183.
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 “Displaying Storage Pool Virtual Device Information” on page 54.
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 243.

ZFS Component Naming Requirements

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

- Each component can only contain alphanumeric characters in addition to the following four special characters:
 - Underscore (_)
 - Hyphen (-)
 - Colon (:)
 - Period (.)
- Pool names must begin with a letter, except for the following 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 (%).

In addition, empty components are not allowed.

Getting Started With Oracle Solaris ZFS

This chapter provides step-by-step instructions on setting up a basic Oracle Solaris ZFS configuration. 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 does not provide a comprehensive overview and refers to later chapters for more detailed information.

The following sections are provided in this chapter:

- “ZFS Rights Profiles” on page 29
- “ZFS Hardware and Software Requirements and Recommendations” on page 30
- “Creating a Basic ZFS File System” on page 30
- “Creating a Basic ZFS Storage Pool” on page 31
- “Creating a ZFS File System Hierarchy” on page 32

ZFS Rights Profiles

If you want to perform ZFS management tasks without using the superuser (root) account, you can assume a role with either of the following profiles to perform ZFS administration tasks:

- 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

For more information about creating or assigning roles, see *Oracle Solaris Administration: Security Services*.

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 9, “Oracle Solaris ZFS Delegated Administration.”](#)

ZFS Hardware and Software Requirements and Recommendations

Ensure that you review the following hardware and software requirements and recommendations before attempting to use the ZFS software:

- Use a SPARC or x86 based system that is running a supported Oracle Solaris release.
- The minimum amount of disk space required for a storage pool is 64 MB. The minimum disk size is 128 MB.
- The minimum amount of memory needed to install a Solaris system is 1 GB. However, for good ZFS performance, size the memory requirements based on your workload.
- If you create a mirrored pool configuration, use multiple controllers.

Creating a Basic ZFS File System

ZFS administration has been designed with simplicity in mind. Among the design goals is to reduce the number of commands needed to create a usable file system. For example, when you create a new pool, a new ZFS file system is created and mounted automatically.

The following example shows how to create a basic mirrored storage pool named `tank` and a ZFS file system named `tank` in one command. Assume that the whole disks `/dev/dsk/c1t0d0` and `/dev/dsk/c2t0d0` are available for use.

```
# zpool create tank mirror c1t0d0 c2t0d0
```

For more information about redundant ZFS pool configurations, see [“Replication Features of a ZFS Storage Pool” on page 45](#).

The new ZFS file system, `tank`, can use available disk space as needed, and is automatically mounted at `/tank`.

```
# mkfile 100m /tank/foo
# df -h /tank
Filesystem      size  used  avail capacity  Mounted on
tank            80G   100M   80G     1%     /tank
```

Within a pool, you probably want to create additional file systems. File systems provide points of administration that enable you to manage different sets of data within the same pool.

The following example shows how to create a file system named `fs` in the storage pool `tank`.

```
# zfs create tank/fs
```

The new ZFS file system, `tank/fs`, can use available disk space as needed, and is automatically mounted at `/tank/fs`.

```
# mkfile 100m /tank/fs/foo
# df -h /tank/fs
Filesystem      size  used  avail capacity  Mounted on
tank/fs         80G   100M   80G     1%        /tank/fs
```

Typically, you want to create and organize a hierarchy of file systems that matches your organizational needs. For information about creating a hierarchy of ZFS file systems, see [“Creating a ZFS File System Hierarchy” on page 32](#).

Creating a Basic ZFS Storage Pool

The previous example illustrates the simplicity of ZFS. The remainder of this chapter provides a more complete example, similar to what you would encounter in your environment. The first tasks are to identify your storage requirements and create a storage pool. The pool describes the physical characteristics of the storage and must be created before any file systems are created.

▼ How to Identify Storage Requirements for Your ZFS Storage Pool

1 Determine available devices for your storage pool.

Before creating a storage pool, you must 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. The devices can be individual slices on a preformatted disk, or they can be entire disks that ZFS formats as a single large slice.

In the storage example in [“How to Create a ZFS Storage Pool” on page 32](#), assume that the whole disks `/dev/dsk/c1t0d0` and `/dev/dsk/c2t0d0` are available for use.

For more information about disks and how they are used and labeled, see [“Using Disks in a ZFS Storage Pool” on page 41](#).

2 Choose data replication.

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

In the storage example in [“How to Create a ZFS Storage Pool” on page 32](#), basic mirroring of two available disks is used.

For more information about ZFS replication features, see [“Replication Features of a ZFS Storage Pool” on page 45](#).

▼ How to Create a ZFS Storage Pool

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

For more information about the ZFS rights profiles, see [“ZFS Rights Profiles” on page 29](#).

2 Pick a name for your storage pool.

This name is used to identify the storage pool when you are using the `zpool` and `zfs` commands. Pick any pool name that you prefer, but it must satisfy the naming requirements in [“ZFS Component Naming Requirements” on page 28](#).

3 Create the pool.

For example, the following command creates a mirrored pool that is named `tank`:

```
# zpool create tank mirror c1t0d0 c2t0d0
```

If one or more devices contains another file system or is otherwise in use, the command cannot create the pool.

For more information about creating storage pools, see [“Creating ZFS Storage Pools” on page 48](#). For more information about how device usage is determined, see [“Detecting In-Use Devices” on page 55](#).

4 View the results.

You can determine if your pool was successfully created by using the `zpool list` command.

```
# zpool list
NAME                SIZE  ALLOC  FREE  CAP  HEALTH  ALROOT
tank                 80G   137K   80G   0%   ONLINE  -
```

For more information about viewing pool status, see [“Querying ZFS Storage Pool Status” on page 80](#).

Creating a ZFS File System Hierarchy

After creating a storage pool to store your data, you can create your file system hierarchy. Hierarchies are simple yet powerful mechanisms for organizing information. They are also very familiar to anyone who has used a file system.

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

▼ How to Determine Your ZFS File System Hierarchy

1 Pick the file system granularity.

ZFS file systems are the central point of administration. They are lightweight and can be created easily. A good model to use is to establish one file system per user or project, as this model allows properties, snapshots, and backups to be controlled on a per-user or per-project basis.

Two ZFS file systems, `jeff` and `bill`, are created in “[How to Create ZFS File Systems](#)” on page 33.

For more information about managing file systems, see [Chapter 6, “Managing Oracle Solaris ZFS File Systems.”](#)

2 Group similar file systems.

ZFS allows file systems to be organized into hierarchies so that similar file systems can be grouped. This model provides a central point of administration for controlling properties and administering file systems. Similar file systems should be created under a common name.

In the example in “[How to Create ZFS File Systems](#)” on page 33, the two file systems are placed under a file system named `home`.

3 Choose the file system properties.

Most file system characteristics are controlled by properties. These properties control a variety of behaviors, including where the file systems are mounted, how they are shared, if they use compression, and if any quotas are in effect.

In the example in “[How to Create ZFS File Systems](#)” on page 33, all home directories are mounted at `/export/zfs/user`, are shared by using NFS, and have compression enabled. In addition, a quota of 10 GB on user `jeff` is enforced.

For more information about properties, see “[Introducing ZFS Properties](#)” on page 127.

▼ How to Create ZFS File Systems

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

For more information about the ZFS rights profiles, see “[ZFS Rights Profiles](#)” on page 29.

2 Create the desired hierarchy.

In this example, a file system that acts as a container for individual file systems is created.

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

3 Set the inherited properties.

After the file system hierarchy is established, set up any properties to be shared among all users:

```
# zfs set mountpoint=/export/zfs tank/home
# zfs set share=name=home,path=/export/zfs,prot=nfs tank/home
name=home,path=/export/zfs,prot=nfs
# zfs set sharenfs=on tank/home
# zfs set compression=on tank/home
# zfs get compression tank/home
NAME                PROPERTY          VALUE                SOURCE
tank/home           compression       on                   local
```

You can set file system properties when the file system is created. For example:

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

For more information about properties and property inheritance, see [“Introducing ZFS Properties” on page 127](#).

Next, individual file systems are grouped under the home file system in the pool tank.

4 Create the individual file systems.

File systems could have been created and then the properties could have been changed at the home level. All properties can be changed dynamically while file systems are in use.

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

These file systems inherit their property values from their parent, so they are automatically mounted at `/export/zfs/user` and are NFS shared. You do not need to edit the `/etc/vfstab` or `/etc/dfs/dfstab` file.

For more information about creating file systems, see [“Creating a ZFS File System” on page 124](#).

For more information about mounting and sharing file systems, see [“Mounting ZFS File Systems” on page 154](#).

5 Set the file system-specific properties.

In this example, user `jeff` is assigned a quota of 10 GBs. This property places a limit on the amount of space he can consume, regardless of how much space is available in the pool.

```
# zfs set quota=10G tank/home/jeff
```

6 View the results.

View available file system information by using the `zfs list` command:

```
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
tank                92.0K 67.0G   9.5K   /tank
tank/home          24.0K 67.0G    8K   /export/zfs
tank/home/bill      8K    67.0G   8K   /export/zfs/bill
tank/home/jeff      8K   10.0G   8K   /export/zfs/jeff
```

Note that user `jeff` only has 10 GB of space available, while user `bill` can use the full pool (67 GB).

For more information about viewing file system status, see [“Querying ZFS File System Information”](#) on page 146.

For more information about how disk space is used and calculated, see [“ZFS Disk Space Accounting”](#) on page 38.

Oracle Solaris ZFS and Traditional File System Differences

This chapter discusses some significant differences between Oracle Solaris ZFS and traditional file systems. Understanding these key differences can help reduce confusion when you use traditional tools to interact with ZFS.

The following sections are provided in this chapter:

- “ZFS File System Granularity” on page 37
- “ZFS Disk Space Accounting” on page 38
- “Out of Space Behavior” on page 39
- “Mounting ZFS File Systems” on page 39
- “Traditional Volume Management” on page 40
- “New Solaris ACL Model” on page 40

ZFS File System Granularity

Historically, file systems have been constrained to one device and thus to the size of that device. Creating and re-creating traditional file systems because of size constraints are time-consuming and sometimes difficult. Traditional volume management products help manage this process.

Because ZFS file systems are not constrained to specific devices, they can be created easily and quickly, similar to the way directories are created. ZFS file systems grow automatically within the disk space allocated to the storage pool in which they reside.

Instead of creating one file system, such as `/export/home`, to manage many user subdirectories, you can create one file system per user. You can easily set up and manage many file systems by applying properties that can be inherited by the descendent file systems contained within the hierarchy.

For an example that shows how to create a file system hierarchy, see “[Creating a ZFS File System Hierarchy](#)” on page 32.

ZFS Disk Space Accounting

ZFS is based on the concept of pooled storage. Unlike typical file systems, which are mapped to physical storage, all ZFS file systems in a pool share the available storage in the pool. So, the available disk space reported by utilities such as `df` might change even when the file system is inactive, as other file systems in the pool consume or release disk space.

Note that the maximum file system size can be limited by using quotas. For information about quotas, see [“Setting Quotas on ZFS File Systems” on page 168](#). A specified amount of disk space can be guaranteed to a file system by using reservations. For information about reservations, see [“Setting Reservations on ZFS File Systems” on page 171](#). This model is very similar to the NFS model, where multiple directories are mounted from the same file system (consider `/home`).

All metadata in ZFS is allocated dynamically. Most other file systems preallocate much of their metadata. As a result, at file system creation time, an immediate space cost for this metadata is required. This behavior also means that the total number of files supported by the file systems is predetermined. Because ZFS allocates its metadata as it needs it, no initial space cost is required, and the number of files is limited only by the available disk space. The output from the `df -g` command must be interpreted differently for ZFS than other file systems. The `total files` reported is only an estimate based on the amount of storage that is available in the pool.

ZFS is a transactional file system. Most file system modifications are bundled into transaction groups and committed to disk asynchronously. Until these modifications are committed to disk, they are called *pending changes*. The amount of disk space used, available, and referenced by a file or file system does not consider pending changes. Pending changes are generally accounted for within a few seconds. Even committing a change to disk by using `fsync(3c)` or `O_SYNC` does not necessarily guarantee that the disk space usage information is updated immediately.

On a UFS file system, the `du` command reports the size of the data blocks within the file. On a ZFS file system, `du` reports the actual size of the file as stored on disk. This size includes metadata as well as compression. This reporting really helps answer the question of “how much more space will I get if I remove this file?” So, even when compression is off, you will still see different results between ZFS and UFS.

When you compare the space consumption that is reported by the `df` command with the `zfs list` command, consider that `df` is reporting the pool size and not just file system sizes. In addition, `df` doesn’t understand descendent file systems or whether snapshots exist. If any ZFS properties, such as compression and quotas, are set on file systems, reconciling the space consumption that is reported by `df` might be difficult.

Consider the following scenarios that might also impact reported space consumption:

- For files that are larger than `recordsize`, the last block of the file is generally about 1/2 full. With the default `recordsize` set to 128 KB, approximately 64 KB is wasted per file, which might be a large impact. The integration of RFE 6812608 would resolve this scenario. You

can work around this by enabling compression. Even if your data is already compressed, the unused portion of the last block will be zero-filled, and compresses very well.

- On a RAIDZ-2 pool, every block consumes at least 2 sectors (512-byte chunks) of parity information. The space consumed by the parity information is not reported, but because it can vary, and be a much larger percentage for small blocks, an impact to space reporting might be seen. The impact is more extreme for a `recordsize` set to 512 bytes, where each 512-byte logical block consumes 1.5 KB (3 times the space). Regardless of the data being stored, if space efficiency is your primary concern, you should leave the `recordsize` at the default (128 KB), and enable compression (to the default of `lzjb`).
- The `df` command is not aware of deduplicated file data.

Out of Space Behavior

File system snapshots are inexpensive and easy to create in ZFS. Snapshots are common in most ZFS environments. For information about ZFS snapshots, see [Chapter 7, “Working With Oracle Solaris ZFS Snapshots and Clones.”](#)

The presence of snapshots can cause some unexpected behavior when you attempt to free disk space. Typically, given appropriate permissions, you can remove a file from a full file system, and this action results in more disk space becoming available in the file system. However, if the file to be removed exists in a snapshot of the file system, then no disk space is gained from the file deletion. The blocks used by the file continue to be referenced from the snapshot.

As a result, the file deletion can consume more disk space because a new version of the directory needs to be created to reflect the new state of the namespace. This behavior means that you can receive an unexpected `ENOSPC` or `EDQUOT` error when attempting to remove a file.

Mounting ZFS File Systems

ZFS reduces complexity and eases administration. For example, with traditional file systems, you must edit the `/etc/vfstab` file every time you add a new file system. ZFS has eliminated this requirement by automatically mounting and unmounting file systems according to the properties of the file system. You do not need to manage ZFS entries in the `/etc/vfstab` file.

For more information about mounting and sharing ZFS file systems, see [“Mounting ZFS File Systems” on page 154.](#)

Traditional Volume Management

As described in [“ZFS Pooled Storage” on page 24](#), ZFS eliminates the need for a separate volume manager. ZFS operates on raw devices, so it is possible to create a storage pool comprised of logical volumes, either software or hardware. This configuration is not recommended, as ZFS works best when it uses raw physical devices. Using logical volumes might sacrifice performance, reliability, or both, and should be avoided.

New Solaris ACL Model

Previous versions of the Solaris OS supported an ACL implementation that was primarily based on the POSIX ACL draft specification. The POSIX-draft based ACLs are used to protect UFS files. A new Solaris ACL model that is based on the NFSv4 specification is used to protect ZFS files.

The main differences of the new Solaris ACL model are as follows:

- The model is based on the NFSv4 specification and is similar to NT-style ACLs.
- This model provides a much more granular set of access privileges.
- ACLs are set and displayed with the `chmod` and `ls` commands rather than the `setfacl` and `getfacl` commands.
- Richer inheritance semantics designate how access privileges are applied from directory to subdirectories, and so on.

For more information about using ACLs with ZFS files, see [Chapter 8, “Using ACLs and Attributes to Protect Oracle Solaris ZFS Files.”](#)

Managing Oracle Solaris ZFS Storage Pools

This chapter describes how to create and administer storage pools in Oracle Solaris ZFS.

The following sections are provided in this chapter:

- “Components of a ZFS Storage Pool” on page 41
- “Replication Features of a ZFS Storage Pool” on page 45
- “Creating and Destroying ZFS Storage Pools” on page 48
- “Managing Devices in ZFS Storage Pools” on page 58
- “Managing ZFS Storage Pool Properties” on page 77
- “Querying ZFS Storage Pool Status” on page 80
- “Migrating ZFS Storage Pools” on page 92
- “Upgrading ZFS Storage Pools” on page 100

Components of a ZFS Storage Pool

The following sections provide detailed information about the following storage pool components:

- “Using Disks in a ZFS Storage Pool” on page 41
- “Using Slices in a ZFS Storage Pool” on page 43
- “Using Files in a ZFS Storage Pool” on page 44

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`). The recommended mode of operation is to use an entire disk, in which case the disk does not

require special formatting. ZFS formats the disk using an EFI label to contain a single, large slice. When used in this way, the partition table that is displayed by the `format` command appears similar to the following:

Current partition table (original):

Total disk sectors available: 286722878 + 16384 (reserved sectors)

Part	Tag	Flag	First Sector	Size	Last Sector
0	usr	wm	34	136.72GB	286722911
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	286722912	8.00MB	286739295

Review the following considerations when using whole disks in your ZFS storage pools:

- When using a whole disk, the disk is generally named by using 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 provide a slice to ZFS.
- On x86 based systems, the disk must have a valid Solaris `fdisk` partition. For more information about creating or changing a Solaris `fdisk` partition, see [Chapter 13, “x86: Setting Up Disks \(Tasks\),”](#) in *Oracle Solaris Administration: Devices and File Systems*.
- ZFS applies an EFI label when you create a storage pool with whole disks. For more information about EFI labels, see [“EFI Disk Label”](#) in *Oracle Solaris Administration: Devices and File Systems*.
- A disk that is intended for a ZFS root pool must be created with an SMI (VTOC) label, not an EFI label. You can relabel a disk with an SMI label by using the `format -e` command. Or, you can use the following command shortcuts to relabel a disk. Be cautioned that the shortcut commands do not include error checking.

The following commands can be used on a x86 system to relabel with an SMI label. The second command creates one Solaris `fdisk` partition that uses the entire disk.

```
x86# format -L vtoc -d c0t1d0
x86# fdisk -B /dev/rdisk/c0t1d0p0
```

The following command relabels the disk with an SMI label and the default partition table. The `s0` slice in the default partition table might not be large enough for the root pool.

```
sparc# format -L vtoc -d c0t1d0
```

For more information about converting a EFI label to an SMI (VTOC) label or changing the default partition table, see [Chapter 12, “SPARC: Setting Up Disks \(Tasks\),”](#) in *Oracle Solaris Administration: Devices and File Systems*.

Disks can be specified by using either the full path, such as `/dev/dsk/c1t0d0`, or a shorthand name that consists of the device name within the `/dev/dsk` directory, such as `c1t0d0`. For example, the following are valid disk names:

- `c1t0d0`
- `/dev/dsk/c1t0d0`
- `/dev/foo/disk`

Using Slices in a ZFS Storage Pool

Disks can be labeled with a traditional Solaris VTOC (SMI) label when you create a storage pool with a disk slice.

For a bootable ZFS root pool, the disks in the pool must contain slices and the disks must be labeled with an SMI label. The simplest configuration would be to put the entire disk capacity in slice 0 and use that slice for the root pool.

On a SPARC based system, a 72-GB disk has 68 GB of usable space located in slice 0 as shown in the following format output:

```
# format
.
.
.
Specify disk (enter its number): 4
selecting c1t1d0
partition> p
Current partition table (original):
Total disk cylinders available: 14087 + 2 (reserved cylinders)
```

Part	Tag	Flag	Cylinders	Size	Blocks
0	root	wm	0 - 14086	68.35GB	(14087/0/0) 143349312
1	unassigned	wm	0	0	(0/0/0) 0
2	backup	wm	0 - 14086	68.35GB	(14087/0/0) 143349312
3	unassigned	wm	0	0	(0/0/0) 0
4	unassigned	wm	0	0	(0/0/0) 0
5	unassigned	wm	0	0	(0/0/0) 0
6	unassigned	wm	0	0	(0/0/0) 0
7	unassigned	wm	0	0	(0/0/0) 0

On an x86 based system, a 72-GB disk has 68 GB of usable disk space located in slice 0, as shown in the following format output. A small amount of boot information is contained in slice 8. Slice 8 requires no administration and cannot be changed.

```
# format
.
.
.
selecting c1t0d0
partition> p
Current partition table (original):
```

Total disk cylinders available: 49779 + 2 (reserved cylinders)

Part	Tag	Flag	Cylinders	Size	Blocks
0	root	wm	1 - 49778	68.36GB	(49778/0/0) 143360640
1	unassigned	wu	0	0	(0/0/0) 0
2	backup	wm	0 - 49778	68.36GB	(49779/0/0) 143363520
3	unassigned	wu	0	0	(0/0/0) 0
4	unassigned	wu	0	0	(0/0/0) 0
5	unassigned	wu	0	0	(0/0/0) 0
6	unassigned	wu	0	0	(0/0/0) 0
7	unassigned	wu	0	0	(0/0/0) 0
8	boot	wu	0 - 0	1.41MB	(1/0/0) 2880
9	unassigned	wu	0	0	(0/0/0) 0

An `fdisk` partition also exists on Solaris x86 systems. An `fdisk` partition is represented by a `/dev/dsk/cN[tN]dNpN` device name and acts as a container for the disk's available slices. Do not use a `cN[tN]dNpN` device for a ZFS storage pool component because this configuration is neither tested nor supported.

Using Files in a ZFS Storage Pool

ZFS also allows you to use files as virtual devices in your storage pool. This feature is aimed primarily at testing and enabling simple experimentation, not for production use.

- 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.
- If you create a ZFS pool backed by files or volumes that are created on another ZFS pool, then the system might deadlock or panic.

However, files can be quite useful when you are first trying out ZFS or experimenting with more complicated configurations when insufficient physical devices are present. All files must be specified as complete paths and must be at least 64 MB in size.

Considerations for ZFS Storage Pools

Review the following considerations when creating and managing ZFS storage pools.

- Using whole physical disks is the easiest way to create ZFS storage pools. ZFS configurations become progressively more complex, from management, reliability, and performance perspectives, when you build pools from disk slices, LUNs in hardware RAID arrays, or volumes presented by software-based volume managers. The following considerations might help you determine how to configure ZFS with other hardware or software storage solutions:
 - If you construct a ZFS configuration on top of LUNs from hardware RAID arrays, you need to understand the relationship between ZFS redundancy features and the redundancy features offered by the array. Certain configurations might provide adequate redundancy and performance, but other configurations might not.

- You can construct logical devices for ZFS using volumes presented by software-based volume managers. However, these configurations are not recommended. Although ZFS functions properly on such devices, less-than-optimal performance might be the result. For additional information about storage pool recommendations, see [Chapter 13, “Recommended Oracle Solaris ZFS Practices.”](#)
- 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.

Replication Features of a ZFS Storage Pool

ZFS provides data redundancy, as well as self-healing properties, in mirrored and RAID-Z configurations.

- [“Mirrored Storage Pool Configuration” on page 45](#)
- [“RAID-Z Storage Pool Configuration” on page 46](#)
- [“Self-Healing Data in a Redundant Configuration” on page 47](#)
- [“Dynamic Striping in a Storage Pool” on page 47](#)
- [“ZFS Hybrid Storage Pool” on page 47](#)

Mirrored Storage Pool Configuration

A mirrored storage pool configuration requires at least two disks, preferably on separate controllers. Many disks can be used in a mirrored configuration. In addition, you can create more than one mirror in each pool. Conceptually, a basic mirrored configuration would look similar to the following:

```
mirror c1t0d0 c2t0d0
```

Conceptually, a more complex mirrored configuration would look similar to the following:

```
mirror c1t0d0 c2t0d0 c3t0d0 mirror c4t0d0 c5t0d0 c6t0d0
```

For information about creating a mirrored storage pool, see [“Creating a Mirrored Storage Pool” on page 48.](#)

RAID-Z Storage Pool Configuration

In addition to a mirrored storage pool configuration, ZFS provides a RAID-Z configuration with either single-, double-, or triple-parity fault tolerance. Single-parity RAID-Z (`raidz` or `raidz1`) is similar to RAID-5. Double-parity RAID-Z (`raidz2`) is similar to RAID-6.

For more information about RAIDZ-3 (`raidz3`), see the following blog:

http://blogs.oracle.com/ahl/entry/triple_parity_raid_z

All traditional RAID-5-like algorithms (RAID-4, RAID-6, RDP, and EVEN-ODD, for example) might experience a problem known as the *RAID-5 write hole*. If only part of a RAID-5 stripe is written, and power is lost before all blocks have been written to disk, the parity will remain unsynchronized with the data, and therefore forever useless, (unless a subsequent full-stripe write overwrites it). In RAID-Z, ZFS uses variable-width RAID stripes so that all writes are full-stripe writes. This design is only possible because 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. RAID-Z is the world's first software-only solution to the RAID-5 write hole.

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 device(s) 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.

Conceptually, a RAID-Z configuration with three disks would look similar to the following:

```
raidz c1t0d0 c2t0d0 c3t0d0
```

Conceptually, a more complex RAID-Z configuration would look similar to the following:

```
raidz c1t0d0 c2t0d0 c3t0d0 c4t0d0 c5t0d0 c6t0d0 c7t0d0
raidz c8t0d0 c9t0d0 c10t0d0 c11t0d0c12t0d0 c13t0d0 c14t0d0
```

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 should perform better.

For information about creating a RAID-Z storage pool, see “[Creating a RAID-Z Storage Pool](#)” on page 50.

For more information about choosing between a mirrored configuration or a RAID-Z configuration based on performance and disk space considerations, see the following blog entry:

http://blogs.oracle.com/roch/entry/when_to_and_not_to

For additional information about RAID-Z storage pool recommendations, see [Chapter 13](#), “Recommended Oracle Solaris ZFS Practices.”

ZFS Hybrid Storage Pool

The ZFS hybrid storage pool, available in Oracle's Sun Storage 7000 product series, is a special storage pool that 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 about this product, see the *Sun Storage Unified Storage System Administration Guide*.

Self-Healing Data in a Redundant Configuration

ZFS provides self-healing data in a mirrored or RAID-Z configuration.

When a bad data block is detected, not only does ZFS fetch the correct data from another redundant copy, but it also repairs the bad data by replacing it with the good copy.

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

Although ZFS supports combining different types of virtual devices within the same pool, avoid this practice. For example, you can create a pool with a two-way mirror and a three-way RAID-Z configuration. However, your fault tolerance is as good as your worst virtual device, RAID-Z in this case. A best practice is to use top-level virtual devices of the same type with the same redundancy level in each device.

Creating and Destroying ZFS Storage Pools

The following sections describe different scenarios for creating and destroying ZFS storage pools:

- “Creating ZFS Storage Pools” on page 48
- “Displaying Storage Pool Virtual Device Information” on page 54
- “Handling ZFS Storage Pool Creation Errors” on page 55
- “Destroying ZFS Storage Pools” on page 57

Creating and destroying pools is fast and easy. However, be cautious when performing these operations. Although checks are performed to prevent using devices known to be in use in a new pool, ZFS cannot always know when a device is already in use. Destroying a pool is easier than creating one. Use `zpool destroy` with caution. This simple command has significant consequences.

Creating ZFS Storage Pools

To create a storage pool, use the `zpool create` command. This command takes a pool name and any number of virtual devices as arguments. The pool name must satisfy the naming requirements in “ZFS Component Naming Requirements” on page 28.

Creating a Basic Storage Pool

The following command creates a new pool named `tank` that consists of the disks `c1t0d0` and `c1t1d0`:

```
# zpool create tank c1t0d0 c1t1d0
```

Device names representing the whole disks are found in the `/dev/dsk` directory and are labeled appropriately by ZFS to contain a single, large slice. Data is dynamically striped across both disks.

Creating a Mirrored Storage Pool

To create a mirrored pool, use the `mirror` keyword, followed by any number of storage devices that will comprise the mirror. Multiple mirrors can be specified by repeating the `mirror` keyword on the command line. The following command creates a pool with two, two-way mirrors:

```
# zpool create tank mirror c1d0 c2d0 mirror c3d0 c4d0
```

The second `mirror` keyword indicates that a new top-level virtual device is being specified. 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.”](#)

Currently, the following operations are supported in a ZFS mirrored configuration:

- Adding another set of disks for an additional top-level virtual device (vdev) to an existing mirrored configuration. For more information, see [“Adding Devices to a Storage Pool” on page 59.](#)
- Attaching additional disks to an existing mirrored configuration. Or, attaching additional disks to a non-replicated configuration to create a mirrored configuration. For more information, see [“Attaching and Detaching Devices in a Storage Pool” on page 63.](#)
- Replacing a disk or disks in an existing mirrored configuration as long as the replacement disks are greater than or equal to the size of the device to be replaced. For more information, see [“Replacing Devices in a Storage Pool” on page 70.](#)
- Detaching a disk in a mirrored configuration as long as the remaining devices provide adequate redundancy for the configuration. For more information, see [“Attaching and Detaching Devices in a Storage Pool” on page 63.](#)
- Splitting a mirrored configuration by detaching one of the disks to create a new, identical pool. For more information, see [“Creating a New Pool By Splitting a Mirrored ZFS Storage Pool” on page 65.](#)

You cannot outright remove a device that is not a spare, a log device, or a cache device from a mirrored storage pool.

Creating a ZFS Root Pool

Consider the following root pool configuration requirements:

- Disks used for the root pool must have a VTOC (SMI) label, and the pool must be created with disk slices.
- The root pool must be created as a mirrored configuration or as a single-disk configuration. 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.
- A RAID-Z or a striped configuration is not supported.
- The root pool cannot have a separate log device.
- If you attempt to use an unsupported configuration for a root pool, you see messages similar to the following:

```
ERROR: ZFS pool <pool-name> does not support boot environments
```

```
# zpool add -f rpool log c0t6d0s0
```

```
cannot add to 'rpool': root pool can not have multiple vdevs or separate logs
```

For more information about installing and booting a ZFS root file system, see [Chapter 5, “Managing ZFS Root Pool Components.”](#)

Creating a RAID-Z Storage Pool

Creating a single-parity RAID-Z pool is identical to creating a mirrored pool, except that the `raidz` or `raidz1` keyword is used instead of `mirror`. The following example shows how to create a pool with a single RAID-Z device that consists of five disks:

```
# zpool create tank raidz c1t0d0 c2t0d0 c3t0d0 c4t0d0 /dev/dsk/c5t0d0
```

This example illustrates that disks can be specified by using their shorthand device names or their full device names. Both `/dev/dsk/c5t0d0` and `c5t0d0` refer to the same disk.

You can create a double-parity or triple-parity RAID-Z configuration by using the `raidz2` or `raidz3` keyword when creating the pool. For example:

```
# zpool create tank raidz2 c1t0d0 c2t0d0 c3t0d0 c4t0d0 c5t0d0
# zpool status -v tank
pool: tank
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
tank	ONLINE	0	0	0
raidz2-0	ONLINE	0	0	0
c1t0d0	ONLINE	0	0	0
c2t0d0	ONLINE	0	0	0
c3t0d0	ONLINE	0	0	0
c4t0d0	ONLINE	0	0	0
c5t0d0	ONLINE	0	0	0

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

```
# zpool create tank raidz3 c0t0d0 c1t0d0 c2t0d0 c3t0d0 c4t0d0
c5t0d0 c6t0d0 c7t0d0 c8t0d0
# zpool status -v tank
pool: tank
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
tank	ONLINE	0	0	0
raidz3-0	ONLINE	0	0	0
c0t0d0	ONLINE	0	0	0
c1t0d0	ONLINE	0	0	0
c2t0d0	ONLINE	0	0	0
c3t0d0	ONLINE	0	0	0
c4t0d0	ONLINE	0	0	0
c5t0d0	ONLINE	0	0	0
c6t0d0	ONLINE	0	0	0
c7t0d0	ONLINE	0	0	0
c8t0d0	ONLINE	0	0	0

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

Currently, the following operations are supported in a ZFS RAID-Z configuration:

- Adding another set of disks for an additional top-level virtual device to an existing RAID-Z configuration. For more information, see “[Adding Devices to a Storage Pool](#)” on page 59.
- Replacing a disk or disks in an existing RAID-Z configuration as long as the replacement disks are greater than or equal to the size of the device to be replaced. For more information, see “[Replacing Devices in a Storage Pool](#)” on page 70.

Currently, the following operations are *not* supported in a RAID-Z configuration:

- Attaching an additional disk to an existing RAID-Z configuration.
- Detaching a disk from a RAID-Z configuration, except when you are detaching a disk that is replaced by a spare disk or when you need to detach a spare disk.
- You cannot outright remove a device that is not a log device or a cache device from a RAID-Z configuration. An RFE is filed for this feature.

For more information about a RAID-Z configuration, see “[RAID-Z Storage Pool Configuration](#)” on page 46.

Creating a ZFS Storage Pool With Log Devices

The ZFS intent log (ZIL) is provided to satisfy 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, better performance might be possible by using separate intent log devices, such as NVRAM or a dedicated disk.

Consider the following points when determining whether setting up a ZFS log device is appropriate for your environment:

- Log devices for the ZFS intent log are not related to database log files.
- Any performance improvement seen by implementing a separate log device depends on the device type, the hardware configuration of the pool, and the application workload. For preliminary performance information, see this blog: http://blogs.oracle.com/perrin/entry/slog_blog_or_blogging_on
- Log devices can be unreplicated or mirrored, but RAID-Z is not supported for log devices.
- If a separate log device is not mirrored and the device that contains the log fails, storing log blocks reverts to the storage pool.
- Log devices can be added, replaced, removed, attached, detached, imported, and exported as part of the larger storage pool.
- You can attach a log device to an existing log device to create a mirrored log device. This operation is identical to attaching a device in a unmirrored storage pool.

- 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 in-play data that might be stored on a log device is relatively small. Log blocks are freed when the log transaction (system call) is committed.
- The maximum size of a log device should be approximately 1/2 the size of physical memory because that is the maximum amount of potential in-play 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.

You can set up a ZFS log device when the storage pool is created or after the pool is created.

The following example shows how to create a mirrored storage pool with mirrored log devices:

```
# zpool create datap mirror c1t1d0 c1t2d0 mirror c1t3d0 c1t4d0
log mirror c1t5d0 c1t8d0
# 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
c1t1d0	ONLINE	0	0	0
c1t2d0	ONLINE	0	0	0
mirror-1	ONLINE	0	0	0
c1t3d0	ONLINE	0	0	0
c1t4d0	ONLINE	0	0	0
logs				
mirror-2	ONLINE	0	0	0
c1t5d0	ONLINE	0	0	0
c1t8d0	ONLINE	0	0	0

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

For information about recovering from a log device failure, see [Example 11-2](#).

Creating a ZFS Storage Pool With Cache Devices

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

You can create a storage pool with cache devices to cache storage pool data. For example:

```
# zpool create tank mirror c2t0d0 c2t1d0 c2t3d0 cache c2t5d0 c2t8d0
# zpool status tank
pool: tank
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
tank	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

After cache devices are added, they gradually fill with content from main memory. Depending on the size of your cache device, it could take over an hour for the device to fill. Capacity and reads can be monitored by using the `zpool iostat` command as follows:

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

Cache devices can be added or removed from a pool after the pool is created.

Consider the following points when determining whether to create a ZFS storage pool with cache devices:

- Using cache devices provides the greatest performance improvement for random-read workloads of mostly static content.
- Capacity and reads can be monitored by using the `zpool iostat` command.
- Single or multiple cache devices can be added when the pool is created. They can also be added and removed after the pool is created. For more information, see [Example 4-4](#).
- Cache devices cannot be mirrored or be part of a RAID-Z configuration.
- 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 the cache devices is considered volatile, similar to other system caches.

Cautions For Creating Storage Pools

Review the following cautions when creating and managing ZFS storage pools.

- Do not repartition or relabel disks that are part of an existing storage pool. If you attempt to repartition or relabel a root pool disk, you might have to reinstall the OS.
- Do not create a storage pool that contains components from another storage pool, such files or volumes. Deadlocks can occur in this unsupported configuration.
- A pool created with a single slice or single disk has no redundancy and is at risk for data loss. A pool created with multiple slices but no redundancy is also at risk for data loss. A pool created with multiple slices across disks is harder to manage than a pool created with whole disks.
- A pool that is not created with ZFS redundancy (RAIDZ or mirror) can only report data inconsistencies. It cannot repair data inconsistencies.

- 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. Performing routine backups of pool data on non-enterprise grade hardware is important.
- A pool cannot be shared across systems. ZFS is not a cluster file system.

Displaying Storage Pool Virtual Device Information

Each storage pool contains one or more virtual devices. A *virtual device* is an internal representation of the storage pool that describes the layout of physical storage and the storage pool's fault characteristics. As such, a virtual device represents the disk devices or files that are used to create the storage pool. A pool can have any number of virtual devices at the top of the configuration, known as a *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. These virtual devices consist of disks, disk slices, or files. A spare is a special virtual dev that tracks available hot spares for a pool.

The following example shows how to create a pool that consists of two top-level virtual devices, each a mirror of two disks:

```
# zpool create tank mirror c1d0 c2d0 mirror c3d0 c4d0
```

The following example shows how to create a pool that consists of one top-level virtual device of four disks:

```
# zpool create mypool raidz2 c1d0 c2d0 c3d0 c4d0
```

You can add another top-level virtual device to this pool by using the `zpool add` command. For example:

```
# zpool add mypool raidz2 c2d1 c3d1 c4d1 c5d1
```

Disks, disk slices, or files that are used in nonredundant pools function as top-level virtual devices. Storage pools typically contain multiple top-level virtual devices. ZFS dynamically stripes data among all of the top-level virtual devices in a pool.

Virtual devices and the physical devices that are contained in a ZFS storage pool are displayed with the `zpool status` command. For example:

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

          NAME                STATE                READ WRITE CKSUM
```

```

tank          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

Handling ZFS Storage Pool Creation Errors

Pool creation errors can occur for many reasons. Some reasons are obvious, such as when a specified device doesn't exist, while other reasons are more subtle.

Detecting In-Use Devices

Before formatting a device, ZFS first determines if the disk is in-use by ZFS or some other part of the operating system. If the disk is in use, you might see errors such as the following:

```

# zpool create tank c1t0d0 c1t1d0
invalid vdev specification
use '-f' to override the following errors:
/dev/dsk/c1t0d0s0 is currently mounted on /. Please see umount(1M).
/dev/dsk/c1t0d0s1 is currently mounted on swap. Please see swap(1M).
/dev/dsk/c1t1d0s0 is part of active ZFS pool zeeppool. Please see zpool(1M).

```

Some errors can be overridden by using the `-f` option, but most errors cannot. The following conditions cannot be overridden by using the `-f` option, and you must manually correct them:

- | | |
|--|--|
| Mounted file system | The disk or one of its slices contains a file system that is currently mounted. To correct this error, use the <code>umount</code> command. |
| File system in <code>/etc/vfstab</code> | The disk contains a file system that is listed in the <code>/etc/vfstab</code> file, but the file system is not currently mounted. To correct this error, remove or comment out the line in the <code>/etc/vfstab</code> file. |
| Dedicated dump device | The disk is in use as the dedicated dump device for the system. To correct this error, use the <code>dumpadm</code> command. |
| Part of a ZFS pool | The disk or file is part of an active ZFS storage pool. To correct this error, use the <code>zpool destroy</code> command to destroy the other pool, if it is no longer needed. Or, use the <code>zpool detach</code> command to detach the disk from the other pool. You can only detach a disk from a mirrored storage pool. |

The following in-use checks serve as helpful warnings and can be overridden by using the `-f` option to create the pool:

Contains a file system	The disk contains a known file system, though it is not mounted and doesn't appear to be in use.
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, as the disk might or might not be a network-attached drive in use by another system. Be cautious when overriding a potentially active pool.

The following example demonstrates how the `-f` option is used:

```
# zpool create tank c1t0d0
invalid vdev specification
use '-f' to override the following errors:
/dev/dsk/c1t0d0s0 contains a ufs filesystem.
# zpool create -f tank c1t0d0
```

Ideally, correct the errors rather than use the `-f` option to override them.

Mismatched Replication Levels

Creating pools with virtual devices of different replication levels is not recommended. The `zpool` command tries to prevent you from accidentally creating a pool with mismatched levels of redundancy. If you try to create a pool with such a configuration, you see errors similar to the following:

```
# zpool create tank c1t0d0 mirror c2t0d0 c3t0d0
invalid vdev specification
use '-f' to override the following errors:
mismatched replication level: both disk and mirror vdevs are present
# zpool create tank 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
```

You can override these errors with the `-f` option, but you should avoid this practice. The command also warns you about creating a mirrored or RAID-Z pool using devices of different sizes. Although this configuration is allowed, mismatched levels of redundancy result in unused disk space on the larger device. The `-f` option is required to override the warning.

Doing a Dry Run of Storage Pool Creation

Attempts to create a pool can fail unexpectedly in different ways, and formatting disks is a potentially harmful action. For these reasons, the `zpool create` command has an additional option, `-n`, which simulates creating the pool without actually writing to the device. This *dry run* option performs the device in-use checking and replication-level validation, and reports any errors in the process. If no errors are found, you see output similar to the following:


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

```
    tank
      mirror
        c1t0d0
        c1t1d0
```

Some errors cannot be detected without actually creating the pool. The most common example is specifying the same device twice in the same configuration. This error cannot be reliably detected without actually writing the data, so the `zpool create -n` command can report success and yet fail to create the pool when the command is run without this option.

Default Mount Point for Storage Pools

When a pool is created, the default mount point for the top-level file system is `/pool-name`. This directory must either not exist or be empty. If the directory does not exist, it is automatically created. If the directory is empty, the root file system is mounted on top of the existing directory. To create a pool with a different default mount point, use the `-m` option of the `zpool create` command. For example:

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

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

For more information about mount points, see [“Managing ZFS Mount Points” on page 154](#).

Destroying ZFS Storage Pools

Pools are destroyed by using the `zpool destroy` command. This command destroys the pool even if it contains mounted datasets.

```
# zpool destroy tank
```



Caution – Be very careful when you destroy a pool. Ensure that you are destroying the right pool and you always have copies of your data. If you accidentally destroy the wrong pool, you can attempt to recover the pool. For more information, see [“Recovering Destroyed ZFS Storage Pools” on page 98](#).

If you destroy a pool with the `zpool destroy` command, the pool is still available for import as described in [“Recovering Destroyed ZFS Storage Pools” on page 98](#). This means that

confidential data might still be available on the disks that were part of the pool. If you want to destroy data on the destroyed pool's disks, you must use a feature like the `format` utility's `analyze->purge` option on every disk in the destroyed pool.

Another option for keeping file system data confidential is to create encrypted ZFS file systems. When a pool with an encrypted file system is destroyed, the data would not be accessible without the encryption keys, even if the destroyed pool was recovered. For more information, see [“Encrypting ZFS File Systems” on page 173](#).

Destroying a Pool With Faulted Devices

The act of destroying a pool requires data to be written to disk to indicate that the pool is no longer valid. This state information prevents the devices from showing up as a potential pool when you perform an import. If one or more devices are unavailable, the pool can still be destroyed. However, the necessary state information won't be written to these unavailable devices.

These devices, when suitably repaired, are reported as *potentially active* when you create a new pool. They appear as valid devices when you search for pools to import. If a pool has enough faulted devices such that the pool itself is faulted (meaning that a top-level virtual device is faulted), then the command prints a warning and cannot complete without the `-f` option. This option is necessary because the pool cannot be opened, so whether data is stored there is unknown. For example:

```
# zpool destroy tank
cannot destroy 'tank': pool is faulted
use '-f' to force destruction anyway
# zpool destroy -f tank
```

For more information about pool and device health, see [“Determining the Health Status of ZFS Storage Pools” on page 87](#).

For more information about importing pools, see [“Importing ZFS Storage Pools” on page 95](#).

Managing Devices in ZFS Storage Pools

Most of the basic information regarding devices is covered in [“Components of a ZFS Storage Pool” on page 41](#). After a pool has been created, you can perform several tasks to manage the physical devices within the pool.

- [“Adding Devices to a Storage Pool” on page 59](#)
- [“Attaching and Detaching Devices in a Storage Pool” on page 63](#)
- [“Creating a New Pool By Splitting a Mirrored ZFS Storage Pool” on page 65](#)
- [“Onlining and Offlining Devices in a Storage Pool” on page 68](#)
- [“Clearing Storage Pool Device Errors” on page 70](#)

- “Replacing Devices in a Storage Pool” on page 70
- “Designating Hot Spares in Your Storage Pool” on page 73

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

```
# zpool add zeepool mirror c2t1d0 c2t2d0
```

The format for specifying the virtual devices is the same as for the `zpool create` command. Devices are checked to determine if they are in use, and the command cannot change the level of redundancy without the `-f` option. The command also supports the `-n` option so that you can perform a dry run. For example:

```
# zpool add -n zeepool mirror c3t1d0 c3t2d0
would update 'zeepool' to the following configuration:
zeepool
  mirror
    c1t0d0
    c1t1d0
  mirror
    c2t1d0
    c2t2d0
  mirror
    c3t1d0
    c3t2d0
```

This command syntax would add mirrored devices `c3t1d0` and `c3t2d0` to the `zeepool` pool's existing configuration.

For more information about how virtual device validation is done, see “[Detecting In-Use Devices](#)” on page 55.

EXAMPLE 4-1 Adding Disks to a Mirrored ZFS Configuration

In the following example, another mirror is added to an existing mirrored ZFS configuration.

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

NAME	STATE	READ	WRITE	CKSUM
tank	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c0t1d0	ONLINE	0	0	0
c1t1d0	ONLINE	0	0	0

EXAMPLE 4-1 Adding Disks to a Mirrored ZFS Configuration *(Continued)*

```

mirror-1 ONLINE      0      0      0
  c0t2d0 ONLINE      0      0      0
  c1t2d0 ONLINE      0      0      0
    
```

```

errors: No known data errors
# zpool add tank mirror c0t3d0 c1t3d0
# zpool status tank
  pool: tank
  state: ONLINE
  scrub: none requested
config:
    
```

NAME	STATE	READ	WRITE	CKSUM
tank	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
    
```

EXAMPLE 4-2 Adding Disks to a RAID-Z Configuration

Additional disks can be added similarly to a RAID-Z configuration. The following example shows how to convert a storage pool with one RAID-Z device that contains three disks to a storage pool with two RAID-Z devices that contains three disks each.

```

# 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

```

errors: No known data errors
# zpool add rzpool raidz c2t2d0 c2t3d0 c2t4d0
# zpool status rzpool
  pool: rzpool
  state: ONLINE
  scrub: none requested
config:
    
```

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

EXAMPLE 4-2 Adding Disks to a RAID-Z Configuration *(Continued)*

```

rzpool      ONLINE      0      0      0
raidz1-0    ONLINE      0      0      0
c1t0d0     ONLINE      0      0      0
c1t2d0     ONLINE      0      0      0
c1t3d0     ONLINE      0      0      0
raidz1-1    ONLINE      0      0      0
c2t2d0     ONLINE      0      0      0
c2t3d0     ONLINE      0      0      0
c2t4d0     ONLINE      0      0      0

```

errors: No known data errors

EXAMPLE 4-3 Adding and Removing a Mirrored Log Device

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

```

# 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

```
# 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

```

errors: No known data errors

You can attach a log device to an existing log device to create a mirrored log device. This operation is identical to attaching a device in an unmirrored storage pool.

You can remove log devices by using the `zpool remove` command. The mirrored log device in the previous example can be removed by specifying the `mirror-1` argument. For example:

EXAMPLE 4-3 Adding and Removing a Mirrored Log Device *(Continued)*

```
# 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
```

If your pool configuration contains only one log device, you remove the log device by specifying the device name. For example:

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

    NAME          STATE          READ WRITE CKSUM
    pool          ONLINE         0     0     0
      raidz1-0    ONLINE         0     0     0
        c0t8d0    ONLINE         0     0     0
        c0t9d0    ONLINE         0     0     0
    logs
      c0t10d0     ONLINE         0     0     0

errors: No known data errors
# zpool remove pool c0t10d0
```

EXAMPLE 4-4 Adding and Removing Cache Devices

You can add cache devices to your ZFS storage pool and remove them if they are no longer required.

Use the `zpool add` command to add cache devices. For example:

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

    NAME          STATE          READ WRITE CKSUM
    tank          ONLINE         0     0     0
      mirror-0    ONLINE         0     0     0
```

EXAMPLE 4-4 Adding and Removing Cache Devices *(Continued)*

```

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

Cache devices cannot be mirrored or be part of a RAID-Z configuration.

Use the `zpool remove` command to remove cache devices. For example:

```

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

```

```

NAME          STATE      READ WRITE CKSUM
tank          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

Currently, the `zpool remove` command only supports removing hot spares, log devices, and cache devices. Devices that are part of the main mirrored pool configuration can be removed by using the `zpool detach` command. Nonredundant and RAID-Z devices cannot be removed from a pool.

For more information about using cache devices in a ZFS storage pool, see [“Creating a ZFS Storage Pool With Cache Devices” on page 52](#).

Attaching and Detaching Devices in a Storage Pool

In addition to the `zpool add` command, you can use the `zpool attach` command to add a new device to an existing mirrored or nonmirrored device.

If you are attaching a disk to create a mirrored root pool, see [“How to Configure a Mirrored Root Pool” on page 109](#).

If you are replacing a disk in a ZFS root pool, see [“How to Replace a Disk in a ZFS Root Pool” on page 110](#).

EXAMPLE 4-5 Converting a Two-Way Mirrored Storage Pool to a Three-way Mirrored Storage Pool

In this example, `zpool` is an existing two-way mirror that is converted to a three-way mirror by attaching `c2t1d0`, the new device, to the existing device, `c1t1d0`.

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

    NAME      STATE      READ WRITE CKSUM
    zpool     ONLINE    0     0     0
      mirror-0 ONLINE    0     0     0
        c0t1d0 ONLINE    0     0     0
        c1t1d0 ONLINE    0     0     0

errors: No known data errors
# zpool attach zpool c1t1d0 c2t1d0
# zpool status zpool
pool: zpool
state: ONLINE
scrub: resilver completed after 0h0m with 0 errors on Fri Jan  8 12:59:20 2010
config:

    NAME      STATE      READ WRITE CKSUM
    zpool     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

errors: No known data errors
```

If the existing device is part of a three-way mirror, attaching the new device creates a four-way mirror, and so on. Whatever the case, the new device begins to resilver immediately.

EXAMPLE 4-6 Converting a Nonredundant ZFS Storage Pool to a Mirrored ZFS Storage Pool

In addition, you can convert a nonredundant storage pool to a redundant storage pool by using the `zpool attach` command. For example:

```
# zpool create tank c0t1d0
# zpool status tank
pool: tank
state: ONLINE
scrub: none requested
config:

    NAME      STATE      READ WRITE CKSUM
    tank     ONLINE    0     0     0
      c0t1d0  ONLINE    0     0     0

errors: No known data errors
# zpool attach tank c0t1d0 c1t1d0
# zpool status tank
```


EXAMPLE 4-6 Converting a Nonredundant ZFS Storage Pool to a Mirrored ZFS Storage Pool
(Continued)

```
pool: tank
state: ONLINE
scrub: resilver completed after 0h0m with 0 errors on Fri Jan  8 14:28:23 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM	
tank	ONLINE	0	0	0	
mirror-0	ONLINE	0	0	0	
c0t1d0	ONLINE	0	0	0	
c1t1d0	ONLINE	0	0	0	73.5K resilvered

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

You can use the `zpool detach` command to detach a device from a mirrored storage pool. For example:

```
# zpool detach zeepool c2t1d0
```

However, this operation fails if no other valid replicas of the data exist. For example:

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

Creating a New Pool By Splitting a Mirrored ZFS Storage Pool

A mirrored ZFS storage pool can be quickly cloned as a backup pool by using the `zpool split` command. You can use this feature to split a mirrored root pool, but the pool that is split off is not bootable.

You can use the `zpool split` command to detach one or more disks from a mirrored ZFS storage pool to create a new pool with the detached disk or disks. The new pool will have identical contents to the original mirrored ZFS storage pool.

By default, a `zpool split` operation on a mirrored pool detaches the last disk for the newly created pool. After the split operation, you then import the new pool. For example:

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

NAME	STATE	READ	WRITE	CKSUM
tank	ONLINE	0	0	0

```

        mirror-0 ONLINE      0    0    0
        c1t0d0  ONLINE      0    0    0
        c1t2d0  ONLINE      0    0    0

errors: No known data errors
# zpool split tank tank2
# zpool import tank2
# zpool status tank tank2
  pool: tank
  state: ONLINE
  scrub: none requested
config:

    NAME      STATE    READ WRITE CKSUM
    tank      ONLINE   0     0     0
    c1t0d0    ONLINE   0     0     0

errors: No known data errors

  pool: tank2
  state: ONLINE
  scrub: none requested
config:

    NAME      STATE    READ WRITE CKSUM
    tank2     ONLINE   0     0     0
    c1t2d0    ONLINE   0     0     0

errors: No known data errors
    
```

You can identify which disk should be used for the newly created pool by specifying it with the `zpool split` command. For example:

```
# zpool split tank tank2 c1t0d0
```

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. A new pool GUID is generated 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 /tank2 tank tank2
```

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.

Review the following considerations before using the `zpool split` feature:

- This feature is not available for a RAIDZ configuration or a non-redundant pool of multiple disks.
- Data and application operations should be quiesced before attempting a `zpool split` operation.
- Having disks that honor, rather than ignore, the disk's flush write cache command is important.
- A pool cannot be split if resilvering is in process.
- Splitting a mirrored pool is optimal when the pool contains two to three disks, where the last disk in the original pool is used for the newly created pool. Then, you can use the `zpool attach` command to re-create your original mirrored storage pool or convert your newly created pool into a mirrored storage pool. No method currently exists to create a *new* mirrored pool from an *existing* mirrored pool by using this feature.
- If the existing pool is a three-way mirror, then the new pool will contain one disk after the split operation. If the existing pool is a two-way mirror of two disks, then the outcome is two non-redundant pools of two disks. You must attach two additional disks to convert the non-redundant pools to mirrored pools.
- A good way to keep your data redundant during a split operation is to split a mirrored storage pool that contains three disks so that the original pool contains two mirrored disks after the split operation.

EXAMPLE 4-7 Splitting a Mirrored ZFS Pool

In the following example, a mirrored storage pool called `trinity`, with three disks, `c1t0d0`, `c1t2d0` and `c1t3d0`, is split. The two resulting pools are the mirrored pool `trinity`, with disks `c1t0d0` and `c1t2d0`, and the new pool, `neo`, with disk `c1t3d0`. Each pool has identical content.

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

    NAME      STATE    READ WRITE CKSUM
    trinity   ONLINE    0     0     0
    mirror-0  ONLINE    0     0     0
        c1t0d0  ONLINE    0     0     0
        c1t2d0  ONLINE    0     0     0
        c1t3d0  ONLINE    0     0     0

errors: No known data errors
# zpool split trinity neo
# zpool import neo
# zpool status trinity neo
pool: neo
state: ONLINE
```

EXAMPLE 4-7 Splitting a Mirrored ZFS Pool (Continued)

```

scrub: none requested
config:

    NAME          STATE      READ WRITE CKSUM
    neo            ONLINE    0     0     0
    c1t3d0         ONLINE    0     0     0

errors: No known data errors

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

        NAME          STATE      READ WRITE CKSUM
        trinity        ONLINE    0     0     0
        mirror-0       ONLINE    0     0     0
        c1t0d0         ONLINE    0     0     0
        c1t2d0         ONLINE    0     0     0

errors: No known data errors

```

Onlining and Offlining Devices in a Storage Pool

ZFS allows individual devices to be taken offline or brought online. When hardware is unreliable or not functioning properly, ZFS continues to read data from or write data to the device, assuming the condition is only temporary. If the condition is not temporary, you can instruct ZFS to ignore the device by taking it offline. ZFS does not send any requests to an offline device.

Note – Devices do not need to be taken offline in order to replace them.

Taking a Device Offline

You can take a device offline by using the `zpool offline` command. The device can be specified by path or by short name, if the device is a disk. For example:

```
# zpool offline tank c1t0d0
bringing device c1t0d0 offline
```

Consider the following points when taking a device offline:

- You cannot take a pool offline to the point where it becomes faulted. For example, you cannot take offline two devices in a `raidz1` configuration, nor can you take offline a top-level virtual device.

```
# zpool offline tank c1t0d0
cannot offline c1t0d0: no valid replicas
```

- By default, the OFFLINE state is persistent. The device remains offline when the system is rebooted.

To temporarily take a device offline, use the `zpool offline -t` option. For example:

```
# zpool offline -t tank c1t0d0
bringing device 'c1t0d0' offline
```

When the system is rebooted, this device is automatically returned to the ONLINE state.

- When a device is taken offline, it is not detached from the storage pool. If you attempt to use the offline device in another pool, even after the original pool is destroyed, you see a message similar to the following:

```
device is part of exported or potentially active ZFS pool. Please see zpool(1M)
```

If you want to use the offline device in another storage pool after destroying the original storage pool, first bring the device online, then destroy the original storage pool.

Another way to use a device from another storage pool, while keeping the original storage pool, is to replace the existing device in the original storage pool with another comparable device. For information about replacing devices, see [“Replacing Devices in a Storage Pool” on page 70](#).

Offline devices are in the OFFLINE state when you query pool status. For information about querying pool status, see [“Querying ZFS Storage Pool Status” on page 80](#).

For more information on device health, see [“Determining the Health Status of ZFS Storage Pools” on page 87](#).

Bringing a Device Online

After a device is taken offline, it can be brought online again by using the `zpool online` command. For example:

```
# zpool online tank c1t0d0
bringing device c1t0d0 online
```

When a device is brought online, any data that has been written to the pool is resynchronized with the newly available device. Note that you cannot use `bring` a device online to replace a disk. If you take a device offline, replace the device, and try to bring it online, it remains in the faulted state.

If you attempt to bring online a faulted device, a message similar to the following is displayed:

```
# zpool online tank c1t0d0
warning: device 'c1t0d0' onlined, but remains in faulted state
use 'zpool replace' to replace devices that are no longer present
```

You might also see the faulted disk message displayed on the console or written to the `/var/adm/messages` file. For example:

```
SUNW-MSG-ID: ZFS-8000-D3, TYPE: Fault, VER: 1, SEVERITY: Major
EVENT-TIME: Wed Sep 21 11:11:27 GMT 2011
PLATFORM: Sun-Fire-X4140, CSN: 0904QAD02C, HOSTNAME: tardis
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: d9e3469f-8d84-4a03-b8a3-d0beb178c017
DESC: A ZFS device failed. Refer to http://sun.com/msg/ZFS-8000-D3
for more information.
AUTO-RESPONSE: No automated response will occur.
IMPACT: Fault tolerance of the pool may be compromised.
REC-ACTION: Run 'zpool status -x' and replace the bad device.
```

For more information about replacing a faulted device, see [“Resolving a Missing Device” on page 262](#).

You can use the `zpool online -e` command to expand a LUN. By default, a LUN that is added to a pool is not expanded to its full size unless the `autoexpand` pool property is enabled. You can expand the LUN automatically by using the `zpool online -e` command even if the LUN is already online or if the LUN is currently offline. For example:

```
# zpool online -e tank c1t13d0
```

Clearing Storage Pool Device Errors

If a device is taken offline due to a failure that causes errors to be listed in the `zpool status` output, you can clear the error counts with the `zpool clear` command.

If specified with no arguments, this command clears all device errors within the pool. For example:

```
# zpool clear tank
```

If one or more devices are specified, this command only clears errors associated with the specified devices. For example:

```
# zpool clear tank c1t0d0
```

For more information about clearing `zpool` errors, see [“Clearing Transient Errors” on page 266](#).

Replacing Devices in a Storage Pool

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

If you are physically replacing a device with another device in the same location in a redundant pool, then you might only need to identify the replaced device. On some hardware, ZFS recognizes that the device is a different disk in the same location. For example, to replace a failed disk (`c1t1d0`) by removing the disk and replacing it in the same location, use the following syntax:

```
# zpool replace tank c1t1d0
```

If you are replacing a device in a storage pool with a disk in a different physical location, you must specify both devices. For example:

```
# zpool replace tank c1t1d0 c1t2d0
```

If you are replacing a disk in the ZFS root pool, see [“How to Replace a Disk in a ZFS Root Pool” on page 110](#).

The following are the basic steps for replacing a disk:

1. Offline the disk, if necessary, with the `zpool offline` command.
2. Remove the disk to be replaced.
3. Insert the replacement disk.
4. Run the `zpool replace` command. For example:

```
# zpool replace tank c1t1d0
```

5. Bring the disk online with the `zpool online` command.
6. Notify FMA that the device is replaced.

```
# fmadm faulty
# fmadm repair fmri
```

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 the first example in this section.

For an example of replacing a SATA disk, see [Example 11-1](#).

Consider the following when replacing devices in a ZFS storage pool:

- If you set the `autoreplace pool` property 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. You are not required to use the `zpool replace` command when this property is enabled. This feature might not be available on all hardware types.
- The storage pool state `REMOVED` is provided when a device or hot spare has been physically removed while the system was running. A hot spare device is substituted for the removed device, if available.

- If a device is removed and then reinserted, the device is placed online. If a hot spare was activated when the device was reinserted, the hot spare is removed when the online operation completes.
- Automatic detection when devices are removed or inserted is hardware-dependent and might not be supported on all platforms. For example, USB devices are automatically configured upon insertion. However, you might have to use the `cfgadm -c configure` command to configure a SATA drive.
- Hot spares are checked periodically to ensure that they are online and available.
- The size of the replacement device must be equal to or larger than the smallest disk in a mirrored or RAID-Z configuration.
- When a replacement device that is larger in size than the device it is replacing is added to a pool, it is not automatically expanded to its full size. The `autoexpand` pool property value determines whether a replacement LUN is expanded to its full size when the disk is added to the pool. By default, the `autoexpand` property is disabled. You can enable this property to expand the LUN size before or after the larger LUN is added to the pool.

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

```
# zpool create pool mirror c1t16d0 c1t17d0
# zpool status
pool: pool
state: ONLINE
scrub: none requested
config:

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

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

- Replacing many disks in a large pool is time-consuming due to resilvering the data onto the new disks. In addition, you might consider running the `zpool scrub` command between disk replacements to ensure that the replacement devices are operational and that the data is written correctly.

- If a failed disk has been replaced automatically with a hot spare, then you might need to detach the spare after the failed disk is replaced. You can use the `zpool detach` command to detach a spare in a mirrored or RAID-Z pool. For information about detaching a hot spare, see “[Activating and Deactivating Hot Spares in Your Storage Pool](#)” on page 74.

For more information about replacing devices, see “[Resolving a Missing Device](#)” on page 262 and “[Replacing or Repairing a Damaged Device](#)” on page 265.

Designating Hot Spares in Your Storage Pool

The hot spares feature enables you to identify disks that could be used to replace a failed or faulted device in a storage pool. Designating a device as a *hot spare* means that the device is not an active device in the pool, but if an active device in the pool fails, the hot spare automatically replaces the failed device.

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

- When the pool is created with the `zpool create` command.
- After the pool is created with the `zpool add` command.

The following example shows how to designate devices as hot spares when the pool is created:

```
# zpool create trinity mirror c1t1d0 c2t1d0 spare c1t2d0 c2t2d0
# zpool status trinity
  pool: trinity
  state: ONLINE
  scrub: none requested
  config:

    NAME        STATE      READ WRITE CKSUM
    trinity     ONLINE    0     0     0
    mirror-0    ONLINE    0     0     0
      c1t1d0    ONLINE    0     0     0
      c2t1d0    ONLINE    0     0     0
    spares
      c1t2d0    AVAIL
      c2t2d0    AVAIL

errors: No known data errors
```

The following example shows how to designate hot spares by adding them to a pool after the pool is created:

```
# zpool add neo spare c5t3d0 c6t3d0
# zpool status neo
  pool: neo
  state: ONLINE
  scrub: none requested
  config:
```

```

NAME          STATE    READ WRITE CKSUM
neo           ONLINE   0     0     0
  mirror-0    ONLINE   0     0     0
    c3t3d0    ONLINE   0     0     0
    c4t3d0    ONLINE   0     0     0
  spares
    c5t3d0    AVAIL
    c6t3d0    AVAIL

```

errors: No known data errors

Hot spares can be removed from a storage pool by using the `zpool remove` command. For example:

```

# zpool remove zeepool c2t3d0
# zpool status zeepool
pool: zeepool
state: ONLINE
scrub: none requested
config:

```

```

NAME          STATE    READ WRITE CKSUM
zeepool       ONLINE   0     0     0
  mirror-0    ONLINE   0     0     0
    c1t1d0    ONLINE   0     0     0
    c2t1d0    ONLINE   0     0     0
  spares
    c1t3d0    AVAIL

```

errors: No known data errors

A hot spare cannot be removed if it is currently used by a storage pool.

Consider the following when using ZFS hot spares:

- Currently, the `zpool remove` command can only be used to remove hot spares, cache devices, and log devices.
- To add a disk as a hot spare, the hot spare must be equal to or larger than the size of the largest disk in the pool. Adding a smaller disk as a spare to a pool is allowed. However, when the smaller spare disk is activated, either automatically or with the `zpool replace` command, the operation fails with an error similar to the following:

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

Activating and Deactivating Hot Spares in Your Storage Pool

Hot spares are activated in the following ways:

- Manual replacement – You replace a failed device in a storage pool with a hot spare by using the `zpool replace` command.

- Automatic replacement – When a fault is detected, an FMA agent examines the pool to determine if it has any available hot spares. If so, it replaces the faulted device with an available spare.

If a hot spare that is currently in use fails, the FMA agent detaches the spare and thereby cancels the replacement. The agent then attempts to replace the device with another hot spare, if one is available. This feature is currently limited by the fact that the ZFS diagnostic engine only generates faults when a device disappears from the system.

If you physically replace a failed device with an active spare, you can reactivate the original device by using the `zpool detach` command to detach the spare. 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.

A faulted device is automatically replaced if a hot spare is available. For example:

```
# zpool status -x
pool: zeepool
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: http://www.sun.com/msg/ZFS-8000-2Q
scrub: resilver completed after 0h0m with 0 errors on Mon Jan 11 10:20:35 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM	
zeepool	DEGRADED	0	0	0	
mirror-0	DEGRADED	0	0	0	
c1t2d0	ONLINE	0	0	0	
spare-1	DEGRADED	0	0	0	
c2t1d0	UNAVAIL	0	0	0	cannot open
c2t3d0	ONLINE	0	0	0	88.5K resilvered
spares					
c2t3d0	INUSE	currently in use			

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

Currently, you can deactivate a hot spare in the following ways:

- By removing the hot spare from the storage pool.
- By detaching a hot spare after a failed disk is physically replaced. See [Example 4-8](#).
- By temporarily or permanently swapping in another hot spare. See [Example 4-9](#).

EXAMPLE 4-8 Detaching a Hot Spare After the Failed Disk Is Replaced

In this example, the failed disk (c2t1d0) is physically replaced and ZFS is notified by using the `zpool replace` command.

```
# zpool replace zeepool c2t1d0
# zpool status zeepool
pool: zeepool
state: ONLINE
```

EXAMPLE 4-8 Detaching a Hot Spare After the Failed Disk Is Replaced (Continued)

```
scrub: resilver completed after 0h0m with 0 errors on Wed Jan 20 10:08:44 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM	
zeepool	ONLINE	0	0	0	
mirror-0	ONLINE	0	0	0	
c1t2d0	ONLINE	0	0	0	
spare-1	ONLINE	0	0	0	
c2t3d0	ONLINE	0	0	0	90K resilvered
c2t1d0	ONLINE	0	0	0	
spares					
c2t3d0	INUSE	currently in use			

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

Then, you can use the `zpool detach` command to return the hot spare back to the spare pool. For example:

```
# zpool detach zeepool c2t3d0
# zpool status zeepool
pool: zeepool
state: ONLINE
scrub: resilver completed with 0 errors on Wed Jan 20 10:08:44 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM
zeepool	ONLINE	0	0	0
mirror	ONLINE	0	0	0
c1t2d0	ONLINE	0	0	0
c2t1d0	ONLINE	0	0	0
spares				
c2t3d0	AVAIL			

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

EXAMPLE 4-9 Detaching a Failed Disk and Using the Hot Spare

If you want to replace a failed disk by temporarily or permanently swapping in the hot spare that is currently replacing it, then detach the original (failed) disk. If the failed disk is eventually replaced, then you can add it back to the storage pool as a spare. For example:

```
# zpool status zeepool
pool: zeepool
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: http://www.sun.com/msg/ZFS-8000-2Q
scrub: resilver in progress for 0h0m, 70.47% done, 0h0m to go
config:
```

NAME	STATE	READ	WRITE	CKSUM
zeepool	DEGRADED	0	0	0

EXAMPLE 4-9 Detaching a Failed Disk and Using the Hot Spare (Continued)

```

mirror-0    DEGRADED    0    0    0
c1t2d0     ONLINE         0    0    0
spare-1    DEGRADED    0    0    0
  c2t1d0   UNAVAIL      0    0    0  cannot open
  c2t3d0   ONLINE       0    0    0  70.5M resilvered
spares
  c2t3d0   INUSE       currently in use

errors: No known data errors
# zpool detach zeepool c2t1d0
# zpool status zeepool
  pool: zeepool
  state: ONLINE
  scrub: resilver completed after 0h0m with 0 errors on Wed Jan 20 13:46:46 2010
config:

NAME        STATE      READ WRITE CKSUM
zeepool     ONLINE    0     0     0
  mirror-0  ONLINE    0     0     0
    c1t2d0  ONLINE    0     0     0
    c2t3d0  ONLINE    0     0     0  70.5M resilvered

errors: No known data errors
(Original failed disk c2t1d0 is physically replaced)
# zpool add zeepool spare c2t1d0
# zpool status zeepool
  pool: zeepool
  state: ONLINE
  scrub: resilver completed after 0h0m with 0 errors on Wed Jan 20 13:48:46 2010
config:

NAME        STATE      READ WRITE CKSUM
zeepool     ONLINE    0     0     0
  mirror-0  ONLINE    0     0     0
    c1t2d0  ONLINE    0     0     0
    c2t3d0  ONLINE    0     0     0  70.5M resilvered
spares
  c2t1d0    AVAIL

errors: No known data errors

```

Managing ZFS Storage Pool Properties

You can use the `zpool get` command to display pool property information. For example:

```

# zpool get all zeepool
NAME    PROPERTY    VALUE                SOURCE
zeepool size      33.8G              -
zeepool capacity 0%                 -
zeepool altroot -                  default
zeepool health  ONLINE            -
zeepool guid    8588873752016230819 default

```

```

zeepool version      31          default
zeepool bootfs      -           default
zeepool delegation  on          default
zeepool autoreplace off         default
zeepool cachefile   -           default
zeepool failmode    wait        default
zeepool listsnapshots off         default
zeepool autoexpand  off         default
zeepool dedupditto  0           default
zeepool dedupratio  1.00x      -
zeepool free        33.7G     -
zeepool allocated   104K       -
zeepool readonly    off         -
    
```

Storage pool properties can be set with the `zpool set` command. For example:

```

# zpool set autoreplace=on zeepool
# zpool get autoreplace zeepool
NAME      PROPERTY  VALUE  SOURCE
zeepool   autoreplace  on      local
    
```

If you attempt to set a pool property on a pool that is 100% full, you will see a message similar to the following:

```

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

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

TABLE 4-1 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 when you are examining 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 <code>off</code> , device replacement must be initiated by using the <code>zpool replace</code> command. If set to <code>on</code> , 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.

TABLE 4-1 ZFS Pool Property Descriptions (Continued)

Property Name	Type	Default Value	Description
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, and if the reference count for a deduped block 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 9, “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.

TABLE 4-1 ZFS Pool Property Descriptions *(Continued)*

Property Name	Type	Default Value	Description
<code>free</code>	String	N/A	Read-only value that identifies the number of blocks within the pool that are not allocated.
<code>guid</code>	String	N/A	Read-only property that identifies the unique identifier for the pool.
<code>health</code>	String	N/A	Read-only property that identifies the current health of the pool, as either ONLINE, DEGRADED, FAULTED, OFFLINE, REMOVED, or UNAVAIL.
<code>listsnapshots</code>	String	<code>off</code>	Controls whether snapshot information that is associated with this pool is displayed with the <code>zfs list</code> command. If this property is disabled, snapshot information can be displayed with the <code>zfs list -t snapshot</code> command.
<code>readonly</code>	Boolean	<code>off</code>	Identifies whether a pool can be modified. This property is only enabled 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.
<code>size</code>	Number	N/A	Read-only property that identifies the total size of the storage pool.
<code>version</code>	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 this property can be used when a specific version is needed for backwards compatibility. This property can be set 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. All three types of storage pool information are covered in this section.

- [“Displaying Information About ZFS Storage Pools” on page 80](#)
- [“Viewing I/O Statistics for ZFS Storage Pools” on page 85](#)
- [“Determining the Health Status of ZFS Storage Pools” on page 87](#)

Displaying Information About ZFS Storage Pools

You can use the `zpool list` command to display basic information about pools.

Displaying Information About All Storage Pools or a Specific Pool

With no arguments, the `zpool list` command displays the following information for all pools on the system:

```
# zpool list
NAME                SIZE    ALLOC    FREE    CAP  HEALTH    ALROOT
tank                80.0G   22.3G   47.7G   28%  ONLINE    -
dozer                1.2T    384G    816G   32%  ONLINE    -
```

This command output displays the following information:

NAME	The name of the pool.
SIZE	The total size of the pool, equal to the sum of the sizes of all top-level virtual devices.
ALLOC	The 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. For more information about determining available file system space, see “ZFS Disk Space Accounting” on page 38 .
FREE	The amount of unallocated space in the pool.
CAP (CAPACITY)	The amount of disk space used, expressed as a percentage of the total disk space.
HEALTH	The current health status of the pool. For more information about pool health, see “Determining the Health Status of ZFS Storage Pools” on page 87 .
ALROOT	The alternate root of the pool, if one exists. For more information about alternate root pools, see “Using ZFS Alternate Root Pools” on page 251 .

You can also gather statistics for a specific pool by specifying the pool name. For example:

```
# zpool list tank
NAME    SIZE    ALLOC    FREE    CAP  HEALTH    ALROOT
tank    80.0G   22.3G   47.7G   28%  ONLINE    -
```

You can use the `zpool list 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 list -T d 3 2
Tue Nov  2 10:36:11 MDT 2010
NAME    SIZE    ALLOC    FREE    CAP  DEDUP  HEALTH    ALROOT
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

You can 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 Lab10Rack5...
```

```
# zpool status -l tank
```

```
pool: tank
state: ONLINE
scan: scrub repaired 0 in 0h0m with 0 errors on Fri May 27 08:24:17 2011
config:
```

NAME	STATE	READ	WRITE	CKSUM
tank	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
mirror-3	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_01/disk	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_21/disk	ONLINE	0	0	0
mirror-4	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_23/disk	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_15/disk	ONLINE	0	0	0
mirror-5	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_09/disk	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_04/disk	ONLINE	0	0	0
mirror-6	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_08/disk	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_05/disk	ONLINE	0	0	0
mirror-7	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_07/disk	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_11/disk	ONLINE	0	0	0
mirror-8	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_06/disk	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_19/disk	ONLINE	0	0	0
mirror-9	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_00/disk	ONLINE	0	0	0
/dev/chassis/Lab10Rack5.../DISK_13/disk	ONLINE	0	0	0
mirror-10	ONLINE	0	0	0

```

/dev/chassis/Lab10Rack5.../DISK_03/disk ONLINE      0      0      0
/dev/chassis/Lab10Rack5.../DISK_18/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 Specific Storage Pool Statistics

Specific statistics can be requested by using the `-o` option. This option provides custom reports or a quick way to list pertinent information. For example, to list only the name and size of each pool, you use the following syntax:

```

# zpool list -o name,size
NAME          SIZE
tank          80.0G
dozer         1.2T

```

The column names correspond to the properties that are listed in [“Displaying Information About All Storage Pools or a Specific Pool”](#) on page 81.

Scripting ZFS Storage Pool Output

The default output for the `zpool list` command is designed for readability and is not easy to use as part of a shell script. To aid programmatic uses of the command, the `-H` option can be used to suppress the column headings and separate fields by tabs, rather than by spaces. For example, to request a list of all pool names on the system, you would use the following syntax:

```

# zpool list -Ho name
tank
dozer

```

Here is another example:

```

# zpool list -H -o name,size
tank  80.0G
dozer 1.2T

```

Displaying ZFS Storage Pool Command History

ZFS automatically logs successful `zfs` and `zpool` commands that modify pool state information. This information can be displayed by using the `zpool history` command.

For example, the following syntax displays the command output for the root pool:

```

# zpool history
History for 'rpool':
2010-05-11.10:18:54 zpool create -f -o failmode=continue -R /a -m legacy -o
cachefile=/tmp/root/etc/zfs/zpool.cache rpool mirror c1t0d0s0 c1t1d0s0

```

```

2010-05-11.10:18:55 zfs set canmount=noauto rpool
2010-05-11.10:18:55 zfs set mountpoint=/rpool rpool
2010-05-11.10:18:56 zfs create -o mountpoint=legacy rpool/ROOT
2010-05-11.10:18:57 zfs create -b 8192 -V 2048m rpool/swap
2010-05-11.10:18:58 zfs create -b 131072 -V 1536m rpool/dump
2010-05-11.10:19:01 zfs create -o canmount=noauto rpool/ROOT/zfsBE
2010-05-11.10:19:02 zpool set bootfs=rpool/ROOT/zfsBE rpool
2010-05-11.10:19:02 zfs set mountpoint=/ rpool/ROOT/zfsBE
2010-05-11.10:19:03 zfs set canmount=on rpool
2010-05-11.10:19:04 zfs create -o mountpoint=/export rpool/export
2010-05-11.10:19:05 zfs create rpool/export/home
2010-05-11.11:11:10 zpool set bootfs=rpool rpool
2010-05-11.11:11:10 zpool set bootfs=rpool/ROOT/zfsBE rpool
    
```

You can use similar output on your system to identify the *actual* ZFS commands that were executed to troubleshoot an error condition.

The features of the history log are as follows:

- The log cannot be disabled.
- The log is saved persistently on disk, which means that the log is saved 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.
- The log requires no administration, which means that tuning the size of the log or changing the location of the log is unnecessary.

To identify the command history of a specific storage pool, use syntax similar to the following:

```

# zpool history tank
History for 'tank':
2011-05-27.13:10:43 zpool create tank mirror c8t1d0 c8t2d0
2011-06-01.12:05:23 zpool scrub tank
2011-06-13.16:26:07 zfs create tank/users
2011-06-13.16:26:27 zfs create tank/users/finance
2011-06-13.16:27:15 zfs set users:dept=finance tank/users/finance
    
```

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 tank
2011-05-27.13:10:43 zpool create tank mirror c8t1d0 c8t2d0 [user root on neo:global]
2011-06-01.12:05:23 zpool scrub tank [user root on neo:global]
2011-06-13.16:26:07 zfs create tank/users [user root on neo:global]
2011-06-13.16:26:27 zfs create tank/users/finance [user root on neo:global]
2011-06-13.16:27:15 zfs set users:dept=finance tank/users/finance [user root ...]
    
```

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

```
# zpool history -i tank
History for 'tank':
2011-05-27.13:10:43 zpool create tank mirror c8t1d0 c8t2d0
2011-05-27.13:10:43 [internal pool create txg:5] pool spa 33; zfs spa 33; zpl 5;...
2011-05-31.15:02:39 [internal pool scrub done txg:11828] complete=1
2011-06-01.12:04:50 [internal pool scrub txg:14353] func=1 mintxg=0 maxtxg=14353
2011-06-01.12:05:23 zpool scrub tank
2011-06-13.16:26:06 [internal create txg:29879] dataset = 52
2011-06-13.16:26:07 zfs create tank/users
2011-06-13.16:26:07 [internal property set txg:29880] $share2=2 dataset = 52
2011-06-13.16:26:26 [internal create txg:29881] dataset = 59
2011-06-13.16:26:27 zfs create tank/users/finance
2011-06-13.16:26:27 [internal property set txg:29882] $share2=2 dataset = 59
2011-06-13.16:26:45 [internal property set txg:29883] users:dept=finance dataset = 59
2011-06-13.16:27:15 zfs set users:dept=finance tank/users/finance
```

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:

<code>alloc capacity</code>	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.
	For more information about the differences between pool space and dataset space, see “ZFS Disk Space Accounting” on page 38 .
<code>free capacity</code>	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.
<code>read operations</code>	The number of read I/O operations sent to the pool or device, including metadata requests.
<code>write operations</code>	The number of write I/O operations sent to the pool or device.
<code>read bandwidth</code>	The bandwidth of all read operations (including metadata), expressed as units per second.
<code>write bandwidth</code>	The bandwidth of all write operations, expressed as units per second.

Listing Pool-Wide I/O Statistics

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

In the above example, the command displays usage statistics for the pool `tank` every two seconds until you type 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. If there is only a single pool, then the statistics are displayed on consecutive lines. If more than one pool exists, then an additional dashed line delineates each iteration to provide visual separation.

Listing Virtual Device I/O Statistics

In addition to pool-wide I/O statistics, the `zpool iostat` command can display I/O statistics for virtual devices. This command can be used to identify abnormally slow devices or to observe the distribution of I/O generated by ZFS. To request the complete virtual device layout as well as all I/O statistics, use the `zpool iostat -v` command. 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
    c1t0d0s0 -    -      0     0     578   109
    c1t1d0s0 -    -      0     0     595   109
-----
tank      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:

- First, disk space usage statistics are only available 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.
- Second, the numbers might not add up exactly as you would expect them to. 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, as 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 the same set of options (interval and count) when examining virtual device statistics.

You can also display physical location information about the pool's virtual devices. For example:

```
# 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
mirror    490G  438G    2    5  8.60K  60.2K
  /dev/chassis/lab10rack15/SCSI_Device__6/disk  -  -  1    0  4.50K  60.2K
  /dev/chassis/lab10rack15/SCSI_Device__7/disk  -  -  1    0  4.49K  60.2K
mirror    490G  438G    2    5  8.47K  60.1K
  /dev/chassis/lab10rack15/SCSI_Device__8/disk  -  -  1    0  4.42K  60.1K
  /dev/chassis/lab10rack15/SCSI_Device__9/disk  -  -  1    0  4.43K  60.1K
.
.
.
```

Determining the Health Status of ZFS Storage Pools

ZFS provides an integrated method of examining pool and device health. The health of a pool is determined from the state of all its devices. This state information is displayed by using the `zpool status` command. In addition, potential pool and device failures are reported by `fmd`, displayed on the system console, and logged in the `/var/adm/messages` file.

This section describes how to determine pool and device health. This chapter does not document how to repair or recover from unhealthy pools. For more information about troubleshooting and data recovery, see [Chapter 11, “Oracle Solaris ZFS Troubleshooting and Pool Recovery.”](#)

Each device can fall into one of the following states:

ONLINE	The device or virtual device is in normal working order. Although some transient errors might still occur, the device is otherwise in working order.
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, as a subsequent fault in another device might be unrecoverable.
FAULTED	The device or virtual device is completely inaccessible. This status typically indicates total failure of the device, such that ZFS is incapable of sending data to it or receiving data from it. If a top-level virtual device is in this state, then the pool is completely inaccessible.
OFFLINE	The device has been explicitly taken offline by the administrator.
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.
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.

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 FAULTED or OFFLINE, then the pool is also FAULTED. A pool in the FAULTED 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 provides details about resilver and scrub operations.

- Resilver in-progress report. For example:

```
scan: resilver in progress since Thu May 26 11:26:32 2011
1.26G scanned out of 2.40G at 6.15M/s, 0h3m to go
1.26G resilvered, 56.3% done
```

- Scrub in-progress report. For example:

```
scan: scrub in progress since Fri May 27 08:24:17 2011
18.0M scanned out of 2.35G at 8.99M/s, 0h4m to go
0 repaired, 0.75% done
```


- Resilver completion message. For example:
scan: resilvered 2.34G in 1h2m with 0 errors on Thu May 26 11:56:40 2011
- Scrub completion message. For example:
scan: scrub repaired 512B in 1h2m with 0 errors on Fri May 27 08:54:50 2011
- Ongoing scrub cancellation message. For example:
scan: scrub canceled on Wed Fri Jun 10 09:06:24 2011
- Scrub and resilver completion messages persist across system reboots

Basic Storage Pool Health Status

You can quickly review pool health status by using the `zpool status` command as follows:

```
# zpool status -x
all pools are healthy
```

Specific pools can be examined by specifying a pool name in the command syntax. Any pool that is not in the `ONLINE` state should be investigated for potential problems, as described in the next section.

Detailed Health Status

You can request a more detailed health summary status by using the `-v` option. For example:

```
# zpool status -v tank
pool: tank
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: http://www.sun.com/msg/ZFS-8000-2Q
scrub: scrub completed after 0h0m with 0 errors on Wed Jan 20 15:13:59 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM	
tank	DEGRADED	0	0	0	
mirror-0	DEGRADED	0	0	0	
c1t0d0	ONLINE	0	0	0	
c1t1d0	UNAVAIL	0	0	0	cannot open

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

This output displays a complete description of why the pool is in its current state, including a readable description of the problem and a link to a knowledge article for more information. Each knowledge article provides up-to-date information about the best way to recover from your current problem. Using the detailed configuration information, you can determine which device is damaged and how to repair the pool.

In the preceding example, the faulted device should be replaced. After the device is replaced, use the `zpool online` command to bring the device online. For example:

```
# zpool online tank c1t0d0
Bringing device c1t0d0 online
# zpool status -x
all pools are healthy
```

If the `autoreplace` property is on, you might not have to online the replaced device.

If a pool has an offline device, the command output identifies the problem pool. For example:

```
# zpool status -x
pool: tank
state: DEGRADED
status: One or more devices has been taken offline by the administrator.
        Sufficient replicas exist for the pool to continue functioning in a
        degraded state.
action: Online the device using 'zpool online' or replace the device with
        'zpool replace'.
scrub: resilver completed after 0h0m with 0 errors on Wed Jan 20 15:15:09 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM	
tank	DEGRADED	0	0	0	
mirror-0	DEGRADED	0	0	0	
c1t0d0	ONLINE	0	0	0	
c1t1d0	OFFLINE	0	0	0	48K resilvered

```
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, and 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 `errors:` field identifies any known data errors.

In the preceding example output, the offline device is not causing data errors.

For more information about diagnosing and repairing faulted pools and data, see [Chapter 11, “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
zpool status -T d 3 2
```

Tue Nov 2 10:38:18 MDT 2010

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

NAME	STATE	READ	WRITE	CKSUM
pool	ONLINE	0	0	0
c3t3d0	ONLINE	0	0	0

errors: No known data errors

pool: rpool
state: ONLINE
scan: resilvered 12.2G in 0h14m with 0 errors on Thu Oct 28 14:55:57 2010
config:

NAME	STATE	READ	WRITE	CKSUM
rpool	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c3t0d0s0	ONLINE	0	0	0
c3t2d0s0	ONLINE	0	0	0

errors: No known data errors

Tue Nov 2 10:38:21 MDT 2010

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

NAME	STATE	READ	WRITE	CKSUM
pool	ONLINE	0	0	0
c3t3d0	ONLINE	0	0	0

errors: No known data errors

pool: rpool
state: ONLINE
scan: resilvered 12.2G in 0h14m with 0 errors on Thu Oct 28 14:55:57 2010
config:

NAME	STATE	READ	WRITE	CKSUM
rpool	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c3t0d0s0	ONLINE	0	0	0
c3t2d0s0	ONLINE	0	0	0

errors: No known data errors

Migrating ZFS Storage Pools

Occasionally, you might need to move a storage pool between systems. To do so, the storage devices must be disconnected from the original system and reconnected to the destination system. This task can be accomplished 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 [“Sending and Receiving ZFS Data” on page 192](#).

- [“Preparing for ZFS Storage Pool Migration” on page 92](#)
- [“Exporting a ZFS Storage Pool” on page 92](#)
- [“Determining Available Storage Pools to Import” on page 93](#)
- [“Importing ZFS Storage Pools From Alternate Directories” on page 95](#)
- [“Importing ZFS Storage Pools” on page 95](#)
- [“Recovering Destroyed ZFS Storage Pools” on page 98](#)

Preparing for ZFS Storage Pool Migration

Storage pools should be explicitly exported to indicate that they are ready to be migrated. This operation flushes any unwritten data to disk, writes data to the disk indicating that the export was done, 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, and the pool will appear faulted on the original system because the devices are no longer present. 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 `zpool export` command. For example:

```
# zpool export tank
```

The command attempts to unmount any mounted file systems within the pool before continuing. If any of the file systems fail to unmount, you can forcefully unmount them by using the `-f` option. For example:

```
# zpool export tank
cannot unmount '/export/home/eric': Device busy
# zpool export -f tank
```

After this command is executed, the pool tank 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 working devices, it appears as “potentially active.”

If ZFS volumes are in use in the pool, the pool cannot be exported, 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 about ZFS volumes, see [“ZFS Volumes” on page 243](#).

Determining Available Storage Pools to Import

After the pool has been removed from the system (either through an explicit export or by forcefully removing the devices), you can attach the devices to the target system. ZFS can handle some situations in which only some of the devices are available, but a successful pool migration depends on the overall health of the devices. In addition, the devices do not necessarily have to be attached under the same device name. ZFS detects any moved or renamed devices, and adjusts the configuration appropriately. To discover available pools, run the `zpool import` command with no options. For example:

```
# zpool import
pool: tank
  id: 11809215114195894163
  state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

    tank          ONLINE
    mirror-0     ONLINE
    c1t0d0       ONLINE
    c1t1d0       ONLINE
```

In this example, the pool tank is available to be imported on the target system. Each pool is identified by a name as well as a unique numeric identifier. If multiple pools with the same name are available to import, you can use the numeric identifier to distinguish between them.

Similar to the `zpool status` command output, the `zpool import` output includes a link to a knowledge article with the most up-to-date information regarding repair procedures for the problem that is preventing a pool from being imported. In this case, the user can force the pool to be imported. However, importing a pool that is currently in use by another system over a storage network can result in data corruption and panics as both systems attempt to write to the

same storage. If some devices in the pool are not available but sufficient redundant data exists to provide a usable pool, the pool appears in the DEGRADED state. For example:

```
# zpool import
pool: tank
  id: 11809215114195894163
  state: DEGRADED
status: One or more devices are missing from the system.
action: The pool can be imported despite missing or damaged devices. The
        fault tolerance of the pool may be compromised if imported.
  see: http://www.sun.com/msg/ZFS-8000-2Q
config:
```

NAME	STATE	READ	WRITE	CKSUM	
tank	DEGRADED	0	0	0	
mirror-0	DEGRADED	0	0	0	
c1t0d0	UNAVAIL	0	0	0	cannot open
c1t3d0	ONLINE	0	0	0	

In this example, the first disk is damaged or missing, though you can still import the pool because the mirrored data is still accessible. If too many faulted or missing devices are present, the pool cannot be imported. For example:

```
# zpool import
pool: dozer
  id: 9784486589352144634
  state: FAULTED
action: The pool cannot be imported. Attach the missing
        devices and try again.
  see: http://www.sun.com/msg/ZFS-8000-6X
config:
  raidz1-0      FAULTED
  c1t0d0        ONLINE
  c1t1d0        FAULTED
  c1t2d0        ONLINE
  c1t3d0        FAULTED
```

In this example, two disks are missing from a RAID-Z virtual device, which means that sufficient redundant data is not available to reconstruct the pool. In some cases, not enough devices are present to determine the complete configuration. In this case, ZFS cannot determine what other devices were part of the pool, though ZFS does report as much information as possible about the situation. For example:

```
# zpool import
pool: dozer
  id: 9784486589352144634
  state: FAULTED
status: One or more devices are missing from the system.
action: The pool cannot be imported. Attach the missing
        devices and try again.
  see: http://www.sun.com/msg/ZFS-8000-6X
config:
  dozer          FAULTED  missing device
  raidz1-0      ONLINE
```

```

c1t0d0    ONLINE
c1t1d0    ONLINE
c1t2d0    ONLINE
c1t3d0    ONLINE

```

Additional devices are known to be part of this pool, though their exact configuration cannot be determined.

Importing ZFS Storage Pools From Alternate Directories

By default, the `zpool import` command only searches devices 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 dozer mirror /file/a /file/b
# zpool export dozer
# zpool import -d /file
pool: dozer
  id: 7318163511366751416
  state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

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

```

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

Importing ZFS Storage Pools

After a pool has been identified for import, you can import it by specifying the name of the pool or its numeric identifier as an argument to the `zpool import` command. For example:

```
# zpool import tank
```

If multiple available pools have the same name, you must specify which pool to import by using the numeric identifier. For example:

```

# zpool import
pool: dozer
  id: 2704475622193776801
  state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

```

```
dozer      ONLINE
  c1t9d0   ONLINE

pool: dozer
  id: 6223921996155991199
  state: ONLINE
action: The pool can be imported using its name or numeric identifier.
config:

dozer      ONLINE
  c1t8d0   ONLINE
# zpool import dozer
cannot import 'dozer': more than one matching pool
import by numeric ID instead
# zpool import 6223921996155991199
```

If the pool name conflicts with an existing pool name, you can import the pool under a different name. For example:

```
# zpool import dozer zeepool
```

This command imports the exported pool `dozer` using the new name `zeepool`. The new pool name is persistent.

If the pool was not cleanly exported, ZFS requires the `-f` flag to prevent users from accidentally importing a pool that is still in use on another system. For example:

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

Note – 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 hosts.

Pools can also be imported under an alternate root by using the `-R` option. For more information on alternate root pools, see [“Using ZFS Alternate Root Pools” on page 251](#).

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. For example:

```
# zpool import dozer
The devices below are missing, use '-m' to import the pool anyway:
  c3t3d0 [log]

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


Import the pool with the missing log device. For example:

```
# 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: http://www.sun.com/msg/ZFS-8000-2Q
scan: scrub repaired 0 in 0h0m with 0 errors on Fri Oct 15 16:43:03 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				
14685044587769991702	UNAVAIL	0	0	0 was c3t3d0

After attaching the missing log device, run the `zpool clear` command to clear the pool errors.

A similar recovery can be attempted with missing mirrored log devices. For example:

```
# 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

```
# 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: http://www.sun.com/msg/ZFS-8000-2Q
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

After attaching the missing log devices, run the `zpool clear` command to clear the pool errors.

Importing a Pool in Read-Only Mode

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

```
# zpool import -o readonly=on tank
# zpool scrub tank
cannot scrub tank: 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. This also means that any pending synchronous writes in the intent log are not played until the pool is imported read-write.
- Attempts to set a pool property during the read-only import are ignored.

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

```
# zpool export tank
# zpool import tank
# zpool scrub tank
```

Importing a Pool By a Specific Device Path

The following command imports the pool `dpool` by identifying one of the pool's specific devices, `/dev/dsk/c2t3d0`, in this example.

```
# zpool import -d /dev/dsk/c2t3d0s0 dpool
# zpool status dpool
pool: dpool
state: ONLINE
scan: resilvered 952K in 0h0m with 0 errors on Thu Mar 10 10:28:46 2011
config:
```

NAME	STATE	READ	WRITE	CKSUM
dpool	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c2t3d0	ONLINE	0	0	0
c2t1d0	ONLINE	0	0	0

Even though this pool is comprised of whole disks, the command must include the specific device's slice identifier.

Recovering Destroyed ZFS Storage Pools

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

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

      tank          ONLINE
      mirror-0     ONLINE
        c1t0d0     ONLINE
        c1t1d0     ONLINE
```

In this `zpool import` output, you can identify the `tank` pool as the destroyed pool because of the following state information:

```
state: ONLINE (DESTROYED)
```

To recover the destroyed pool, run the `zpool import -D` command again with the pool to be recovered. For example:

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

      NAME          STATE      READ WRITE CKSUM
      tank          ONLINE
        mirror-0   ONLINE
          c1t0d0   ONLINE
          c1t1d0   ONLINE
```

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

If one of the devices in the destroyed pool is faulted or unavailable, you might be able to recover the destroyed pool anyway 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 destroy dozer
# zpool import -D
  pool: dozer
    id: 13643595538644303788
   state: DEGRADED (DESTROYED)
  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: http://www.sun.com/msg/ZFS-8000-2Q
 config:

      NAME          STATE      READ WRITE CKSUM
      dozer         DEGRADED    0     0     0
        raidz2-0   DEGRADED    0     0     0
          c2t8d0   ONLINE      0     0     0
```

```

c2t9d0 ONLINE      0    0    0
c2t10d0 ONLINE     0    0    0
c2t11d0 UNAVAIL    0   35    1 cannot open
c2t12d0 ONLINE      0    0    0

```

errors: No known data errors

```
# zpool import -Df dozer
```

```
# zpool status -x
```

```
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: http://www.sun.com/msg/ZFS-8000-2Q
```

```
scrub: scrub completed after 0h0m with 0 errors on Thu Jan 21 15:38:48 2010
```

```
config:
```

NAME	STATE	READ	WRITE	CKSUM	
dozer	DEGRADED	0	0	0	
raidz2-0	DEGRADED	0	0	0	
c2t8d0	ONLINE	0	0	0	
c2t9d0	ONLINE	0	0	0	
c2t10d0	ONLINE	0	0	0	
c2t11d0	UNAVAIL	0	37	0	cannot open
c2t12d0	ONLINE	0	0	0	

errors: No known data errors

```
# zpool online dozer c2t11d0
```

```
Bringing device c2t11d0 online
```

```
# zpool status -x
```

```
all pools are healthy
```

Upgrading ZFS Storage Pools

If you have ZFS storage pools from a previous Solaris release, you can upgrade your pools with the `zpool upgrade` command to take advantage of the pool features in the current release. In addition, the `zpool status` command notifies you when your pools are running older versions. For example:

```
# zpool status
```

```
pool: tank
```

```
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
tank	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

```

You can use the following syntax to identify additional information about a particular version and supported releases:

```

# zpool upgrade -v
This system is currently running ZFS pool version 33.

```

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
 6  bootfs pool property
 7  Separate intent log devices
 8  Delegated administration
 9  refquota and refreservation properties
10  Cache devices
11  Improved scrub performance
12  Snapshot properties
13  snapused property
14  passthrough-x aclinherit
15  user/group space accounting
16  stmf property support
17  Triple-parity RAID-Z
18  Snapshot user holds
19  Log device removal
20  Compression using zle (zero-length encoding)
21  Deduplication
22  Received properties
23  Slim ZIL
24  System attributes
25  Improved scrub stats
26  Improved snapshot deletion performance
27  Improved snapshot creation performance
28  Multiple vdev replacements
29  RAID-Z/mirror hybrid allocator
30  Encryption
31  Improved 'zfs list' performance
32  One MB blocksize
33  Improved share support

```

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

Then, you can run the `zpool upgrade` command to upgrade all of your pools. For example:

```

# zpool upgrade -a

```

Note – If you upgrade your pool to a later ZFS version, the pool will not be accessible on a system that runs an older ZFS version.

Managing ZFS Root Pool Components

This chapter describes how to manage your Oracle Solaris ZFS root pool components, such as attaching a root pool mirror, cloning a ZFS boot environment, and resizing swap and dump devices.

The following sections are provided in this chapter:

- “Managing ZFS Root Pool Components (Overview)” on page 103
- “ZFS Root Pool Requirements” on page 104
- “Managing Your ZFS Root Pool” on page 106
- “Managing Your ZFS Swap and Dump Devices” on page 113
- “Booting From a ZFS Root File System” on page 116

For information about root pool recovery, see [Chapter 12, “Archiving Snapshots and Root Pool Recovery.”](#)

For any late-breaking issues, see the Oracle Solaris 11 release notes.

Managing ZFS Root Pool Components (Overview)

ZFS is the default root file system in the Oracle Solaris 11 release. Review the following considerations when installing the Oracle Solaris release.

- **Installation** – In the Oracle Solaris 11 release, you can install and boot from a ZFS root file system in the following ways:
 - Live CD (x86 only) – Installs a ZFS root pool on a single disk. You can use the `fdisk` partition menu during the installation to partition the disk for your environment.
 - Text installation (SPARC and x86) – Installs a ZFS root pool on a single disk from media or over the network. You can use the `fdisk` partition menu during the installation to partition the disk for your environment.

- Automated Installer (AI) (SPARC and x86) – Automatically installs a ZFS root pool. You can use an AI manifest to determine the disk and the disk partitions to be used for the ZFS root pool.
- **Swap and dump devices** – Automatically created on ZFS volumes in the ZFS root pool by all of the above installation methods. For more information about managing ZFS swap and dump devices, see “[Managing Your ZFS Swap and Dump Devices](#)” on page 113.
- **Mirrored root pool configuration** – You can configure a mirrored root pool during an automatic installation. For more information about configuring a mirrored root pool after an installation, see “[How to Configure a Mirrored Root Pool](#)” on page 109.
- **Root pool space management** – After the system is installed, consider setting 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, which allows 1 GB of remaining file system space.

ZFS Root Pool Requirements

Review the following sections that describe ZFS root pool space and configuration requirements.

ZFS Root Pool Space Requirements

When a system is installed, the size of the swap volume and the dump volume are dependent upon 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 boot environments (BEs) to be created.

Review the following ZFS storage pool space requirements:

- For a description of memory requirements for the different installation methods, see [Oracle Solaris 11 Release Notes](#).
- At least 7-13 GB of disk space is recommended. The space is consumed as follows:
 - **Swap area and dump device** – The default sizes of the swap and dump volumes that are created by the Solaris installation programs vary based on the amount of memory on the system and other variables. Consider that the swap device size is generally 1/4 of physical memory and the dump device size is approximately half the size of physical memory.

You can adjust the sizes of your swap and dump volumes to sizes of your choosing as long as the new sizes support system operation, during or after installation. For more information, see “[Adjusting the Sizes of Your ZFS Swap and Dump Devices](#)” on page 114.

- **Boot environment (BE)** – A ZFS BE is approximately 4-6 GB. Each ZFS BE that is cloned from another ZFS BE doesn't need additional disk space. Consider that BE size will increase when the BE is updated, depending on the updates. All ZFS BEs in the same root pool use the same swap and dump devices.
- **Oracle Solaris OS Components** – All subdirectories of the root file system that are part of the OS image, with the exception of `/var`, must be in the root file system. In addition, all Solaris OS components must reside in the root pool with the exception of the swap and dump devices.

ZFS Root Pool Configuration Requirements

Review the following ZFS storage pool configuration requirements:

- The disk that is intended for the root pool must have an SMI (VTOC) label.
- The disk that is intended for the root pool must be less than 2 TBs in size so that the Solaris OS can boot successfully.
- The pool must exist either on a disk slice or on disk slices that are mirrored. If you attempt to use an unsupported pool configuration during a `beadm` operation, you will see a message similar to the following:

```
ERROR: ZFS pool name does not support boot environments
```

For a detailed description of supported ZFS root pool configurations, see [“Creating a ZFS Root Pool” on page 49](#).

- On an x86 based system, the disk must contain a Solaris `fdisk` partition. A Solaris `fdisk` partition is created automatically when the x86 based system is installed. For more information about Solaris `fdisk` partitions, see [“Guidelines for Creating an fdisk Partition” in Oracle Solaris Administration: Devices and File Systems](#).
- Pool properties or file system properties can be set on a root pool during an automatic installation. 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.

Troubleshooting ZFS Root Pool Installation Problems

Review the following issues if you attempt to install the Oracle Solaris 11 release.

- **Root pool disk is too small** - If the disk slice that is intended for the root pool is too small, you might need to exit the installer, expand the size of the disk slice by using the `format` utility, and restart the installation. For example, during an automatic installation, you might see a message similar to the following on the system console:

```
15:43:54   Space required for installation: 5.00gb
15:43:54   Total available space: 4.55gb
15:43:54   Error occurred during execution of 'target-selection' checkpoint.
```

```
15:43:54 Failed Checkpoints:
15:43:54
15:43:54 target-selection
15:43:54
15:43:54 Checkpoint execution error:
15:43:54
15:43:54 Error determining swap/dump requirements.
15:43:54
15:43:54 Automated Installation Failed. See install log at
/system/volatile/install_log
```

- For information about expanding the root pool disk slice, see [“How to Create a Disk Slice for a ZFS Root File System” in *Oracle Solaris Administration: Devices and File Systems*](#) or [“How to Create a Disk Slice for a ZFS Root File System” in *Oracle Solaris Administration: Devices and File Systems*](#).

- After the root pool disk slice is expanded, restart the auto installer:

```
# svcadm clear auto-installer
```

Press return. You can observe the installation by using this command:

```
# tail -f /system/volatile/install_log
```

- **Installation hangs** - If the installation hangs and the root pool disk is small, such as 16 GB, and the system memory is large, such as 32 GB, then the disk is too small to create a swap volume and a dump volume. On a large memory system, the root pool disk must be large enough to contain the BE, the swap volume, and a dump device that is 1/2 to 3/4 the size of physical memory.

Managing Your ZFS Root Pool

The following sections provide information about installing and updating a ZFS root pool and configuring a mirrored root pool.

Installing a ZFS Root Pool

The Oracle Solaris 11 Live CD installation method installs a default ZFS root pool on a single disk. With the Oracle Solaris 11 automated installation (AI) method, you can create an AI manifest to identify the disk or mirrored disks for the ZFS root pool.

The AI installer provides the flexibility of installing a ZFS root pool on the default boot disk or on a target disk that you identify. You can specify the logical device, such as `c1t0d0s0`, or the physical device path. In addition, you can use the MPxIO identifier or the device ID for the device to be installed.

After the installation, review your ZFS storage pool and file system information, which can vary by installation type and customizations. For example:

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

    NAME            STATE      READ WRITE CKSUM
    rpool            ONLINE    0     0     0
    c1t3d0s0        ONLINE    0     0     0

errors: No known data errors
# zfs list
# zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rpool                                6.49G 60.4G   40K    /rpool
rpool/ROOT                           3.46G 60.4G   31K    legacy
rpool/ROOT/solaris                    3.46G 60.4G   3.16G  /
rpool/ROOT/solaris/var                 303M 60.4G   216M   /var
rpool/dump                             2.00G 60.5G   1.94G  -
rpool/export                          96.5K 60.4G   32K    /rpool/export
rpool/export/home                      64.5K 60.4G   32K    /rpool/export/home
rpool/export/home/admin                 32.5K 60.4G   32.5K  /rpool/export/home/admin
rpool/swap                              1.03G 60.5G   1.00G  -
```

Review your ZFS BE information. For example:

```
# beadm list
# beadm list
BE      Active Mountpoint Space Policy Created
--      -
solaris NR    /                3.85G static 2011-09-26 08:37
```

In the above output, the Active field indicates whether the BE is active now represented by N and active on reboot represented by R, or both represented by NR.

▼ How to Update Your ZFS Boot Environment

The default ZFS boot environment (BE) is named `solaris` by default. You can identify your BEs by using the `beadm list` command. For example:

```
# beadm list
BE      Active Mountpoint Space Policy Created
--      -
solaris NR    /                8.41G static 2011-01-13 15:31
```

In the above output, NR means the BE is active now and will be the active BE on reboot.

You can use the `pkg update` command to update your ZFS boot environment. If you update your ZFS BE by using the `pkg update` command, a new BE is created and activated automatically, unless the updates to the existing BE are very minimal.

1 Update your 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.

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

```
# init 6
```

```
.
.
.
```

```
# beadm list
```

```
BE      Active Mountpoint Space Policy Created
--      -
solaris - - 6.25M static 2011-09-26 08:37
solaris-1 NR / 3.92G static 2011-09-26 09:32
```

3 If an error occurs when booting the new BE, activate and boot back to the previous BE.

```
# beadm activate solaris
```

```
# init 6
```

▼ How to Mount an Alternate BE

You might need to copy or access a file from another BE for recovery purposes.

1 Become an administrator.

2 Mount the alternate BE.

```
# beadm mount solaris-1 /mnt
```

3 Access the BE.

```
# ls /mnt
```

```
bin      export  media   pkg      rpool    tmp
boot     home    mine    platform sbin     usr
dev      import  mnt     proc     scde     var
devices  java    net     project  shared
doe      kernel  nfs4    re       src
etc      lib     opt     root     system
```

4 Unmount the alternate BE when you're finished with it.

```
# beadm umount solaris-1
```

▼ How to Configure a Mirrored Root Pool

If you do not configure a mirrored root pool during an automatic installation, you can easily configure a mirrored root pool after the installation.

For information about replacing a disk in a root pool, see [“How to Replace a Disk in a ZFS Root Pool” on page 110](#).

1 Display your current root pool status.

```
# 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 Prepare a second disk for attachment to the root pool, if necessary.

- SPARC: Confirm that the disk has an SMI (VTOC) disk label and a slice 0. If you need to relabel the disk and create a slice 0, see [“Creating a Disk Slice for a ZFS Root File System” in *Oracle Solaris Administration: Devices and File Systems*](#).
- 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 [“Creating a Disk Slice for a ZFS Root File System” in *Oracle Solaris Administration: Devices and File Systems*](#).

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

```
# zpool attach rpool c2t0d0s0 c2t1d0s0
```

Make sure to wait until resilver is done before rebooting.

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

```
# zpool status rpool
pool: rpool
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 Thu Sep 29 18:09:09 2011
      1.55G scanned out of 5.36G at 36.9M/s, 0h1m to go
      1.55G scanned out of 5.36G at 36.9M/s, 0h1m to go
      1.55G resilvered, 28.91% done
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 (resilvering)
```

errors: No known data errors

In the above output, the resilvering process is not complete. Resilvering is complete when you see messages similar to the following:

```
resilvered 5.36G in 0h10m with 0 errors on Thu Sep 29 18:19:09 2011
```

- 5 **Verify that you can boot successfully from the new disk.**
- 6 **Set up the system to boot automatically from the new disk.**
 - SPARC: Set up the system to boot automatically from the new disk, either by using the `eprom` command or the `setenv` command from the boot PROM.
 - x86: Reconfigure the system BIOS.

▼ How to Replace a Disk 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 so that the system won't boot, you'll need to boot from an alternate media, such as a CD or the network, before you replace the root pool disk.

In a mirrored root pool configuration, you might be able to attempt a disk replacement without having to boot from alternate media. You can replace a failed disk by using the `zpool replace` command or if you have an additional disk, you can use the `zpool attach` command. See the steps below for an example of attaching an additional disk and detaching a root pool disk.

Systems with SATA disks require that you offline and unconfigure a disk before attempting the `zpool replace` operation to replace a failed disk. For example:

```
# zpool offline rpool c1t0d0s0
# cfgadm -c unconfigure c1::dsk/c1t0d0
<Physically remove failed disk c1t0d0>
<Physically insert replacement disk c1t0d0>
# cfgadm -c configure c1::dsk/c1t0d0
<Confirm that the new disk has an SMI label and a slice 0>
# zpool replace rpool c1t0d0s0
# zpool online rpool c1t0d0s0
# zpool status rpool
<Let disk resilver before installing the boot blocks>
SPARC# installboot -F zfs /usr/platform/`uname -i`/lib/fs/zfs/bootblk /dev/rdisk/c1t0d0s0
x86# installgrub /boot/grub/stage1 /boot/grub/stage2 /dev/rdisk/c1t0d0s0
```

On some hardware, you do not have to online or reconfigure the replacement disk after it is inserted.

1 Physically connect the replacement disk.**2 Confirm that the replacement (new) disk has an SMI (VTOC) label and a slice 0.**

For information about relabeling a disk that is intended for the root pool, see “How to Label a Disk” in *Oracle Solaris Administration: Devices and File Systems*.

3 Attach the new disk to the root pool.

For example:

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

4 Confirm the root pool status.

For example:

```
# zpool status rpool
pool: rpool
state: ONLINE
scan: resilvered 5.36G in 0h2m with 0 errors on Thu Sep 29 18:11:53 2011
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
```

5 Verify that you can boot from the new disk after resilvering is complete.

For example, on a SPARC based system:

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

Identify the boot device pathnames of the current and new disk so that you can test booting from the replacement disk and also manually boot from the existing disk, if necessary, if the replacement disk fails. In the example below, the current root pool disk (c2t0d0s0) is:

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

In the example below, the replacement boot disk is (c2t1d0s0):

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

6 If the system boots from the new disk, detach the old disk.

For example:

```
# zpool detach rpool c2t0d0s0
```

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

- SPARC: Set up the system to boot automatically from the new disk, either by using the `eprom` command or the `setenv` command from the boot PROM.

- x86: Reconfigure the system BIOS.

▼ How to Create a BE in Another Root Pool

If you want to re-create your existing BE in another root pool, follow the steps below. You can modify the steps based on whether you want two root pools with similar BEs that have independent swap and dump devices or whether you just want a BE in another root pool that shares the swap and dump devices.

After you activate and boot from the new BE in the second root pool, it will have no information about the previous BE in the first root pool. If you want to boot back to the original BE, you will need to boot the system manually from the original root pool's boot disk.

- 1 **Create a second root pool with an SMI (VTOC)-labeled disk. For example:**

```
# zpool create rpool2 c4t2d0s0
```

- 2 **Create the new BE in the second root pool. For example:**

```
# beadm create -p rpool2 solaris2
```

- 3 **Set the bootfs property on the second root pool. For example:**

```
# zpool set bootfs=rpool2/ROOT/solaris2 rpool2
```

- 4 **Activate the new BE. For example:**

```
# beadm activate solaris2
```

- 5 **Boot from the new BE but you must boot specifically from the second root pool's boot device.**

```
ok boot disk2
```

Your system should be running under the new BE.

- 6 **Re-create the swap volume. For example:**

```
# zfs create -V 4g rpool2/swap
```

- 7 **Update the /etc/vfstab entry for the new swap device. For example:**

```
/dev/zvol/dsk/rpool2/swap - - swap - no -
```

- 8 **Re-create the dump volume. For example:**

```
# zfs create -V 4g rpool2/dump
```

- 9 **Reset the dump device. For example:**

```
# dumpadm -d /dev/zvol/dsk/rpool2/dump
```

- 10 **Reset your default boot device to boot from the second root pool's boot disk.**

- SPARC – Set up the system to boot automatically from the new disk, either by using the `eprom` command or the `setenv` command from the boot PROM.
- x86 – Reconfigure the system BIOS.

11 Reboot to clear the original root pool's swap and dump devices.

```
# init 6
```

Managing Your ZFS Swap and Dump Devices

During the installation process, a swap area is created on a ZFS volume in the ZFS root pool. For example:

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

During the installation process, a dump device is created on a ZFS volume in the ZFS root pool. In general, a dump device requires no administration because it is set up automatically at installation time. 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
```

If you disable and remove the dump device, then you will need to enable it with the `dumpadm` command after it is recreated. In most cases, you will only have to adjust the size of the dump device by using the `zfs` command.

For information about the swap and dump volume sizes that are created by the installation programs, see [“ZFS Root Pool Requirements” on page 104](#).

Both the swap volume size and the dump volume size can be adjusted after installation. For more information, see [“Adjusting the Sizes of Your ZFS Swap and Dump Devices” on page 114](#).

Consider the following issues when working with ZFS swap and dump devices:

- Separate ZFS volumes must be used for the swap area and dump devices.
- Currently, using a swap file on a ZFS file system is not supported.
- 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 Solaris releases. For more information, see [Chapter 19, “Configuring Additional Swap Space \(Tasks\),” in *Oracle Solaris Administration: Devices and File Systems*](#) and [Chapter 17, “Managing System Crash Information \(Tasks\),” in *Oracle Solaris Administration: Common Tasks*](#).

Adjusting the Sizes of Your ZFS Swap and Dump Devices

You might need to adjust the size of swap and dump devices after installation or possibly, recreate the swap and dump volumes.

- Adjust the size of your swap and dump volumes.
- You can reset the `volsize` property of the dump device after a system is installed. For example:

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

- You can resize the swap volume, or just recreate it if the system is not busy. For example:

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

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 *Oracle Solaris Administration: Devices and File Systems*.

- If you need more swap space on a system that is already installed and the swap device is busy, just add another swap volume. For example:

```
# zfs create -V 2G rpool/swap2
```

- Activate the new swap volume. For example:

```
# swap -a /dev/zvol/dsk/rpool/swap2
# swap -l
swapfile          dev  swaplo  blocks  free
/dev/zvol/dsk/rpool/swap  256,1    16 1058800 1058800
/dev/zvol/dsk/rpool/swap2 256,3    16 4194288 4194288
```

- Add an entry for the second swap volume to the `/etc/vfstab` file. For example:

```
/dev/zvol/dsk/rpool/swap    -          -          swap    -    no    -
```

- Select one of the following if you need to recreate your swap area:
 - On a SPARC based system, create your swap area. Set the block size to 8 KB.

```
# zfs create -V 2G -b 8k rpool/swap
```

- On an x86 based system, create your swap area. Set the block size to 4 KB.

```
# zfs create -V 2G -b 4k rpool/swap
```

- You must enable the swap area when a new swap device is added or changed.
- Add an entry for the swap volume to the `/etc/vfstab` file.

Troubleshooting ZFS Dump Device Issues

Review the following items if you have problems either capturing a system crash dump or resizing the dump device.

- If a crash dump was not created automatically, you can use the `savecore` command to save the crash dump.
- A dump device is created automatically when you initially install a ZFS root file system or migrate to a ZFS root file system. In most cases, you will only need to adjust the size of the dump device if the default dump device size is too small. For example, on a large-memory system, the dump device size is increased to 40 GB as follows:

```
# zfs set volsize=40G rpool/dump
```

Resizing a large dump device can be a time-consuming process.

If, for any reason, you need to enable a dump device after you create a dump device manually, use syntax similar to the following:

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

- A system with 128 GB or greater memory will need a larger dump device than the dump device that is created by default. If the dump device is too small to capture an existing crash dump, a message similar to the following is displayed:

```
# dumpadm -d /dev/zvol/dsk/rpool/dump
dumpadm: dump device /dev/zvol/dsk/rpool/dump is too small to hold a system dump
dump size 36255432704 bytes, device size 34359738368 bytes
```

For information on sizing the swap and dump devices, see “Planning for Swap Space” in *Oracle Solaris Administration: Devices and File Systems*.

- You cannot currently add a dump device to a pool with multiple top level-devices. You will see a message similar to the following:

```
# dumpadm -d /dev/zvol/dsk/datapool/dump
dump is not supported on device '/dev/zvol/dsk/datapool/dump':
'datapool' has multiple top level vdevs
```

Add the dump device to the root pool, which cannot have multiple top-level devices.

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. When booting from a ZFS root file system, the path names of both the boot archive and the kernel file are resolved in the root file system that is selected for booting.

Booting from a ZFS file system differs from booting from a UFS file system because with ZFS, a device specifier identifies a storage pool, not a single root file system. A storage pool can contain multiple bootable ZFS root file systems. When booting from ZFS, you must specify a boot device and a root file system within the pool that was identified by the boot device.

By default, the file system selected for booting is the one identified by the pool's `boot fs` property. This default selection can be overridden by specifying an alternate bootable file system that is included in the `boot -Z` command on a SPARC system or by selecting an alternate boot device from the BIOS on an x86 based system.

Booting From an Alternate Disk in a Mirrored ZFS Root Pool

You can attach a disk to create a mirrored ZFS root pool after installation. For more information about creating a mirrored root pool, see [“How to Configure a Mirrored Root Pool” on page 109](#).

Review the following known issues regarding mirrored ZFS root pools:

- If you replace a root pool disk by using the `zpool replace` command, you must install the boot information on the newly replaced disk by using the `installboot` or `installgrub` commands. If you create a mirrored ZFS root pool with the initial installation method or if you use the `zpool attach` command to attach a disk to the root pool, then this step is unnecessary. The `installboot` or `installgrub` command is as follows:

```
sparc# installboot -F zfs /usr/platform/'uname -i'/lib/fs/zfs/bootblk
/dev/rdisk/c0t1d0s0
```

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

- You can boot from different devices in a mirrored ZFS root pool. Depending on the hardware configuration, you might need to update the PROM or the BIOS to specify a different boot device.

For example, you can boot from either disk (`c1t0d0s0` or `c1t1d0s0`) in this pool.

```
# 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
c1t0d0s0	ONLINE	0	0	0
c1t1d0s0	ONLINE	0	0	0

On a SPARC based system, enter the alternate disk at the ok prompt.

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

After the system is rebooted, confirm the active boot device. For example:

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

On an x86 based system, use syntax similar to the following:

```
x86# 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'
```

- On an x86 based system, select an alternate disk in the mirrored ZFS root pool from the appropriate BIOS menu.

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

On a SPARC based system with multiple ZFS BEs, you can boot from any BE by using the `beadm activate` command.

During an installation and `beadm` activation process, the ZFS root file system is automatically designated with the `bootfs` property.

Multiple bootable file systems can exist within a pool. By default, the bootable file system entry in the `/pool-name/boot/menu.lst` file is identified by the pool's `bootfs` property. However, a `menu.lst` entry can contain a `bootfs` command, which specifies an alternate file system in the pool. In this way, the `menu.lst` file can contain entries for multiple root file systems within the pool.

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

```
title Oracle Solaris 11 solaris SPARC
bootfs rpool/ROOT/solaris
```

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

On a SPARC based system, two boot options are available:

- After a ZFS BE is activated, you can use the `boot -L` command to display a list of bootable file systems within a ZFS pool. Then, you can select one of the bootable file systems in the list. Detailed instructions for booting that file system are displayed. You can boot the selected file system by following the instructions.
- Use the `boot -Z file system` command to boot a specific ZFS file system.

EXAMPLE 5-1 Booting From a Specific ZFS Boot Environment

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

For example, the following ZFS BEs are available as described by the `beadm` output:

```
# beadm list
BE      Active Mountpoint Space Policy Created
--      -
solaris -      -      19.18M static 2011-01-13 15:31
solaris-1 NR    /      8.48G  static 2011-01-13 15:44
```

If you have multiple ZFS BEs on your SPARC based system, you can use the `boot -L` command. For example:

```
ok boot -L
Boot device: /pci@0/pci@0/pci@2/scsi@0/disk@3,0:a File and args: -L
1 solaris
2 solaris-1
Select environment to boot: [ 1 - 2 ]: 2
```

```
To boot the selected entry, invoke:
boot [<root-device>] -Z rpool/ROOT/solaris-1
```

```
Program terminated
ok boot -Z rpool/ROOT/solaris-1
```

Keep in mind that the BE that is booted with the above command is not activated for the next reboot. If you want to continue to boot automatically from the BE that is selected during the `boot -Z` operation, you will need to activate it.

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

The following entries are added to the `/pool-name/boot/grub/menu.lst` file during the installation process or `beadm activate` operation to boot ZFS automatically:

```
title solaris
bootfs rpool/ROOT/solaris
kernel$ /platform/i86pc/kernel/amd64/unix -B $ZFS-BOOTFS
module$ /platform/i86pc/amd64/boot_archive
title solaris-1
```

```
bootfs rpool/ROOT/solaris-1
kernel$ /platform/i86pc/kernel/amd64/unix -B $ZFS-BOOTFS
module$ /platform/i86pc/amd64/boot_archive
```

If the device identified by GRUB as the boot device contains a ZFS storage pool, the menu.lst file is used to create the GRUB menu.

On an x86 based system with multiple ZFS BEs, you can select a BE from the GRUB menu. If the root file system corresponding to this menu entry is a ZFS file system, the following option is added.

```
-B $ZFS-BOOTFS
```

EXAMPLE 5-2 x86: Booting a ZFS File System

When booting from a ZFS file system, the root device is specified by the boot -B \$ZFS-BOOTFS parameter. For example:

```
title solaris
bootfs rpool/ROOT/solaris
kernel$ /platform/i86pc/kernel/amd64/unix -B $ZFS-BOOTFS
module$ /platform/i86pc/amd64/boot_archive
title solaris-1
bootfs rpool/ROOT/solaris-1
kernel$ /platform/i86pc/kernel/amd64/unix -B $ZFS-BOOTFS
module$ /platform/i86pc/amd64/boot_archive
```

EXAMPLE 5-3 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. The ability to fast reboot a system drastically reduces down time and improves efficiency.

You must still use the `init 6` command when transitioning between BEs with the `beadm activate` command. For other system operations where the `reboot` command is appropriate, you can use the `reboot -f` command. For example:

```
# reboot -f
```

Booting For Recovery Purposes in a ZFS Root Environment

Use the following procedure if you need to boot the system so that you can recover from a lost root password or similar problem.

▼ How to Boot the System For Recovery Purposes

Use the procedure below to resolve a problem with a menu.lst problem or a root password problem. If you need to replace a disk in root pool, see [“How to Replace a Disk in a ZFS Root Pool” on page 110](#). If you need to perform complete system (bare metal) recovery, see [Chapter 12, “Archiving Snapshots and Root Pool Recovery.”](#)

1 Select the appropriate boot method:

- x86: Live Media – Boot from the installation media and use a GNOME terminal for the recovery procedure.
- SPARC: Text installation – Boot from the install media or from the network, and select option 3 Shell from the text installation screen.
- x86: Text installation – From the GRUB menu, select the Text Installer and command line boot entry, then select the option 3 Shell from the text installation screen.
- SPARC: Automated installation – Use the following command to boot directly from an installation menu that allows you to exit to a shell.


```
ok boot net:dhcp
```
- x86: Automated installation – Booting from an install server on the network requires a PXE boot. Select the Text Installer and command line entry from the GRUB menu. Then, select the option 3 Shell from the text installation screen.

For example, after the system is booted, select option 3 Shell.

```
1 Install Oracle Solaris
2 Install Additional Drivers
3 Shell
4 Terminal type (currently xterm)
5 Reboot
```

```
Please enter a number [1]: 3
To return to the main menu, exit the shell
#
```

2 Select the boot recovery problem:

- Resolve a bad root shell by booting the system to single-user mode and correcting the shell entry in the /etc/passwd file.

On an x86 system, edit the selected boot entry and add the -s option.

For example, on a SPARC system, shut down the system and boot to single-mode. After you log in as root, edit the /etc/passwd file, and fix the root shell entry.

```
# init 0
ok boot -s
```

```
Boot device: /pci@780/pci@0/pci@9/scsi@0/disk@0,0:a File and args: -s
SunOS Release 5.11 Version 11.0 64-bit
Copyright (c) 1983, 2011, Oracle and/or its affiliates. All rights
```



```

reserved.
Booting to milestone "milestone/single-user:default".
Hostname: tardis.central
Requesting System Maintenance Mode
SINGLE USER MODE

Enter user name for system maintenance (control-d to bypass): root
Enter root password (control-d to bypass): xxxx
single-user privilege assigned to root on /dev/console.
Entering System Maintenance Mode

Jan 24 13:23:54 su: 'su root' succeeded for root on /dev/console
Oracle Corporation SunOS 5.11 11.0 November 2011
su: No shell /usr/bin/mybash. Trying fallback shell /sbin/sh.
root@tardis.central:~# TERM =vt100; export TERM
root@tardis.central:~# vi /etc/passwd
root@tardis.central:~# <Press control-d>
logout
svc.startd: Returning to milestone all.

```

- Resolve a problem with a menu.lst boot entry.

First, you must boot from media or the network by using one of the boot methods listed in step 1. Then, import the root pool and fix the menu.lst entry.

```

x86# zpool import -f rpool
x86# cd /rpool/boot/grub
x86# vi menu.lst
x86# exit
    1 Install Oracle Solaris
    2 Install Additional Drivers
    3 Shell
    4 Terminal type (currently sun-color)
    5 Reboot

```

Please enter a number [1]: 5

Confirm that the system boots successfully.

- Resolve an unknown root password that prevents you from logging into the system.

First, you must boot from media or the network by using one of the boot methods listed in step 1. Then, import the root pool (rpool) and mount the BE to remove the root password entry. This process is identical on SPARC and x86 platforms.

```

# zpool import -f rpool
# beadm list
be_find_current_be: failed to find current BE name
be_find_current_be: failed to find current BE name
BE          Active Mountpoint Space  Policy Created
--          -
solaris     -          -          11.45M static 2011-10-22 00:30
solaris-2   R          -          12.69G static 2011-10-21 21:04
# mkdir /a
# beadm mount solaris-2 /a
# TERM=vt100
# export TERM
# cd /a/etc
# vi shadow

```

```
<Carefully remove the unknown password>
# cd /
# beadm umount solaris-2
# halt
```

Go to the next step to set the root password.

3 Set the root password by booting to single-user mode and setting the password.

This step assumes that you have removed an unknown root password in the previous step.

On an x86 system, edit the selected boot entry and add the `-s` option.

On a SPARC system, boot the system to single-user mode, log in as root, and set the root password. For example:

```
ok boot -s
```

```
Boot device: /pci@780/pci@0/pci@9/scsi@0/disk@0,0:a File and args: -s
SunOS Release 5.11 Version 11.0 64-bit
Copyright (c) 1983, 2011, Oracle and/or its affiliates. All rights
reserved.
Booting to milestone "milestone/single-user:default".
Hostname: tardis.central
Requesting System Maintenance Mode
SINGLE USER MODE
```

```
Enter user name for system maintenance (control-d to bypass): root
Enter root password (control-d to bypass): <Press return>
single-user privilege assigned to root on /dev/console.
Entering System Maintenance Mode
```

```
Jan 24 13:23:54 su: 'su root' succeeded for root on /dev/console
Oracle Corporation SunOS 5.11 11.0 November 2011
root@tardis.central:~# passwd -r files root
New Password: xxxxxx
Re-enter new Password: xxxxxx
passwd: password successfully changed for root
root@tardis.central:~# <Press control-d>
logout
svc.startd: Returning to milestone all.
```

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.

The following sections are provided in this chapter:

- “Managing ZFS File Systems (Overview)” on page 123
- “Creating, Destroying, and Renaming ZFS File Systems” on page 124
- “Introducing ZFS Properties” on page 127
- “Querying ZFS File System Information” on page 146
- “Managing ZFS Properties” on page 149
- “Mounting ZFS File Systems” on page 154
- “Sharing and Unsharing ZFS File Systems” on page 158
- “Setting ZFS Quotas and Reservations” on page 167
- “Encrypting ZFS File Systems” on page 173
- “Migrating ZFS File Systems” on page 178
- “Upgrading ZFS File Systems” on page 181

Managing ZFS File Systems (Overview)

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

ZFS file systems are administered 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, volumes, and clones 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 7, “Working With Oracle Solaris ZFS Snapshots and Clones.”](#) For detailed information about ZFS volumes, see [“ZFS Volumes” on page 243.](#)

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

Creating, Destroying, and Renaming ZFS File Systems

ZFS file systems can be created and destroyed by using the `zfs create` and `zfs destroy` commands. ZFS file systems can be renamed by using the `zfs rename` command.

- “Creating a ZFS File System” on page 124
- “Destroying a ZFS File System” on page 125
- “Renaming a ZFS File System” on page 126

Creating a ZFS File System

ZFS file systems are created by using the `zfs create` command. The `create` subcommand takes a single argument: the name of the file system to be created. The file system name is specified as a path name starting from the name of the pool as follows:

pool-name/[filesystem-name/]filesystem-name

The pool name and initial file system names in the path identify the location in the hierarchy where the new file system will be created. The last name in the path identifies the name of the file system to be created. The file system name must satisfy the naming requirements in “ZFS Component Naming Requirements” on page 28.

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 173.

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

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

ZFS automatically mounts the newly created file system if it is created successfully. By default, file systems are mounted as *dataset*, using the path provided for the file system name in 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 154.

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

You can set file system properties when the 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 127.

Destroying a ZFS File System

To destroy a ZFS file system, use the `zfs destroy` command. 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 155.

In the following example, the `tank/home/mark` file system is destroyed:

```
# zfs destroy tank/home/mark
```



Caution – No confirmation prompt appears with the `destroy` subcommand. Use it with extreme caution.

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 tank/home/matt
cannot unmount 'tank/home/matt': Device busy
```

```
# zfs destroy -f tank/home/matt
```

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. Note that a recursive destroy also destroys snapshots, so use this option with caution.

```
# 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
```

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
```



Caution – No confirmation prompt appears with the `-f`, `-r`, or `-R` options to the `zfs destroy` command, so use these options carefully.

For more information about snapshots and clones, see [Chapter 7, “Working With Oracle Solaris ZFS Snapshots and Clones.”](#)

Renaming 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 following example uses the `rename` subcommand to rename of a file system from `eric` to `eric_old`:

```
# zfs rename tank/home/eric tank/home/eric_old
```

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 167.

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.

For information about renaming snapshots, see [“Renaming ZFS Snapshots”](#) on page 186.

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 138
- “Settable ZFS Native Properties” on page 139
- “ZFS User Properties” on page 145

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 145.

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 descendants.

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 149.
<code>inherited from <i>dataset-name</i></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 145.

TABLE 6-1 ZFS Native Property Descriptions

Property Name	Type	Default Value	Description
<code>aclinherit</code>	String	<code>secure</code>	Controls how ACL entries are inherited when files and directories are created. The values are <code>discard</code> , <code>noallow</code> , <code>secure</code> , and <code>passthrough</code> . For a description of these values, see “ACL Properties” on page 209.

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
<code>aclmode</code>	String	groupmask	Controls how an ACL entry is modified during a <code>chmod</code> operation. The values are <code>discard</code> , <code>groupmask</code> , and <code>passthrough</code> . For a description of these values, see “ACL Properties” on page 209 .
<code>atime</code>	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.
<code>available</code>	Number	N/A	<p>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.</p> <p>The property abbreviation is <code>avail</code>.</p> <p>For more information about disk space accounting, see “ZFS Disk Space Accounting” on page 38.</p>
<code>canmount</code>	Boolean	on	<p>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 <code>off</code>, a mount point can be inherited to descendent file systems, but the file system itself is never mounted.</p> <p>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.</p> <p>For more information, see “The <code>canmount</code> Property” on page 141.</p>

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
<code>casesensitivity</code>	String	<code>mixed</code>	<p>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.</p> <p>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 <code>casesensitivity</code> Property” on page 141.</p> <p>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.</p>
<code>checksum</code>	String	<code>on</code>	<p>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.</p>
<code>compression</code>	String	<code>off</code>	<p>Enables or disables compression for a dataset. The values are <code>on</code>, <code>off</code>, <code>lzjb</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.</p> <p>The property abbreviation is <code>compress</code>.</p>
<code>compressratio</code>	Number	N/A	<p>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.</p> <p>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.</p>

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
<code>copies</code>	Number	1	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 used 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 143 .
<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.
<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>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</code> , <code>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.

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
logbias	String	latency	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> .
mlslabel	String	None	Provides a sensitivity label that determines if a file system can be mounted in a Trusted Extensions 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 when Trusted Extensions is enabled and only with the appropriate privilege.
mounted	Boolean	N/A	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> .
mountpoint	String	N/A	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 154 .
primarycache	String	all	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.

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
nbmand	Boolean	off	Controls whether the file system should be mounted with nbmand (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.
normalization	String	None	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 none, and the utf8only property was left unspecified, the utf8only property is automatically set to on. The default value of the normalization property is none. This property cannot be changed after the file system is created.
origin	String	N/A	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 -r or -f option) as long as a clone exists. Non-cloned file systems have an origin of none.
quota	Number (or none)	none	Limits the amount of disk space a file system and its descendants can consume. This property enforces a hard limit on the amount of disk space used, including all space consumed by descendants, 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 quota, but rather imposes an additional limit. Quotas cannot be set on volumes, as the volsize property acts as an implicit quota. For information about setting quotas, see “Setting Quotas on ZFS File Systems” on page 168 .
rekeydate	String	N/A	Read-only property that indicates the date of the last data encryption key change from a zfs key -K or zfs clone -K operation on this file system. If no rekey operation has been performed, the value of this property is the same as the creation date.
readonly	Boolean	off	Controls whether a dataset can be modified. When set to on, no modifications can be made. The property abbreviation is rdonly.

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
recordsize	Number	128K	<p>Specifies a suggested block size for files in a file system.</p> <p>The property abbreviation is <code>recs size</code>. For a detailed description, see “The recordsize Property” on page 144.</p>
referenced	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>
refquota	Number (or none)	none	<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 descendents, such as snapshots and clones.</p>
refreservation	Number (or none)	none	<p>Sets the minimum amount of disk space that is guaranteed to a dataset, not including descendents, 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>ref reserv</code>.</p>

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
reservation	Number (or none)	none	<p>Sets the minimum amount of disk space guaranteed to a file system and its descendents. 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> <p>For more information, see “Setting Reservations on ZFS File Systems” on page 171.</p>
rstchown	Boolean	on	<p>Indicates whether the file system owner can grant file ownership changes. The default is to restrict <code>chown</code> operations. When <code>rstchown</code> is set to <code>off</code>, the user has the <code>PRIV_FILE_CHOWN_SELF</code> privilege for <code>chown</code> operations.</p>
secondarycache	String	all	<p>Controls what is cached in the secondary cache (L2ARC). 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.</p>
setuid	Boolean	on	<p>Controls whether the <code>setuid</code> bit is honored in a file system.</p>
shadow	String	None	<p>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 <i>URI</i>. The file system to be migrated must be read-only for a complete migration.</p>

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
sharenfs	String	off	<p>Controls whether a ZFS file system is published as an NFS share. You can also publish and unpublish an NFS share of a ZFS file system by using the <code>zfs share</code> and <code>zfs unshare</code> commands. Both methods of publishing an NFS share require that the NFS share properties are already set. For information about setting NFS share properties, see the <code>zfs set share</code> command.</p> <p>When the <code>sharenfs</code> property is changed, the file system share and any children inheriting the property are re-published with any new options that have been set with the <code>zfs set share</code> command only if the property was previously off, or if the shares were published before the property was changed. If the new property value is off, the file system shares are unpublished.</p> <p>For more information about sharing ZFS file systems, see “Sharing and Unsharing ZFS File Systems” on page 158.</p>
sharesmb	String	off	<p>Controls whether a ZFS file system is published as an SMB share. You can also publish and unpublish an SMB share of a ZFS file system by using the <code>zfs share</code> and <code>zfs unshare</code> commands. Both methods of publishing an SMB share require that the SMB share properties are also set. For information about setting SMB share properties, see the <code>zfs set share</code> command.</p>
snapdir	String	hidden	<p>Controls whether the <code>.zfs</code> 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 183.</p>

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
sync	String	standard	<p>Determines the synchronous behavior of a file system's transactions. Possible values are:</p> <ul style="list-style-type: none"> ■ <code>standard</code>, the default value, which means synchronous file system transactions, such as <code>fsync</code>, <code>O_DSYNC</code>, <code>O_SYNC</code>, and so on, are written to the intent log. ■ <code>always</code>, ensures that <i>every</i> file system transaction is written and flushed to stable storage by a returning system call. This value has a significant performance penalty. ■ <code>disabled</code>, 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. <p>Caution – This <code>disabled</code> 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 <code>/var</code> 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.</p>
type	String	N/A	Read-only property that identifies the dataset type as <code>filesystem</code> (file system or clone), <code>volume</code> , or <code>snapshot</code> .
used	Number	N/A	<p>Read-only property that identifies the amount of disk space consumed by a dataset and all its descendents.</p> <p>For a detailed description, see “The used Property” on page 139.</p>
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 <code>usedchild</code> .

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
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 reservation reservations. The property abbreviation is usedds.
usedbyreservationNumber		off	Read-only property that identifies the amount of disk space that is used by a reservation set on a dataset, which would be freed if the reservation was removed. The property abbreviation is usedreserv.
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.
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.
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 <code>off</code> , the <code>normalization</code> property must either not be explicitly set or be set to <code>none</code> . The default value for the <code>utf8only</code> property is <code>off</code> . This property cannot be changed after the file system is created.
volsize	Number	N/A	For volumes, specifies the logical size of the volume. For a detailed description, see “ The volsize Property ” on page 145.
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 <code>volblock</code> .

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
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 <code>off</code> .
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 a Solaris System With Zones Installed” on page 245.
xattr	Boolean	on	Indicates whether extended attributes are enabled (<code>on</code>) or disabled (<code>off</code>) for this file system.

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 6-1](#).

The read-only native properties are listed here and described in [Table 6-1](#).

- `available`
- `compressratio`
- `creation`
- `keystatus`
- `mounted`
- `origin`
- `referenced`
- `rekeydate`
- `type`
- `used`

For detailed information, see [“The used Property”](#) on page 139.

- `usedbychildren`

- `usedbydataset`
- `usedbyreservation`
- `usedbysnapshots`

For more information about disk space accounting, including the used, referenced, and available properties, see [“ZFS Disk Space Accounting” on page 38](#).

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 descendants. 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. For more information about snapshots and space issues, see [“Out of Space Behavior” on page 39](#).

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`, `usedbyreservation`, 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 descendants. For more information, see [Table 6-1](#).

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 149](#) or by using the `zfs create` command as described in [“Creating a ZFS File System” on page 124](#). 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 167](#).

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 6-1](#). If not specifically mentioned, a property applies to all dataset types: file systems, volumes, clones, and snapshots.

The settable properties are listed here and described in [Table 6-1](#).

- `aclinherit`
For a detailed description, see [“ACL Properties”](#) on page 209.
- `atime`
- `canmount`
- `casesensitivity`
- `checksum`
- `compression`
- `copies`
- `devices`
- `dedup`
- `encryption`
- `exec`
- `keysource`
- `logbias`
- `mlslabel`
- `mountpoint`
- `nbmand`
- `normalization`
- `primarycache`
- `quota`
- `readonly`
- `recordsize`
For a detailed description, see [“The recordsize Property”](#) on page 144.
- `refquota`
- `refreservation`
- `reservation`
- `rstchown`
- `secondarycache`
- `sharesmb`
- `sharenfs`
- `setuid`
- `snapdir`

- version
- vscan
- utf8only
- volsize

For a detailed description, see [“The volsize Property” on page 145](#).

- volblocksize
- zoned
- xattr

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`, `f0o`, `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 232](#).

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. 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	270G

dedup = 1.20, compress = 1.28, copies = 1.03, dedup * compress / 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:

```
in-core DDT size = 2.63M x 320 = 841.60M
```
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  ALTROOT
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 173](#).

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 128 KB. Changing the file system's `recordsize` value only affects files created afterward. Existing files are unaffected.

The property abbreviation is `recsize`.

The sharesmb Property

This property enables sharing of ZFS file systems with the Oracle Solaris SMB service, and identifies options to be used.

Because SMB shares requires a resource name, 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, which would be illegal in the resource name, are replaced with underscore (_) characters. A pseudo property *name* is also supported that allows you to replace the file system name with a specific name. The specific name is then used to replace the prefix file system in the case of inheritance.

For example, if the file system, `data/home/john`, is set to `name=john`, then `data/home/john` has a resource name of `john`. If a child file system of `data/home/john/backups` exists, it has a resource name of `john_backups`. When the `sharesmb` property is changed for a file system, the file system and any children inheriting the property are re-shared with the new options, only if the property was previously set to `off`, or if they were shared before the property was changed. If the new property is set to `off`, the file systems are unshared.

For examples of using the `sharesmb` property, see [“Sharing and Unsharing ZFS File Systems” on page 158](#).

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.

Though not recommended, you can create a sparse volume by specifying the `-s` flag to `zfs create -V` or by changing the reservation after the volume has been created. A *sparse volume* is a volume whose reservation is not equal to the volume size. For a sparse volume, changes to `volsize` are not reflected in the reservation.

For more information about using volumes, see [“ZFS Volumes” on page 243](#).

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: `! , ' + ; : ; _ .`

- 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 127](#).

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 [`zfs\(1M\)`](#).

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 127. 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 `sharenfs` and `mountpoint` property values.

```
# zfs list -r -o name,sharenfs,mountpoint users/home
NAME                                SHARENFS  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 6-2 Types of ZFS Datasets

Type	Description
filesystem	File systems and clones
volume	Volumes
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
```

Managing ZFS Properties

Dataset properties are managed through the `zfs` command's `set`, `inherit`, and `get` subcommands.

- “Setting ZFS Properties” on page 149
- “Inheriting ZFS Properties” on page 150
- “Querying ZFS Properties” on page 151

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 139.

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): `BKMGTP EZ`. 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 [zfs\(1M\)](#).

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 [zfs\(1M\)](#).

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 147. 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 6-3 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 157.
-(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     type          filesystem     -
```

```

tank/home creation      Fri Apr 22 10:42 2011 -
tank/home used         611K -
tank/home available    66.9G -
tank/home referenced   33K -
tank/home compressratio 1.00x -
tank/home mounted      yes -
tank/home quota        none default
tank/home reservation  none default
tank/home recordsize   128K default
tank/home mountpoint   /tank/home default
tank/home sharenfs     off default
tank/home checksum     on default
tank/home compression off default
tank/home atime        on default
tank/home devices     on default
tank/home exec         on default
tank/home setuid       on default
tank/home readonly    off default
tank/home zoned        off default
tank/home snapdir      hidden default
tank/home aclinherit   restricted default
tank/home canmount     on default
tank/home xattr        on default
tank/home copies       1 default
tank/home version      5 -
tank/home utf8only     off -
tank/home normalization none -
tank/home casesensitivity sensitive -
tank/home vscan        off default
tank/home nbmand       off default
tank/home sharesmb     off default
tank/home refquota     none default
tank/home refreservation none default
tank/home primarycache all default
tank/home secondarycache all default
tank/home usedbysnapshots 0 -
tank/home usedbydataset 33K -
tank/home usedbychildren 578K -
tank/home usedbyrefreservation 0 -
tank/home logbias      latency default
tank/home dedup        off default
tank/home mlslabel     none default
tank/home sync         standard default
tank/home encryption   off -
tank/home keysource    none default
tank/home keystatus    none -
tank/home rekeydate    - default
tank/home rstchown     on default
tank/home shadow       none -

```

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 [zfs\(1M\)](#).

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 parseable 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 127 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 descendants. 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 descendants, 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 154](#)
- [“Mounting ZFS File Systems” on page 156](#)
- [“Using Temporary Mount Properties” on page 157](#)
- [“Unmounting ZFS File Systems” on page 158](#)

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 48](#).

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 141](#).

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 155](#).

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. For more information about ZFS data integrity, see [“Transactional Semantics” on page 24](#).

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(1M) 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 6-4 ZFS Mount-Related Properties and Mount Options

Property	Mount Option
atime	atime/noatime

TABLE 6-4 ZFS Mount-Related Properties and Mount Options (Continued)

Property	Mount Option
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 [zfs\(1M\)](#).

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 unmount` command, see [zfs\(1M\)](#).

Sharing and Unsharing ZFS File Systems

In this Solaris release, you create a ZFS file system share and publish the share as follows:

- Create the file system share and define the NFS or SMB share properties by using the `zfs share` command.

Using a separate command to create a share provides the following features:

- Defines the options that are used to share a particular path in a file system.
- Multiple shares can be defined per file system, but a share name is used to uniquely identify each share.
- A share can define options for both NFS and SMB sharing.
- Multiple SMB paths can be defined for a single directory path.
- The share is stored in a file in the `.zfs/share` directory with the name of the share.

The interaction between the defined shares and the `sharenfs` and `sharesmb` properties is as follows:

- The existing `sharenfs` property controls whether the file system is published over NFS. The value is on or off. This property is inheritable by descendent file systems.
- The existing `sharesmb` property controls whether the file system is published over SMB. The value is on or off. This property is inheritable by descendent file systems.
- When the `sharenfs` or `sharesmb` property is set to on, all defined shares for the file system (and all descendent file systems inheriting the property) are published for the appropriate protocol. All defined shares are also published when the `zfs share` command is issued.

1. If no shares are defined, the file system is not shared.
2. If shares are defined for the file system, then only those shares are published. The mount point of the file system is only shared if a share exists that explicitly shares it.

- When the `sharenfs` or `sharesmb` property is set to off, all published shares for the file system (and all descendent file systems inheriting the property) are unpublished for the appropriate protocol. These shares remain unshared until the `sharenfs` or `sharesmb` property is set to on.

None of the defined shares are removed when the property is set to off and are re-shared the next time the `sharenfs` or `sharesmb` property is set to on.

- When the `zfs unshare` command is issued, all published shares for the file system are unpublished. These shares remain unshared until the `zfs share` command is issued for the file system.

None of the defined shares are removed when the `zfs unshare` command is issued, and are re-shared the next time the `zfs share` command is issued.

This section details some of the differences between the new sharing syntax and legacy sharing syntax.

The primary new sharing differences are as follows:

- The `zfs set share` command replaces the `sharemgr` interface for sharing ZFS file systems.
- The `sharemgr` interface is no longer available. The legacy `share` command and the `sharenfs` property are still available. See the examples below.

- The `/etc/dfs/dfstab` file still exists but modifications are ignored. SMF manages ZFS or UFS share information so that file systems are shared automatically when the system is rebooted, similar to the way ZFS mount and share information is managed.
- The `share -a` command is like the `share -ap` command so that the file system share is persistent.
- Descendent file systems do not inherit share properties. If a descendent file system is created with an inherited `sharenfs` property set to `on`, then a share is created for the new descendent file system.

Legacy ZFS Sharing Syntax

Legacy sharing syntax is still supported.

1. Use the `share` command to share a file system.

For example, to share a ZFS file system:

```
# share -F nfs /tank/zfsfs
# cat /etc/dfs/sharetab
/tank/zfsfs      -      nfs      rw
```

The above syntax is identical to sharing a UFS file system:

```
# share -F nfs /ufsfs
# cat /etc/dfs/sharetab
/ufsfs -      nfs      rw
/tank/zfsfs -      nfs      rw
```

2. The `zfs share` command cannot be used to initially share a file system until the `sharenfs` property is set.

```
# zfs share rpool/data
cannot share 'rpool/data': legacy share
use share(1M) to share this filesystem, or
set the 'share' property and set [sharenfs|sharesmb] property on
# zfs set sharenfs=on rpool/data
# cat /etc/dfs/sharetab
/rpool/data     -      nfs      rw
```

All methods publish the file system shares immediately.

New ZFS Sharing Syntax

The new `zfs set share` command is used to share a ZFS file system over the NFS or SMB protocols. The share is not published until the `sharenfs` set property is also set on the file system.

Use the `zfs set share` command to create an NFS or SMB share of ZFS file system and also set the `sharenfs` property.


```
# zfs create rpool/fs1
# zfs set share=name=fs1,path=/rpool/fs1,prot=nfs rpool/fs1
name=fs1,path=/rpool/fs1,prot=nfs
```

The share is not published until the `sharenfs` or `sharesmb` property is set to on. For example:

```
# zfs set sharenfs=on rpool/fs1
# cat /etc/dfs/sharetab
/rpool/fs1 fs1 nfs sec=sys,rw
```

A public NFS share can be created as follows:

```
# zfs set share=name=pp,path=/pub,prot=nfs,sec=sys,rw=*,public rpool/public
name=pp,path=/pub,prot=nfs,public=true,sec=sys,rw=*
# zfs set sharenfs=on rpool/public
# cat /etc/dfs/sharetab
/pub pp nfs public,sec=sys,rw
```

You can also create a share of a newly created ZFS file system by using syntax similar to the following:

```
# zfs create -o mountpoint=/ds -o sharenfs=on rpool/ds
```

When you create a NFS share of a ZFS file system, you must provide the following share components:

<code>share=name</code>	Identify a name for your share. Maximum share name is 80 characters.
<code>path=pathname</code>	Identify a path for your NFS share that must exist within the file system or directory to be shared.
<code>prot=nfs</code> or <code>smb</code>	Identify the protocol as NFS or SMB.
<code>pool/filesystem</code>	Identifies the ZFS file system to be shared.

Additional share options are:

<code>description=string</code>	Provides text that might help identify the share. Spaces or commas in the description must be enclosed in quote marks ("").
<code>rw=</code> or <code>ro=</code>	Identifies whether the share is available as read/write or read-only to all clients. You can also specify a colon-separated list that includes hostnames, IP addresses, or a netgroup.
<code>root=</code>	Identifies a root user from a specified host or list of hosts have root access. By default, no host has root access.
<code>sec=</code>	Identifies a NFS server security mode, such as <code>sys</code> , <code>dh</code> , <code>krb5</code> , and so on. For supported security mode information, see nfssec(5) .

The following NFS properties must be specified after `prot=nfs`, but before any `sec=` properties.

- `anon=user-name|uid`

- nosub=true|false
- nosuid=true|false
- aclok=true|false
- public=true|false
- index=*filename*
- log=*TYPE_LOGTAG*
- cksum=*TYPE_STRINGSET*

The following optional SMB properties must be specified after the `prot=smb` property:

- ad-container=*string*
- abe=[true|false]
- csc=[disabled|manual|auto|vdo]
- catia=[true|false]
- guestok=[true|false]
- ro=*access-list*
- rw=*access-list*
- none=*access-list*

See [share_nfs\(1M\)](#) and [share_smb\(1M\)](#) for a detailed description of NFS and SMB share properties.

Displaying ZFS Share Information

As in the previous releases, you display the value of the `sharenfs` property by using `zfs get sharenfs` property or by using the `zfs get all` command syntax.

```
# zfs get sharenfs rpool/fs1
NAME          PROPERTY  VALUE   SOURCE
rpool/fs1    sharenfs  on      local
```

The new share information is available by using the `zfs get share` command.

```
# zfs get share rpool/fs1
NAME          PROPERTY  VALUE   SOURCE
rpool/fs1    share     name=rpool_fs1,path=/rpool/fs1,prot=nfs  local
```

The new share information is not available in the `zfs get all` command syntax.

If you create a share of a newly created ZFS file system, use the `zfs get share` command to identify the *share-name* name or the *share-path* name. For example:

```
# zfs create -o mountpoint=/data -o sharenfs=on rpool/data
# zfs get share rpool/data
NAME          PROPERTY  VALUE   SOURCE
rpool/data    share     name=data,path=/data,prot=nfs  local
```

ZFS Sharing Inheritance

Inheritance of the `zfs share` property and the `sharenfs` or the `sharesmb` property works as follows:

- The `zfs share` property is not inherited from a parent to a descendent file system. In addition, the `zfs set share` command does not support the `-r` option to set a ZFS property on descendent file systems.
- If the `sharenfs` or the `sharesmb` property is set on a parent file system, the `sharenfs` or the `sharesmb` property is also set on the descendent file systems. For example:

```
# zfs create -o mountpoint=/ds rpool/ds
# zfs set share=name=ds,path=/ds,prot=nfs rpool/ds
name=ds,path=/ds,prot=nfs
# zfs set sharenfs=on rpool/ds
# cat /etc/dfs/sharetab
/ds rpool_ds nfs sec=sys,rw
# zfs create rpool/ds/ds1
# zfs get sharenfs rpool/ds/ds1
NAME          PROPERTY VALUE      SOURCE
rpool/ds/ds1  sharenfs  on         inherited from rpool/ds
```

Any existing child file system also inherits the parent's `sharenfs` or `sharesmb` property value.

If the `sharenfs` or the `sharesmb` property is set to off on the parent file system, the `sharenfs` property or the `sharesmb` property is set to off on the descendent file systems. For example:

```
# zfs set sharenfs=off rpool/ds
$ zfs get -r sharenfs rpool/ds
NAME          PROPERTY VALUE      SOURCE
rpool/ds      sharenfs  off        local
rpool/ds/ds1  sharenfs  off        inherited from rpool/ds
rpool/ds/ds2  sharenfs  off        inherited from rpool/ds
rpool/ds/ds3  sharenfs  off        inherited from rpool/ds
```

Changing a ZFS Share

The name and protocol properties must be specified when you change share property values.

For example, create an NFS share like this:

```
# zfs create -o mountpoint=/ds -o sharenfs=on rpool/ds
# zfs set share=name=ds,path=/ds,prot=nfs rpool/ds
name=ds,path=/ds,prot=nfs
```

Then, add the SMB protocol:

```
# zfs set share=name=ds,prot=nfs,prot=smb rpool/ds
name=ds,path=/ds,prot=nfs,prot=smb
```

Remove the SMB protocol:

```
# zfs set -c share=name=ds,prot=smb rpool/ds
name=ds,path=/ds,prot=nfs
```

Removing a ZFS Share

You can remove an existing share by using the `zfs set -c` command. For example, identify the share name.

```
# zfs get share
NAME          PROPERTY  VALUE  SOURCE
rpool/ds     share     name=ds,path=/ds,prot=nfs  local
```

Then, remove the share by identifying the *share-name* name. For example:

```
# zfs set -c share=name=ds rpool/ds
share 'ds' was removed.
```

If a share is established by creating a default share, when the file system is created, then a share can be removed by the *share-name* name or the *share-path* name. For example, this share is given a default *share-name* name, `data`, and a default *share-path* name, `/data`.

```
# zfs create -o mountpoint=/data -o sharenfs=on rpool/data
# zfs get share rpool/data
NAME          PROPERTY  VALUE  SOURCE
rpool/data    share     name=data,path=/data,prot=nfs  local
```

Remove the share by identifying the *share-name* name. For example:

```
# zfs set -c share=name=data rpool/data
share 'data' was removed.
```

Remove the share by identifying the *share-path* name. For example:

```
# zfs set -c share=path=/data rpool/data
share 'data' was removed.
```

ZFS File Sharing Within a Non-Global Zone

In previous Solaris releases, you could not create and publish NFS or SMB shares in a Oracle Solaris non-global zone. In this Solaris release, you can create and publish NFS shares by using the `zfs set share` command and the `legacy share` command with 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 mounted in a non-global zone or is not shared to a non-global zone.

- If a ZFS file system's `mountpoint` property set to `legacy`, the file system can 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
/export/home/data      export_home_data      nfs      sec=sys,rw

zfszone# zfs set share=name=data1,path=/export/home/data1,prot=nfs
tank/zones/export/home/data1
zfszone# zfs set sharenfs=on tank/zones/export/home/data1
zfszone# cat /etc/dfs/sharetab
/export/home/data1    data1    nfs      sec=sys,rw
```

New ZFS Sharing and Legacy Share Command Summary

This table describes the new ZFS file system sharing syntax and the legacy sharing syntax.

TABLE 6-5 ZFS Sharing and Legacy Share Command Summary

ZFS Share Task	Legacy Share Syntax	New Share Syntax
Share a ZFS file system over NFS.	Set the <code>sharenfs</code> property to on. # <code>zfs set sharenfs=on tank/fs1</code>	<ol style="list-style-type: none"> 1. Create the NFS share. # <code>zfs set share=name=fs1,path=/fs1,prot=nfs tank/fs1</code> 2. Set the <code>sharenfs</code> property to on. # <code>zfs set sharenfs=on tank/fs1</code>
Share ZFS file system over SMB.	Set the <code>sharesmb</code> property to on. # <code>zfs set sharesmb=on tank/fs2</code>	<ol style="list-style-type: none"> 1. Create the SMB share. # <code>zfs set share=name=fs2,path=/fs2,prot=smb tank/fs2</code> 2. Set the <code>sharesmb</code> property to on. # <code>zfs set sharesmb=on tank/fs2</code>
Unshare the ZFS file system.	Set the <code>sharenfs</code> property to off. # <code>zfs set sharenfs=off tank/fs1</code> Set the <code>sharesmb</code> property to off. # <code>zfs set sharesmb=off tank/fs2</code>	Set the <code>sharenfs</code> property to off. # <code>zfs set sharenfs=off tank/fs1</code> Set the <code>sharesmb</code> property to off. # <code>zfs set sharesmb=off tank/fs2</code>

TABLE 6-5 ZFS Sharing and Legacy Share Command Summary (Continued)

ZFS Share Task	Legacy Share Syntax	New Share Syntax
Add share options to an existing share.	Reset the sharenfs property. # zfs set sharenfs=nosuid tank/fs1	Reset the share with the additional property. # zfs set share=name=fs1,prot=nfs,nosuid rpool/fs1 name=fs1,path=/rpool/fs1,prot=nfs,nosuid=true
Create a permanent NFS share.	Set the sharenfs property to on. # zfs set sharenfs=on tank/fs1 For legacy share command syntax, you had to edit the /etc/dfs/dfstab file to create a permanent share.	Set the sharenfs property to on. # zfs set sharenfs=on tank/fs1 The /etc/dfs/dfstab file is not available in this Solaris release.
Create a permanent SMB share.	Set the sharesmb property to on. # zfs set sharesmb=on tank/fs2 Or, create the SMB share with sharemgr. # sharemgr create -P smb fssmb # sharemgr add-share -r fs-smb -s /tank/fs2 fssmb	Set the sharesmb property to on. # zfs set sharesmb=on tank/fs2 The sharemgr feature is not available in this Solaris release.

Troubleshooting ZFS Share Problems

- You can't share a parent file system if a subdirectory or descendent file system is already shared.


```
# share -F nfs /rpool/fs2/dir1
# share -F nfs /rpool/fs2/dir2
# share -F nfs /rpool/fs2
share: NFS: descendant of path is shared: /rpool/fs2/dir1 in rpool_fs2_dir2
```
- Renaming a share that is created with the `zfs set share` command is not supported.
- You can create a file system share with both NFS and SMB protocols by using the `zfs set share` command. For example:

```
# zfs set share=name=ds,path=/ds,prot=nfs,prot=smb rpool/ds
name=ds,path=/ds,prot=nfs,prot=smb
```

If you want to create a file system share with both NFS and SMB protocols by using the legacy share command, you must specify the command twice. For example:

```
# share -F nfs /rpool/ds
# share -F smb /rpool/ds
# zfs get share rpool/df
name=rpool_ds,path=/rpool/ds,prot=nfs,prot=smb
```

- A share path or description that includes a comma (,) must be quoted with double quotes.

ZFS Sharing Migration/Transition Issues

Identify any transition issues in this section.

- **Upgrading your system** – ZFS shares will be incorrect if you boot back to an older BE due to property changes in this release. Non-ZFS shares are unaffected. If you plan to boot back to an older BE, you should first save a copy of the existing share configuration prior to the pkg update operation to be able to restore the share configuration on the ZFS file systems.
 - In the older BE, use the `sharemgr show -vp` command to list all shares and their configuration.
 - Use the `zfs get sharenfs filesystem` command and the `zfs sharesmb filesystem` commands to get the values of the sharing properties.
 - If you back to an older BE, reset the `sharenfs` and `sharesmb` properties to their original values.
- **Legacy unsharing behavior** – Using the `unshare -a` command or `unshareall` command unpublishes a share, 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.

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 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.

For more information about setting quotas and reservations, see “[Setting Quotas on ZFS File Systems](#)” on page 168 and “[Setting Reservations on ZFS File Systems](#)” on page 171.

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 cannot set a quota to an amount less than is currently being used by a file system. For example:

```
# zfs set quota=10K tank/home/jeff
cannot set property for 'tank/home/jeff':
size is less than current used or reserved space
```


You can set a refquota on a file system that limits the amount of disk space that the file system can consume. This hard 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
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  -
```

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
```

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 users might exceed their quota before the system notices that they are 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 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 descendants 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.

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.
- ZFS encryption is inheritable to descendent file systems. Key management can be delegated through ZFS delegated administration
- 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.

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 when the file system is created or mounted
 - ,file:///filename – The key file location in a file system
 - pkcs11 – A URI describing the location of a key in a PKCS#11 token
 - https://location – The key file location on a secure server

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 6-4](#).

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  Tue Oct 12 15:36 2010  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.

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.

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 6-1 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 6-2 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 6-3 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.

EXAMPLE 6-3 Cloning an Encrypted ZFS File System (Continued)

```
# 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 6-4 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

You can use the shadow migration feature to migrate file systems as follows:

- A local or remote ZFS file system to a target ZFS file system
- A local or remote UFS file system to a target ZFS file system

Shadow migration is a process that pulls the data to be migrated:

- Create an empty ZFS file system.
- Set the shadow property on an empty ZFS file system, which is the target (or shadow) file system, to point to the file system to be migrated.
- Data from file system to be migrated is copied over to the shadow file system.

You can use the shadow property URI to identify the file system to be migrated in two ways:

- `shadow=file:///path` – Use this syntax to migrate a local file system
- `shadow=nfs://host:path` – Use this syntax to migrate a NFS file system

Review the following considerations when migrating file systems:

- The file system to be migrated must be set to read-only. If the file system is not set to read-only, in progress changes might not be migrated.
- The target file system must be completely empty.
- If the system is rebooted during a migration, the migration continues after the system is booted.
- Access to directory content that is not completely migrated or access to file content that is not completely migrated is blocked until the entire content is migrated.
- If you want the UID, GID, and ACL information to be migrated to the shadow file system during an NFS migration, make sure that the name service information is accessible between the local and remote systems. You might consider copying a subset of the file system data to be migrated for a test migration to see that all the information is migrated properly before doing a completing a large migration of data over NFS.
- Migrating file system data over NFS can be slow, depending on your network bandwidth. Be patient.
- You can use the `shadowstat` command to monitor a file system migration, which provides the following data:
 - The `BYTES XFRD` column identifies how many bytes have been transferred to the shadow file system.
 - The `BYTES LEFT` column fluctuates continuously until the migration is almost complete. ZFS does not identify how much data needs to be migrated at the beginning of the migration because this process might be too time-consuming.
 - Consider using the `BYTES XFRD` and the `ELAPSED TIME` information to estimate the length of the migration process.

▼ 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 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 Install the shadow-migration package, if necessary, and enable the shadowd service to assist with the migration process.

```
# pkg install shadow-migration
```

```
# svcadm enable shadowd
```

If you do not enable the shadowd process, you will have to reset the shadow property to none when the migration process is complete.

- 3 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 readonly=on tank/home/data
```

If you are migrating a remote file system, share it read-only. For example,

```
# share -F nfs -o ro /export/home/ufsdata
# share
- /export/home/ufsdata ro ""
```

- 4 Create a new ZFS file system with the shadow property set to the file system to be migrated.

For example, if you are migrating a local ZFS file system, `rpool/old`, to a new ZFS file system, `users/home/shadow`, set the shadow property to `rpool/old` when the `users/home/shadow` file system is created.

```
# zfs create -o shadow=file:///rpool/old users/home/shadow
```

For example, to migrate `/export/home/ufsdata` from a remote server, set the shadow property when the ZFS file system is created.

```
# zfs create -o shadow=nfs://v120-brm-02/export/home/ufsdata users/home/shadow2
```

- 5 Check the progress of the migration.

For example:

```
# shadowstat
```

DATASET	BYTES XFRD	EST BYTES LEFT	ERRORS	ELAPSED TIME
users/home/shadow	45.5M	2.75M	-	00:02:31
users/home/shadow	55.8M	-	-	00:02:41
users/home/shadow	69.7M	-	-	00:02:51
No migrations in progress				

When the migration is complete, the shadow property is set to none.

```
# zfs get -r shadow users/home/shadow*
NAME          PROPERTY  VALUE   SOURCE
users/home/shadow  shadow   none    -
users/home/shadow2 shadow    none    -
```

Troubleshooting ZFS File System Migrations

Review the following points when troubleshooting ZFS migration problems:

- If the file system to be migrated is not set to read-only, then not all data will be migrated.
- If the target file system is not empty when the shadow property is set, the data migration will not begin.
- If you add or remove data from the file system to be migrated when the migration is in progress, those changes might not be migrated.
- If you attempt to change the mount of the shadow file system when the migration is in progress, you will see the following message:

```
# zfs set mountpoint=/users/home/data users/home/shadow3
cannot unmount '/users/home/shadow3': Device busy
```

Upgrading ZFS File Systems

If you have ZFS file systems from a previous 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

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

Working With Oracle Solaris ZFS Snapshots and Clones

This chapter describes how to create and manage Oracle Solaris ZFS snapshots and clones. Information about saving snapshots is also provided.

The following sections are provided in this chapter:

- “Overview of ZFS Snapshots” on page 183
- “Creating and Destroying ZFS Snapshots” on page 184
- “Displaying and Accessing ZFS Snapshots” on page 187
- “Rolling Back a ZFS Snapshot” on page 188
- “Overview of ZFS Clones” on page 190
- “Creating a ZFS Clone” on page 191
- “Destroying a ZFS Clone” on page 191
- “Replacing a ZFS File System With a ZFS Clone” on page 191
- “Sending and Receiving ZFS Data” on page 192

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.

ZFS snapshots include the following features:

- The 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 descendent 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 [“Sending and Receiving ZFS Data”](#) on page 192.

- [“Creating and Destroying ZFS Snapshots”](#) on page 184
- [“Displaying and Accessing ZFS Snapshots”](#) on page 187
- [“Rolling Back a ZFS Snapshot”](#) on page 188

Creating and Destroying ZFS Snapshots

Snapshots are created 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 is specified as follows:

```
filesystem@snapname  
volume@snapname
```

The snapshot name must satisfy the naming requirements in [“ZFS Component Naming Requirements”](#) on page 28.

In the following example, a snapshot of `tank/home/matt` that is named `friday` is created.

```
# zfs snapshot tank/home/matt@friday
```

You can create snapshots for all descendent file systems by using the `-r` option. For example:

```
# zfs snapshot -r tank/home@snap1  
# zfs list -t snapshot -r tank/home  
zfs list -t snapshot -r tank/home  
NAME                USED  AVAIL  REFER  MOUNTPOINT  
tank/home@snap1      0      -    34K    -  
tank/home/mark@snap1 0      -    2.00G  -  
tank/home/matt@snap1 0      -    1.00G  -  
tank/home/tom@snap1  0      -    2.00G  -
```

Snapshots have no modifiable properties. Nor can dataset properties be applied to a snapshot. For example:

```
# zfs set compression=on tank/home/matt@friday  
cannot set property for 'tank/home/matt@friday':  
this property can not be modified for snapshots
```

Snapshots are destroyed by using the `zfs destroy` command. For example:

```
# zfs destroy tank/home/matt@friday
```

A dataset cannot be destroyed if snapshots of the dataset exist. For example:

```
# zfs destroy tank/home/matt  
cannot destroy 'tank/home/matt': filesystem has children  
use '-r' to destroy the following datasets:
```



```
tank/home/matt@tuesday
tank/home/matt@wednesday
tank/home/matt@thursday
```

In addition, if clones have been created from a snapshot, then they must be destroyed before the snapshot can be destroyed.

For more information about the `destroy` subcommand, see [“Destroying a ZFS File System” on page 125](#).

Holding ZFS Snapshots

If you have different automatic snapshot policies such that older snapshots are being inadvertently destroyed by `zfs receive` because they no longer exist on the sending side, you might consider using the snapshots hold feature.

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, a snapshot could only be destroyed 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 `tank/home/cindy/snap@1`:

```
# zfs hold keep tank/home/cindy@snap1
```

You can use the `-r` option to recursively hold the snapshots of all descendent file systems. For example:

```
# zfs snapshot -r tank/home@now
# zfs hold -r keep tank/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 tank/home/cindy@snap1
cannot destroy 'tank/home/cindy@snap1': dataset is busy
```

To destroy a held snapshot, use the `-d` option. For example:

```
# zfs destroy -d tank/home/cindy@snap1
```

Use the `zfs holds` command to display a list of held snapshots. For example:

```
# zfs holds tank/home@now
NAME          TAG      TIMESTAMP
tank/home@now keep    Fri May  6 06:34:03 2011
```

```
# zfs holds -r tank/home@now
NAME          TAG      TIMESTAMP
tank/home/cindy@now keep    Fri May  6 06:34:03 2011
tank/home/mark@now keep    Fri May  6 06:34:03 2011
tank/home/matt@now keep    Fri May  6 06:34:03 2011
tank/home/tom@now keep    Fri May  6 06:34:03 2011
tank/home@now keep    Fri May  6 06:34:03 2011
```

You can use the `zfs release` command to release a hold on a snapshot or set of snapshots. For example:

```
# zfs release -r keep tank/home@now
```

If the snapshot is released, the snapshot can be destroyed by using the `zfs destroy` command. For example:

```
# zfs destroy -r tank/home@now
```

Two new properties identify snapshot hold information.

- The `defer_destroy` property is on if the snapshot has been marked for deferred destruction by using the `zfs destroy -d` command. Otherwise, the property is off.
- The `userrefs` property is set to the number of holds on this snapshot, also referred to as the *user-reference* count.

Renaming ZFS Snapshots

You can rename snapshots, but they must be renamed within the same pool and dataset from which they were created. For example:

```
# zfs rename tank/home/cindy@snap1 tank/home/cindy@today
```

In addition, the following shortcut syntax is equivalent to the preceding syntax:

```
# zfs rename tank/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 tank/home/cindy@today pool/home/cindy@saturday
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 are no longer displayed in the `zfs list` output. You must use the `zfs list -t snapshot` command to display snapshot information. Or, enable the `listsnapshots` pool property. For example:

```

# zpool get listsnapshots tank
NAME PROPERTY      VALUE      SOURCE
tank listsnapshots off         default
# zpool set listsnapshots=on tank
# zpool get listsnapshots tank
NAME PROPERTY      VALUE      SOURCE
tank listsnapshots on          local

```

Snapshots of file systems are accessible in the `.zfs/snapshot` directory within the root of the file system. For example, if `tank/home/matt` is mounted on `/home/matt`, then the `tank/home/matt@thursday` snapshot data is accessible in the `/home/matt/.zfs/snapshot/thursday` directory.

```

# ls /tank/home/matt/.zfs/snapshot
tuesday wednesday thursday

```

You can list snapshots as follows:

```

# zfs list -t snapshot -r tank/home
NAME                          USED    AVAIL  REFER  MOUNTPOINT
tank/home/cindy@today          0      -  2.00G  -
tank/home/mark@today           0      -  2.00G  -
tank/home/matt@tuesday         20K    -  1.00G  -
tank/home/matt@wednesday        20K    -  1.00G  -
tank/home/matt@thursday         0      -  1.00G  -

```

You can list snapshots that were created for a particular file system as follows:

```

# zfs list -r -t snapshot -o name,creation tank/home
NAME                          CREATION
tank/home/cindy@today          Fri May 6 6:32 2011
tank/home/mark@today           Fri May 6 6:22 2011
tank/home/matt@tuesday         Tue May 3 6:27 2011

```

```
tank/home/matt@wednesday Wed May 4 6:28 2011
tank/home/matt@thursday Thu May 5 6:28 2011
```

Disk Space Accounting for ZFS Snapshots

When a snapshot is created, 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 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 DS	USED REFRESERV	USED CHILD
rpool	60.0G	6.92G	0	40.5K	0	6.92G
rpool/ROOT	60.0G	3.89G	0	31K	0	3.89G
rpool/ROOT/solaris	60.0G	3.49G	40.4M	3.16G	0	306M
rpool/ROOT/solaris-1	60.0G	403M	0	310M	0	92.7M
rpool/ROOT/solaris-1/var	60.0G	92.7M	0	92.7M	0	0
rpool/ROOT/solaris/var	60.0G	306M	89.9M	216M	0	0
rpool/dump	60.1G	2.00G	0	1.94G	62.7M	0
rpool/export	60.0G	96.5K	0	32K	0	64.5K
rpool/export/home	60.0G	64.5K	0	32K	0	32.5K
rpool/export/home/admin	60.0G	32.5K	0	32.5K	0	0
rpool/swap	60.0G	1.03G	0	1.00G	32.5M	0

For a description of these properties, see [Table 6-1](#).

Rolling Back a ZFS Snapshot

You can use the `zfs rollback` command to discard all changes made to a file system since a specific snapshot was created. 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, all intermediate snapshots must be destroyed. You can destroy earlier snapshots by specifying the `-r` option.

If clones of any intermediate snapshots exist, the `-R` option must be specified to destroy the clones as well.

Note – The file system that you want to roll back is unmounted and remounted, if it is currently mounted. If the file system cannot be unmounted, the rollback fails. The `-f` option forces the file system to be unmounted, if necessary.

In the following example, the `tank/home/matt` file system is rolled back to the tuesday snapshot:

```
# zfs rollback tank/home/matt@tuesday
cannot rollback to 'tank/home/matt@tuesday': more recent snapshots exist
use '-r' to force deletion of the following snapshots:
tank/home/matt@wednesday
tank/home/matt@thursday
# zfs rollback -r tank/home/matt@tuesday
```

In this example, the wednesday and thursday snapshots are destroyed because you rolled back to the earlier tuesday snapshot.

```
# zfs list -r -t snapshot -o name,creation tank/home/matt
NAME                CREATION
tank/home/matt@tuesday  Tue May  3  6:27 2011
```

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 /tank/home/tim
fileA
$ zfs snapshot tank/home/tim@snap1
$ ls /tank/home/tim
fileA fileB
$ zfs snapshot tank/home/tim@snap2
```

For example, to identify the differences between two snapshots, use syntax similar to the following:

```
$ zfs diff tank/home/tim@snap1 tank/home/tim@snap2
M      /tank/home/tim/
+      /tank/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 `M` in the following output indicates that a file in a snapshot has been renamed.

```
$ mv /tank/cindy/fileB /tank/cindy/fileC
$ zfs snapshot tank/cindy@snap2
$ zfs diff tank/cindy@snap1 tank/cindy@snap2
```

```
M      /tank/cindy/
R      /tank/cindy/fileB -> /tank/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 [zfs\(1M\)](#).

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 only be created 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 it was created. Use the `zfs get` and `zfs set` commands to view and change the properties of a cloned dataset. For more information about setting ZFS dataset properties, see [“Setting ZFS Properties” on page 149](#).

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.

- [“Creating a ZFS Clone” on page 191](#)
- [“Destroying a ZFS Clone” on page 191](#)
- [“Replacing a ZFS File System With a ZFS Clone” on page 191](#)

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. The new file system or volume 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.

In the following example, a new clone named `tank/home/matt/bug123` with the same initial contents as the snapshot `tank/ws/gate@yesterday` is created:

```
# zfs snapshot tank/ws/gate@yesterday
# zfs clone tank/ws/gate@yesterday tank/home/matt/bug123
```

In the following example, a cloned workspace is created from the `projects/newproject@today` snapshot for a temporary user as `projects/teamA/tempuser`. Then, properties are set on the cloned workspace.

```
# zfs snapshot projects/newproject@today
# zfs clone projects/newproject@today projects/teamA/tempuser
# zfs set share=name=projectA,path=/projects/teamA/tempuser,prot=nfs
projects/teamA/tempuser
name=projectA,path=/projects/teamA/tempuser,prot=nfs
# zfs set sharenfs=on projects/teamA/tempuser
# zfs set quota=5G projects/teamA/tempuser
```

Destroying a ZFS Clone

ZFS clones are destroyed by using the `zfs destroy` command. For example:

```
# zfs destroy tank/home/matt/bug123
```

Clones must be destroyed before the parent snapshot can be destroyed.

Replacing a ZFS File System With a ZFS Clone

You can use the `zfs promote` command to replace an active ZFS file system with a clone of that file system. 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, this feature makes it possible to 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 about destroying clones, see [“Destroying a ZFS Clone” on page 191](#).

In the following example, the `tank/test/productA` file system is cloned and then the clone file system, `tank/test/productAbeta`, becomes the original `tank/test/productA` file system.

```
# zfs create tank/test
# zfs create tank/test/productA
# zfs snapshot tank/test/productA@today
# zfs clone tank/test/productA@today tank/test/productAbeta
# zfs list -r tank/test
NAME                USED  AVAIL  REFER  MOUNTPOINT
tank/test            104M  66.2G   23K    /tank/test
tank/test/productA  104M  66.2G  104M   /tank/test/productA
tank/test/productA@today    0     -    104M   -
tank/test/productAbeta    0  66.2G  104M   /tank/test/productAbeta
# zfs promote tank/test/productAbeta
# zfs list -r tank/test
NAME                USED  AVAIL  REFER  MOUNTPOINT
tank/test            104M  66.2G   24K    /tank/test
tank/test/productA    0  66.2G  104M   /tank/test/productA
tank/test/productAbeta 104M  66.2G  104M   /tank/test/productAbeta
tank/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 tank/test/productA tank/test/productALegacy
# zfs rename tank/test/productAbeta tank/test/productA
# zfs list -r tank/test
```

Optionally, you can remove the legacy file system. For example:

```
# zfs destroy tank/test/productALegacy
```

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 send ZFS snapshot data and receive ZFS snapshot data and file systems with these commands. See the examples in the next section.

- [“Saving ZFS Data With Other Backup Products” on page 193](#)
- [“Sending a ZFS Snapshot” on page 195](#)
- [“Receiving a ZFS Snapshot” on page 196](#)
- [“Applying Different Property Values to a ZFS Snapshot Stream” on page 197](#)
- [“Sending and Receiving Complex ZFS Snapshot Streams” on page 199](#)
- [“Remote Replication of ZFS Data” on page 202](#)

The following backup solutions for saving ZFS data are available:

- **Enterprise backup products** – If you need the following features, then consider an enterprise backup solution:

- Per-file restoration
- Backup media verification
- Media management
- **File system snapshots and rolling back snapshots** – Use the `zfs snapshot` and `zfs rollback` commands if you want to easily create a copy of a file system and revert to a previous file system version, if necessary. For example, to restore a file or files from a previous version of a file system, you could use this solution.

For more information about creating and rolling back to a snapshot, see [“Overview of ZFS Snapshots” on page 183](#).

- **Saving snapshots** – Use the `zfs send` and `zfs receive` commands to send and receive a ZFS snapshot. You can save incremental changes between snapshots, but you cannot restore files individually. You must restore the entire file system snapshot. These commands do not provide a complete backup solution for saving your ZFS data.
- **Remote replication** – Use the `zfs send` and `zfs receive` commands to 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. The advantage of replicating a ZFS file system is that you can 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.

For up-to-date information about issues with ZFS and third-party backup products, see the Oracle Solaris 11 Release Notes.

Identifying ZFS Snapshot Streams

A snapshot of a ZFS file system or volume is converted into a snapshot stream by using the `zfs send` command. Then, you can use the snapshot stream to re-create a ZFS file system or volume by using the `zfs receive` command.

Depending on the `zfs send` options that were used to create the snapshot stream, different types of stream formats are 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 on the command line.

- 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. Three types of stream packages exist:

- Replication stream package – Consists of the specified dataset and its descendents. It includes all intermediate snapshots. If the origin of a cloned dataset is not a descendent 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.

Consider the following list of datasets and their origins. Assume that they were created in the order that they appear below.

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	-

A replication stream package that is created with the following syntax:

```
# zfs send -R pool/b@send ...
```

Consists 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 preceding output, the `pool/a/1@clone` snapshot is not included in the replication stream package. As such, 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 descendents. 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 descendent of the snapshot specified on the command line, the behavior is the similar to replication streams. However, a self-contained recursive stream, discussed below, are created in such a way that there are no external dependencies.

A recursive stream package that is created with the following syntax:

```
# zfs send -r pool/b@send ...
```

Consists 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 preceding output, the pool/a/1@clone snapshot is not included in the recursive stream package. As such, 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 - Is not dependent on any datasets that are not included in the stream package. This recursive stream package is created with the following syntax:

```
# zfs send -rc pool/b@send ...
```

Consists 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 the 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 syntax similar to the following:

```
# zfs send tank/dana@snap1 | zfs recv spool/ds01
```

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 tank/dana@snap1 | ssh sys2 zfs recv newtank/dana
```

When you send a full stream, the destination file system must not exist.

You can send incremental data by using the `zfs send -i` option. For example:

```
sys1# zfs send -i tank/dana@snap1 tank/dana@snap2 | ssh sys2 zfs recv newtank/dana
```

Note that the first argument (`snap1`) is the earlier snapshot and the second argument (`snap2`) is the later snapshot. In this case, the `newtank/dana` file system must already exist for the incremental receive to be successful.

The incremental `snap1` source can be specified as the last component of the snapshot name. This shortcut means you only have to specify 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 tank/dana@snap2 | ssh sys2 zfs recv newtank/dana
```

This shortcut syntax is equivalent to the incremental syntax in the preceding example.

The following message is displayed if you attempt to generate an incremental stream from a different file system snapshot 1:

```
cannot send 'pool/fs@name': not an earlier snapshot from the same fs
```

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
```

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 descendent file systems are unmounted.
- The file systems are inaccessible while they are being received.
- The original file system to be received must not exist while it is being transferred.
- If the file system name already exists, you can use `zfs rename` command to rename the file system.

For example:

```
# zfs send tank/gozer@0830 > /bkups/gozer.083006
# zfs receive tank/gozer2@today < /bkups/gozer.083006
# zfs rename tank/gozer tank/gozer.old
# zfs rename tank/gozer2 tank/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 newtank/dana/file.1
```

Then, perform an incremental send of tank/dana@snap3. However, you must first roll back the receiving file system to receive the new incremental snapshot. Or, you can eliminate the rollback step by using the -F option. For example:

```
sys1# zfs send -i tank/dana@snap2 tank/dana@snap3 | ssh sys2 zfs recv -F newtank/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 do not use the -F option, you see a message similar to the following:

```
sys1# zfs send -i tank/dana@snap4 tank/dana@snap5 | ssh sys2 zfs recv newtank/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 roll back 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, but you can specify a different local property value when the snapshot stream is received. Or, you can 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.

- Use the `zfs inherit -S` to revert a local property value to the received value, if any. If a property does not have a received value, the behavior of the `zfs inherit -S` command is the same as the `zfs inherit` command without the `-S` 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 use the `zfs get -o` to include the new non-default RECEIVED column. Or, use the `zfs get -o all` command to include all columns, including RECEIVED.
- You can use the `zfs send -p` option to include properties in the send stream without the `-R` option.
- You can use the `zfs send -e` option to use the last element of the sent snapshot name to determine the new snapshot name. The following example sends the `poola/bee/cee@1` snapshot to the `poold/eee` file system and only uses 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    -
```

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, the `tank/data` file system has the `compression` property disabled. A snapshot of the `tank/data` file system is sent with properties (`-p` option) to a backup pool and is received with the `compression` property enabled.

```
# zfs get compression tank/data
NAME          PROPERTY  VALUE  SOURCE
tank/data     compression  off    default
# zfs snapshot tank/data@snap1
# zfs send -p tank/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` command 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 `tank/data` file system.

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` command. 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 tank/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 tank/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 `zfs send -I` and `-R` options to send and receive more complex snapshot streams.

Keep the following points in mind when sending and receiving complex ZFS snapshot streams:

- Use the `zfs send -I` option to send all incremental streams from one snapshot to a cumulative snapshot. Or, 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.
- Use the `zfs send -R` option to send a replication stream of all descendent file systems. When the replication stream is received, all properties, snapshots, descendent file systems, and clones are preserved.
- When using the `zfs send -r` option without the `-c` option and when using the `zfs send -R` option stream packages omit the origin of clones in some circumstances. For more information, see “[Identifying ZFS Snapshot Streams](#)” on page 193.
- Use both options to send an incremental replication stream.
 - Changes to properties are preserved, as are snapshot and file system rename and `destroy` operations are preserved.
 - If `zfs recv -F` is not specified when receiving the replication stream, dataset `destroy` operations are ignored. The `zfs recv -F` syntax in this case also retains its *rollback if necessary* meaning.
 - As with other (non `zfs send -R`) `-i` or `-I` cases, if `-I` is used, all snapshots between `snapA` and `snapD` are sent. If `-i` is used, only `snapD` (for all descendents) are sent.
- To receive any of these new types of `zfs send` streams, the receiving system must be running a software version capable of sending them. The stream version is incremented. However, you can access streams from older pool versions by using a newer software version. For example, you can send and receive streams created with the newer options to and from a version 3 pool. But, you must be running recent software to receive a stream sent with the newer options.

EXAMPLE 7-1 Sending and Receiving Complex ZFS Snapshot Streams

A group of incremental snapshots can be combined into one snapshot by using the `zfs send -I` option. For example:

```
# zfs send -I pool/fs@snapA pool/fs@snapD > /snaps/fs@all-I
```

Then, you would remove `snapB`, `snapC`, and `snapD`.

```
# 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
```


EXAMPLE 7-1 Sending and Receiving Complex ZFS Snapshot Streams (Continued)

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 `zfs send -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/fsc clonesnap-I
# zfs destroy pool/clone@snapA
# zfs destroy pool/clone
# zfs receive -F pool/clone < /snaps/fsc clonesnap-I
```

You can use the `zfs send -R` command to replicate a ZFS file system and all descendent file systems, up to the named snapshot. When this stream is received, all properties, snapshots, descendent file systems, and clones are preserved.

In the following example, snapshots are created 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    -
```

In the following example, the `zfs send -R` command was used to replicate the `users` file system and its descendents, and to send the replicated stream to another pool, `users2`.

EXAMPLE 7-1 Sending and Receiving Complex ZFS Snapshot Streams *(Continued)*

```
# 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 tank/cindy@today | ssh newsys zfs recv sandbox/restfs@today
```

This command sends the `tank/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.

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.

The following sections are provided in this chapter:

- “New Solaris ACL Model” on page 203
- “Setting ACLs on ZFS Files” on page 210
- “Setting and Displaying ACLs on ZFS Files in Verbose Format” on page 212
- “Setting and Displaying ACLs on ZFS Files in Compact Format” on page 223
- “Applying Special Attributes to ZFS Files” on page 229

New Solaris ACL Model

Previous versions of Solaris supported an ACL implementation that was primarily based on the POSIX-draft ACL specification. The POSIX-draft based ACLs are used to protect UFS files and are translated by versions of NFS prior to NFSv4.

With the introduction of NFSv4, a new ACL model fully supports the interoperability that NFSv4 offers between UNIX and non-UNIX clients. The new ACL implementation, as defined in the NFSv4 specification, provides much richer semantics that are based on NT-style ACLs.

The main differences of the new ACL model are as follows:

- Based on the NFSv4 specification and similar to NT-style ACLs.
- Provide much more granular set of access privileges. For more information, see [Table 8–2](#).
- Set and displayed with the `chmod` and `ls` commands rather than the `setfacl` and `getfacl` commands.
- Provide richer inheritance semantics for designating how access privileges are applied from directory to subdirectories, and so on. For more information, see “[ACL Inheritance](#)” on page 208.

Both ACL models provide more fine-grained access control than is available with the standard file permissions. Much like POSIX-draft ACLs, the new ACLs are composed of multiple Access Control Entries (ACEs).

POSIX-draft style ACLs use a single entry to define what permissions are allowed and what permissions are denied. The new ACL model has two types of ACEs that affect access checking: `ALLOW` and `DENY`. As such, you cannot infer from any single ACE that defines a set of permissions whether or not the permissions that weren't defined in that ACE are allowed or denied.

Translation between NFSv4-style ACLs and POSIX-draft ACLs is as follows:

- If you use any ACL-aware utility, such as the `cp`, `mv`, `tar`, `cpio`, or `rcp` commands, to transfer UFS files with ACLs to a ZFS file system, the POSIX-draft ACLs are translated into the equivalent NFSv4-style ACLs.
- Some NFSv4-style ACLs are translated to POSIX-draft ACLs. You see a message similar to the following if an NFSv4-style ACL isn't translated to a POSIX-draft ACL:

```
# cp -p filea /var/tmp
cp: failed to set acl entries on /var/tmp/filea
```

- If you create a UFS `tar` or `cpio` archive with the `preserve ACL` option (`tar -p` or `cpio -P`) on a system that runs a current Solaris release, you will lose the ACLs when the archive is extracted on a system that runs a previous Solaris release.

All of the files are extracted with the correct file modes, but the ACL entries are ignored.

- You can use the `ufsrestore` command to restore data into a ZFS file system. If the original data includes POSIX-style ACLs, they are converted to NFSv4-style ACLs.
- If you attempt to set an NFSv4-style ACL on a UFS file, you see a message similar to the following:

```
chmod: ERROR: ACL type's are different
```

- If you attempt to set a POSIX-style ACL on a ZFS file, you will see messages similar to the following:

```
# getfacl filea
File system doesn't support aclent_t style ACL's.
See acl(5) for more information on Solaris ACL support.
```

For information about other limitations with ACLs and backup products, see [“Saving ZFS Data With Other Backup Products” on page 193](#).

Syntax Descriptions for Setting ACLs

Two basic ACL formats are provided as follows:

Syntax for Setting 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
```

Syntax for Setting 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
```

owner@, group@, everyone@

Identifies the *ACL-entry-type* for trivial ACL syntax. For a description of *ACL-entry-types*, see [Table 8-1](#).

user or group:ACL-entry-ID=username or groupname

Identifies the *ACL-entry-type* for explicit ACL syntax. The user and group *ACL-entry-type* must also contain the *ACL-entry-ID*, *username* or *groupname*. For a description of *ACL-entry-types*, see [Table 8-1](#).

access-permissions/.../

Identifies the access permissions that are granted or denied. For a description of ACL access privileges, see [Table 8-2](#).

inheritance-flags

Identifies an optional list of ACL inheritance flags. For a description of the ACL inheritance flags, see [Table 8-4](#).

deny | allow

Identifies whether the access permissions are granted or denied.

In the following example, no *ACL-entry-ID* value exists for owner@, group@, or everyone@..

```
group@:write_data/append_data/execute:deny
```

The following example includes an *ACL-entry-ID* because a specific user (*ACL-entry-type*) is included in the ACL.

```
0:user:gozer:list_directory/read_data/execute:allow
```

When an ACL entry is displayed, it looks similar to the following:

```
2:group@:write_data/append_data/execute:deny
```

The 2 or the *index-ID* designation in this example identifies the ACL entry in the larger ACL, which might have multiple entries for owner, specific UIDs, group, and everyone. You can

specify the *index-ID* with the `chmod` command to identify which part of the ACL you want to modify. For example, you can identify index ID 3 as A3 to the `chmod` command, similar to the following:

```
chmod A3=user:venkman:read_acl:allow filename
```

ACL entry types, which are the ACL representations of owner, group, and other, are described in the following table.

TABLE 8-1 ACL Entry Types

ACL Entry Type	Description
owner@	Specifies the access granted to the owner of the object.
group@	Specifies the access granted to the owning group of the object.
everyone@	Specifies the access granted to any user or group that does not match any other ACL entry.
user	With a user name, specifies the access granted to an additional user of the object. Must include the <i>ACL-entry-ID</i> , which contains a <i>username</i> or <i>userID</i> . If the value is not a valid numeric UID or <i>username</i> , the ACL entry type is invalid.
group	With a group name, specifies the access granted to an additional group of the object. Must include the <i>ACL-entry-ID</i> , which contains a <i>groupname</i> or <i>groupID</i> . If the value is not a valid numeric GID or <i>groupname</i> , the ACL entry type is invalid.

ACL access privileges are described in the following table.

TABLE 8-2 ACL Access Privileges

Access Privilege	Compact Access Privilege	Description
add_file	w	Permission to add a new file to a directory.
add_subdirectory	p	On a directory, permission to create a subdirectory.
append_data	p	Not currently implemented.
delete	d	Permission to delete a file. For more information about specific delete permission behavior, see Table 8-3 .
delete_child	D	Permission to delete a file or directory within a directory. For more information about specific delete_child permission behavior, see Table 8-3 .
execute	x	Permission to execute a file or search the contents of a directory.
list_directory	r	Permission to list the contents of a directory.

TABLE 8-2 ACL Access Privileges (Continued)

Access Privilege	Compact Access Privilege	Description
read_acl	c	Permission to read the ACL (ls).
read_attributes	a	Permission to read basic attributes (non-ACLs) of a file. Think of basic attributes as the stat level attributes. Allowing this access mask bit means the entity can execute <code>ls(1)</code> and <code>stat(2)</code> .
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	Not currently implemented.
write_xattr	W	Permission to create extended attributes or write to the extended attributes directory. 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. 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 8-3 ACL delete and delete_child Permission Behavior

Parent Directory Permissions	Target Object Permissions		
	ACL allows delete	ACL denies delete	Delete permission unspecified
ACL allows <code>delete_child</code>	Permit	Permit	Permit
ACL denies <code>delete_child</code>	Permit	Deny	Deny

TABLE 8-3 ACL delete and delete_child Permission Behavior (Continued)

Parent Directory Permissions	Target Object Permissions		
ACL allows only write and execute	Permit	Permit	Permit
ACL denies write and execute	Permit	Deny	Deny

ZFS ACL Sets

The following ACL combinations can be applied in an *ACL set* rather than setting individual permissions separately. The following ACL sets are available.

ACL Set Name	Included ACL Permissions
full_set	All 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

These ACL sets are predefined and cannot be modified.

ACL Inheritance

The purpose of using ACL inheritance is so that a newly created file or directory can inherit the ACLs they are intended to inherit, but 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.

The optional inheritance flags are described in the following table.

TABLE 8-4 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.

TABLE 8-4 ACL Inheritance Flags (Continued)

Inheritance Flag	Compact Inheritance Flag	Description
<code>inherit_only</code>	<code>i</code>	Inherit the ACL from the parent directory but applies only to newly created files or subdirectories and not the directory itself. This flag requires the <code>file_inherit</code> flag, the <code>dir_inherit</code> flag, or both, to indicate what to inherit.
<code>no_propagate</code>	<code>n</code>	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 <code>file_inherit</code> flag, the <code>dir_inherit</code> flag, or both, to indicate what to inherit.
<code>-</code>	N/A	No permission granted.
Currently, the following flags are only applicable to a SMB client or server.		
<code>successful_access</code>	<code>S</code>	Indicates whether an alarm or audit record should be initiated upon a successful access. This flag is used with audit or alarm ACE types.
<code>failed_access</code>	<code>F</code>	Indicates whether an alarm or audit record should be initiated when an access fails. This flag is used with audit or alarm ACE types.
<code>inherited</code>	<code>I</code>	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, see the next section.

ACL Properties

The ZFS file system includes the following ACL properties to determine the specific behavior of ACL inheritance and ACL interaction with `chmod` operations.

- `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 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, but only if execute permission is set in the file creation mode and in an inheritable ACE that affects the mode.

The default mode for the `aclinherit` is `restricted`.

- `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` – A file system with an `aclmode` property of `discard` deletes all ACL entries that do not represent the mode of the file. This is the default value.
 - `mask` – A file system with an `aclmode` property of `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 an explicit ACL set operation has not been performed.
 - `passthrough` – A file system with an `aclmode` property of `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 on using the `aclmode` property, see [Example 8–14](#).

Setting ACLs on ZFS Files

As implemented with ZFS, ACLs are composed of an array of ACL entries. ZFS provides a *pure* ACL model, where all files have an ACL. Typically, the ACL is *trivial* in that it only represents the traditional UNIX owner/group/other entries.

ZFS files still have permission bits and a mode, but these values are more of a cache of what the ACL represents. As such, if you change the permissions of the file, the file's ACL is updated accordingly. In addition, if you remove a non-trivial ACL that granted a user access to a file or directory, that user could still have access to the file or directory because of the file or directory's permission bits that grant access to group or everyone. All access control decisions are governed by the permissions represented in a file or directory's ACL.

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 that have a “who” that 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 what access request 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.

If you set a non-trivial ACL on a directory, the ACL is not automatically inherited by the directory's children. If you set a non-trivial ACL and you want it inherited to the directory's children, you have to use the ACL inheritance flags. For more information, see [Table 8-4](#) and “[Setting ACL Inheritance on ZFS Files in Verbose Format](#)” on page 217.

When you create a new file and depending on the `umask` value, a default trivial ACL, similar to the following, is applied:

```
$ ls -v file.1
-rw-r--r--  1 root    root      206663 Jun 23 15:06 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/read_xattr/read_attributes/read_acl/synchronize:allow
 2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
            :allow
```

Each user category (`owner@`, `group@`, `everyone@`) has an ACL entry in this example.

A description of this file ACL is as follows:

0:owner@	The owner can read and modify the contents of the file (<code>read_data/write_data/append_data/read_xattr</code>). The owner can also modify the file's attributes such as timestamps, extended attributes, and ACLs (<code>write_xattr/read_attributes/write_attributes/read_acl/write_acl</code>). In addition, the owner can modify the ownership of the file (<code>write_owner:allow</code>).
	The <code>synchronize</code> access permission is not currently implemented.
1:group@	The group is granted read permissions to the file and the file's attributes (<code>read_data/read_xattr/read_attributes/read_acl:allow</code>).
2:everyone@	Everyone who is not user or group is granted read permissions to the file and the file's attributes (<code>read_data/read_xattr/read_attributes/read_acl/synchronize:allow</code>). The <code>synchronize</code> access permission is not currently implemented.

When a new directory is created and depending on the `umask` value, a default directory ACL is similar to the following:

```
$ ls -dv dir.1
drwxr-xr-x  2 root   root       2 Jul 20 13:44 dir.1
 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
```

A description of this directory ACL is as follows:

- 0:owner@ The owner can read and modify the directory contents (list_directory/read_data/add_file/write_data/add_subdirectory/append_data) and read and modify the file's attributes such as timestamps, extended attributes, and ACLs (/read_xattr/write_xattr/read_attributes/write_attributes/read_acl/write_acl). In addition, the owner can search the contents (execute), delete a file or directory (delete_child), and can modify the ownership of the directory (write_owner:allow).
- The synchronize access permission is not currently implemented.
- 1:group@ The group can list and read the directory contents and the directory's attributes. In addition, the group has execute permission to search the directory contents (list_directory/read_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow).
- 2:everyone@ Everyone who is not user or group is granted read and execute permissions to the directory contents and the directory's attributes (list_directory/read_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow). The synchronize access permission is not currently implemented.

Setting and Displaying ACLs on ZFS Files in Verbose Format

You can use the `chmod` command to modify ACLs on ZFS files. The following `chmod` syntax for modifying ACLs uses *acl-specification* to identify the format of the ACL. For a description of *acl-specification*, see [“Syntax Descriptions for Setting ACLs” on page 204](#).

- Adding ACL entries
 - Adding an ACL entry for a user
 - `% chmod A+acl-specification filename`
 - Adding an ACL entry by *index-ID*

```
% chmod Aindex-ID+acl-specification filename
```

This syntax inserts the new ACL entry at the specified *index-ID* location.

- Replacing an ACL entry

```
% chmod A=acl-specification filename
```

```
% chmod Aindex-ID=acl-specification filename
```

- Removing ACL entries

- Removing an ACL entry by *index-ID*

```
% chmod Aindex-ID- filename
```

- Removing an ACL entry by user

```
% chmod A-acl-specification filename
```

- Removing all non-trivial ACEs from a file

```
% chmod A- filename
```

Verbose ACL information is displayed by using the `ls -v` command. For example:

```
# ls -v file.1
-rw-r--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/read_xattr/read_attributes/read_acl/synchronize:allow
 2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
  :allow
```

For information about using the compact ACL format, see [“Setting and Displaying ACLs on ZFS Files in Compact Format” on page 223](#).

EXAMPLE 8-1 Modifying Trivial ACLs on ZFS Files

This section provides examples of setting and displaying trivial ACLs.

In the following example, a trivial ACL exists on `file.1`:

```
# ls -v file.1
-rw-r--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/read_xattr/read_attributes/read_acl/synchronize:allow
 2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
  :allow
```

In the following example, `write_data` permissions are granted for `group@`.

```
# chmod A1=group@:read_data/write_data:allow file.1
# ls -v file.1
```

EXAMPLE 8-1 Modifying Trivial ACLs on ZFS Files (Continued)

```
-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, permissions on file.1 are set back to 644.

```
# chmod 644 file.1
# ls -v file.1
-rw-r--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/read_xattr/read_attributes/read_acl/synchronize:allow
 2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
  :allow
```

EXAMPLE 8-2 Setting Non-Trivial ACLs on ZFS Files

This section provides examples of setting and displaying non-trivial ACLs.

In the following example, read_data/execute permissions are added for the user gozer on the test.dir directory.

```
# chmod A+user:gozer: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:gozer: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, read_data/execute permissions are removed for user gozer.

```
# 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
```

EXAMPLE 8-2 Setting Non-Trivial ACLs on ZFS Files (Continued)

```
/read_acl/synchronize:allow
```

EXAMPLE 8-3 ACL Interaction With Permissions on ZFS Files

The following ACL examples illustrate the interaction between setting ACLs and then changing the file or directory's permission bits.

In the following example, a trivial ACL exists on file.2:

```
# ls -v file.2
-rw-r--r-- 1 root    root        2693 Jul 20 14:26 file.2
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
```

In the following example, ACL allow permissions are removed from everyone@.

```
# chmod A2- file.2
# ls -v file.2
-rw-r----- 1 root    root        2693 Jul 20 14:26 file.2
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
```

In this output, the file's permission bits are reset from 644 to 640. Read permissions for everyone@ have been effectively removed from the file's permissions bits when the ACL allow permissions are removed for everyone@.

In the following example, the existing ACL is replaced with read_data/write_data permissions for everyone@.

```
# chmod A=everyone@:read_data/write_data:allow file.3
# ls -v file.3
-rw-rw-rw- 1 root    root        2440 Jul 20 14:28 file.3
0:everyone@:read_data/write_data:allow
```

In this output, the chmod syntax effectively replaces the existing ACL with read_data/write_data:allow permissions to read/write permissions for owner, group, and everyone@. In this model, everyone@ specifies access to any user or group. Since no owner@ or group@ ACL entry exists to override the permissions for owner and group, the permission bits are set to 666.

In the following example, the existing ACL is replaced with read permissions for user gozer.

EXAMPLE 8-3 ACL Interaction With Permissions on ZFS Files (Continued)

```
# chmod A=user:gozer:read_data:allow file.3
# ls -v file.3
-----+ 1 root    root        2440 Jul 20 14:28 file.3
      0:user:gozer:read_data:allow
```

In this output, the file permissions are computed to be 000 because no ACL entries exist for owner@, group@, or everyone@, which represent the traditional permission components of a file. The owner of the file can resolve this problem by resetting the permissions (and the ACL) as follows:

```
# chmod 655 file.3
# ls -v file.3
-rw-r-xr-x 1 root    root        2440 Jul 20 14:28 file.3
      0:owner@:execute: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/execute/read_attributes/read_acl
        /synchronize:allow
      3:everyone@:read_data/read_xattr/execute/read_attributes/read_acl
        /synchronize:allow
```

EXAMPLE 8-4 Restoring Trivial ACLs on ZFS Files

You can use the `chmod` command to remove all non-trivial ACLs on a file or directory.

In the following example, two non-trivial ACEs exist on `test5.dir`.

```
# ls -dv test5.dir
drwxr-xr-x+ 2 root    root        2 Jul 20 14:32 test5.dir
      0:user:lp:read_data:file_inherit:deny
      1:user:gozer:read_data:file_inherit:deny
      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
```

In the following example, the non-trivial ACLs for users `gozer` and `lp` are removed. The remaining ACL contains the default values for `owner@`, `group@`, and `everyone@`.

```
# chmod A- test5.dir
# ls -dv test5.dir
drwxr-xr-x 2 root    root        2 Jul 20 14:32 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
```


EXAMPLE 8-4 Restoring Trivial ACLs on ZFS Files (Continued)

```

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

```

EXAMPLE 8-5 Applying an ACL Set to ZFS Files

ACL sets are available so that you do not have to apply ACL permissions separately. For a description of ACL sets, see [“ZFS ACL Sets” on page 208](#).

For example, you can apply the `read_set` as follows:

```

# chmod A+user:otto:read_set:allow file.1
# ls -v file.1
-r--r--r--+ 1 root    root      206695 Jul 20 13:43 file.1
 0:user:otto:read_data/read_xattr/read_attributes/read_acl:allow
 1:owner@:read_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

```

You can apply the `write_set` and `read_set` as follows:

```

# chmod A+user:otto:read_set/write_set:allow file.2
# ls -v file.2
-rw-r--r--+ 1 root    root      2693 Jul 20 14:26 file.2
 0:user:otto:read_data/write_data/append_data/read_xattr/write_xattr
   /read_attributes/write_attributes/read_acl: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

```

Setting ACL Inheritance on ZFS Files in Verbose Format

You can determine how ACLs are inherited or not inherited on files and directories. By default, ACLs are not propagated. If you set a non-trivial ACL on a directory, the ACL is not inherited by any subsequent directory. You must specify the inheritance of an ACL on a file or directory.

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 208](#).

EXAMPLE 8-6 Granting Default ACL Inheritance

By default, ACLs are not propagated through a directory structure.

In the following example, a non-trivial ACE of `read_data/write_data/execute` is applied for user `gozer` on `test.dir`.

```
# chmod A+user:gozer:read_data/write_data/execute:allow test.dir
# ls -dv test.dir
drwxr-xr-x+ 2 root    root          2 Jul 20 14:53 test.dir
 0:user:gozer:list_directory/read_data/add_file/write_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
```

If a `test.dir` subdirectory is created, the ACE for user `gozer` is not propagated. User `gozer` would only have access to `sub.dir` if the permissions on `sub.dir` granted him access as the file owner, group member, or `everyone@`.

```
# mkdir test.dir/sub.dir
# ls -dv test.dir/sub.dir
drwxr-xr-x  2 root    root          2 Jul 20 14:54 test.dir/sub.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
```

EXAMPLE 8-7 Granting ACL Inheritance on Files and Directories

This series of examples identify the file and directory 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 `gozer` so that he has read access on any newly created files.

```
# chmod A+user:gozer: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:gozer: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
```

EXAMPLE 8-7 Granting ACL Inheritance on Files and Directories (Continued)

```

    /read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
    /read_acl/synchronize:allow

```

In the following example, user gozer's permissions are applied on the newly created `test2.dir/file.2` file. The ACL inheritance granted, `read_data:file_inherit:allow`, means user gozer can read the contents of any newly created file.

```

# touch test2.dir/file.2
# ls -v test2.dir/file.2
-rw-r--r--+ 1 root    root          0 Jul 20 14:56 test2.dir/file.2
  0:user:gozer: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 gozer does not have `write_data` permission on `file.2` because the group permission of the file does not allow it.

Note 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 gozer is only granted or denied permission from `everyone@` permissions unless 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:gozer: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

```

The following series of examples 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 gozer is granted read, write, and execute permissions that are inherited for newly created files and directories.

```

# chmod A+user:gozer:read_data/write_data/execute:file_inherit/dir_inherit:allow
test3.dir

```

EXAMPLE 8-7 Granting ACL Inheritance on Files and Directories (Continued)

```
# ls -dv test3.dir
drwxr-xr-x+ 2 root    root          2 Jul 20 15:00 test3.dir
0:user:gozer: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 output below 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:gozer: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 gozer 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 gozer 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:gozer:read_data/write_data/execute:file_inherit/no_propagate:allow
test4.dir
# ls -dv test4.dir
drwxr-xr-x+ 2 root    root          2 Jul 20 15:05 test4.dir
0:user:gozer: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/execute/read_attributes
/read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

EXAMPLE 8-7 Granting ACL Inheritance on Files and Directories (Continued)

As the following example illustrates, when a new subdirectory is created, user gozer's read_data/write_data/execute permission for files are not propagated to the new sub4.dir directory.

```
# mkdir test4.dir/sub4.dir
# ls -dv test4.dir/sub4.dir
drwxr-xr-x  2 root   root           2 Jul 20 15:06 test4.dir/sub4.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
```

As the following example illustrates, gozer'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:gozer: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
```

EXAMPLE 8-8 ACL Inheritance With ACL Inherit Mode Set to Pass Through

If the `aclinherit` property on the tank/cindy file system is set to `passthrough`, then user gozer would inherit the ACL applied on `test4.dir` for the newly created `file.5` as follows:

```
# zfs set aclinherit=passthrough tank/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:gozer: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 8-9 ACL Inheritance With 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:

EXAMPLE 8-9 ACL Inheritance With ACL Inherit Mode Set to Discard (Continued)

```
# zfs set aclinherit=discard tank/cindy
# chmod A+user:gozer: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:gozer: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 8-10 ACL Inheritance With 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 tank/cindy
# chmod A+user:gozer: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:gozer: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
```

EXAMPLE 8-10 ACL Inheritance With ACL Inherit Mode Set to Noallow (Continued)

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:gozer: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
```

Setting and Displaying ACLs on ZFS Files in Compact Format

You can set and display permissions on ZFS files in a compact format that uses 14 unique letters to represent the permissions. The letters that represent the compact permissions are listed in [Table 8-2](#) and [Table 8-4](#).

You can display compact ACL listings for files and directories by using the `ls -V` command. For example:

```
# 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
```

The compact ACL output is described as follows:

owner@ The owner can read and modify the contents of the file (`rw=read_data/write_data`), (`p=append_data`). The owner can also modify the file's attributes such as timestamps, extended attributes, and ACLs (`a=read_attributes`, `A=write_xattr`, `R=read_xattr`, `W=write_attributes`, `c=read_acl`, `C=write_acl`). In addition, the owner can modify the ownership of the file (`o=write_owner`).

The synchronize (`s`) access permission is not currently implemented.

group@ The group is granted read permissions to the file (`r=read_data`) and the file's attributes (`a=read_attributes`, `R=read_xattr`, `c=read_acl`).

The synchronize (`s`) access permission is not currently implemented.

everyone@ Everyone who is not user or group is granted read permissions to the file and the file's attributes (r=read_data, a=append_data, R=read_xattr, c=read_acl, and s=synchronize).

The synchronize (s) access permission is not currently implemented.

Compact ACL format provides the following advantages over verbose ACL format:

- Permissions can be specified as positional arguments to the chmod command.
- The hyphen (-) characters, which identify no permissions, can be removed and only the required letters need to be specified.
- Both permissions and inheritance flags are set in the same fashion.

For information about using the verbose ACL format, see [“Setting and Displaying ACLs on ZFS Files in Verbose Format” on page 212](#).

EXAMPLE 8-11 Setting and Displaying ACLs in Compact Format

In the following example, a trivial ACL exists on file.1:

```
# 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
```

In this example, read_data/execute permissions are added for the user gozer on file.1.

```
# chmod A+user:gozer:rx:allow file.1
# ls -V file.1
-rw-r--r--+ 1 root    root      206695 Jul 20 14:27 file.1
  user:gozer:r-x-----:-----:allow
  owner@:rw-p--aARWcCos:-----:allow
  group@:r-----a-R-c--s:-----:allow
  everyone@:r-----a-R-c--s:-----:allow
```

In the following example, user gozer is granted read, write, and execute permissions that are inherited for newly created files and directories by using the compact ACL format.

```
# chmod A+user:gozer:rwx:fd:allow dir.2
# ls -dV dir.2
drwxr-xr-x+ 2 root    root      2 Jul 20 14:33 dir.2
  user:gozer:rwx-----:fd-----:allow
  owner@:rwxp--DaARWcCos:-----:allow
  group@:r-x---a-R-c--s:-----:allow
  everyone@:r-x---a-R-c--s:-----:allow
```

You can also cut and paste permissions and inheritance flags from the ls -V output into the compact chmod format. For example, to duplicate the permissions and inheritance flags on

EXAMPLE 8-11 Setting and Displaying ACLs in Compact Format (Continued)

`dir.2` for user `gozer` to user `cindy` on `dir.2`, copy and paste the permission and inheritance flags (`rwX-----:fd-----:allow`) into your `chmod` command. For example:

```
# chmod A+user:cindy:rwX-----:fd-----:allow dir.2
# ls -dV dir.2
drwxr-xr-x+ 2 root    root          2 Jul 20 14:33 dir.2
      user:cindy:rwX-----:fd-----:allow
      user:gozer:rwX-----:fd-----:allow
      owner@:rwxp-DaARwCcos:-----:allow
      group@:r-x---a-R-c--s:-----:allow
      everyone@:r-x---a-R-c--s:-----:allow
```

EXAMPLE 8-12 ACL Inheritance With ACL Inherit Mode Set to Pass Through

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. When this property is set to `passthrough`, 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.

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 tank/cindy
# pwd
/tank/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
```

EXAMPLE 8-12 ACL Inheritance With ACL Inherit Mode Set to Pass Through (Continued)

```
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 /tank/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
```

EXAMPLE 8-13 ACL Inheritance With ACL Inherit Mode Set to Pass Through-X

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 `aclinherit` mode to `passthrough-x`.

```
# zfs set aclinherit=passthrough-x tank/cindy
```

The following ACL is set on `/tank/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
```

EXAMPLE 8-13 ACL Inheritance With ACL Inherit Mode Set to Pass Through-X (Continued)

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-pDaARWcCos:-----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.

EXAMPLE 8-14 ACL Interaction With `chmod` Operations on ZFS Files

The following examples illustrate how specific `aclmode` and `aclinherit` property values influence the interaction of existing ACLs with a `chmod` operation that changes file or directory permissions to either reduce or expand any existing ACL permissions to be consistent with the owning group.

In this example, the `aclmode` property is set to `mask` and the `aclinherit` property is set to `restricted`. The ACL permissions in this example are displayed in compact mode, which more easily illustrates changing permissions.

The original file and group ownership and ACL permissions are as follows:

```
# zfs set aclmode=mask pond/whoville
# zfs set aclinherit=restricted pond/whoville

# 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
```

EXAMPLE 8-14 ACL Interaction With `chmod` Operations on ZFS Files (Continued)

A `chown` operation changes the file ownership on `file.1` and the output is now seen by the owning user, `amy`. For example:

```
# chown amy:staff file.1
# su - amy
$ ls -lV file.1
-rwxrwx---+ 1 amy      staff      206695 Aug 30 16:03 file.1
      user:amy:r-----a-R-c---:-----:allow
      user:roxy: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
```

The following `chmod` operation changes the permissions to a more restrictive mode. In this example, the modified `sysadmin` group's and `staff` group's ACL permissions do not exceed the owning group's permissions.

```
$ 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:roxy: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
```

The following `chmod` operation changes the permissions to a less restrictive mode. In this example, the modified `sysadmin` group's and `staff` group's ACL permissions are restored to allow the same permissions as the owning group.

```
$ 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:roxy: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
```

Applying Special Attributes to ZFS Files

The following examples show how to apply and display special attributes, such as immutability or read-only access, to ZFS files.

For more information about displaying and applying special attributes, see [ls\(1\)](#) and [chmod\(1\)](#).

EXAMPLE 8-15 Apply 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
```

EXAMPLE 8-16 Apply 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
```

EXAMPLE 8-17 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,nosparsed}
# 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,nosparsed}
```

EXAMPLE 8-17 Displaying and Changing ZFS File Attributes *(Continued)*

Some of these attributes only apply 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 231
- “Delegating ZFS Permissions” on page 232
- “Displaying ZFS Delegated Permissions (Examples)” on page 240
- “Delegating ZFS Permissions (Examples)” on page 236
- “Removing ZFS Delegated Permissions (Examples)” on page 241

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 [Part III, “Roles, Rights Profiles, and Privileges,”](#) in *Oracle Solaris Administration: Security Services*.

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 (Subcommand)	Description	Dependencies
<code>allow</code>	The permission to grant permissions that you have to another user.	Must also have the permission that is being allowed.

Permission (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 descendent 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 zfs diff 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 descendent file systems with the zfs receive 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.	
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
- devices
- encryption
- exec
- keysource
- logbias
- mountpoint
- nbmand
- normalization
- primarycache
- quota
- readonly
- recordsize
- refquota
- refreservation
- reservation
- rstchown
- secondarycache
- setuid
- shadow
- sharenfs
- sharesmb
- 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 [“Introducing ZFS Properties” on page 127](#).

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 descendents. 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 descendents, unless the `-d` option is also specified. The `-d` option indicates that the permissions are allowed for the descendent 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 descendents.

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

To remove these permissions, you would use the following syntax:

```
# zfs unallow cindy tank/home/cindy
# zfs allow tank/home/cindy
```

Delegating ZFS Permissions (Examples)

EXAMPLE 9-1 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 tank file system, set the permissions first:

```
# chmod A+user:mark:add_subdirectory:fd:allow /tank/home
```

Then, use the zfs allow command to delegate create, destroy, and mount permissions. For example:

```
# zfs allow mark create,destroy,mount tank/home
```

Now, user mark can create his own file systems in the tank/home file system. For example:

```
# su mark
mark$ zfs create tank/home/mark
mark$ ^D
# su lp
$ zfs create tank/home/lp
cannot create 'tank/home/lp': permission denied
```

EXAMPLE 9-2 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 tank/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 tank/home
# zfs allow -c create,destroy tank/home
# zfs allow tank/home
```

EXAMPLE 9-2 Delegating create and destroy Permissions to a Group *(Continued)*

```

---- Permissions on tank/home -----
Create time permissions:
    create,destroy
Local+Descendent permissions:
    group staff create,mount
# su cindy
cindy% zfs create tank/home/cindy/files
cindy% exit
# su mark
mark% zfs create tank/home/mark/data
mark% exit
cindy% zfs destroy tank/home/mark/data
cannot destroy 'tank/home/mark/data': permission denied

```

EXAMPLE 9-3 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 descendent file systems. User mark is delegated local permission to snapshot the tank/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 tank/home
# zfs allow tank/home
---- Permissions on tank/home -----
Create time permissions:
    create,destroy
Local permissions:
    user mark snapshot
Local+Descendent permissions:
    group staff create,mount
# su mark
mark$ zfs snapshot tank/home@snap1
mark$ zfs snapshot tank/home/mark@snap1
cannot create snapshot 'tank/home/mark@snap1': permission denied

```

To delegate user mark permission at the descendent file system level, use the `zfs allow -d` option. For example:

```

# zfs unallow -l mark snapshot tank/home
# zfs allow -d mark snapshot tank/home
# zfs allow tank/home
---- Permissions on tank/home -----
Create time permissions:
    create,destroy
Descendent permissions:
    user mark snapshot
Local+Descendent permissions:
    group staff create,mount
# su mark
$ zfs snapshot tank/home@snap2
cannot create snapshot 'tank/home@snap2': permission denied
$ zfs snapshot tank/home/mark@snappy

```

EXAMPLE 9-3 Delegating Permissions at the Correct File System Level *(Continued)*

Now, user mark can only create a snapshot below the tank/home file system level.

EXAMPLE 9-4 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 `tank/home` file systems are created.

```
# zfs allow staff create,mount tank/home
# zfs allow -c destroy,snapshot tank/home
# zfs allow tank/home
---- Permissions on tank/home -----
Create time permissions:
    create,destroy,snapshot
Local+Descendent permissions:
    group staff create,mount
```

Because user mark is a member of the `staff` group, he can create file systems in `tank/home`. In addition, user mark can create a snapshot of `tank/home/mark2` because he has specific permissions to do so. For example:

```
# su mark
$ zfs create tank/home/mark2
$ zfs allow tank/home/mark2
---- Permissions on tank/home/mark2 -----
Local permissions:
    user mark create,destroy,snapshot
---- Permissions on tank/home -----
Create time permissions:
    create,destroy,snapshot
Local+Descendent permissions:
    group staff create,mount
```

But, user mark cannot create a snapshot in `tank/home/mark` because he doesn't have specific permissions to do so. For example:

```
$ zfs snapshot tank/home/mark@snap1
cannot create snapshot 'tank/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 /tank/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
```

EXAMPLE 9-4 Defining and Using Complex Delegated Permissions (Continued)

```

$ pwd
/tank/home/mark2/.zfs/snapshot
$ mkdir snap2
$ zfs list
# zfs list -r tank/home
NAME                                USED  AVAIL  REFER  MOUNTPOINT
tank/home/mark                      63K   62.3G   32K    /tank/home/mark
tank/home/mark2                     49K   62.3G   31K    /tank/home/mark2
tank/home/mark2@snap1              18K         -   31K    -
tank/home/mark2@snap2                0         -   31K    -
$ ls
snap1 snap2
$ rmdir snap2
$ ls
snap1

```

EXAMPLE 9-5 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 `tank` file system. User `cindy`, a `staff` group member, has the permission to create a file system in `tank`. However, user `lp` does not have permission to create a file system in `tank`.

```

# zfs allow -s @myset create,destroy,mount,snapshot,promote,clone,readonly tank
# zfs allow tank
---- Permissions on tank -----
Permission sets:
    @myset clone,create,destroy,mount,promote,readonly,snapshot
# zfs allow staff @myset,rename tank
# zfs allow tank
---- Permissions on tank -----
Permission sets:
    @myset clone,create,destroy,mount,promote,readonly,snapshot
Local+Descendent permissions:
    group staff @myset,rename
# chmod A+group:staff:add_subdirectory:fd:allow tank
# su cindy
cindy% zfs create tank/data
cindy% zfs allow tank
---- Permissions on tank -----
Permission sets:
    @myset clone,create,destroy,mount,promote,readonly,snapshot
Local+Descendent permissions:
    group staff @myset,rename
cindy% ls -l /tank
total 15
drwxr-xr-x  2 cindy  staff          2 Jun 24 10:55 data
cindy% exit
# su lp
$ zfs create tank/lp
cannot create 'tank/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 descendent datasets
- Descendent datasets only

EXAMPLE 9-6 Displaying Basic Delegated Administration Permissions

The following output indicates that user `cindy` has `create`, `destroy`, `mount`, `snapshot` permissions on the `tank/cindy` file system.

```
# zfs allow tank/cindy
-----
Local+Descendent permissions on (tank/cindy)
  user cindy create,destroy,mount,snapshot
```

EXAMPLE 9-7 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 descendent file systems.
- User `barney` and the `staff` group are delegated the `@basic` permission set for descendent file systems only.

For the `pool` file system:

- The permission set `@simple` (`create`, `destroy`, `mount`) is defined.

EXAMPLE 9-7 Displaying Complex Delegated Administration Permissions (Continued)

- 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+Descendent 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 `tank/cindy` file system.

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

The following `zfs unallow` syntax removes user `cindy`'s `snapshot` permission from the `tank/home/cindy` file system:

```
# zfs unallow cindy snapshot tank/home/cindy
# zfs allow tank/home/cindy
---- Permissions on tank/home/cindy -----
Local+Descendent permissions:
    user cindy create,destroy,mount
cindy% zfs create tank/home/cindy/data
cindy% zfs snapshot tank/home/cindy@today
cannot create snapshot 'tank/home/cindy@today': permission denied
```

As another example, user `mark` has the following permissions on the `tank/home/mark` file system:

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

The following `zfs unallow` syntax removes all permissions for user mark from the tank/home/mark file system:

```
# zfs unallow mark tank/home/mark
```

The following `zfs unallow` syntax removes a permission set on the tank file system.

```
# zfs allow tank
---- Permissions on tank -----
Permission sets:
    @myset clone,create,destroy,mount,promote,readonly,snapshot
Create time permissions:
    create,destroy,mount
Local+Descendent permissions:
    group staff create,mount
# zfs unallow -s @myset tank
# zfs allow tank
---- Permissions on tank -----
Create time permissions:
    create,destroy,mount
Local+Descendent permissions:
    group staff create,mount
```

Oracle Solaris ZFS Advanced Topics

This chapter describes ZFS volumes, using ZFS on a Solaris system with zones installed, ZFS alternate root pools, and ZFS rights profiles.

The following sections are provided in this chapter:

- “ZFS Volumes” on page 243
- “Using ZFS on a Solaris System With Zones Installed” on page 245
- “Using ZFS Alternate Root Pools” on page 251

ZFS Volumes

A ZFS volume is a dataset that represents a block device. ZFS volumes are identified as devices in the `/dev/zvol/{dsk, rdisk}/pool` directory.

In the following example, a 5-GB ZFS volume, `tank/vol`, is created:

```
# zfs create -V 5gb tank/vol
```

When you create a volume, a reservation is automatically set to the initial size of the volume so that unexpected behavior doesn't occur. For example, if the size of the volume shrinks, data corruption might occur. You must be careful when changing the size of the volume.

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.

For information about file system properties that can be applied to volumes, see [Table 6-1](#).

If you are using a Solaris system with zones installed, you cannot create or clone a ZFS volume in a non-global zone. Any attempt to do so will fail. For information about using ZFS volumes in a global zone, see “[Adding ZFS Volumes to a Non-Global Zone](#)” on page 248.

Using a ZFS Volume as a Swap or Dump Device

During installation of a ZFS root file system or a migration from a UFS root file system, a swap device is created on a ZFS volume in the ZFS root pool. For example:

```
# swap -l
swapfile          dev  swaplo  blocks  free
/dev/zvol/dsk/rpool/swap 253,3      16  8257520  8257520
```

During installation of a ZFS root file system or a migration from a UFS root file system, a dump device is created on a ZFS volume in the ZFS root pool. The dump device requires no administration after it is set up. For example:

```
# 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 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. 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 Your ZFS Swap and Dump Devices”](#) on page 114.

Using a ZFS Volume as an iSCSI LUN

The Common Multiprotocol SCSI Target (COMSTAR) software framework enables you to convert any Oracle Solaris host 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).

First, install the COMSTAR package.

```
# pkg install group/feature/storage-server
```

Next, create a ZFS volume to be used as an iSCSI target and then create the SCSI-block-device-based LUN. For example:

```
# zfs create -V 2g tank/volumes/v2
# sbdadm create-lu /dev/zvol/rdisk/tank/volumes/v2
Created the following LU:

-----
          GUID                DATA SIZE          SOURCE
-----
600144f000144f1dafaa4c0faff20001  2147483648      /dev/zvol/rdisk/tank/volumes/v2
# sbdadm list-lu
Found 1 LU(s)

-----
          GUID                DATA SIZE          SOURCE
-----
600144f000144f1dafaa4c0faff20001  2147483648      /dev/zvol/rdisk/tank/volumes/v2
```

You can expose the LUN views to all clients or selected clients. Identify the LUN GUID and then share the LUN view. In the following example, the LUN view is shared to all clients.

```
# stmfadm list-lu
LU Name: 600144F000144F1DAFAAA4C0FAFF20001
# stmfadm add-view 600144F000144F1DAFAAA4C0FAFF20001
# stmfadm list-view -l 600144F000144F1DAFAAA4C0FAFF20001
View Entry: 0
  Host group   : All
  Target group : All
  LUN          : 0
```

The next step is to create the iSCSI targets. For information about creating the iSCSI targets, see [Chapter 14, “Configuring Storage Devices With COMSTAR,”](#) in *Oracle Solaris Administration: Devices and File Systems*.

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. You will see messages similar to the following:

```
# zfs rename tank/volumes/v2 tank/volumes/v1
cannot rename 'tank/volumes/v2': dataset is busy
# zpool export tank
cannot export 'tank': 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.

Using ZFS on a Solaris System With Zones Installed

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 246
- “Delegating Datasets to a Non-Global Zone” on page 247
- “Adding ZFS Volumes to a Non-Global Zone” on page 248
- “Using ZFS Storage Pools Within a Zone” on page 248
- “Managing ZFS Properties Within a Zone” on page 248

- [“Understanding the zoned Property” on page 249](#)

Keep the following points in mind when associating ZFS datasets with zones:

- You can add a ZFS file system or a clone to a non-global zone with or without delegating administrative control.
- You can add a ZFS volume as a device to non-global zones.
- You cannot associate ZFS snapshots with zones at this time.

In the following sections, a ZFS dataset refers to a file system or a clone.

Adding a dataset allows the non-global zone to share disk space with the global zone, though 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 and should be used when the primary purpose is solely to share common disk space.

ZFS also allows datasets to be delegated to a non-global zone, giving complete control over the dataset and all its children to the zone administrator. 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.

Consider the following when working with ZFS on a system with Oracle Solaris zones installed:

- A ZFS file system that is added to a non-global zone must have its `mountpoint` property set to `legacy`.
- When both a source `zonepath` and a target `zonepath` reside on a ZFS file system and are in the same pool, `zoneadm clone` will now automatically use the ZFS clone to clone a zone. The `zoneadm clone` command will create a ZFS snapshot of the source `zonepath` and set up the target `zonepath`. You cannot use the `zfs clone` command to clone a zone. For more information, see [Part II, “Oracle Solaris Zones,” in *Oracle Solaris Administration: Oracle Solaris Zones, Oracle Solaris 10 Zones, and Resource Management*](#).

Adding ZFS File Systems to a Non-Global Zone

You can add a ZFS file system as a generic file system when the goal is solely to share space with the global zone. A ZFS file system that is added to a non-global zone must have its `mountpoint` property set to `legacy`. For example, if the `tank/zone/zion` file system will be added to a non-global zone, set the `mountpoint` property in the global zone as follows:

```
# zfs set mountpoint=legacy tank/zone/zion
```

You can add a ZFS file system to a non-global zone by using the `zonecfg` command's `add fs` subcommand.

In the following example, a ZFS file system is added to a non-global zone by a global zone administrator from the global zone:

```
# zonecfg -z zion
zonecfg:zion> add fs
zonecfg:zion:fs> set type=zfs
zonecfg:zion:fs> set special=tank/zone/zion
zonecfg:zion:fs> set dir=/export/shared
zonecfg:zion:fs> end
```

This syntax adds the ZFS file system, `tank/zone/zion`, to the already configured `zion` zone, which is mounted at `/export/shared`. The `mountpoint` property of the file system must be set to `legacy`, and the file system cannot already be mounted in another location. The zone administrator can create and destroy files within the file system. The file system cannot be remounted in a different location, nor can the zone administrator 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 [Part II, “Oracle Solaris Zones,”](#) in *Oracle Solaris Administration: Oracle Solaris Zones, Oracle Solaris 10 Zones, and Resource Management*.

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 non-global zone through the use of the `zonecfg` command's `add dataset` subcommand.

In the following example, a ZFS file system is delegated to a non-global zone by a global zone administrator from the global zone.

```
# zonecfg -z zion
zonecfg:zion> add dataset
zonecfg:zion:dataset> set name=tank/zone/zion
zonecfg:zion:dataset> set alias=tank
zonecfg:zion:dataset> end
```

Unlike adding a file system, this syntax causes the ZFS file system `tank/zone/zion` to be visible within the already configured `zion` zone. Within the `zion` zone, this file system is not accessible as `tank/zone/zion`, but as a *virtual pool* named `tank`. 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. If a specific alias is not provided, the default alias in the above example would have been `zion`.

Within delegated datasets, the zone administrator can set file system properties, as well as create descendent 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, it is possible for them to conflict with ZFS volumes that are added as device resources. For more information, see the next section and [dev\(7FS\)](#).

Adding ZFS Volumes to a Non-Global Zone

You can add or create a ZFS volume in a non-global zone or you can add access to a volume's data in a non-global zone in the following ways:

- In a non-global zone, a privileged zone administrator can create a ZFS volume as descendent of a previously delegated file system. For example:

```
# zfs create -V 2g tank/zone/zion/vol1
```

The above syntax means that the zone administrator can manage the volume's properties and data in the non-global zone.

- In a global zone, use the `zonecfg add dataset` subcommand and specify a ZFS volume to be added to a non-global zone. For example:

```
# zonecfg -z zion
zonecfg:zion> add dataset
zonecfg:zion:dataset> set name=tank/volumes/vol1
zonecfg:zion:dataset> end
```

The above syntax means that the zone administrator can manage the volume's properties and data in the non-global zone.

- In a global zone, use the `zonecfg add device` subcommand and specify a ZFS volume whose data can be accessed in a non-global zone. For example:

```
# zonecfg -z zion
zonecfg:zion> add device
zonecfg:zion:device> set match=/dev/zvol/dsk/tank/volumes/vol2
zonecfg:zion:device> end
```

The above syntax means that only the volume data can be accessed in the non-global zone.

Using ZFS Storage Pools Within a Zone

ZFS storage pools cannot be created or modified within a zone. The delegated administration model centralizes control of physical storage devices within the global zone and control of virtual storage to non-global zones. Although a pool-level dataset can be added to a zone, any command that modifies the physical characteristics of the pool, such as creating, adding, or removing devices, is not allowed from within a zone. Even if physical devices are added to a zone by using the `zonecfg` command's `add device` subcommand, or if files are used, the `zpool` command does not allow the creation of any new pools within the 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 descendents. For example, consider the following configuration:


```
global# zfs list -Ho name
tank
tank/home
tank/data
tank/data/matrix
tank/data/zion
tank/data/zion/home
```

If `tank/data/zion` were added to a zone with the default `zion` alias, each dataset would have the following properties.

Dataset	Visible	Writable	Immutable Properties
<code>tank</code>	No	-	-
<code>tank/home</code>	No	-	-
<code>tank/data</code>	No	-	-
<code>tank/data/zion</code>	Yes	Yes	zoned, quota, reservation
<code>tank/data/zion/home</code>	Yes	Yes	zoned

Note that every parent of `tank/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 non-global zone.

In addition, the `sharenfs` and `mountpoint` properties cannot be changed by the global zone administrator after a dataset has been delegated to a non-global zone.

Understanding the zoned Property

When a dataset is delegated to a non-global zone, the dataset must be specially marked so that certain properties are not interpreted within the context of the global zone. After a dataset has been delegated to a non-global zone and is under the control of a zone administrator, its contents can no longer be trusted. As with any file system, there might be `setuid` binaries, symbolic links, or otherwise questionable contents that might adversely affect the security of the global zone. In addition, the `mountpoint` property cannot be interpreted in the context of the global zone. Otherwise, the zone administrator could affect the global zone's namespace. To address the latter, ZFS uses the `zoned` property to indicate that a dataset has been delegated to a non-global 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 turn on this property. If the `zoned` property is set, the dataset cannot be mounted or shared in the global zone. In the following example, `tank/zone/zion` has been delegated to a zone, while `tank/zone/global` has not:

```
# zfs list -o name,zoned,mountpoint -r tank/zone
NAME                ZONED  MOUNTPOINT
tank/zone/global    off    /tank/zone/global
tank/zone/zion      on     /tank/zone/zion
# zfs mount
tank/zone/global    /tank/zone/global
tank/zone/zion      /export/zone/zion/root/tank/zone/zion
```

Note the difference between the `mountpoint` property and the directory where the `tank/zone/zion` dataset is currently mounted. The `mountpoint` property reflects the property as it is stored on disk, not where the dataset is currently mounted on the system.

When a dataset is removed from a zone or a zone is destroyed, the `zoned` property is *not* automatically cleared. This behavior is due to the inherent security risks associated with these tasks. Because an untrusted user has had complete access to the dataset and its descendents, 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 descendents 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 non-global zone.

Copying Zones to Other Systems

When you need to migrate one or more zones needs to another system, consider using the `zfs send` and `zfs receive` commands. Depending on the scenario, it may be best to use a replication streams or recursive streams.

The examples in this section describe how to copy zone data between systems. Additional steps are required to transfer each zone's configuration and attach each zone to the new system. For more information, see [Part II, “Oracle Solaris Zones,” in *Oracle Solaris Administration: Oracle Solaris Zones, Oracle Solaris 10 Zones, and Resource Management*](#).

If all zones on one system need to move to another 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 tank/zones file system on sys. 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 tank
```

In the following example, one of several zones is copied from sysC to the sysD. Assume that the ssh command is not available but an NFS server instance is available. The following commands might be used to generate a recursive zfs send stream without worrying about whether the zone is a clone of another zone.

```
sysC# zfs snapshot -r rpool/zones/zone1@send-to-nfs
sysC# zfs send -rc rpool/zones/zone1@send-to-nfs > /net/nfssrv/export/scratch/zone1.zfs
sysD# zfs create tank/zones
sysD# zfs receive -d tank/zones < /net/nfssrv/export/scratch/zone1.zfs
```

Using ZFS Alternate Root Pools

When a pool is created, it 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* pool feature. An alternate root pool does not persist across system reboots, and all mount points are modified to be relative to the root of the pool.

Creating ZFS Alternate Root Pools

The most common reason for creating an alternate root pool 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 an alternate root 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 path:

```
# 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 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 pool 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 Alternate Root Pools

Pools can also be imported using an alternate root. This feature allows for recovery situations, where the mount points should not be interpreted in context of the current root, 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 path. 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
```

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 Failures” on page 253
- “Checking ZFS File System Integrity” on page 255
- “Resolving Problems With ZFS” on page 257
- “Repairing a Damaged ZFS Configuration” on page 262
- “Resolving a Missing Device” on page 262
- “Replacing or Repairing a Damaged Device” on page 265
- “Repairing Damaged Data” on page 274
- “Repairing an Unbootable System” on page 278

Identifying ZFS Failures

As a combined file system and volume manager, ZFS can exhibit many different failures. This chapter begins by outlining the various failures, then discusses how to identify them on a running system. This chapter concludes by discussing how to repair the problems. ZFS can encounter three basic types of errors:

- “Missing Devices in a ZFS Storage Pool” on page 254
- “Damaged Devices in a ZFS Storage Pool” on page 254
- “Corrupted ZFS Data” on page 254

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.

Missing Devices in a ZFS Storage Pool

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 `FAULTED`, which means no data is accessible until the device is reattached, under the following conditions:

- If all components of a mirror are removed
- If more than one device in a RAID-Z (`raidz1`) device is removed
- If top-level device is removed in a single-disk configuration

Damaged Devices in a ZFS Storage Pool

The term “damaged” covers a wide variety of possible errors. Examples include the following:

- 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.

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.

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 277](#).

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 tank
```

The status of the current scrubbing operation can be displayed by using the `zpool status` command. For example:

```
# zpool status -v tank
pool: tank
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
tank	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 tank
```

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.

For more information about interpreting `zpool status` output, see [“Querying ZFS Storage Pool Status” on page 80](#).

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 272](#).

Resolving Problems With ZFS

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 258](#)
- [“Reviewing zpool status Output” on page 258](#)
- [“System Reporting of ZFS Error Messages” on page 261](#)

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.

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 tank
History for 'tank':
2010-07-15.12:06:50 zpool create tank mirror c0t1d0 c0t2d0 c0t3d0
```

```
2010-07-15.12:06:58 zfs create tank/eric
2010-07-15.12:07:01 zfs set checksum=off tank/eric
```

In this output, note that checksums are disabled for the tank/eric file system. This configuration is not recommended.

- Identify the errors through the `fmt` 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 faulted 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 80](#).

Reviewing zpool status Output

The complete `zpool status` output looks similar to the following:

```
# zpool status tank
# zpool status tank
  pool: tank
  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: http://www.sun.com/msg/ZFS-8000-2Q
scrub: none requested
config:

```

NAME	STATE	READ	WRITE	CKSUM	
tank	DEGRADED	0	0	0	
mirror-0	DEGRADED	0	0	0	
c1t0d0	ONLINE	0	0	0	
c1t1d0	UNAVAIL	0	0	0	cannot open

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

This output is described next:

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:

<code>pool</code>	Identifies the name of the pool.
<code>state</code>	Indicates the current health of the pool. This information refers only to the ability of the pool to provide the necessary replication level.
<code>status</code>	Describes what is wrong with the pool. This field is omitted if no errors are found.
<code>action</code>	A recommended action for repairing the errors. This field is omitted if no errors are found.
<code>see</code>	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.
<code>scrub</code>	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.
<code>errors</code>	Identifies known data errors or the absence of known data errors.

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`, `UNAVAIL`, or `OFFLINE`. 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 265](#).

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 `FAULTED`, 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 272](#).

Scrubbing Status

The scrub section of the `zpool status` output describes the current status of any explicit 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 Mon Jun  7 08:56:04 2010
      1.90G scanned out of 16.2G at 9.33M/s, 0h26m to go
      0 repaired, 11.69% done
```

- Scrub completion message. For example:

```
scrub repaired 0 in 0h12m with 0 errors on Mon Jun  7 09:08:48 2010
```

- Ongoing scrub cancellation message. For example:

```
scan: scrub canceled on Thu Jun  3 09:39:39 2010
```

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 255](#).

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
pool: tank
state: UNAVAIL
status: One or more devices are faulted in response to IO failures.
action: Make sure the affected devices are connected, then run 'zpool clear'.
       see: http://www.sun.com/msg/ZFS-8000-HC
       scrub: scrub completed after 0h0m with 0 errors on Tue Feb  2 13:08:42 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM	
tank	UNAVAIL	0	0	0	insufficient replicas
c1t0d0	ONLINE	0	0	0	
c1t1d0	UNAVAIL	4	1	0	cannot open

```
errors: Permanent errors have been detected in the following files:
```

```
/tank/data/aaa
/tank/data/bbb
/tank/data/ccc
```

A similar message is also displayed by `cmd` on the system console and the `/var/adm/messages` file. These messages can also be tracked by using the `cmdump` command.

For more information about interpreting data corruption errors, see [“Identifying the Type of Data Corruption” on page 274](#).

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.

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 92](#).

Resolving a Missing 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.

For example, you might see a message similar to the following from `fmdd` after a device failure:

```
SUNW-MSG-ID: ZFS-8000-FD, TYPE: Fault, VER: 1, SEVERITY: Major
EVENT-TIME: Thu Jun 24 10:42:36 PDT 2010
```

```

PLATFORM: SUNW,Sun-Fire-T200, CSN: -, HOSTNAME: daleks
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: a1fb66d0-cc51-cd14-a835-961c15696fcb
DESC: The number of I/O errors associated with a ZFS device exceeded
acceptable levels. Refer to http://sun.com/msg/ZFS-8000-FD for more information.
AUTO-RESPONSE: The device has been offlined and marked as faulted. An attempt
will be made to activate a hot spare if available.
IMPACT: Fault tolerance of the pool may be compromised.
REC-ACTION: Run 'zpool status -x' and replace the bad device.

```

To view more detailed information about the device problem and the resolution, use the `zpool status -x` command. For example:

```

# zpool status -x
pool: tank
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: http://www.sun.com/msg/ZFS-8000-2Q
scan: scrub repaired 0 in 0h00m with 0 errors on Tue Sep 27 16:59:07 2011
config:

```

NAME	STATE	READ	WRITE	CKSUM	
tank	DEGRADED	0	0	0	
mirror-0	DEGRADED	0	0	0	
c2t2d0	ONLINE	0	0	0	
c2t1d0	UNAVAIL	0	0	0	cannot open

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

You can see from this output that the missing `c2t1d0` device is not functioning. If you determine that the device is faulty, replace it.

If necessary, use the `zpool online` command to bring online the replaced device. For example:

```
# zpool online tank c2t1d0
```

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
-----
Sep 27 16:58:50    e6bb52c3-5fe0-41a1-9ccc-c2f8a6b56100    ZFS-8000-D3          Major

```

```

Host       : t2k-brm-10
Platform   : SUNW,Sun-Fire-T200           Chassis_id  :
Product_sn :

```

```

Fault class : fault.fs.zfs.device
Affects     : zfs://pool=tank/vdev=c75a8336cda03110
              faulted and taken out of service
Problem in  : zfs://pool=tank/vdev=c75a8336cda03110

```

faulted and taken out of service

Description : A ZFS device failed. Refer to <http://sun.com/msg/ZFS-8000-D3> for more information.

Response : No automated response will occur.

Impact : Fault tolerance of the pool may be compromised.

Action : Run 'zpool status -x' and replace the bad device.

```
# fmadm repair zfs://pool=tank/vdev=c75a8336cda03110
```

As a last step, confirm that the pool with the replaced device is healthy. For example:

```
# zpool status -x tank  
pool 'tank' is healthy
```

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 265.

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 faulted, 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 tank c0t1d0
```

For more information about bringing devices online, see [“Bringing a Device Online”](#) on page 69.

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 tank
pool: tank
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://www.sun.com/msg/ZFS-8000-8A
scan: scrub in progress since Tue Sep 27 17:12:40 2011
63.9M scanned out of 528M at 10.7M/s, 0h0m to go
0 repaired, 12.11% done
config:
```

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

```

tank          ONLINE      2      0      0
mirror-0     ONLINE      2      0      0
c2t2d0      ONLINE      2      0      0
c2t1d0      ONLINE      2      0      0

```

errors: Permanent errors have been detected in the following files:

```
/tank/words
```

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 Damaged Data” on page 274](#).

Clearing Transient 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 tank 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 tank
```

For more information about clearing pool errors, see [“Clearing Storage Pool Device Errors” on page 70](#).

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 267](#)
- [“Devices That Cannot be Replaced” on page 268](#)
- [“Replacing a Device in a ZFS Storage Pool” on page 268](#)
- [“Viewing Resilvering Status” on page 272](#)

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 faulted, 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 faulted, 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 `FAULTED` 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          FAULTED
```

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 faulted disk can be replaced. The `ONLINE` disks cannot be replaced either because the pool itself is faulted.

```
raidz          FAULTED
c1t0d0          ONLINE
c2t0d0          FAULTED
c3t0d0          FAULTED
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 faulted, then no replacement can be performed because the pool itself is faulted.

Devices That Cannot be Replaced

If the loss of a device causes the pool to become faulted 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 277](#).

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 tank 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 tank 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 70](#).

EXAMPLE 11-1 Replacing a Device in a ZFS Storage Pool

The following example shows how to replace a device (`c1t3d0`) in a mirrored storage pool tank 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. The basic steps follow:

EXAMPLE 11-1 Replacing a Device in a ZFS Storage Pool (Continued)

- Take offline the disk (c1t3d0) to be replaced. You cannot unconfigure a disk that is currently being used.
- Use the `cfgadm` command to identify the 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 faulted drive.
- Reconfigure the 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 tank c2t4d0
```

- If FMA is reporting the failed device, then you should clear the device failure.

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

The following example walks through the steps to replace a disk in a ZFS storage pool.

```
# zpool offline tank c1t3d0
# cfgadm | grep c1t3d0
sata1/3::dsk/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::dsk/c1t3d0          disk          connected    configured   ok
# zpool online tank c1t3d0
# zpool replace tank c1t3d0
# zpool status tank
```

EXAMPLE 11-1 Replacing a Device in a ZFS Storage Pool (Continued)

```
pool: tank
state: ONLINE
scrub: resilver completed after 0h0m with 0 errors on Tue Feb  2 13:17:32 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM
tank	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 tank c1t3d0 c4t3d0
# zpool status
pool: tank
state: DEGRADED
scrub: resilver completed after 0h0m with 0 errors on Tue Feb  2 13:35:41 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM
tank	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
```

EXAMPLE 11-1 Replacing a Device in a ZFS Storage Pool (Continued)

You might need to run the `zpool status` command several times until the disk replacement is completed.

```
# zpool status tank
pool: tank
state: ONLINE
scrub: resilver completed after 0h0m with 0 errors on Tue Feb  2 13:35:41 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM
tank	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 11-2 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 (`pool`). The basic steps follow:

- Review the `zpool status -x` output and FMA diagnostic message, described here: <https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&doctype=REFERENCE&alias=EVENT:ZFS-8000-K4>
- 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: pool
state: FAULTED
status: One or more of the intent logs could not be read.
       Waiting for administrator intervention to fix the faulted pool.
```

EXAMPLE 11-2 Replacing a Failed Log Device (Continued)

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
pool	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 pool c0t5d0
# zpool clear pool
# 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 pool c0t5d0
```

- Reset the failed log device error condition.

```
# zpool clear pool
```

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 out of 16.2G at 18.5M/s, 0h2m to go
      13.3G resilvered, 82.34% done
```

- 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.
- Resilvering is interruptible and safe. 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 tank
pool: tank
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 scanned out of 222M at 5.46M/s, 0h0m to go
      54.5M resilvered, 24.64% done
config:

        NAME                STATE          READ WRITE CKSUM
        tank                 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 tank
pool: tank
```

```
state: ONLINE
scrub: resilver completed after 0h1m with 0 errors on Tue Feb  2 13:54:30 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM	
tank	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.

Repairing Damaged Data

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 274](#)
- [“Repairing a Corrupted File or Directory” on page 275](#)
- [“Repairing ZFS Storage Pool-Wide Damage” on page 277](#)

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 255](#).

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 monkey
pool: monkey
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://www.sun.com/msg/ZFS-8000-8A
scrub: scrub completed after 0h0m with 8 errors on Tue Jul 13 13:17:32 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM
monkey	ONLINE	8	0	0
c1t1d0	ONLINE	2	0	0
c2t5d0	ONLINE	6	0	0

errors: 8 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: 1422736890544688191
state: FAULTED
status: The pool metadata is corrupted.
action: The pool cannot be imported due to damaged devices or data.
see: http://www.sun.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 -v
pool: monkey
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://www.sun.com/msg/ZFS-8000-8A
scrub: scrub completed after 0h0m with 8 errors on Tue Jul 13 13:17:32 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM
monkey	ONLINE	8	0	0
c1t1d0	ONLINE	2	0	0
c2t5d0	ONLINE	6	0	0

errors: Permanent errors have been detected in the following files:

```
/monkey/a.txt
/monkey/bananas/b.txt
/monkey/sub/dir/d.txt
monkey/ghost/e.txt
/monkey/ghost/boo/f.txt
```

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:


```
/monkey/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:


```
monkey/ghost/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:


```
monkey/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.

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 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: tpool
state: FAULTED
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 Wed Jul 14 11:44:10 2010
        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://www.sun.com/msg/ZFS-8000-72
        scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM	
tpool	FAULTED	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 tpool
```

If you attempt to import a damaged storage pool, you will see messages similar to the following:

```
# zpool import tpool
cannot import 'tpool': I/O error
Recovery is possible, but will result in some data loss.
Returning the pool to its state as of Wed Jul 14 11:44:10 2010
should correct the problem. Approximately 5 seconds of data
must be discarded, irreversibly. Recovery can be attempted
by executing 'zpool import -F tpool'. 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 tpool
Pool tpool returned to its state as of Wed Jul 14 11:44:10 2010.
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 tpool
```

For more information about importing a pool read-only, see [“Importing a Pool in Read-Only Mode” on page 98](#).

- 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 96](#).
- 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 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.

Archiving Snapshots and Root Pool Recovery

This chapter describes how to archive snapshots that can be used to migrate or restore an Oracle Solaris 11 system in the event of a system failure. You can use these steps to create the core of basic disaster recovery plan, or the steps might be used to migrate a system's configuration to a new boot device.

The following sections are provided in this chapter:

- [“Overview of ZFS Recovery Process” on page 281](#)
- [“Creating a ZFS Snapshot Archive for Recovery” on page 282](#)
- [“Recreating Your Root Pool and Recovering Root Pool Snapshots” on page 284](#)

Overview of ZFS Recovery Process

At a minimum, all file system data should be backed up on a regular basis to reduce down time due to system failures. If a catastrophic system failure occurs, you could restore your ZFS root pool snapshots rather than reinstall the OS and recreate your system configuration. Then, restore any non-root pool data.

Any system that runs Oracle Solaris 11 is a candidate for backup and archival. The overall process involves the following steps:

- Create a ZFS snapshot archive for the root pool file systems and for any non-root pools that need to be migrated or recovered.
You should re-archive your root pool snapshots after the OS is updated.
- Save the snapshot archive on local removable media, such as a USB drive, or send the snapshots to a remote system for potential retrieval.
- Failed disk or other system components are replaced.
- The target system is booted from Oracle Solaris 11 installation media, new storage pools are created and the file systems are recovered.

- Perform minimal boot configuration, and then the system is usable, offering all of the services that were running at the time of archival.

ZFS Pool Recovery Requirements

- The archived system and recovery system must be the same architecture and must meet the Oracle Solaris 11 minimum requirements for supported platforms.
- The replacement disks that will contain the new ZFS storage pool must be at least as large in capacity as the data used in the archived pools (see below).
- Root access is required on both system that contains the archived snapshots and the recovery system. If you are using `ssh` to access the remote system, you will need to configure it for privileged access.

Creating a ZFS Snapshot Archive for Recovery

Before you create the ZFS root pool snapshot, consider saving the following information:

- Capture the root pool properties.

```
sysA# zpool get all rpool
```

- Identify the size and current capacity of the root pool disk.

```
sysA# zpool list
NAME  SIZE  ALLOC  FREE  CAP  DEDUP  HEALTH  ALROOT
rpool  74G  5.42G  68.6G  7%  1.00x  ONLINE  -
```

- Identify the root pool components.

```
sysA# zfs list -r rpool
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rpool                                5.48G  67.4G  75.5K  /rpool
rpool/ROOT                           3.44G  67.4G   31K  legacy
rpool/ROOT/solaris                    3.44G  67.4G  3.14G  /
rpool/ROOT/solaris/var                 303M  67.4G  214M  /var
rpool/dump                             1.01G  67.4G 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  67.4G  1.00G  -
```

▼ How to Create a ZFS Snapshot Archive

The following steps describe how to create a recursive snapshot of the root pool that will include all file systems in the root pool. Other non-root pools can be archived in this same way.

Consider the following points:

- For complete system recovery, send the snapshots to a pool on a remote system.

- Create an NFS share from the remote system and also configure ssh to allow privileged access, if necessary.
- The recursive root pool snapshot is sent as one large snapshot file to a remote system, but you could send the recursive snapshots to be stored as individual snapshots on a remote system.

In the steps that follow, the recursive snapshot is named `rpool@snap1`. The local system to be recovered is `sysA` and the remote system is `sysB`. Note that `rpool` is the default root pool name and might be different on your system.

1 Become an administrator.

2 Create a recursive snapshot of the root pool.

```
sysA# zfs snapshot -r rpool@rpool.snap1
```

3 Reduce the snapshot archive by removing the swap and dump snapshots, if desired.

```
sysA# zfs destroy rpool/dump@rpool.snap1
sysA# zfs destroy rpool/swap@rpool.snap1
```

The swap volume does not contain data that is relevant to a system migration or recovery. Do not remove the dump volume snapshot if you wish to preserve any crash dumps.

4 Send the recursive root pool snapshot to another pool on another system.

a. Share a file system on a remote system for receiving the snapshot or snapshots:

In the following steps, the `/tank/snaps` file system is shared for storing the recursive root snapshot.

```
sysB# zfs set share=name=snapf,path=/tank/snaps,prot=nfs,root=sysA tank/snaps
sysB# zfs set sharenfs=on tank/snaps
```

b. Send the recursive root pool snapshot to a remote system.

Send the recursive snapshot to the remote file system that was shared in the previous step.

```
sysA# zfs send -Rv rpool@rpool.snap1 | gzip > /net/sysB/tank/snaps/
rpool.snap1.gz
sending from @ to rpool@rpool.snap1
sending from @ to rpool/export@rpool.snap1
sending from @ to rpool/export/home@rpool.snap1
sending from @ to rpool/export/home/admin@rpool.snap1
sending from @ to rpool/ROOT@rpool.snap1
sending from @ to rpool/ROOT/solaris@install
sending from @ to rpool/ROOT/solaris@install
sending from @install to rpool/ROOT/solaris@rpool.snap1
sending from @ to rpool/ROOT/solaris/var@install
sending from @install to rpool/ROOT/solaris/var@rpool.snap1
```

Recreating Your Root Pool and Recovering Root Pool Snapshots

If you need to recreate your root pool and recover your root pool snapshots, the general steps are as follows:

- Prepare replacement root pool disk or disks and recreate the root pool
- Restore the root pool file system snapshots
- Select and activate the desired boot environment
- Boot the system

▼ How to Recreate the Root Pool on the Recovery System

Review the following considerations when recovering your root pool.

- If a non-redundant root pool disk fails, you will need to boot the system from installation media or an install server to either re-install the OS or restore your root pool snapshots that you archived previously.

For information about replacing a disk on your system, see your hardware documentation.

- If a mirrored root pool disk fails, then you can replace the failed disk while the system is still operational. For information on replacing a failed disk in a mirrored root pool, see [“How to Replace a Disk in a ZFS Root Pool” on page 110](#).

1 Identify and replace the failed root pool disk or system component.

This disk is generally the default boot device or you can select another disk and then reset the default boot device.

2 Boot the system from the Oracle Solaris 11 installation media by selecting one of the following.

- DVD or USB install media (SPARC or x86) – Insert the media and select the appropriate device as the boot device.
If text-based media is used, select the Shell option from the text installer menu.
- Live media (x86 only) – The GNOME desktop session can be used during the recovery procedure.
- Automated installer or a local copy of AI media (SPARC or x86) – From the text installer menu, select the shell option. On a SPARC system, boot the AI media (either locally or over the network), and select the Shell option:

```
ok boot net:dhcp
.
.
>Welcome to the Oracle Solaris 11 installation menu
```

```
1 Install Oracle Solaris
```

- 2 Install Additional Drivers
- 3 Shell
- 4 Terminal type (currently xterm)
- 5 Reboot

Please enter a number [1]: 3

3 Prepare the root pool disk.

a. Confirm that the replacement root pool disk is visible from the format utility.

```
# format
Searching for disks...done
AVAILABLE DISK SELECTIONS:
  0. c2t0d0 <FUJITSU-MAY2073RCSUN72G-0401 cyl 14087 alt 2 hd 24 sec 424>
     /pci@780/pci@0/pci@9/scsi@0/sd@0,0
  1. c2t1d0 <FUJITSU-MAY2073RCSUN72G-0401 cyl 14087 alt 2 hd 24 sec 424>
     /pci@780/pci@0/pci@9/scsi@0/sd@1,0
  2. c2t2d0 <SEAGATE-ST973402SSUN72G-0400-68.37GB>
     /pci@780/pci@0/pci@9/scsi@0/sd@2,0
  3. c2t3d0 <SEAGATE-ST973401LSUN72G-0556-68.37GB>
     /pci@780/pci@0/pci@9/scsi@0/sd@3,0
Specify disk (enter its number): 0
```

b. Confirm that the root pool disk has an SMI (VTOC) label and a slice 0 with the bulk of the disk space.

Review the partition table to confirm that the root pool disk has an SMI label and a slice 0.

```
selecting c2t0d0
[disk formatted]
format> partition
partition> print
```

c. Relabel the disk with an SMI (VTOC) label, if necessary.

Use the following shortcut commands to relabel the disk. A caution is that these commands do not provide any error checking so make sure that you are relabeling the correct disk.

- SPARC:

```
sysA# format -L vtoc -d c2t0d0
```

Confirm that slice 0 has disk space allocated appropriately. The default partition is applied in the above command, which might be too small for the root pool slice 0. For information about modifying the default partition table, see [“How to Create a Disk Slice for a ZFS Root File System”](#) in *Oracle Solaris Administration: Devices and File Systems*.

- x86:

```
sysA# fdisk -B /dev/rdisk/c2t0d0p0
sysA# format -L vtoc -d c2t0d0
```

Confirm that slice 0 has disk space allocated appropriately. The default partition is applied in the above command, which might be too small for the root pool slice 0. For information about modifying the default partition table, see [“How to Create a Disk Slice for a ZFS Root File System”](#) in *Oracle Solaris Administration: Devices and File Systems*.

4 Recreate the root pool.

```
sysA# zpool create rpool c2t0d0s0
```

5 Mount the file system that contains the snapshots from the remote system.

```
sysA# mount -F nfs sysB:/tank/snaps /mnt
```

6 Restore the root pool snapshots.

```
sysA# gzcat /mnt/rpool.snap1.gz | zfs receive -Fv rpool
receiving full stream of rpool@rpool.snap1 into rpool@rpool.snap1
received 92.7KB stream in 1 seconds (92.7KB/sec)
receiving full stream of rpool/export@rpool.snap1 into rpool/export@rpool.snap1
received 47.9KB stream in 1 seconds (47.9KB/sec)
.
.
.
```

7 Set the bootfs property.

```
sysA# zpool set bootfs=rpool/ROOT/solaris rpool
```

8 Recreate swap and dump devices, if necessary.

For example:

```
sysA# zfs create -V 4G rpool/swap
sysA# zfs create -V 4G rpool/dump
```

For information about sizing swap and dump volumes, see “[Planning for Swap Space](#)” in *Oracle Solaris Administration: Devices and File Systems*.

9 Mount the BE.

The next step requires that the BE is mounted so that the boot blocks can be installed.

```
sysA# beadm mount solaris /tmp/mnt
```

10 Install the boot blocks on the new disk.

- SPARC:

```
sysA# installboot /tmp/mnt/usr/platform/'uname -i'/lib/fs/zfs/bootblk /dev/rdisk/c2t0d0s0
```

- x86:

```
sysA# installgrub /tmp/mnt/boot/grub/stage1 /tmp/mnt/boot/grub/stage2
/dev/rdisk/c2t0d0s0
```

11 If the same devices will not be used or the devices will be configured in a different way on the original system, clear the existing device information. Then, direct the system to reconfigure the new device information.

```
# devfsadm -Cn -r /tmp/mnt
# touch /tmp/mnt/reconfigure
```

12 Unmount the BE.

```
#beadm unmount solaris
```

13 Activate the boot environment, if necessary.

For example:

```
sysA# beadm list
BE      Active Mountpoint Space  Policy Created
--      -
solaris-1 -      -      13.26M static 2011-09-28 15:23
solaris  -      -      3.87G  static 2011-09-29 08:20
# beadm activate solaris
```

14 Verify that you can boot successfully from the replacement root pool disk.

If necessary, reset the default boot device:

- SPARC: Set up the system to boot automatically from the new disk, either by using the `eprom` command or the `setenv` command from the boot PROM.
- x86: Reconfigure the system BIOS.

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 289
- “Recommended File System Practices” on page 294

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 11, “Oracle Solaris ZFS Troubleshooting and Pool Recovery.”](#)

General System Practices

- Keep system up-to-date with latest Solaris releases and patches
- 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			

- 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 RAIDZ, RAIDZ-2, RAIDZ-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.
- 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 tank c4t1d0 c4t3d0
'tank' successfully created, but with no redundancy; failure
of one device will cause loss of the pool
```

- For mirrored pools, use mirrored disk pairs
- For RAIDZ pools, group 3-9 disks per VDEV
- 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

Root Pool Creation Practices

- 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:

```
# zpool create rpool c0t1d0s0
```

Or, create a mirrored root pool. For example:

```
# zpool create rpool mirror c0t1d0s0 c0t2d0s0
```

- The root pool must be created as a mirrored configuration or as a single-disk configuration. A RAID-Z or a striped configuration is not 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.

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 tank 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 c7t2d
```

- 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 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, see the following white paper:

http://blogs.oracle.com/storage/entry/new_white_paper_configuring_oracle

Storage Pool Practices for Performance

- Keep pool capacity below 80% for best performance
- 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 80% 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 80% pool capacity.
- Monitor pool health
 - Redundant pools, monitor pool with `zpool status` and `fmddump` on a weekly basis
 - Non-redundant pools, monitor pool with `zpool status` and `fmddump` on a biweekly basis
- 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 biweekly 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:

```
----- Original Message -----
Subject: Fault Management Event: tardis:SMF-8000-YX
Date: Wed, 21 Sep 2011 11:11:27 GMT
From: No Access User <noaccess@tardis.drwho.COM>
Reply-To: root@tardis.drwho.COM
To: root@tardis.drwho.COM

SUNW-MSG-ID: ZFS-8000-D3, TYPE: Fault, VER: 1, SEVERITY: Major
EVENT-TIME: Wed Sep 21 11:11:27 GMT 2011
PLATFORM: Sun-Fire-X4140, CSN: 0904QAD02C, HOSTNAME: tardis
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: d9e3469f-8d84-4a03-b8a3-d0beb178c017
DESC: A ZFS device failed. Refer to http://sun.com/msg/ZFS-8000-D3
for more information.
AUTO-RESPONSE: No automated response will occur.
IMPACT: Fault tolerance of the pool may be compromised.
REC-ACTION: Run 'zpool status -x' and replace the bad device.
```

- 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.

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 descendent 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

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 descendent file systems or snapshots.
- 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

Oracle Solaris ZFS Version Descriptions

This appendix describes available ZFS versions, features of each version, and the Solaris OS that provides the ZFS version and feature.

The following sections are provided in this appendix:

- [“Overview of ZFS Versions” on page 297](#)
- [“ZFS Pool Versions” on page 297](#)
- [“ZFS File System Versions” on page 299](#)

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 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 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 181](#) and [“Upgrading ZFS Storage Pools” on page 100](#).

ZFS Pool Versions

The following table provides a list of ZFS pool versions that are available in the Oracle Solaris release.

Version	Oracle Solaris 11	Description
1	snv_36	Initial ZFS version
2	snv_38	Ditto blocks (replicated metadata)

Version	Oracle Solaris 11	Description
3	snv_42	Hot spares and double parity RAID-Z
4	snv_62	zpool history
5	snv_62	gzip compression algorithm
6	snv_62	bootfs pool property
7	snv_68	Separate intent log devices
8	snv_69	Delegated administration
9	snv_77	refquota and refreservation properties
10	snv_78	Cache devices
11	snv_94	Improved scrub performance
12	snv_96	Snapshot properties
13	snv_98	snapped property
14	snv_103	aclinherit passthrough-x property
15	snv_114	user and group space accounting
16	snv_116	stmf property
17	snv_120	Triple-parity RAID-Z
18	snv_121	Snapshot user holds
19	snv_125	Log device removal
20	snv_128	zle (zero-length encoding) compression algorithm
21	snv_128	Deduplication
22	snv_128	Received properties
23	snv_135	Slim ZIL
24	snv_137	System attributes
25	snv_140	Improved scrub stats
26	snv_141	Improved snapshot deletion performance
27	snv_145	Improved snapshot creation performance
28	snv_147	Multiple vdev replacements
29	snv_148	RAID-Z/mirror hybrid allocator
30	snv_149	Encryption

Version	Oracle Solaris 11	Description
31	snv_150	Improved 'zfs list' performance
32	snv_151	One MB blocksize
33	snv_163	Improved share support

ZFS File System Versions

The following table lists the ZFS file system versions that are available in the Oracle Solaris release.

Version	Oracle Solaris 11	Description
1	snv_36	Initial ZFS file system version
2	snv_69	Enhanced directory entries
3	snv_77	Case insensitivity and file system unique identifier (FUID)
4	snv_114	userquota and groupquota properties
5	snv_137	System attributes

Index

A

accessing

- ZFS snapshot
(example of), 187

ACL model, Solaris, differences between ZFS and traditional file systems, 40

ACL property mode

- `aclinherit`, 127
- `aclmode`, 128

`aclinherit` property, 209

ACLs

- access privileges, 206
- ACL inheritance, 208
- ACL inheritance flags, 208
- ACL on ZFS directory
 - detailed description, 212
- ACL on ZFS file
 - detailed description, 211
- ACL property, 209
- `aclinherit` property, 209
- description, 203
- differences from POSIX-draft ACLs, 204
- entry types, 206
- format description, 204
- modifying trivial ACL on ZFS file (verbose mode)
(example of), 213
- restoring trivial ACL on ZFS file (verbose mode)
(example of), 216
- setting ACL inheritance on ZFS file (verbose mode)
(example of), 217
- setting ACLs on ZFS file (compact mode)
(example of), 224

ACLs, setting ACLs on ZFS file (compact mode)

(*Continued*)

- description, 223
- setting ACLs on ZFS file (verbose mode)
description, 212
- setting on ZFS files
description, 210

adding

- cache devices (example of), 62
- devices to a ZFS storage pool (`zpool add`)
(example of), 59
- disks to a RAID-Z configuration (example of), 60
- mirrored log device (example of), 61
- ZFS file system to a non-global zone
(example of), 246
- ZFS volume to a non-global zone
(example of), 248

adjusting, sizes of swap and dump devices, 114

`allocated` property, description, 78

alternate root pools

- creating
(example of), 251
- description, 251
- importing
(example of), 252

`altroot` property, description, 78

`atime` property, description, 128

attaching

- devices to ZFS storage pool (`zpool attach`)
(example of), 63

`autoreplace` property, description, 78

`available` property, description, 128

B

- bootblocks, installing with `installboot` and `installgrub`, 116
- bootfs property, description, 78
- booting
 - a ZFS BE with `boot -L` and `boot -Z` on SPARC systems, 118
 - root file system, 116

C

- cache devices
 - considerations for using, 52
 - creating a ZFS storage pool with (example of), 52
- cache devices, adding, (example of), 62
- cache devices, removing, (example of), 62
- cache file property, description, 79
- canmount property
 - description, 128
 - detailed description, 141
- capacity property, description, 79
- casesensitivity property, description, 129
- checking, ZFS data integrity, 255
- checksum, definition, 26
- checksum property, description, 129
- checksummed data, description, 25
- clearing
 - a device in a ZFS storage pool (`zpool clear`)
 - description, 70
 - device errors (`zpool clear`)
 - (example of), 266
- clearing a device
 - ZFS storage pool
 - (example of), 70
- clone, definition, 26
- clones
 - creating (example of), 191
 - destroying (example of), 191
 - features, 190
- components of, ZFS storage pool, 41
- components of ZFS, naming requirements, 28
- compression property, description, 129
- compressratio property, description, 129
- controlling, data validation (scrubbing), 255

- copies property, description, 130
- crash dump, saving, 115
- creating
 - a basic ZFS file system (`zpool create`)
 - (example of), 30
 - a new pool by splitting a mirrored storage pool (`zpool split`)
 - (example of), 65
 - a ZFS storage pool (`zpool create`)
 - (example of), 30
 - alternate root pools
 - (example of), 251
 - double-parity RAID-Z storage pool (`zpool create`)
 - (example of), 50
 - mirrored ZFS storage pool (`zpool create`)
 - (example of), 48
 - single-parity RAID-Z storage pool (`zpool create`)
 - (example of), 50
 - triple-parity RAID-Z storage pool (`zpool create`)
 - (example of), 50
 - ZFS clone (example of), 191
 - ZFS file system, 33
 - (example of), 124
 - description, 124
 - ZFS file system hierarchy, 32
 - ZFS snapshot
 - (example of), 184
 - ZFS storage pool
 - description, 48
 - ZFS storage pool (`zpool create`)
 - (example of), 48
 - ZFS storage pool with cache devices (example of), 52
 - ZFS storage pool with log devices (example of), 51
 - ZFS volume
 - (example of), 243
- creation property, description, 130

D

- data
 - corrupted, 254
 - corruption identified (`zpool status -v`)
 - (example of), 261

- data (*Continued*)
 - repair, 255
 - resilvering
 - description, 257
 - scrubbing
 - (example of), 256
 - validation (scrubbing), 255
- dataset
 - definition, 27
 - description, 124
- dataset types, description, 148
- dedup property, description, 130
- dedupdit to property, description, 79
- dedupratio property, description, 79
- delegated administration, overview, 231
- delegating
 - dataset to a non-global zone
 - (example of), 247
 - permissions (example of), 236
- delegating permissions, zfs allow, 235
- delegating permissions to a group, (example of), 236
- delegating permissions to an individual user, (example of), 236
- delegation property, description, 79
- delegation property, disabling, 232
- destroying
 - ZFS clone (example of), 191
 - ZFS file system
 - (example of), 125
 - ZFS file system with dependents
 - (example of), 125
 - ZFS snapshot
 - (example of), 185
 - ZFS storage pool
 - description, 48
 - ZFS storage pool (zpool destroy)
 - (example of), 57
- detaching
 - devices to ZFS storage pool (zpool detach)
 - (example of), 65
- detecting
 - in-use devices
 - (example of), 55
- detecting (*Continued*)
 - mismatched replication levels
 - (example of), 56
- determining
 - if a device can be replaced
 - description, 267
 - type of device failure
 - description, 265
- devices property, description, 130
- differences between ZFS and traditional file systems
 - file system granularity, 37
 - mounting ZFS file systems, 39
 - new Solaris ACL model, 40
 - out of space behavior, 39
 - traditional volume management, 40
 - ZFS space accounting, 38
- disks, as components of ZFS storage pools, 43
- displaying
 - delegated permissions (example of), 240
 - detailed ZFS storage pool health status
 - (example of), 89
 - health status of storage pools
 - description of, 88
 - syslog reporting of ZFS error messages
 - description, 261
 - ZFS storage pool health status
 - (example of), 89
 - ZFS storage pool I/O statistics
 - description, 85
 - ZFS storage pool vdev I/O statistics
 - (example of), 86
 - ZFS storage pool-wide I/O statistics
 - (example of), 85
- dry run
 - ZFS storage pool creation (zpool create -n)
 - (example of), 56
- dumpadm, enabling a dump device, 115
- dynamic striping
 - description, 47
 - storage pool feature, 47

E

- EFI label
 - description, 42
 - interaction with ZFS, 42
- exec property, description, 130
- exporting
 - ZFS storage pool
 - (example of), 93

F

- failmode property, description, 79
- failure modes
 - corrupted data, 254
 - damaged devices, 254
 - missing (faulted) devices, 254
- failures, 253
- file system, definition, 27
- file system granularity, differences between ZFS and traditional file systems, 37
- file system hierarchy, creating, 32
- files, as components of ZFS storage pools, 44
- free property, description, 80

G

- guid property, description, 80

H

- hardware and software requirements, 30
- health property, description, 80
- hot spares
 - creating
 - (example of), 73
 - description of
 - (example of), 73

I

- identifying
 - storage requirements, 31
 - type of data corruption (zpool status -v)
 - (example of), 274
 - ZFS storage pool for import (zpool import -a)
 - (example of), 93
- importing
 - alternate root pools
 - (example of), 252
 - ZFS storage pool
 - (example of), 96
 - ZFS storage pool from alternate directories (zpool import -d)
 - (example of), 95
- in-use devices
 - detecting
 - (example of), 55
- inheriting
 - ZFS properties (zfs inherit)
 - description, 150
- installing bootblocks
 - installboot and installgrub
 - (example of), 116

L

- listing
 - descendents of ZFS file systems
 - (example of), 147
 - types of ZFS file systems
 - (example of), 148
 - ZFS file systems
 - (example of), 147
 - ZFS file systems (zfs list)
 - (example of), 34
 - ZFS file systems without header information
 - (example of), 148
 - ZFS pool information, 32
 - ZFS properties (zfs list)
 - (example of), 151
 - ZFS properties by source value
 - (example of), 152

listing (*Continued*)

- ZFS properties for scripting
 - (example of), 153
- ZFS storage pools
 - (example of), 81
 - description, 80

- listsnapshots property, description, 80
- logbias property, description, 131

M

- migrating ZFS storage pools, description, 92
- mirror, definition, 27
- mirrored configuration
 - conceptual view, 45
 - description, 45
 - redundancy feature, 45
- mirrored log device, adding, (example of), 61
- mirrored log devices, creating a ZFS storage pool with
 - (example of), 51
- mirrored storage pool (`zpool create`), (example of), 48
- mismatched replication levels
 - detecting
 - (example of), 56
- `mlslabel` property, description, 131
- modifying
 - trivial ACL on ZFS file (verbose mode)
 - (example of), 213
- mount point, default for ZFS storage pools, 57
- mount points
 - automatic, 154
 - legacy, 154
 - managing ZFS
 - description, 154
- mounted property, description, 131
- mounting
 - ZFS file systems
 - (example of), 156
- mounting ZFS file systems, differences between ZFS and
 - traditional file systems, 39
- mountpoint, default for ZFS file system, 124
- mountpoint property, description, 131

N

- naming requirements, ZFS components, 28
- NFSv4 ACLs
 - ACL inheritance, 208
 - ACL inheritance flags, 208
 - ACL property, 209
 - differences from POSIX-draft ACLs, 204
 - format description, 204
 - model
 - description, 203
- notifying
 - ZFS of reattached device (`zpool online`)
 - (example of), 264

O

- offlining a device (`zpool offline`)
 - ZFS storage pool
 - (example of), 68
- onlining a device
 - ZFS storage pool (`zpool online`)
 - (example of), 69
- onlining and offlining devices
 - ZFS storage pool
 - description, 68
- origin property, description, 132
- out of space behavior, differences between ZFS and
 - traditional file systems, 39

P

- permission sets, defined, 231
- pool, definition, 27
- pooled storage, description, 24
- POSIX-draft ACLs, description, 204
- `primarycache` property, description, 131
- properties of ZFS
 - description, 127
 - description of heritable properties, 127

Q

- quota property, description, 132
- quotas and reservations, description, 167

R

- RAID-Z, definition, 27
- RAID-Z configuration
 - (example of), 50
 - conceptual view, 46
 - double-parity, description, 46
 - redundancy feature, 46
 - single-parity, description, 46
- RAID-Z configuration, adding disks to, (example of), 60
- read-only properties of ZFS
 - available, 128
 - compression, 129
 - creation, 130
 - description, 138
 - mounted, 131
 - origin, 132
 - referenced, 133
 - type, 136
 - used, 136
 - usedbychildren, 136
 - usedbydataset, 137
 - usedbyreservation, 137
 - usedbysnapshots, 137
- read-only property, description, 132
- receiving
 - ZFS file system data (zfs receive)
 - (example of), 196
- recordsize property
 - description, 133
 - detailed description, 144
- recovering
 - destroyed ZFS storage pool
 - (example of), 99
- recursive stream package, 195
- referenced property, description, 133
- refquota property, description, 133
- refreservation property, description, 133
- removing, cache devices (example of), 62
- removing permissions, zfs unallow, 236
- renaming
 - ZFS file system
 - (example of), 126
 - ZFS snapshot
 - (example of), 186
- repairing
 - a damaged ZFS configuration
 - description, 262
 - an unbootable system
 - description, 278
 - pool-wide damage
 - description, 278
 - repairing a corrupted file or directory
 - description, 275
- replacing
 - a device (zpool replace)
 - (example of), 70, 268, 273
 - a missing device
 - (example of), 262
- replication features of ZFS, mirrored or RAID-Z, 45
- replication stream package, 194
- reservation property, description, 134
- resilvering, definition, 27
- resilvering and data scrubbing, description, 257
- restoring
 - trivial ACL on ZFS file (verbose mode)
 - (example of), 216
- rights profiles, for management of ZFS file systems and storage pools, 29
- rolling back
 - ZFS snapshot
 - (example of), 189

S

- savecore, saving crash dumps, 115
- saving
 - crash dumps
 - savecore, 115
 - ZFS file system data (zfs send)
 - (example of), 195

- scripting
 - ZFS storage pool output
 - (example of), 83
- scrubbing
 - (example of), 256
 - data validation, 255
- secondarycache property, description, 134
- self-healing data, description, 47
- sending and receiving
 - ZFS file system data
 - description, 192
- separate log devices, considerations for using, 51
- settable properties of ZFS
 - aclinherit, 127
 - aclmode, 128
 - atime, 128
 - canmount, 128
 - detailed description, 141
 - casesensitivity, 129
 - checksum, 129
 - compression, 129
 - copies, 130
 - dedup, 130
 - description, 139
 - devices, 130
 - exec, 130
 - mountpoint, 131
 - primarycache, 131
 - quota, 132
 - read-only, 132
 - recordsize, 133
 - detailed description, 144
 - refquota, 133
 - refreservation, 133
 - reservation, 134
 - secondarycache, 134
 - setuid, 134
 - shadow, 134
 - sharenfs, 135
 - sharesmb, 135
 - snapdir, 135
 - sync, 136
 - used
 - detailed description, 139
 - settable properties of ZFS (*Continued*)
 - version, 137
 - volblocksize, 137
 - volsize, 137
 - detailed description, 145
 - xattr, 138
 - zoned, 138
- setting
 - ACL inheritance on ZFS file (verbose mode)
 - (example of), 217
 - ACLs on ZFS file (compact mode)
 - (example of), 224
 - description, 223
 - ACLs on ZFS file (verbose mode)
 - (description, 212
 - ACLs on ZFS files
 - description, 210
 - compression property
 - (example of), 34
 - legacy mount points
 - (example of), 155
 - mountpoint property, 34
 - quota property (example of), 34
 - sharenfs property
 - (example of), 34
 - ZFS atime property
 - (example of), 149
 - ZFS file system quota (zfs set quota)
 - example of, 168
 - ZFS file system reservation
 - (example of), 171
 - ZFS mount points (zfs set mountpoint)
 - (example of), 155
 - ZFS quota
 - (example of), 150
- setuid property, description, 134
- shadow property, description, 134
- sharenfs property, description, 135
- sharesmb property, description, 135
- sharesmb property, description, detailed, 145
- sharing ZFS file systems, sharesmb property, 145
- simplified administration, description, 26
- size property, description, 80
- snapdir property, description, 135

- snapshot
 - accessing
 - (example of), 187
 - creating
 - (example of), 184
 - definition, 28
 - destroying
 - (example of), 185
 - features, 183
 - renaming
 - (example of), 186
 - rolling back
 - (example of), 189
 - space accounting, 188
 - Solaris ACLs
 - ACL inheritance, 208
 - ACL inheritance flags, 208
 - ACL property, 209
 - differences from POSIX-draft ACLs, 204
 - format description, 204
 - new model
 - description, 203
 - splitting a mirrored storage pool
 - (`zpool split`)
 - (example of), 65
 - storage requirements, identifying, 31
 - stream package
 - recursive, 195
 - replication, 194
 - swap and dump devices
 - adjusting sizes of, 114
 - description, 113
 - issues, 113
 - sync property, description, 136
- T**
- terminology
 - checksum, 26
 - clone, 26
 - dataset, 27
 - file system, 27
 - mirror, 27
 - pool, 27
 - terminology (*Continued*)
 - RAID-Z, 27
 - resilvering, 27
 - snapshot, 28
 - virtual device, 28
 - volume, 28
 - traditional volume management, differences between ZFS and traditional file systems, 40
 - transactional semantics, description, 25
 - troubleshooting
 - clear device errors (`zpool clear`)
 - (example of), 266
 - damaged devices, 254
 - data corruption identified (`zpool status -v`)
 - (example of), 261
 - determining if a device can be replaced
 - description, 267
 - determining if problems exist (`zpool status -x`), 258
 - determining type of data corruption (`zpool status -v`)
 - (example of), 274
 - determining type of device failure
 - description, 265
 - identifying problems, 257
 - missing (faulted) devices, 254
 - notifying ZFS of reattached device (`zpool onLine`)
 - (example of), 264
 - overall pool status information
 - description, 259
 - repairing a corrupted file or directory
 - description, 275
 - repairing a damaged ZFS configuration, 262
 - repairing an unbootable system
 - description, 278
 - repairing pool-wide damage
 - description, 278
 - replacing a device (`zpool replace`)
 - (example of), 268, 273
 - replacing a missing device
 - (example of), 262
 - syslog reporting of ZFS error messages, 261
 - ZFS failures, 253
 - type property, description, 136

U

- unmounting
 - ZFS file systems
 - (example of), 158
- upgrading
 - ZFS file systems
 - description, 181
 - ZFS storage pool
 - description, 100
- used property
 - description, 136
 - detailed description, 139
- usedbychildren property, description, 136
- usedbydataset property, description, 137
- usedbyreservation property, description, 137
- usedbysnapshots property, description, 137
- user properties of ZFS
 - (example of), 145
 - detailed description, 145

V

- version property, description, 137
- version property, description, 80
- virtual device, definition, 28
- virtual devices, as components of ZFS storage pools, 54
- volblocksize property, description, 137
- volsize property
 - description, 137
 - detailed description, 145
- volume, definition, 28

W

- whole disks, as components of ZFS storage pools, 43

X

- xattr property, description, 138

Z

- zfs allow
 - description, 235
 - displaying delegated permissions, 240
- zfs create
 - (example of), 33, 124
 - description, 124
- ZFS delegated administration, overview, 231
- zfs destroy, (example of), 125
- zfs destroy -r, (example of), 125
- ZFS file system
 - description, 123
 - versions
 - description, 297
- ZFS file systems
 - ACL on ZFS directory
 - detailed description, 212
 - ACL on ZFS file
 - detailed description, 211
 - adding ZFS file system to a non-global zone
 - (example of), 246
 - adding ZFS volume to a non-global zone
 - (example of), 248
 - booting a root file system
 - description, 116
 - booting a ZFS BE with boot -Land boot -Z
 - (SPARC example of), 118
 - checksum
 - definition, 26
 - checksummed data
 - description, 25
 - clone
 - replacing a file system with (example of), 191
 - clones
 - definition, 26
 - description, 190
 - component naming requirements, 28
 - creating
 - (example of), 124
 - creating a clone, 191
 - creating a ZFS volume
 - (example of), 243
 - dataset
 - definition, 27

ZFS file systems (*Continued*)

- dataset types
 - description, 148
- default mountpoint
 - (example of), 124
- delegating dataset to a non-global zone
 - (example of), 247
- description, 24
- destroying
 - (example of), 125
- destroying a clone, 191
- destroying with dependents
 - (example of), 125
- file system
 - definition, 27
- inheriting property of (`zfs inherit`)
 - (example of), 150
- listing
 - (example of), 147
- listing descendents
 - (example of), 147
- listing properties by source value
 - (example of), 152
- listing properties for scripting
 - (example of), 153
- listing properties of (`zfs list`)
 - (example of), 151
- listing types of
 - (example of), 148
- listing without header information
 - (example of), 148
- managing automatic mount points, 154
- managing legacy mount points
 - description, 154
- managing mount points
 - description, 154
- modifying trivial ACL on ZFS file (verbose mode)
 - (example of), 213
- mounting
 - (example of), 156
- pooled storage
 - description, 24
- property management within a zone
 - description, 248

ZFS file systems (*Continued*)

- receiving data streams (`zfs receive`)
 - (example of), 196
- renaming
 - (example of), 126
- restoring trivial ACL on ZFS file (verbose mode)
 - (example of), 216
- rights profiles, 29
- saving data streams (`zfs send`)
 - (example of), 195
- sending and receiving
 - description, 192
- setting a reservation
 - (example of), 171
- setting ACL inheritance on ZFS file (verbose mode)
 - (example of), 217
- setting ACLs on ZFS file (compact mode)
 - (example of), 224
 - description, 223
- setting ACLs on ZFS file (verbose mode)
 - description, 212
- setting ACLs on ZFS files
 - description, 210
- setting a time property
 - (example of), 149
- setting legacy mount point
 - (example of), 155
- setting mount point (`zfs set mountpoint`)
 - (example of), 155
- setting quota property
 - (example of), 150
- simplified administration
 - description, 26
- snapshot
 - accessing, 187
 - creating, 184
 - definition, 28
 - description, 183
 - destroying, 185
 - renaming, 186
 - rolling back, 189
- snapshot space accounting, 188
- swap and dump devices
 - adjusting sizes of, 114

- ZFS file systems, swap and dump devices (*Continued*)
 - description, 113
 - issues, 113
 - transactional semantics
 - description, 25
 - unmounting
 - (example of), 158
 - upgrading
 - description, 181
 - using on a Solaris system with zones installed
 - description, 246
 - volume
 - definition, 28
- ZFS file systems (zfs set quota)
 - setting a quota
 - example of, 168
- zfs get, (example of), 151
- zfs get -H -o, (example of), 153
- zfs get -s, (example of), 152
- zfs inherit, (example of), 150
- ZFS intent log (ZIL), description, 51
- zfs list
 - (example of), 34, 147
- zfs list -H, (example of), 148
- zfs list -r, (example of), 147
- zfs list -t, (example of), 148
- zfs mount, (example of), 156
- ZFS pool properties
 - allocated, 78
 - alroot, 78
 - autoreplace, 78
 - bootfs, 78
 - cachefile, 79
 - capacity, 79
 - dedupditto, 79
 - dedupratio, 79
 - delegation, 79
 - failmode, 79
 - free, 80
 - guid, 80
 - health, 80
 - listsnapshots, 80
 - size, 80
 - version, 80
- zfs promote, clone promotion (example of), 191
- ZFS properties
 - aclinherit, 127
 - aclmode, 128
 - atime, 128
 - available, 128
 - canmount, 128
 - detailed description, 141
 - casesensitivity, 129
 - checksum, 129
 - compression, 129
 - compressratio, 129
 - copies, 130
 - creation, 130
 - dedup, 130
 - description, 127
 - devices, 130
 - exec, 130
 - inheritable, description of, 127
 - logbias, 131
 - management within a zone
 - description, 248
 - mlslabel, 131
 - mounted, 131
 - mountpoint, 131
 - origin, 132
 - quota, 132
 - read-only, 132
 - read-only, 138
 - recordsize, 133
 - detailed description, 144
 - referenced, 133
 - refquota, 133
 - reservation, 133
 - reservation, 134
 - secondarycache, 131, 134
 - settable, 139
 - setuid, 134
 - shadow, 134
 - sharenfs, 135
 - sharesmb, 135
 - snappdir, 135
 - sync, 136
 - type, 136

ZFS properties (*Continued*)

- used, 136
 - detailed description, 139
- usedbychildren, 136
- usedbydataset, 137
- usedbyreservation, 137
- usedbysnapshots, 137
- user properties
 - detailed description, 145
- version, 137
- volblocksize, 137
- volsize, 137
 - detailed description, 145
- xattr, 138
- zoned, 138
- zoned property
 - detailed description, 249
- zfs receive, (example of), 196
- zfs rename, (example of), 126
- zfs send, (example of), 195
- zfs set atime, (example of), 149
- zfs set compression, (example of), 34
- zfs set mountpoint
 - (example of), 34, 155
- zfs set mountpoint=legacy, (example of), 155
- zfs set quota
 - (example of), 34
- zfs set quota, (example of), 150
- zfs set quota
 - example of, 168
- zfs set reservation, (example of), 171
- zfs set sharenfs, (example of), 34
- ZFS space accounting, differences between ZFS and traditional file systems, 38
- ZFS storage pool
 - versions
 - description, 297
- ZFS storage pools
 - adding devices to (zpool add)
 - (example of), 59
 - alternate root pools, 251
 - attaching devices to (zpool attach)
 - (example of), 63

ZFS storage pools (*Continued*)

- clearing a device
 - (example of), 70
- clearing device errors (zpool clear)
 - (example of), 266
- components, 41
- corrupted data
 - description, 254
- creating (zpool create)
 - (example of), 48
- creating a RAID-Z configuration (zpool create)
 - (example of), 50
- creating mirrored configuration (zpool create)
 - (example of), 48
- damaged devices
 - description, 254
- data corruption identified (zpool status -v)
 - (example of), 261
- data repair
 - description, 255
- data scrubbing
 - (example of), 256
 - description, 255
- data scrubbing and resilvering
 - description, 257
- data validation
 - description, 255
- default mount point, 57
- destroying (zpool destroy)
 - (example of), 57
- detaching devices from (zpool detach)
 - (example of), 65
- determining if a device can be replaced
 - description, 267
- determining if problems exist (zpool status -x)
 - description, 258
- determining type of device failure
 - description, 265
- displaying detailed health status
 - (example of), 89
- displaying health status, 88
 - (example of), 89
- doing a dry run (zpool create -n)
 - (example of), 56

ZFS storage pools (*Continued*)

- dynamic striping, 47
- exporting
 - (example of), 93
- failures, 253
- identifying for import (`zpool import -a`)
 - (example of), 93
- identifying problems
 - description, 257
- identifying type of data corruption (`zpool status -v`)
 - (example of), 274
- importing
 - (example of), 96
- importing from alternate directories (`zpool import -d`)
 - (example of), 95
- listing
 - (example of), 81
- migrating
 - description, 92
- mirror
 - definition, 27
- mirrored configuration, description, 45
- missing (faulted) devices
 - description, 254
- notifying ZFS of reattached device (`zpool onLine`)
 - (example of), 264
- offlining a device (`zpool offline`)
 - (example of), 68
- onlining and offlining devices
 - description, 68
- overall pool status information for troubleshooting
 - description, 259
- pool
 - definition, 27
- pool-wide I/O statistics
 - (example of), 85
- RAID-Z
 - definition, 27
- RAID-Z configuration, description, 46
- recovering a destroyed pool
 - (example of), 99

ZFS storage pools (*Continued*)

- repairing a corrupted file or directory
 - description, 275
- repairing a damaged ZFS configuration, 262
- repairing an unbootable system
 - description, 278
- repairing pool-wide damage
 - description, 278
- replacing a device (`zpool replace`)
 - (example of), 70, 268
- replacing a missing device
 - (example of), 262
- resilvering
 - definition, 27
- rights profiles, 29
- scripting storage pool output
 - (example of), 83
- splitting a mirrored storage pool (`zpool split`)
 - (example of), 65
- system error messages
 - description, 261
- upgrading
 - description, 100
- using files, 44
- using whole disks, 43
- vdev I/O statistics
 - (example of), 86
- viewing resilvering process
 - (example of), 273
- virtual device
 - definition, 28
- virtual devices, 54
- ZFS storage pools (`zpool online`)
 - onlining a device
 - (example of), 69
- `zfs unallow`, description, 236
- `zfs unmount`, (example of), 158
- `zfs upgrade`, 181
- ZFS version
 - ZFS feature and Solaris OS
 - description, 297
- ZFS volume, description, 243
- zoned property
 - description, 138

- zoned property (*Continued*)
 - detailed description, 249
- zones
 - adding ZFS file system to a non-global zone
 - (example of), 246
 - adding ZFS volume to a non-global zone
 - (example of), 248
 - delegating dataset to a non-global zone
 - (example of), 247
 - using with ZFS file systems
 - description, 246
 - ZFS property management within a zone
 - description, 248
 - zoned property
 - detailed description, 249
- zpool add, (example of), 59
- zpool attach, (example of), 63
- zpool clear
 - (example of), 70
 - description, 70
- zpool create
 - (example of), 30, 32
 - basic pool
 - (example of), 48
 - mirrored storage pool
 - (example of), 48
 - RAID-Z storage pool
 - (example of), 50
- zpool create -n, dry run (example of), 56
- zpool destroy, (example of), 57
- zpool detach, (example of), 65
- zpool export, (example of), 93
- zpool import -a, (example of), 93
- zpool import -D, (example of), 99
- zpool import -d, (example of), 95
- zpool import *name*, (example of), 96
- zpool iostat, pool-wide (example of), 85
- zpool iostat -v, vdev (example of), 86
- zpool list
 - (example of), 32, 81
 - description, 80
- zpool list -Ho *name*, (example of), 83
- zpool offline, (example of), 68
- zpool online, (example of), 69
- zpool replace, (example of), 70
- zpool split, (example of), 65
- zpool status -v, (example of), 89
- zpool status -x, (example of), 89
- zpool upgrade, 100