

Oracle® Solaris ZFS Administration Guide

Copyright © 2006, 2010, 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 software 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 which 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.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. 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. UNIX is a registered trademark licensed through X/Open Company, Ltd.

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.

Copyright © 2006, 2010, Oracle et/ou ses affiliés. Tous droits réservés.

Ce logiciel et la documentation qui l'accompagne sont protégés par les lois sur la propriété intellectuelle. Ils sont concédés sous licence et soumis à des restrictions d'utilisation et de divulgation. Sauf 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 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.

AMD, Opteron, le logo AMD et le logo AMD Opteron sont des marques ou des marques déposées d'Advanced Micro Devices. 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. UNIX est une marque déposée concédée sous licence par X/Open Company, Ltd.

Contents

Preface	11
1 Oracle Solaris ZFS File System (Introduction)	15
What's New in ZFS?	15
Splitting a Mirrored ZFS Storage Pool (<code>zpool split</code>)	16
New ZFS System Process	17
Changes to the <code>zpool list</code> Command	17
ZFS Storage Pool Recovery	17
ZFS Log Device Enhancements	17
Triple Parity RAIDZ (<code>raidz3</code>)	18
Holding ZFS Snapshots	18
ZFS Device Replacement Enhancements	19
ZFS and Flash Installation Support	20
ZFS User and Group Quotas	20
ZFS ACL Pass Through Inheritance for Execute Permission	21
ZFS Property Enhancements	22
ZFS Log Device Recovery	24
Using Cache Devices in Your ZFS Storage Pool	25
Zone Migration in a ZFS Environment	26
ZFS Installation and Boot Support	26
Rolling Back a Dataset Without Unmounting	26
Enhancements to the <code>zfs send</code> Command	26
ZFS Quotas and Reservations for File System Data Only	27
ZFS Storage Pool Properties	28
ZFS Command History Enhancements (<code>zpool history</code>)	29
Upgrading ZFS File Systems (<code>zfs upgrade</code>)	30
ZFS Delegated Administration	30
Setting Up Separate ZFS Log Devices	30

Creating Intermediate ZFS Datasets	31
ZFS Hot-Plugging Enhancements	32
Recursively Renaming ZFS Snapshots (zfs rename -r)	33
gzip Compression Is Available for ZFS	33
Storing Multiple Copies of ZFS User Data	34
Improved zpool status Output	35
ZFS and Solaris iSCSI Improvements	35
ZFS Command History (zpool history)	35
ZFS Property Improvements	36
Displaying All ZFS File System Information	37
New zfs receive -F Option	37
Recursive ZFS Snapshots	37
Double-Parity RAID-Z (raidz2)	38
Hot Spares for ZFS Storage Pool Devices	38
Replacing a ZFS File System With a ZFS Clone (zfs promote)	38
Upgrading ZFS Storage Pools (zpool upgrade)	38
ZFS Backup and Restore Commands Are Renamed	39
Recovering Destroyed Storage Pools	39
ZFS Is Integrated With Fault Manager	39
The zpool clear Command	40
Compact NFSv4 ACL Format	40
File System Monitoring Tool (fsstat)	40
ZFS Web-Based Management	40
What Is ZFS?	41
ZFS Pooled Storage	41
Transactional Semantics	42
Checksums and Self-Healing Data	42
Unparalleled Scalability	43
ZFS Snapshots	43
Simplified Administration	43
ZFS Terminology	43
ZFS Component Naming Requirements	45
2 Getting Started With Oracle Solaris ZFS	47
ZFS Hardware and Software Requirements and Recommendations	47

Creating a Basic ZFS File System	48
Creating a ZFS Storage Pool	49
▼ How to Identify Storage Requirements for Your ZFS Storage Pool	49
▼ How to Create a ZFS Storage Pool	49
Creating a ZFS File System Hierarchy	50
▼ How to Determine Your ZFS File System Hierarchy	50
▼ How to Create ZFS File Systems	51
3 Oracle Solaris ZFS and Traditional File System Differences	53
ZFS File System Granularity	53
ZFS Disk Space Accounting	54
Out of Space Behavior	54
Mounting ZFS File Systems	55
Traditional Volume Management	55
New Solaris ACL Model	55
4 Managing Oracle Solaris ZFS Storage Pools	57
Components of a ZFS Storage Pool	57
Using Disks in a ZFS Storage Pool	57
Using Slices in a ZFS Storage Pool	59
Using Files in a ZFS Storage Pool	60
Replication Features of a ZFS Storage Pool	60
Mirrored Storage Pool Configuration	61
RAID-Z Storage Pool Configuration	61
ZFS Hybrid Storage Pool	62
Self-Healing Data in a Redundant Configuration	62
Dynamic Striping in a Storage Pool	62
Creating and Destroying ZFS Storage Pools	63
Creating a ZFS Storage Pool	63
Displaying Storage Pool Virtual Device Information	68
Handling ZFS Storage Pool Creation Errors	69
Destroying ZFS Storage Pools	71
Managing Devices in ZFS Storage Pools	72
Adding Devices to a Storage Pool	73
Attaching and Detaching Devices in a Storage Pool	77

Creating a New Pool By Splitting a Mirrored ZFS Storage Pool	79
Onlining and Offlining Devices in a Storage Pool	82
Clearing Storage Pool Device Errors	84
Replacing Devices in a Storage Pool	85
Designating Hot Spares in Your Storage Pool	87
Managing ZFS Storage Pool Properties	92
Querying ZFS Storage Pool Status	95
Displaying Information About ZFS Storage Pools	95
Viewing I/O Statistics for ZFS Storage Pools	98
Determining the Health Status of ZFS Storage Pools	100
Migrating ZFS Storage Pools	103
Preparing for ZFS Storage Pool Migration	103
Exporting a ZFS Storage Pool	103
Determining Available Storage Pools to Import	104
Importing ZFS Storage Pools From Alternate Directories	106
Importing ZFS Storage Pools	106
Recovering Destroyed ZFS Storage Pools	107
Upgrading ZFS Storage Pools	109
5 Installing and Booting an Oracle Solaris ZFS Root File System	111
Installing and Booting an Oracle Solaris ZFS Root File System (Overview)	111
ZFS Installation Features	112
Oracle Solaris Installation and Oracle Solaris Live Upgrade Requirements for ZFS Support	113
Installing a ZFS Root File System (Initial Installation)	115
▼ How to Create a Mirrored Root Pool (Post Installation)	121
Installing a ZFS Root File System (Oracle Solaris Flash Archive Installation)	122
Installing a ZFS Root File System (Oracle Solaris JumpStart Installation)	124
JumpStart Keywords for ZFS	125
JumpStart Profile Examples for ZFS	126
JumpStart Issues for ZFS	127
Migrating a UFS Root File System to a ZFS Root File System (Oracle Solaris Live Upgrade) ..	128
ZFS Migration Issues With Oracle Solaris Live Upgrade	129
Using Oracle Solaris Live Upgrade to Migrate to a ZFS Root File System (Without Zones)	129
Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (Solaris 10	

10/08)	134
Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (at Least Solaris 10 5/09)	139
ZFS Support for Swap and Dump Devices	148
Adjusting the Sizes of Your ZFS Swap Device and Dump Device	149
Troubleshooting ZFS Dump Device Issues	151
Booting From a ZFS Root File System	152
Booting From an Alternate Disk in a Mirrored ZFS Root Pool	152
SPARC: Booting From a ZFS Root File System	153
x86: Booting From a ZFS Root File System	155
Resolving ZFS Mount-Point Problems That Prevent Successful Booting (Solaris 10 10/08)	156
Booting For Recovery Purposes in a ZFS Root Environment	157
Recovering the ZFS Root Pool or Root Pool Snapshots	159
▼ How to Replace a Disk in the ZFS Root Pool	159
▼ How to Create Root Pool Snapshots	161
▼ How to Recreate a ZFS Root Pool and Restore Root Pool Snapshots	162
▼ How to Roll Back Root Pool Snapshots From a Failsafe Boot	164
6 Managing Oracle Solaris ZFS File Systems	167
Managing ZFS File Systems (Overview)	167
Creating, Destroying, and Renaming ZFS File Systems	168
Creating a ZFS File System	168
Destroying a ZFS File System	169
Renaming a ZFS File System	170
Introducing ZFS Properties	171
ZFS Read-Only Native Properties	178
Settable ZFS Native Properties	179
ZFS User Properties	182
Querying ZFS File System Information	183
Listing Basic ZFS Information	183
Creating Complex ZFS Queries	184
Managing ZFS Properties	185
Setting ZFS Properties	185
Inheriting ZFS Properties	186
Querying ZFS Properties	187

Mounting and Sharing ZFS File Systems	190
Managing ZFS Mount Points	190
Mounting ZFS File Systems	192
Using Temporary Mount Properties	193
Unmounting ZFS File Systems	194
Sharing and Unsharing ZFS File Systems	194
Setting ZFS Quotas and Reservations	196
Setting Quotas on ZFS File Systems	196
Setting Reservations on ZFS File Systems	200
7 Working With Oracle Solaris ZFS Snapshots and Clones	203
Overview of ZFS Snapshots	203
Creating and Destroying ZFS Snapshots	204
Displaying and Accessing ZFS Snapshots	207
Rolling Back a ZFS Snapshot	208
Overview of ZFS Clones	209
Creating a ZFS Clone	210
Destroying a ZFS Clone	210
Replacing a ZFS File System With a ZFS Clone	210
Sending and Receiving ZFS Data	211
Saving ZFS Data With Other Backup Products	212
Sending a ZFS Snapshot	212
Receiving a ZFS Snapshot	213
Sending and Receiving Complex ZFS Snapshot Streams	214
8 Using ACLs to Protect Oracle Solaris ZFS Files	219
New Solaris ACL Model	219
Syntax Descriptions for Setting ACLs	220
ACL Inheritance	223
ACL Properties	224
Setting ACLs on ZFS Files	225
Setting and Displaying ACLs on ZFS Files in Verbose Format	228
Setting ACL Inheritance on ZFS Files in Verbose Format	233
Setting and Displaying ACLs on ZFS Files in Compact Format	239

9 Oracle Solaris ZFS Delegated Administration	245
Overview of ZFS Delegated Administration	245
Disabling ZFS Delegated Permissions	246
Delegating ZFS Permissions	246
Delegating ZFS Permissions (zfs allow)	248
Removing ZFS Delegated Permissions (zfs unallow)	249
Delegating ZFS Permissions (Examples)	250
Displaying ZFS Delegated Permissions (Examples)	253
Removing ZFS Delegated Permissions (Examples)	255
10 Oracle Solaris ZFS Advanced Topics	257
ZFS Volumes	257
Using a ZFS Volume as a Swap or Dump Device	258
Using a ZFS Volume as a Solaris iSCSI Target	258
Using ZFS on a Solaris System With Zones Installed	259
Adding ZFS File Systems to a Non-Global Zone	260
Delegating Datasets to a Non-Global Zone	261
Adding ZFS Volumes to a Non-Global Zone	262
Using ZFS Storage Pools Within a Zone	262
Managing ZFS Properties Within a Zone	263
Understanding the zoned Property	264
Using ZFS Alternate Root Pools	265
Creating ZFS Alternate Root Pools	265
Importing Alternate Root Pools	265
ZFS Rights Profiles	266
11 Oracle Solaris ZFS Troubleshooting and Pool Recovery	267
Identifying ZFS Failures	267
Missing Devices in a ZFS Storage Pool	268
Damaged Devices in a ZFS Storage Pool	268
Corrupted ZFS Data	268
Checking ZFS File System Integrity	269
File System Repair	269
File System Validation	269
Controlling ZFS Data Scrubbing	269

Resolving Problems With ZFS	271
Determining If Problems Exist in a ZFS Storage Pool	272
Reviewing <code>zpool status</code> Output	272
System Reporting of ZFS Error Messages	275
Repairing a Damaged ZFS Configuration	276
Resolving a Missing Device	276
Physically Reattaching a Device	277
Notifying ZFS of Device Availability	277
Replacing or Repairing a Damaged Device	278
Determining the Type of Device Failure	278
Clearing Transient Errors	279
Replacing a Device in a ZFS Storage Pool	280
Repairing Damaged Data	287
Identifying the Type of Data Corruption	287
Repairing a Corrupted File or Directory	288
Repairing ZFS Storage Pool-Wide Damage	289
Repairing an Unbootable System	291
A Oracle Solaris ZFS Version Descriptions	293
Overview of ZFS Versions	293
ZFS Pool Versions	293
ZFS File System Versions	294
Index	297

Preface

The *Oracle Solaris ZFS Administration Guide* 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: UltraSPARC, SPARC64, AMD64, Pentium, and Xeon EM64T. The supported systems appear in the *Solaris 10 Hardware Compatibility List* at <http://www.sun.com/bigadmin/hcl>. This document cites any implementation differences between the platform types.

In this document these x86 terms mean the following:

- “x86” refers to the larger family of 64-bit and 32-bit x86 compatible products.
- “x64” points out specific 64-bit information about AMD64 or EM64T systems.
- “32-bit x86” points out specific 32-bit information about x86 based systems.

For supported systems, see the *Solaris 10 Hardware Compatibility List*.

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	Description
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 5, “Installing and Booting an Oracle Solaris ZFS Root File System”	Describes how to install and boot a ZFS file system. Migrating a UFS root file system to a ZFS root file system by using Oracle Solaris Live Upgrade is also covered.
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 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.
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:

- *System Administration Guide: Basic Administration*
- *System Administration Guide: Advanced Administration*
- *System Administration Guide: Devices and File Systems*
- *System Administration Guide: Security Services*

Documentation, Support, and Training

See the following web sites for additional resources:

- [Documentation](http://docs.sun.com) (<http://docs.sun.com>)
- [Support](http://www.oracle.com/us/support/systems/index.html) (<http://www.oracle.com/us/support/systems/index.html>)
- [Training](http://education.oracle.com) (<http://education.oracle.com>) – Click the Sun link in the left navigation bar.

Oracle Welcomes Your Comments

Oracle welcomes your comments and suggestions on the quality and usefulness of its documentation. If you find any errors or have any other suggestions for improvement, go to <http://docs.sun.com> and click Feedback. Indicate the title and part number of the documentation along with the chapter, section, and page number, if available. Please let us know if you want a reply.

[Oracle Technology Network](http://www.oracle.com/technetwork/index.html) (<http://www.oracle.com/technetwork/index.html>) offers a range of resources related to Oracle software:

- Discuss technical problems and solutions on the [Discussion Forums](http://forums.oracle.com) (<http://forums.oracle.com>).
- Get hands-on step-by-step tutorials with [Oracle By Example](http://www.oracle.com/technology/obe/start/index.html) (<http://www.oracle.com/technology/obe/start/index.html>).
- Download [Sample Code](http://www.oracle.com/technology/sample_code/index.html) (http://www.oracle.com/technology/sample_code/index.html).

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 <code>rm filename</code> .

TABLE P-1 Typographic Conventions (Continued)

Typeface	Meaning	Example
<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	\$
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 ZFS?” on page 41
- “ZFS Terminology” on page 43
- “ZFS Component Naming Requirements” on page 45

What's New in ZFS?

This section summarizes new features in the ZFS file system.

- “Splitting a Mirrored ZFS Storage Pool (`zpool split`)” on page 16
- “New ZFS System Process” on page 17
- “Changes to the `zpool list` Command” on page 17
- “ZFS Storage Pool Recovery” on page 17
- “ZFS Log Device Enhancements” on page 17
- “Triple Parity RAIDZ (`raidz3`)” on page 18
- “Holding ZFS Snapshots” on page 18
- “ZFS Device Replacement Enhancements” on page 19
- “ZFS and Flash Installation Support” on page 20
- “ZFS User and Group Quotas” on page 20
- “ZFS ACL Pass Through Inheritance for Execute Permission” on page 21
- “ZFS Property Enhancements” on page 22
- “ZFS Log Device Recovery” on page 24
- “Using Cache Devices in Your ZFS Storage Pool” on page 25
- “Zone Migration in a ZFS Environment” on page 26
- “ZFS Installation and Boot Support” on page 26
- “Rolling Back a Dataset Without Unmounting” on page 26

- “Enhancements to the `zfs send` Command” on page 26
- “ZFS Quotas and Reservations for File System Data Only” on page 27
- “ZFS Storage Pool Properties” on page 28
- “ZFS Command History Enhancements (`zpool history`)” on page 29
- “Upgrading ZFS File Systems (`zfs upgrade`)” on page 30
- “ZFS Delegated Administration” on page 30
- “Setting Up Separate ZFS Log Devices” on page 30
- “Creating Intermediate ZFS Datasets” on page 31
- “ZFS Hot-Plugging Enhancements” on page 32
- “Recursively Renaming ZFS Snapshots (`zfs rename -r`)” on page 33
- “gzip Compression Is Available for ZFS” on page 33
- “Storing Multiple Copies of ZFS User Data” on page 34
- “Improved `zpool status` Output” on page 35
- “ZFS and Solaris iSCSI Improvements” on page 35
- “ZFS Command History (`zpool history`)” on page 35
- “ZFS Property Improvements” on page 36
- “Displaying All ZFS File System Information” on page 37
- “New `zfs receive -F` Option” on page 37
- “Recursive ZFS Snapshots” on page 37
- “Double-Parity RAID-Z (`raidz2`)” on page 38
- “Hot Spares for ZFS Storage Pool Devices” on page 38
- “Replacing a ZFS File System With a ZFS Clone (`zfs promote`)” on page 38
- “Upgrading ZFS Storage Pools (`zpool upgrade`)” on page 38
- “ZFS Backup and Restore Commands Are Renamed” on page 39
- “Recovering Destroyed Storage Pools” on page 39
- “ZFS Is Integrated With Fault Manager” on page 39
- “The `zpool clear` Command” on page 40
- “Compact NFSv4 ACL Format” on page 40
- “File System Monitoring Tool (`fsstat`)” on page 40
- “ZFS Web-Based Management” on page 40

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

Oracle Solaris 10 9/10 Release: In this Solaris 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 79.

New ZFS System Process

Oracle Solaris 10 9/10 Release: In this Solaris 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 checksumming, 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 process can be reviewed by using the `ps` and `prstat` commands. These processes are only available in the global zone. For more information, see [SDC\(7\)](#).

Changes to the `zpool list` Command

Oracle Solaris 10 9/10 Release: In this Solaris release, the `zpool list` output has changed to provide better space allocation information. For example:

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

The previous `USED` and `AVAIL` fields have been replaced with `ALLOC` and `FREE`.

The `ALLOC` field identifies the amount of physical space allocated to all datasets and internal metadata. The `FREE` field identifies the amount of unallocated space in the pool.

For more information, see [“Displaying Information About ZFS Storage Pools”](#) on page 95.

ZFS Storage Pool Recovery

Oracle Solaris 10 9/10 Release: A storage pool can become damaged if underlying devices become unavailable, a power failure occurs, or if more than the supported number of devices fail in a redundant ZFS configuration. This release provides new command features for recovering your damaged storage pool. However, using this recovery feature means that the last few transactions that occurred prior to the pool outage might be lost.

Both the `zpool clear` and `zpool import` commands support the `-F` option to possibly recover a damaged pool. In addition, running the `zpool status`, `zpool clear`, or `zpool import` command automatically report a damaged pool and these commands describe how to recover the pool.

For more information, see [“Repairing ZFS Storage Pool-Wide Damage”](#) on page 289.

ZFS Log Device Enhancements

Oracle Solaris 10 9/10 release: The following log device enhancements are available:

- The `logbias` property – You can use this property to provide a hint to ZFS about handling synchronous requests for a specific dataset. If `logbias` is set to `latency`, ZFS uses the pool's separate log devices, if any, to handle the requests at low latency. If `logbias` is set to `throughput`, 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 `latency`. For most configurations, the default value is recommended. Using the `logbias=throughput` value might improve performance for writing database files.
- Log device removal – You can now remove a log device from a ZFS storage pool by using the `zpool remove` command. A single log device can be removed by specifying the device name. A mirrored log device can be removed by specifying the top-level mirror for the log. When a separate log device is removed from the system, ZIL transaction records are written to the main pool.

Redundant top-level virtual devices are now identified with a numeric identifier. For example, in a mirrored storage pool of two disks, the top level virtual device is `mirror-0`.

For more information, see [Example 4-3](#).

Triple Parity RAIDZ (raidz3)

Oracle Solaris 10 9/10 Release: In this Solaris release, a redundant RAID-Z configuration can now have either single-, double-, or triple-parity, which means that one, two, three device failures can be sustained respectively, without any data loss. You can specify the `raidz3` keyword for a triple-parity RAID-Z configuration. For more information, see [“Creating a RAID-Z Storage Pool”](#) on page 65.

Holding ZFS Snapshots

Oracle Solaris 10 9/10 Release: If you implement different automatic snapshot policies so 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 in this Solaris release.

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.

You can hold a snapshot or set of snapshots. For example, the following syntax puts a hold tag, `keep`, on `tank/home/cindys/snap@1`.

```
# zfs hold keep tank/home/cindys@snap1
```

For more information, see [“Holding ZFS Snapshots”](#) on page 205.

ZFS Device Replacement Enhancements

Oracle Solaris 10 9/10 Release: In this Solaris release, a system event or *sysevent* is provided when an a disk is replaced with larger disk or the disks in the pool are replaced with larger disks. ZFS has been enhanced to recognize these events and adjusts the pool based on the new size of the disk, depending on the setting of the `autoexpand` property. You can use the `autoexpand` pool property to enable or disable automatic pool expansion when a larger disk replaces a smaller disk.

These features enable you to increase the pool size without having to export and import pool or reboot the system.

For example, the `autoexpand` property is enabled on the `tank` pool.

```
# zpool set autoexpand=on tank
```

Or, you can create the pool with the `autoexpand` property enabled.

```
# zpool create -o autoexpand=on tank c1t13d0
```

The `autoexpand` property is disabled by default so you can decide whether you want the pool size expanded when a larger disk replaces a smaller disk.

The pool size also be expanded by using the `zpool online -e` command. For example:

```
# zpool online -e tank c1t6d0
```

Or, you can reset the `autoexpand` property after the larger disk is attached or made available by using the `zpool replace` feature. For example, the following pool is created with one 8-GB disk (`c0t0d0`). The 8-GB disk is replaced with a 16-GB disk (`c1t13d0`), but the pool size is not expanded until the `autoexpand` property is enabled.

```
# zpool create pool c0t0d0
# zpool list
NAME  SIZE  ALLOC  FREE   CAP  HEALTH  ALROOT
pool  8.44G 76.5K  8.44G   0%  ONLINE  -
# zpool replace pool c0t0d0 c1t13d0
# zpool list
NAME  SIZE  ALLOC  FREE   CAP  HEALTH  ALROOT
pool  8.44G 91.5K  8.44G   0%  ONLINE  -
# zpool set autoexpand=on pool
# zpool list
NAME  SIZE  ALLOC  FREE   CAP  HEALTH  ALROOT
pool  16.8G 91.5K  16.8G   0%  ONLINE  -
```

Another way to expand the LUN in the above example without enabling the `autoexpand` property, is to use the `zpool online -e` command even though the device is already online. For example:

```
# zpool create tank c0t0d0
# zpool list tank
NAME  SIZE  ALLOC  FREE   CAP  HEALTH  ALTRoot
tank  8.44G 76.5K  8.44G   0%  ONLINE  -
# zpool replace tank c0t0d0 c1t13d0
# zpool list tank
NAME  SIZE  ALLOC  FREE   CAP  HEALTH  ALTRoot
tank  8.44G 91.5K  8.44G   0%  ONLINE  -
# zpool online -e tank c1t13d0
# zpool list tank
NAME  SIZE  ALLOC  FREE   CAP  HEALTH  ALTRoot
tank  16.8G  90K  16.8G   0%  ONLINE  -
```

Additional device replacement enhancements in this release include the following features:

- In previous releases, ZFS was not able to replace an existing disk with another disk or attach a disk if the replacement disk was a slightly different size. In this release, you can replace an existing disk with another disk or attach a new disk that is nominally the same size provided that the pool is not already full.
- In this release, you do not need to reboot the system or export and import a pool to expand the pool size. As described above, you can enable the `autoexpand` property or use the `zpool online -e` command to expand the pool size.

For more information about replacing devices, see [“Replacing Devices in a Storage Pool”](#) on page 85.

ZFS and Flash Installation Support

Solaris 10 10/09 Release: In this Solaris release, you can set up a JumpStart profile to identify a flash archive of a ZFS root pool. For more information, see [“Installing a ZFS Root File System \(Oracle Solaris Flash Archive Installation\)”](#) on page 122.

ZFS User and Group Quotas

Solaris 10 10/09 Release: In previous Solaris releases, you could apply quotas and reservations to ZFS file systems to manage and reserve disk space.

In this Solaris release, you can set a quota on the amount of disk space consumed by files that are owned by a particular user or group. You might consider setting user and group quotas in an environment with a large number of users or groups.

You can set a user quota by using the `zfs userquota` property. To set a group quota, use the `zfs groupquota` property. For example:

```
# zfs set userquota@user1=5G tank/data
# zfs set groupquota@staff=10G tank/staff/admins
```

You can display a user's or a group's current quota setting as follows:

```
# zfs get userquota@user1 tank/data
NAME      PROPERTY      VALUE      SOURCE
tank/data userquota@user1 5G         local
# zfs get groupquota@staff tank/staff/admins
NAME      PROPERTY      VALUE      SOURCE
tank/staff/admins groupquota@staff 10G       local
```

Display general quota information as follows:

```
# zfs userspace tank/data
TYPE      NAME  USED  QUOTA
POSIX User root   3K    none
POSIX User user1  0     5G

# zfs groupspace tank/staff/admins
TYPE      NAME  USED  QUOTA
POSIX Group root   3K    none
POSIX Group staff  0     10G
```

You can display an individual user's disk space usage by viewing the `userused@user` property. A group's disk space usage can be viewed by using the `groupused@group` property. For example:

```
# zfs get userused@user1 tank/staff
NAME      PROPERTY      VALUE      SOURCE
tank/staff userused@user1 213M       local
# zfs get groupused@staff tank/staff
NAME      PROPERTY      VALUE      SOURCE
tank/staff groupused@staff 213M       local
```

For more information about setting user quotas, see [“Setting ZFS Quotas and Reservations” on page 196](#).

ZFS ACL Pass Through Inheritance for Execute Permission

Solaris 10 10/09 Release: In previous Solaris releases, you could apply ACL inheritance so that all files are created with `0664` or `0666` permissions. In this release, if you want to optionally include the execute bit from the file creation mode into the inherited ACL, you can set the `aclinherit` mode to pass the execute permission to the inherited ACL.

If `aclinherit=passthrough-x` is enabled on a ZFS dataset, you can include execute permission for an output file that is generated from `cc` or `gcc` compiler tools. If the inherited ACL does not include execute permission, then the executable output from the compiler won't be executable until you use the `chmod` command to change the file's permissions.

For more information, see [Example 8–12](#).

ZFS Property Enhancements

Solaris 10 10/09 and Oracle Solaris 10 9/10: The following ZFS file system enhancements are included in these releases.

- ZFS Snapshot Stream Property Enhancements** – You can set a received property that is different from its local property setting. For example, you might receive a stream with the compression property disabled, but you want compression enabled in the receiving file system. This means that the received stream has a received compression value of `off` and a local compression value of `on`. Since the local value overrides the received value, you don't have to worry about the setting on the sending side replacing the received side value. The `zfs get` command shows the effective value of the compression property under the `VALUE` column.

New ZFS command options and properties to support send and local property values are as follows:

- 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.

In addition, 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    -
```

- Setting ZFS file system properties at pool creation time** – You can set ZFS file system properties when a storage pool is created. In the following example, compression is enabled on the ZFS file system that is created when the pool is created:

```
# zpool create -O compression=on pool mirror c0t1d0 c0t2d0
```

- **Setting cache properties on a ZFS file system** – Two new ZFS file system properties enable you to control what is cached in the primary cache (ARC) and the secondary cache (L2ARC). The cache properties are set as follows:
 - `primarycache` – Controls what is cached in the ARC.
 - `secondarycache` – Controls what is cached in the L2ARC.
 - Possible values for both properties – `all`, `none`, and `metadata`. If set to `all`, both user data and metadata are cached. If set to `none`, neither user data nor metadata is cached. If set to `metadata`, only metadata is cached. The default is `all`.

You can set these properties on an existing file system or when a file system is created. For example:

```
# zfs set primarycache=metadata tank/datab
# zfs create -o primarycache=metadata tank/newdatab
```

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.

- **Viewing disk space accounting properties** – New read-only file system properties help you identify disk space usage for clones, file systems, and volumes, and snapshots. The properties are as follows:
 - `usedbychildren` – Identifies the amount of disk space that is used by children of this dataset, which would be freed if all the dataset's children were destroyed. The property abbreviation is `usedchild`.
 - `usedbydataset` – Identifies the amount of disk space that is used by this dataset itself, which would be freed if the dataset was destroyed, after first destroying any snapshots and removing any reservation. The property abbreviation is `usedds`.
 - `usedbyreservation` – Identifies the amount of disk space that is used by a reservation set on this dataset, which would be freed if the reservation was removed. The property abbreviation is `usedrefreserv`.
 - `usedbysnapshots` – Identifies the amount of disk space that is consumed by snapshots of this dataset, which would be freed if all of this dataset's snapshots were destroyed. Note that this is not the sum of the snapshots' used properties, because disk space can be shared by multiple snapshots. The property abbreviation is `usedsnap`.

These new properties break down the value of the `used` property into the various elements that consume disk space. In particular, the value of the `used` property breaks down as follows:

```
used property = usedbychildren + usedbydataset + usedbyreservation + usedbysnapshots
```

You can view these properties by using the `zfs list -o space` command. For example:

```
$ zfs list -o space
NAME                AVAIL  USED  USED SNAP  USED DS  USED REFRESERV  USED CHILD
rpool               25.4G  7.79G  0          64K      0              7.79G
rpool/ROOT          25.4G  6.29G  0          18K      0              6.29G
rpool/ROOT/snv_98  25.4G  6.29G  0          6.29G    0              0
rpool/dump          25.4G  1.00G  0          1.00G    0              0
rpool/export        25.4G  38K    0          20K      0              18K
rpool/export/home   25.4G  18K    0          18K      0              0
rpool/swap          25.8G  512M   0          111M     401M          0
```

The preceding command is equivalent to the `zfs list`

`-o name,avail,used,usedsnap,usedds,usedrefreserv,usedchild -t filesystem,volume` command.

- **Listing snapshots** – The `listsnapshots` pool property controls whether snapshot information is displayed by the `zfs list` command. The default value is `on`, which means snapshot information is displayed by default.

If your system has many ZFS snapshots and you wish to disable the display of snapshot information in the `zfs list` command, disable the `listsnapshots` property as follows:

```
# zpool get listsnapshots pool
NAME PROPERTY VALUE SOURCE
pool listsnapshots on default
# zpool set listsnapshots=off pool
```

If you disable the `listsnapshots` property, you can use the `zfs list -t snapshots` command to list snapshot information. For example:

```
# zfs list -t snapshot
NAME                USED  AVAIL  REFER  MOUNTPOINT
pool/home@today     16K   -      22K   -
pool/home/user1@today 0     -      18K   -
pool/home/user2@today 0     -      18K   -
pool/home/user3@today 0     -      18K   -
```

ZFS Log Device Recovery

Solaris 10 10/09 Release: In this release, 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.

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.
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	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

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.

Using Cache Devices in Your ZFS Storage Pool

Solaris 10 10/09 Release: In this release, when you create a pool, you can specify *cache devices*, which are used to cache storage pool data.

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.

One or more cache devices can be specified when the pool is created. For example:

```
# zpool create pool mirror c0t2d0 c0t4d0 cache c0t0d0
# zpool status pool
pool: pool
state: ONLINE
scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM	
pool	ONLINE	0	0	0	
mirror	ONLINE	0	0	0	
c0t2d0	ONLINE	0	0	0	
c0t4d0	ONLINE	0	0	0	
cache					
c0t0d0	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.

For more information, see “[Creating a ZFS Storage Pool With Cache Devices](#)” on page 67 and [Example 4–4](#).

Zone Migration in a ZFS Environment

Solaris 10 5/09 Release: This release extends support for migrating zones in a ZFS environment with Oracle Solaris Live Upgrade. For more information, see “[Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones \(at Least Solaris 10 5/09\)](#)” on page 139.

For a list of known issues with this release, see the Solaris 10 5/09 release notes.

ZFS Installation and Boot Support

Solaris 10 10/08 Release: This release enables you to install and boot a ZFS root file system. You can use the initial installation option or the JumpStart feature to install a ZFS root file system. Or, you can use Oracle Solaris Live Upgrade to migrate a UFS root file system to a ZFS root file system. ZFS support for swap and dump devices is also provided. For more information, see [Chapter 5, “Installing and Booting an Oracle Solaris ZFS Root File System.”](#)

For a list of known issues with this release, go to the following site:

<http://hub.opensolaris.org/bin/view/Community+Group+zfs/boot>

Also, see the Solaris 10 10/08 release notes.

Rolling Back a Dataset Without Unmounting

Solaris 10 10/08 Release: This release enables you to roll back a dataset without unmounting it first. Thus, the `zfs rollback -f` option is no longer needed to force an unmount operation. The `-f` option is no longer supported and is ignored, if specified.

Enhancements to the `zfs send` Command

Solaris 10 10/08 Release: This release includes the following enhancements to the `zfs send` command. Using this command, you can now perform the following tasks:

- Send all incremental streams from one snapshot to a cumulative snapshot. For example:

```
# zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
pool                                428K  16.5G  20K    /pool
pool/fs                              71K   16.5G  21K    /pool/fs
pool/fs@snapA                        16K   -      18.5K  -
pool/fs@snapB                        17K   -      20K    -
pool/fs@snapC                        17K   -      20.5K  -
pool/fs@snapD                         0     -      21K    -
# zfs send -I pool/fs@snapA pool/fs@snapD > /snaps/fs@combo
```

This syntax sends all incremental snapshots between fs@snapA to fs@snapD to fs@combo.

- 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. For example:

```
# zfs send -I pool/fs@snap1 pool/clone@snapA > /snaps/fsc clonesnap-I
.
.
# zfs receive -F pool/clone < /snaps/fsc clonesnap-I
```

- Send a replication stream of all descendent file systems, up to the named snapshots. When received, all properties, snapshots, descendent file systems, and clones are preserved. For example:

```
# zfs send -R pool/fs@snap > snaps/fs-R
```

For an extended example, see [Example 7-1](#).

- Send an incremental replication stream. For example:

```
# zfs send -R -[iI] @snapA pool/fs@snapD
```

For an extended example, see [Example 7-1](#).

For more information, see “[Sending and Receiving Complex ZFS Snapshot Streams](#)” on [page 214](#).

ZFS Quotas and Reservations for File System Data Only

Solaris 10 10/08 Release : In addition to the existing ZFS quota and reservation features, this release includes dataset quotas and reservations that do not include descendents, such as snapshots and clones, in the disk space accounting.

- The `refquota` property enforces a hard limit on the amount of disk space that a dataset can consume. This hard limit does not include disk space used by descendents, such as snapshots and clones.
- The `refreservation` property sets the minimum amount of disk space that is guaranteed for a dataset, not including its descendents.

For example, you can set a 10-GB `refquota` limit for `studentA` that sets a 10-GB hard limit of *referenced* disk space. For additional flexibility, you can set a 20-GB quota that enables you to manage `studentA`'s snapshots.

```
# zfs set refquota=10g tank/studentA
# zfs set quota=20g tank/studentA
```

For more information, see [“Setting ZFS Quotas and Reservations”](#) on page 196.

ZFS Storage Pool Properties

Solaris 10 10/08 Release: ZFS storage pool properties were introduced in an earlier release. This release provides two properties, `cachefile` and `failmode`.

The following describes the new storage pool properties in this release:

- The `cachefile` property – This property 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 in a different location that can be imported later by using the `zpool import -c` command. For most ZFS configurations, this property would not be used.

The `cachefile` property is not persistent and is not stored on disk. This property replaces the temporary property that was used to indicate that pool information should not be cached in previous Solaris releases.

- The `failmode` property – This property determines the behavior of a catastrophic pool failure due to a loss of device connectivity or the failure of all devices in the pool. The `failmode` property can be set to these values: `wait`, `continue`, or `panic`. The default value is `wait`, which means you must reconnect the device or replace a failed device, and then clear the error with the `zpool clear` command.

The `failmode` property is set like other settable ZFS properties, which can be set either before or after the pool is created. For example:

```
# zpool set failmode=continue tank
# zpool get failmode tank
NAME PROPERTY VALUE SOURCE
tank failmode continue local

# zpool create -o failmode=continue users mirror c0t1d0 c1t1d0
```

For a description of pool properties, see [Table 4-1](#).

ZFS Command History Enhancements (zpool history)

Solaris 10 10/08 Release: The `zpool history` command has been enhanced to provide the following new features:

- ZFS file system event information is now displayed. For example:

```
# zpool history
History for 'rpool':
2010-06-23.09:30:12 zpool create -f -o failmode=continue -R /a -m legacy -o
cachefile=/tmp/root/etc/zfs/zpool.cache rpool c1t0d0s0
2010-06-23.09:30:13 zfs set canmount=noauto rpool
2010-06-23.09:30:13 zfs set mountpoint=/rpool rpool
2010-06-23.09:30:13 zfs create -o mountpoint=legacy rpool/ROOT
2010-06-23.09:30:14 zfs create -b 8192 -V 2048m rpool/swap
2010-06-23.09:30:14 zfs create -b 131072 -V 1024m rpool/dump
2010-06-23.09:30:15 zfs create -o canmount=noauto rpool/ROOT/zfsBE
2010-06-23.09:30:16 zpool set bootfs=rpool/ROOT/zfsBE rpool
2010-06-23.09:30:16 zfs set mountpoint=/ rpool/ROOT/zfsBE
2010-06-23.09:30:16 zfs set canmount=on rpool
2010-06-23.09:30:16 zfs create -o mountpoint=/export rpool/export
2010-06-23.09:30:17 zfs create rpool/export/home
```

- The `-l` option can be used 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 rpool
History for 'tank':
2010-06-24.13:07:58 zpool create tank mirror c2t2d0 c2t5d0 [user root on neo:global]
2010-06-24.13:08:23 zpool scrub tank [user root on neo:global]
2010-06-24.13:38:42 zpool clear tank [user root on neo:global]
2010-06-29.11:44:18 zfs create tank/home [user root on neo:global]
2010-06-29.13:28:51 zpool clear tank c2t5d0 [user root on neo:global]
2010-06-30.14:07:40 zpool add tank spare c2t1d0 [user root on neo:global]
```

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

```
# zpool history -i tank
History for 'tank':
2010-06-24.13:07:58 zpool create tank mirror c2t2d0 c2t5d0
2010-06-24.13:08:23 [internal pool scrub txg:6] func=1 mintxg=0 maxtxg=6
2010-06-24.13:08:23 [internal pool create txg:6] pool spa 22; zfs spa 22; zpl 4; uts neo 5.10 Generic_142909-13 s
2010-06-24.13:08:23 [internal pool scrub done txg:6] complete=1
2010-06-24.13:08:23 zpool scrub tank
2010-06-24.13:38:42 zpool clear tank
2010-06-24.13:38:42 [internal pool scrub txg:69] func=1 mintxg=3 maxtxg=8
2010-06-24.13:38:42 [internal pool scrub done txg:69] complete=1
2010-06-29.11:44:18 [internal create txg:14241] dataset = 34
2010-06-29.11:44:18 zfs create tank/home
2010-06-29.13:28:51 zpool clear tank c2t5d0
2010-06-30.14:07:40 zpool add tank spare c2t1d0
```

For more information about using the `zpool history` command, see [“Resolving Problems With ZFS” on page 271](#).

Upgrading ZFS File Systems (zfs upgrade)

Solaris 10 10/08 Release: The `zfs upgrade` command is included in this release to provide future ZFS file system enhancements to existing file systems. ZFS storage pools have a similar upgrade feature to provide pool enhancements to existing storage pools.

For example:

```
# zfs upgrade
This system is currently running ZFS filesystem version 3.

All filesystems are formatted with the current version.
```

Note – File systems that are upgraded and any streams created from those upgraded file systems by the `zfs send` command are not accessible on systems that are running older software releases.

ZFS Delegated Administration

Solaris 10 10/08 Release: In this release, you can grant fine-grained permissions to allow nonprivileged users to perform ZFS administration tasks.

You can use the `zfs allow` and `zfs unallow` commands to delegate and remove permissions.

You can modify delegated administration with the 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.

For more information, see [Chapter 9, “Oracle Solaris ZFS Delegated Administration,”](#) and [zfs\(1M\)](#).

Setting Up Separate ZFS Log Devices

Solaris 10 10/08 Release: 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

storage pool. In this Solaris release, you can decide if you want the ZIL blocks to continue to be allocated from the main storage pool or from a separate log device. Better performance might be possible by using separate intent log devices in your ZFS storage pool, such as with NVRAM or a dedicated disk.

Log devices for the ZFS intent log are not related to database log files.

You can set up a ZFS log device when the storage pool is created or after the pool is created. For examples of setting up log devices, see “[Creating a ZFS Storage Pool With Log Devices](#)” on page 66 and “[Adding Devices to a Storage Pool](#)” on page 73.

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.

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

- 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.sun.com/perrin/entry/slog_blog_or_blogginng_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, attached, detached, imported, and exported as part of the larger storage pool. Log devices can be removed starting in the Solaris 10 9/10 release.
- 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.

Creating Intermediate ZFS Datasets

Solaris 10 10/08 Release: You can use the `-p` option with the `zfs create`, `zfs clone`, and `zfs rename` commands to quickly create a non-existent intermediate dataset, if it doesn't already exist.

In the following example, ZFS datasets (`users/area51`) are created in the `datab` storage pool.

```
# zfs list
NAME                                USED AVAIL REFER MOUNTPOINT
datab                               106K 16.5G  18K  /datab
# zfs create -p -o compression=on datab/users/area51
```

If the intermediate dataset already exists during the create operation, the operation completes successfully.

Properties specified apply to the target dataset, not to the intermediate dataset. For example:

```
# zfs get mountpoint,compression datab/users/area51
NAME                                PROPERTY  VALUE                                SOURCE
datab/users/area51                 mountpoint /datab/users/area51                default
datab/users/area51                 compression on                                       local
```

The intermediate dataset is created with the default mount point. Any additional properties are disabled for the intermediate dataset. For example:

```
# zfs get mountpoint,compression datab/users
NAME                                PROPERTY  VALUE                                SOURCE
datab/users                         mountpoint /datab/users                        default
datab/users                         compression off                                   default
```

For more information, see [zfs\(1M\)](#).

ZFS Hot-Plugging Enhancements

Solaris 10 10/08 Release: In this release, ZFS more effectively responds to devices that are removed and can now automatically identify devices that are inserted.

- You can replace an existing device with an equivalent device without having to use the `zpool replace` command.

The `autoreplace` property controls automatic device replacement. If set to `off`, device replacement must be initiated by the administrator by using the `zpool replace` command. If set to `on`, any new device that is found in the same physical location as a device that previously belonged to the pool is automatically formatted and replaced. The default behavior is `off`.

- 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.

For more information, see [zpool\(1M\)](#).

Recursively Renaming ZFS Snapshots (`zfs rename -r`)

Solaris 10 10/08 Release: You can recursively rename all descendent ZFS snapshots by using the `zfs rename -r` command. For example:

First, a snapshot of a set of ZFS file systems is created.

```
# zfs snapshot -r users/home@today
# zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
users                                216K  16.5G  20K    /users
users/home                            76K  16.5G  22K    /users/home
users/home@today                       0     -     22K    -
users/home/markm                       18K  16.5G  18K    /users/home/markm
users/home/markm@today                  0     -     18K    -
users/home/marks                       18K  16.5G  18K    /users/home/marks
users/home/marks@today                  0     -     18K    -
users/home/neil                        18K  16.5G  18K    /users/home/neil
users/home/neil@today                   0     -     18K    -
```

Then, the snapshots are renamed the following day.

```
# zfs rename -r users/home@today @yesterday
# zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
users                                216K  16.5G  20K    /users
users/home                            76K  16.5G  22K    /users/home
users/home@yesterday                   0     -     22K    -
users/home/markm                       18K  16.5G  18K    /users/home/markm
users/home/markm@yesterday              0     -     18K    -
users/home/marks                       18K  16.5G  18K    /users/home/marks
users/home/marks@yesterday              0     -     18K    -
users/home/neil                        18K  16.5G  18K    /users/home/neil
users/home/neil@yesterday               0     -     18K    -
```

A snapshot is the only type of dataset that can be renamed recursively.

For more information about snapshots, see “[Overview of ZFS Snapshots](#)” on page 203 and this blog entry that describes how to create rolling snapshots:

http://blogs.sun.com/mmusante/entry/rolling_snapshots_made_easy

gzip Compression Is Available for ZFS

Solaris 10 10/08 Release: In this Solaris release, you can set `gzip` compression on ZFS file systems, in addition to `lzjb` compression. You can specify compression as `gzip`, or `gzip-N`, where `N` equals 1 through 9. For example:

```
# zfs create -o compression=gzip users/home/snapshots
# zfs get compression users/home/snapshots
NAME                PROPERTY    VALUE          SOURCE
users/home/snapshots  compression gzip           local
# zfs create -o compression=gzip-9 users/home/oldfiles
# zfs get compression users/home/oldfiles
NAME                PROPERTY    VALUE          SOURCE
users/home/oldfiles   compression gzip-9         local
```

For more information about setting ZFS properties, see “Setting ZFS Properties” on page 185.

Storing Multiple Copies of ZFS User Data

Solaris 10 10/08 Release: 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 Solaris 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.

For a detailed description of how storing multiple copies on a system with a single-disk pool or a multiple-disk pool might impact overall data protection, see this blog:

http://blogs.sun.com/relling/entry/zfs_copies_and_data_protection

For more information about setting ZFS properties, see [“Setting ZFS Properties”](#) on page 185.

Improved zpool status Output

Solaris 10 8/07 Release: You can use the `zpool status -v` command to display a list of files with persistent errors. Previously, you had to use the `find -inum` command to identify the file names from the list of displayed inodes.

For more information about displaying a list of files with persistent errors, see [“Repairing a Corrupted File or Directory”](#) on page 288.

ZFS and Solaris iSCSI Improvements

Solaris 10 8/07 Release: In this Solaris release, you can create a ZFS volume as a Solaris iSCSI target device by setting the `shareiscsi` property on the ZFS volume. This method is a convenient way to quickly set up a Solaris iSCSI target. For example:

```
# zfs create -V 2g tank/volumes/v2
# zfs set shareiscsi=on tank/volumes/v2
# iscsitadm list target
Target: tank/volumes/v2
  iSCSI Name: iqn.1986-03.com.sun:02:984fe301-c412-ccc1-cc80-cf9a72aa062a
  Connections: 0
```

After the iSCSI target is created, you can set up the iSCSI initiator. For information about setting up a Solaris iSCSI initiator, see [Chapter 14, “Configuring Oracle Solaris iSCSI Targets and Initiators \(Tasks\)”](#), in *System Administration Guide: Devices and File Systems*.

For more information about managing a ZFS volume as an iSCSI target, see [“Using a ZFS Volume as a Solaris iSCSI Target”](#) on page 258.

ZFS Command History (zpool history)

Solaris 10 8/07 Release: In this Solaris release, ZFS automatically logs successful `zfs` and `zpool` commands that modify pool state information. For example:

```
# zpool history
History for 'newpool':
2007-04-25.11:37:31 zpool create newpool mirror c0t8d0 c0t10d0
2007-04-25.11:37:46 zpool replace newpool c0t10d0 c0t9d0
2007-04-25.11:38:04 zpool attach newpool c0t9d0 c0t11d0
2007-04-25.11:38:09 zfs create newpool/user1
2007-04-25.11:38:15 zfs destroy newpool/user1

History for 'tank':
2007-04-25.11:46:28 zpool create tank mirror c1t0d0 c2t0d0 mirror c3t0d0 c4t0d0
```

This feature enables you or Oracle support personnel to identify the *actual* ZFS commands that were executed to troubleshoot an error scenario.

You can identify a specific storage pool with the `zpool history` command. For example:

```
# zpool history newpool
History for 'newpool':
2007-04-25.11:37:31 zpool create newpool mirror c0t8d0 c0t10d0
2007-04-25.11:37:46 zpool replace newpool c0t10d0 c0t9d0
2007-04-25.11:38:04 zpool attach newpool c0t9d0 c0t11d0
2007-04-25.11:38:09 zfs create newpool/user1
2007-04-25.11:38:15 zfs destroy newpool/user1
```

In this Solaris release, the `zpool history` command does not record *user-ID*, *hostname*, or *zone-name*. However, this information is recorded starting in the Solaris 10 10/08 release. For more information, see [“ZFS Command History Enhancements \(zpool history\)” on page 29](#).

For more information about troubleshooting ZFS problems, see [“Resolving Problems With ZFS” on page 271](#).

ZFS Property Improvements

ZFS `xattr` Property

Solaris 10 8/07 Release: You can use the `xattr` property to disable or enable extended attributes for a specific ZFS file system. The default value is `on`. For a description of ZFS properties, see [“Introducing ZFS Properties” on page 171](#).

ZFS `canmount` Property

Solaris 10 8/07 Release: The new `canmount` property enables you to specify whether a dataset can be mounted by using the `zfs mount` command. For more information, see [“`canmount` Property” on page 180](#).

ZFS User Properties

Solaris 10 8/07 Release: In addition to the standard native properties that can be used to either export internal statistics or control ZFS file system behavior, ZFS provides 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.

For more information, see [“ZFS User Properties” on page 182](#).

Setting Properties When Creating ZFS File Systems

Solaris 10 8/07 Release: In this Solaris release, you can set properties when you create a file system, not just after the file system is created.

The following examples illustrate equivalent syntax:

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

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

Displaying All ZFS File System Information

Solaris 10 8/07 Release: In this Solaris release, you can use various forms of the `zfs get` command to display information about all datasets if you do not specify a dataset or if you specify `all`. In previous releases, all dataset information was not retrievable with the `zfs get` command.

For example:

```
# zfs get -s local all
tank/home                atime                off                  local
tank/home/bonwick        atime                off                  local
tank/home/marks          quota                50G                 local
```

New `zfs receive -F` Option

Solaris 10 8/07 Release: In this Solaris release, you can use the new `-F` option to the `zfs receive` command to force a rollback of the file system to the most recent snapshot before the receive is initiated. Using this option might be necessary when the file system is modified after a rollback occurs but before the receive is initiated.

For more information, see [“Receiving a ZFS Snapshot” on page 213](#).

Recursive ZFS Snapshots

Solaris 10 11/06 Release: When you use the `zfs snapshot` command to create a file system snapshot, you can use the `-r` option to recursively create snapshots for all descendent file systems. In addition, you can use the `-r` option to recursively destroy all descendent snapshots when a snapshot is destroyed.

Recursive ZFS snapshots are created quickly as one atomic operation. The snapshots are created together (all at once) or not created at all. The benefit of such an operation is that the snapshot data is always taken at one consistent time, even across descendent file systems.

For more information, see [“Creating and Destroying ZFS Snapshots” on page 204](#).

Double-Parity RAID-Z (raidz2)

Solaris 10 11/06 Release: A redundant RAID-Z configuration can now have either a single- or double-parity configuration, which means that one or two device failures, respectively, can be sustained, without any data loss. You can specify the `raidz2` keyword for a double-parity RAID-Z configuration. Or, you can specify the `raidz` or `raidz1` keyword for a single-parity RAID-Z configuration.

For more information, see [“Creating a RAID-Z Storage Pool”](#) on page 65 or `zpool(1M)`.

Hot Spares for ZFS Storage Pool Devices

Solaris 10 11/06 Release: The ZFS hot spares feature enables you to identify disks that could be used to replace a failed or faulted device in one or more storage pools. Designating a device as a *hot spare* means that if an active device in the pool fails, the hot spare automatically replaces the failed device. Or, you can manually replace a device in a storage pool with a hot spare.

For more information, see [“Designating Hot Spares in Your Storage Pool”](#) on page 87 and `zpool(1M)`.

Replacing a ZFS File System With a ZFS Clone (`zfs promote`)

Solaris 10 11/06 Release: The `zfs promote` command enables you to replace an existing ZFS file system with a clone of that file system. This feature is helpful when you want to run tests on an alternative version of a file system and then make that alternative version the active file system.

For more information, see [“Replacing a ZFS File System With a ZFS Clone”](#) on page 210 and `zfs(1M)`.

Upgrading ZFS Storage Pools (`zpool upgrade`)

Solaris 10 6/06 Release: You can upgrade your storage pools to a newer version of ZFS to take advantage of the latest features by using the `zpool upgrade` command. In addition, the `zpool status` command has been modified to notify you when your pools are running older versions of ZFS.

For more information, see [“Upgrading ZFS Storage Pools”](#) on page 109 and `zpool(1M)`.

If you want to use the ZFS Administration console on a system with a pool from a previous Solaris release, ensure that you upgrade your pools before using the console. To determine if your pools need to be upgraded, use the `zpool status` command. For information about the ZFS Administration console, see [“ZFS Web-Based Management”](#) on page 40.

ZFS Backup and Restore Commands Are Renamed

Solaris 10 6/06 Release: In this Solaris release, the `zfs backup` and `zfs restore` commands are renamed to `zfs send` and `zfs receive` to more accurately describe their functions. These commands send and receive ZFS data stream representations.

For more information about these commands, see [“Sending and Receiving ZFS Data” on page 211](#).

Recovering Destroyed Storage Pools

Solaris 10 6/06 Release: This release includes the `zpool import -D` command, which enables you to recover pools that were previously destroyed with the `zpool destroy` command.

For more information, see [“Recovering Destroyed ZFS Storage Pools” on page 107](#).

ZFS Is Integrated With Fault Manager

Solaris 10 6/06 Release: This release includes a ZFS diagnostic engine that is capable of diagnosing and reporting pool failures and device failures. Checksum, I/O, device, and pool errors associated with pool or device failures are also reported.

The diagnostic engine does not include predictive analysis of checksum and I/O errors, nor does it include proactive actions based on fault analysis.

If a ZFS failure occurs, you might see a message similar to the following:

```
SUNW-MSG-ID: ZFS-8000-D3, TYPE: Fault, VER: 1, SEVERITY: Major
EVENT-TIME: Wed Jun 30 14:53:39 MDT 2010
PLATFORM: SUNW,Sun-Fire-880, CSN: -, HOSTNAME: neo
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: 504a1188-b270-4ab0-af4e-8a77680576b8
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.
```

By reviewing the recommended action, which is to follow the more specific directions in the `zpool status` command, you can quickly identify and resolve the failure.

For an example of recovering from a reported ZFS problem, see [“Resolving a Missing Device” on page 276](#).

The `zpool clear` Command

Solaris 10 6/06 Release: This release includes the `zpool clear` command for clearing error counts associated with a device or a pool. Previously, error counts were cleared when a device in a pool was brought online with the `zpool online` command. For more information, see [“Clearing Storage Pool Device Errors” on page 84](#) and `zpool(1M)`.

Compact NFSv4 ACL Format

Solaris 10 6/06 Release: In this release, you can set and display NFSv4 ACLs in two formats: verbose and compact. You can use the `chmod` command to set either ACL formats. You can use the `ls -V` command to display the compact ACL format. You can use the `ls -v` command to display the verbose ACL format.

For more information, see [“Setting and Displaying ACLs on ZFS Files in Compact Format” on page 239](#), `chmod(1)`, and `ls(1)`.

File System Monitoring Tool (`fsstat`)

Solaris 10 6/06 Release: A new file system monitoring tool, `fsstat`, reports file system operations. Activity can be reported by mount point or by file system type. The following example shows general ZFS file system activity:

```
$ fsstat zfs
new name name attr attr lookup rddir read read write write
file remov chng get set ops ops ops bytes ops bytes
7.82M 5.92M 2.76M 1.02G 3.32M 5.60G 87.0M 363M 1.86T 20.9M 251G zfs
```

For more information, see `fsstat(1M)`.

ZFS Web-Based Management

Solaris 10 6/06 Release: A web-based ZFS management tool, the ZFS Administration console, enables you to perform the following administrative tasks:

- Create a new storage pool.
- Add capacity to an existing pool.
- Move (export) a storage pool to another system.
- Import a previously exported storage pool to make it available on another system.
- View information about storage pools.
- Create a file system.

- Create a volume.
- Create a snapshot of a file system or a volume.
- Roll back a file system to a previous snapshot.

You can access the ZFS Administration console through a secure web browser at:

```
https://system-name:6789/zfs
```

If you type the appropriate URL and are unable to reach the ZFS Administration console, the server might not be started. To start the server, run the following command:

```
# /usr/sbin/smcwebserver start
```

If you want the server to run automatically when the system boots, run the following command:

```
# /usr/sbin/smcwebserver enable
```

Note – You cannot use the Solaris Management Console (smc) to manage ZFS storage pools or file systems.

What Is ZFS?

The ZFS file system is a revolutionary new file system that fundamentally changes the way file systems are administered, with features and benefits not found in any other file system 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 the 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:

alternate boot environment	A boot environment that is created by the <code>lucreate</code> command and possibly updated by the <code>luupgrade</code> command, but it is not the active or primary boot environment. The alternate boot environment can become the primary boot environment by running the <code>luactivate</code> command.
----------------------------	--

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	<p>A file system whose initial contents are identical to the contents of a snapshot.</p> <p>For information about clones, see “Overview of ZFS Clones” on page 209.</p>						
dataset	<p>A generic name for the following ZFS components: clones, file systems, snapshots, and volumes.</p> <p>Each dataset is identified by a unique name in the ZFS namespace. Datasets are identified using the following format:</p> <p><i>pool/path[@snapshot]</i></p> <table><tr><td><i>pool</i></td><td>Identifies the name of the storage pool that contains the dataset</td></tr><tr><td><i>path</i></td><td>Is a slash-delimited path name for the dataset component</td></tr><tr><td><i>snapshot</i></td><td>Is an optional component that identifies a snapshot of a dataset</td></tr></table> <p>For more information about datasets, see Chapter 6, “Managing Oracle Solaris ZFS File Systems.”</p>	<i>pool</i>	Identifies the name of the storage pool that contains the dataset	<i>path</i>	Is a slash-delimited path name for the dataset component	<i>snapshot</i>	Is an optional component that identifies a snapshot of a dataset
<i>pool</i>	Identifies the name of the storage pool that contains the dataset						
<i>path</i>	Is a slash-delimited path name for the dataset component						
<i>snapshot</i>	Is an optional component that identifies a snapshot of a dataset						
file system	<p>A ZFS dataset of type <code>filesystem</code> that is mounted within the standard system namespace and behaves like other file systems.</p> <p>For more information about file systems, see Chapter 6, “Managing Oracle Solaris ZFS File Systems.”</p>						
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	<p>A logical group of devices describing the layout and physical characteristics of the available storage. Disk space for datasets is allocated from a pool.</p> <p>For more information about storage pools, see Chapter 4, “Managing Oracle Solaris ZFS Storage Pools.”</p>						
primary boot environment	A boot environment that is used by the <code>lucreate</code> command to build the alternate boot environment. By default, the primary						

	boot environment is the current boot environment. This default can be overridden by using the <code>lucreate -s</code> option.
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 61 .
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 285 .
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 203 .
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 68 .
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 257 .

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 Hardware and Software Requirements and Recommendations” on page 47
- “Creating a Basic ZFS File System” on page 48
- “Creating a ZFS Storage Pool” on page 49
- “Creating a ZFS File System Hierarchy” on page 50

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 at least the Solaris 10 6/06 release or later 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 768 MB. However, for good ZFS performance, use at least one GB or more of memory.
- If you create a mirrored disk 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 60.

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 50.

Creating a 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 49](#), 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 57](#).

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 49](#), 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 60](#).

▼ 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 266](#).

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. Most systems require only a single pool, so you can pick any name that you prefer, but it must satisfy the naming requirements in [“ZFS Component Naming Requirements” on page 45](#).

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 a ZFS Storage Pool” on page 63](#). For more information about how device usage is determined, see [“Detecting In-Use Devices” on page 69](#).

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  ALTRoot
tank                 80G   137K   80G   0%   ONLINE  -
```

For more information about viewing pool status, see [“Querying ZFS Storage Pool Status” on page 95](#).

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, `bonwick` and `billm`, are created in “[How to Create ZFS File Systems](#)” on page 51.

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 51, 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 51, 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 `bonwick` is enforced.

For more information about properties, see “[Introducing ZFS Properties](#)” on page 171.

▼ 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 266.

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 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 171](#).

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/bonwick
# zfs create tank/home/billm
```

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 168](#).

For more information about mounting and sharing file systems, see [“Mounting and Sharing ZFS File Systems” on page 190](#).

5 Set the file system-specific properties.

In this example, user `bonwick` 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/bonwick
```

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/billm                      8K 67.0G    8K   /export/zfs/billm
tank/home/bonwick                     8K 10.0G    8K   /export/zfs/bonwick
```

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

For more information about viewing file system status, see [“Querying ZFS File System Information” on page 183](#).

For more information about how disk space is used and calculated, see [“ZFS Disk Space Accounting” on page 54](#).

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 53
- “ZFS Disk Space Accounting” on page 54
- “Out of Space Behavior” on page 54
- “Mounting ZFS File Systems” on page 55
- “Traditional Volume Management” on page 55
- “New Solaris ACL Model” on page 55

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 50.

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 196. 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 200. 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.

For additional details about ZFS disk space consumption as reported by the `du` and `df` commands, see:

<http://hub.opensolaris.org/bin/view/Community+Group+zfs/faq/#whydusize>

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 dataset. 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 and Sharing ZFS File Systems” on page 190](#).

Traditional Volume Management

As described in [“ZFS Pooled Storage” on page 41](#), 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 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 57
- “Replication Features of a ZFS Storage Pool” on page 60
- “Creating and Destroying ZFS Storage Pools” on page 63
- “Managing Devices in ZFS Storage Pools” on page 72
- “Managing ZFS Storage Pool Properties” on page 92
- “Querying ZFS Storage Pool Status” on page 95
- “Migrating ZFS Storage Pools” on page 103
- “Upgrading ZFS Storage Pools” on page 109

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 57
- “Using Slices in a ZFS Storage Pool” on page 59
- “Using Files in a ZFS Storage Pool” on page 60

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
```

To use a whole disk, the disk must be 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.

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 *System Administration Guide: Devices and File Systems*.

A disk that is intended for a ZFS root pool must be created with an SMI label, not an EFI label. You can relabel a disk with an SMI label by using the `format -e` command.

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 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, such as Solaris Volume Manager (SVM) or Veritas Volume Manager (VxVM). 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 the ZFS best practices site:

http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide

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.

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 clt1d0
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

Using Files in a ZFS Storage Pool

ZFS also allows you to use UFS files as virtual devices in your storage pool. This feature is aimed primarily at testing and enabling simple experimentation, not for production use. The reason is that **any use of files relies on the underlying file system for consistency**. If you create a ZFS pool backed by files on a UFS file system, then you are implicitly relying on UFS to guarantee correctness and synchronous semantics.

However, 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.

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 61](#)
- [“RAID-Z Storage Pool Configuration” on page 61](#)
- [“Self-Healing Data in a Redundant Configuration” on page 62](#)
- [“Dynamic Striping in a Storage Pool” on page 62](#)
- [“ZFS Hybrid Storage Pool” on page 62](#)

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 64](#).

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.sun.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. 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 c11t0d0  
c12t0d0 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 65](#).

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.sun.com/roch/entry/when_to_and_not_to

For additional information about RAID-Z storage pool recommendations, see the ZFS best practices site:

http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide

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 a ZFS Storage Pool” on page 63](#)
- [“Displaying Storage Pool Virtual Device Information” on page 68](#)
- [“Handling ZFS Storage Pool Creation Errors” on page 69](#)
- [“Destroying ZFS Storage Pools” on page 71](#)

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 a ZFS Storage Pool

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 45](#).

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 the following site:

http://www.solarisinternals.com/wiki/index.php/ZFS_Best_Practices_Guide

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 73.
- 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 77.
- 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 85.
- 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 77.
- 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 79.

You cannot outright remove a device that is not a log or a cache device from a mirrored storage pool. An RFE is filed for this feature.

Creating a ZFS Root Pool

You can install and boot from a ZFS root file system. Review the following root pool configuration information:

- 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, “Installing and Booting an Oracle Solaris ZFS Root File System.”](#)

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
# 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

```
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 73](#).
- 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 85](#).

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.
- You cannot outright remove a device that is not a log 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 61](#).

Creating a ZFS Storage Pool With Log Devices

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. For more information about ZFS log devices, see [“Setting Up Separate ZFS Log Devices” on page 30](#).

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

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

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.

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 provide 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 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 zeepool. 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.
Live upgrade	The disk is in use as an alternate boot environment for Oracle Solaris Live Upgrade.
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 dataset 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 dataset 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` dataset with a mount point of `/export/zfs`.

For more information about mount points, see [“Managing ZFS Mount Points” on page 190](#).

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 107](#).

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 100](#).

For more information about importing pools, see [“Importing ZFS Storage Pools” on page 106](#).

Managing Devices in ZFS Storage Pools

Most of the basic information regarding devices is covered in [“Components of a ZFS Storage Pool” on page 57](#). 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 73](#)
- [“Attaching and Detaching Devices in a Storage Pool” on page 77](#)
- [“Creating a New Pool By Splitting a Mirrored ZFS Storage Pool” on page 79](#)
- [“Onlining and Offlining Devices in a Storage Pool” on page 82](#)
- [“Clearing Storage Pool Device Errors” on page 84](#)
- [“Replacing Devices in a Storage Pool” on page 85](#)
- [“Designating Hot Spares in Your Storage Pool” on page 87](#)

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 69](#).

EXAMPLE 4-1 Adding Disks to a Mirrored ZFS Configuration

In the following example, another mirror is added to an existing mirrored ZFS configuration on Oracle's Sun Fire x4500 system.

```
# 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

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

```

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
    rzpool          ONLINE           0     0     0
      raidz1-0      ONLINE           0     0     0
        c1t0d0       ONLINE           0     0     0
        c1t2d0       ONLINE           0     0     0
    
```

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

```

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 mirrored storage pool. For more information about using log devices in your storage pool, see [“Setting Up Separate ZFS Log Devices” on page 30](#).

```

# 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 a unmirrored storage pool.

Log devices can be removed 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 only contains one log device, you would 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 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
c2t0d0	ONLINE	0	0	0
c2t1d0	ONLINE	0	0	0

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

```

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 67](#).

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 Create a Mirrored Root Pool \(Post Installation\)” on page 121](#).

If you are replacing a disk in a ZFS root pool, see [“How to Replace a Disk in the ZFS Root Pool” on page 159](#).

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.

Currently, this feature cannot be used to split a mirrored root pool.

You can use the `zpool split` command to detach disks from a mirrored ZFS storage pool to create a new pool with one of the detached 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, 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 dataset mount points and the new pool is created on the same system, then you will need to 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 should not be 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 composed of 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 recreate your original mirrored storage pool or convert your newly created pool into a mirrored storage pool. No way 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 will need to 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 is composed of three disks so that the original pool is comprised of 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.

You can use the `zpool offline` command when you need to temporarily disconnect storage. For example, if you need to physically disconnect an array from one set of Fibre Channel switches and connect the array to a different set, you can take offline the LUNs from the array that is used in the ZFS storage pools. After the array is reconnected and operational on the new set of switches, you can then bring the same LUNs online. Data that had been added to the storage pools while the LUNs were offline would resilver to the LUNs after they are brought back online.

This scenario is possible assuming that the systems in question can detect the storage after it is attached to the new switches, possibly through different controllers than before, and your pools are set up as RAID-Z or mirrored configurations.

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 85](#).

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 95](#).

For more information on device health, see [“Determining the Health Status of ZFS Storage Pools” on page 100](#).

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 Jun 30 14:53:39 MDT 2010  
PLATFORM: SUNW,Sun-Fire-880, CSN: -, HOSTNAME: neo  
SOURCE: zfs-diagnosis, REV: 1.0  
EVENT-ID: 504a1188-b270-4ab0-af4e-8a77680576b8  
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 276](#).

You can use the `zpool online -e` command to expand the pool's size when a smaller disk is replaced by a larger disk. By default, pool size is not expanded to its full size unless the `autoexpand pool` property is enabled. You can expand the pool size automatically when a smaller disk is replaced by a larger disk by using the `zpool online -e` command even if the disk is already online or if the disk 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 clear 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 279](#).

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. ZFS recognizes that the device is a different disk in the same location on some hardware. 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 will need to 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 the ZFS Root Pool” on page 159](#).

The following are the basic steps for replacing a disk:

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

```
# zpool replace tank c1t1d0
```

- Bring the disk online with the `zpool online` command.

On some systems, such as the Sun Fire x4500, 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 disk on a Sun Fire X4500 system, 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 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 greater in size than the device it is replacing is added to a pool, is not automatically expanded to its full size. The `autoexpand` pool property value determines whether a pool 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 a pool's size before or after the larger disk 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. For information about detaching a hot spare, see [“Activating and Deactivating Hot Spares in Your Storage Pool”](#) on page 88.

For more information about replacing devices, see [“Resolving a Missing Device” on page 276](#) and [“Replacing or Repairing a Damaged Device” on page 278](#).

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 one or more storage pools. 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.

You can manually replace a device with a hot spare by using the `zpool replace` command. See [Example 4–8](#).

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 4–9](#).
- By temporarily or permanently swapping in the hot spare. See [Example 4–10](#).

EXAMPLE 4–8 Manually Replacing a Disk With a Hot Spare

In this example, the `zpool replace` command is used to replace disk `c2t1d0` with the hot spare `c2t3d0`.

```
# zpool replace zeepool c2t1d0 c2t3d0
# zpool status zeepool
pool: zeepool
state: ONLINE
```

EXAMPLE 4-8 Manually Replacing a Disk With a Hot Spare *(Continued)*

```
scrub: resilver completed after 0h0m with 0 errors on Wed Jan 20 10:00:50 2010
config:
```

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

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

Then, detach the disk c2t1d0.

```
# zpool detach zpool c2t1d0
# zpool status zpool
pool: zpool
state: ONLINE
scrub: resilver completed after 0h0m with 0 errors on Wed Jan 20 10:00:50 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM	
zpool	ONLINE	0	0	0	
mirror-0	ONLINE	0	0	0	
c1t2d0	ONLINE	0	0	0	
c2t3d0	ONLINE	0	0	0	90K resilvered

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

EXAMPLE 4-9 Detaching a Hot Spare After the Failed Disk Is Replaced

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

```
# zpool replace zpool c2t1d0
# zpool status zpool
pool: zpool
state: ONLINE
scrub: resilver completed after 0h0m with 0 errors on Wed Jan 20 10:08:44 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM	
zpool	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			

EXAMPLE 4-9 Detaching a Hot Spare After the Failed Disk Is Replaced (Continued)

```
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 4-10 Detaching a Failed Disk and Using the Hot Spare

If you want to replace a failed disk by temporarily or permanently swap 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	
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
```

EXAMPLE 4-10 Detaching a Failed Disk and Using the Hot Spare *(Continued)*

```
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 mpool
NAME PROPERTY VALUE SOURCE
pool size 68G -
pool capacity 0% -
pool altroot - default
pool health ONLINE -
pool guid 601891032394735745 default
pool version 22 default
pool bootfs - default
pool delegation on default
pool autoreplace off default
pool cachefile - default
pool failmode wait default
pool listsnapshots on default
pool autoexpand off default
pool free 68.0G -
pool allocated 76.5K -
```

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

```
# zpool set autoreplace=on mpool
# zpool get autoreplace mpool
NAME PROPERTY    VALUE    SOURCE
mpool autoreplace on        default
```

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 dataset for the root pool. This property is typically set by the installation and upgrade programs.
cachefile	String	N/A	Controls where pool configuration information is cached. All pools in the cache are automatically imported when the system boots. However, installation and clustering environments might require this information to be cached in a different location so that pools are not automatically imported. You can set this property to cache pool configuration information in a different location. This information can be imported later by using the <code>zpool import -c</code> command. For most ZFS configurations, this property is not used.
capacity	Number	N/A	Read-only value that identifies the percentage of pool space used. The property abbreviation is <code>cap</code> .
delegation	Boolean	on	Controls whether a nonprivileged user can be granted access permissions that are defined for a dataset. For more information, see Chapter 9, “Oracle Solaris ZFS Delegated Administration.”

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

Property Name	Type	Default Value	Description
<code>failmode</code>	String	<code>wait</code>	<p>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:</p> <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.
<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>on</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>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 95
- “Viewing I/O Statistics for ZFS Storage Pools” on page 98
- “Determining the Health Status of ZFS Storage Pools” on page 100

Displaying Information About ZFS Storage Pools

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

Listing 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    ALTROOT
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 54 .
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 100 .
ALTROOT	The alternate root of the pool, if one exists.

For more information about alternate root pools, see [“Using ZFS Alternate Root Pools” on page 265](#).

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  -
```

Listing 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 [“Listing Information About All Storage Pools or a Specific Pool” on page 95](#).

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':
2010-05-13.14:13:15 zpool create tank mirror c1t2d0 c1t3d0
2010-05-13.14:21:19 zfs create tank/snaps
2010-05-14.08:10:29 zfs create tank/ws01
2010-05-14.08:10:54 zfs snapshot tank/ws01@now
2010-05-14.08:11:05 zfs clone tank/ws01@now tank/ws01bugfix

```

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
History for 'tank':
2010-05-13.14:13:15 zpool create tank mirror c1t2d0 c1t3d0 [user root on neo]
2010-05-13.14:21:19 zfs create tank/snaps [user root on neo]
2010-05-14.08:10:29 zfs create tank/ws01 [user root on neo]
2010-05-14.08:10:54 zfs snapshot tank/ws01@now [user root on neo]
2010-05-14.08:11:05 zfs clone tank/ws01@now tank/ws01bugfix [user root on neo]

```

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

```
# zpool history -i tank
2010-05-13.14:13:15 zpool create -f tank mirror c1t2d0 c1t23d0
2010-05-13.14:13:45 [internal pool create txg:6] pool spa 19; zfs spa 19; zpl 4;...
2010-05-13.14:21:19 zfs create tank/snaps
2010-05-13.14:22:02 [internal replay_inc_sync txg:20451] dataset = 41
2010-05-13.14:25:25 [internal snapshot txg:20480] dataset = 52
2010-05-13.14:25:25 [internal destroy_begin_sync txg:20481] dataset = 41
2010-05-13.14:25:26 [internal destroy txg:20488] dataset = 41
2010-05-13.14:25:26 [internal reservation set txg:20488] 0 dataset = 0
2010-05-14.08:10:29 zfs create tank/ws01
2010-05-14.08:10:54 [internal snapshot txg:53992] dataset = 42
2010-05-14.08:10:54 zfs snapshot tank/ws01@now
2010-05-14.08:11:04 [internal create txg:53994] dataset = 58
2010-05-14.08:11:05 zfs clone tank/ws01@now tank/ws01bugfix
```

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 54 .
<code>free capacity</code>	The amount of disk space available in the pool or device. As with 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 this 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.

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.

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.”](#)

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 machine and import it on the destination system, even if the system are of different architectural endianness. For information about replicating or migrating file systems between different storage pools, which might reside on different machines, see [“Sending and Receiving ZFS Data” on page 211](#).

- [“Preparing for ZFS Storage Pool Migration” on page 103](#)
- [“Exporting a ZFS Storage Pool” on page 103](#)
- [“Determining Available Storage Pools to Import” on page 104](#)
- [“Importing ZFS Storage Pools From Alternate Directories” on page 106](#)
- [“Importing ZFS Storage Pools” on page 106](#)
- [“Recovering Destroyed ZFS Storage Pools” on page 107](#)

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/eschrock': 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 257](#).

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`.

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 265](#).

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, such as the Solaris 10 10/09 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 has been modified to notify 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 22.

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	Reserved
22	Received properties

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.

Installing and Booting an Oracle Solaris ZFS Root File System

This chapter describes how to install and boot a Oracle Solaris ZFS file system. Migrating a UFS root file system to a ZFS file system by using Oracle Solaris Live Upgrade is also covered.

The following sections are provided in this chapter:

- “Installing and Booting an Oracle Solaris ZFS Root File System (Overview)” on page 111
- “Oracle Solaris Installation and Oracle Solaris Live Upgrade Requirements for ZFS Support” on page 113
- “Installing a ZFS Root File System (Initial Installation)” on page 115
- “Installing a ZFS Root File System (Oracle Solaris Flash Archive Installation)” on page 122
- “Installing a ZFS Root File System (Oracle Solaris JumpStart Installation)” on page 124
- “Migrating a UFS Root File System to a ZFS Root File System (Oracle Solaris Live Upgrade)” on page 128
- “ZFS Support for Swap and Dump Devices” on page 148
- “Booting From a ZFS Root File System” on page 152
- “Recovering the ZFS Root Pool or Root Pool Snapshots” on page 159

For a list of known issues in this release, see *Oracle Solaris 10 9/10 Release Notes*.

For up-to-date troubleshooting information, go to the following site:

http://www.solarisinternals.com/wiki/index.php/ZFS_Troubleshooting_Guide

Installing and Booting an Oracle Solaris ZFS Root File System (Overview)

Starting in the Solaris 10 10/08 release, you can install and boot from a ZFS root file system in the following ways:

- You can perform an initial installation during which ZFS is selected as the root file system.

- You can use Oracle Solaris Live Upgrade to migrate a UFS root file system to a ZFS root file system. In addition, you can use Oracle Solaris Live Upgrade to perform the following tasks:
 - Create a new boot environment within an existing ZFS root pool.
 - Create a new boot environment in a new ZFS root pool.
- You can use an Oracle Solaris JumpStart profile to automatically install a system with a ZFS root file system.
- Starting in the Solaris 10 10/09 release, you can use a JumpStart profile to automatically install a system with a ZFS Flash archive.

After a SPARC based or an x86 based system is installed with or migrated to a ZFS root file system, the system boots automatically from the ZFS root file system. For more information about boot changes, see [“Booting From a ZFS Root File System” on page 152](#).

ZFS Installation Features

The following ZFS installation features are provided in this Solaris release:

- Using the Solaris interactive text installer, you can install a UFS or a ZFS root file system. The default file system is still UFS for this Solaris release. You can access the interactive text installer option in the following ways:
 - SPARC: Use the following syntax from the Solaris Installation DVD:

```
ok boot cdrom - text
```
 - SPARC: Use the following syntax when booting from the network:

```
ok boot net - text
```
 - x86: Select the text-mode installation option.
- A Custom JumpStart profile provides the following features:
 - You can set up a profile to create a ZFS storage pool and designate a bootable ZFS file system.
 - You can set up a profile to identify a flash archive of a ZFS root pool.
- Using Oracle Solaris Live Upgrade, you can migrate a UFS root file system to a ZFS root file system. The `lucreate` and `luactivate` commands have been enhanced to support ZFS pools and file systems.
- You can set up a mirrored ZFS root pool by selecting two disks during installation. Or, you can attach additional disks after installation to create a mirrored ZFS root pool.
- Swap and dump devices are automatically created on ZFS volumes in the ZFS root pool.

The following installation features are not provided in this release:

- The GUI installation feature for installing a ZFS root file system is not currently available.
- The Oracle Solaris Flash installation feature for installing a ZFS root file system is not available by selecting the flash installation option from the initial installation option. However, you can create a JumpStart profile to identify a flash archive of a ZFS root pool. For more information, see [“Installing a ZFS Root File System \(Oracle Solaris Flash Archive Installation\)”](#) on page 122.
- You cannot use the standard upgrade program to upgrade your UFS root file system to a ZFS root file system.

Oracle Solaris Installation and Oracle Solaris Live Upgrade Requirements for ZFS Support

Ensure that the following requirements are met before attempting to install a system with a ZFS root file system or attempting to migrate a UFS root file system to a ZFS root file system.

Oracle Solaris Release Requirements

You can install and boot a ZFS root file system or migrate to a ZFS root file system in the following ways:

- Install a ZFS root file system – Available starting in the Solaris 10 10/08 release.
- Migrate from a UFS root file system to a ZFS root file system with Oracle Solaris Live Upgrade – You must have installed at least the Solaris 10 10/08 release or you must have upgraded to at least the Solaris 10 10/08 release.

General ZFS Storage Pool Requirements

The following sections describe ZFS root pool space and configuration requirements.

Disk Space Requirements for ZFS Storage Pools

The required minimum amount of available pool space for a ZFS root file system is larger than for a UFS root file system because swap and dump devices must be separate devices in a ZFS root environment. By default, swap and dump devices are the same device in a UFS root file system.

When a system is installed or upgraded with a ZFS root file system, the size of the swap area and the dump device are dependent upon the amount of physical memory. The minimum amount of available 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 disk space requirements for ZFS storage pools:

- 768 MB is the minimum amount of memory required to install a ZFS root file system.
- 1 GB of memory is recommended for better overall ZFS performance.
- At least 16 GB of disk space is recommended. The disk 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 are as follows:
 - **Solaris initial installation** – In the new ZFS boot environment, the default swap volume size is calculated as half the size of physical memory, generally in the 512 MB to 2 GB range. You can adjust the swap size during an initial installation.
 - The default dump volume size is calculated by the kernel based on `dumpadm` information and the size of physical memory. You can adjust the dump size during an initial installation.
 - **Oracle Solaris Live Upgrade** – When a UFS root file system is migrated to a ZFS root file system, the default swap volume size for the ZFS BE is calculated as the size of the swap device of the UFS BE. The default swap volume size calculation adds the sizes of all the swap devices in the UFS BE, and creates a ZFS volume of that size in the ZFS BE. If no swap devices are defined in the UFS BE, then the default swap volume size is set to 512 MB.
 - In the ZFS BE, the default dump volume size is set to half the size of physical memory, between 512 MB and 2 GB.

You can adjust the sizes of your swap and dump volumes to sizes of your choosing as long as the new sizes support system operations. For more information, see [“Adjusting the Sizes of Your ZFS Swap Device and Dump Device” on page 149](#).

- **Boot environment (BE)** – In addition to either new swap and dump space requirements or adjusted swap and dump device sizes, a ZFS BE that is migrated from a UFS BE requires approximately 6 GB. Each ZFS BE that is cloned from another ZFS BE doesn't require additional disk space, but consider that the BE size will increase when patches are applied. All ZFS BEs in the same root pool use the same swap and dump devices.
- **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 same dataset as 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.

Another restriction is that the `/var` directory or dataset must be a single dataset. For example, you cannot create a descendent `/var` dataset, such as `/var/tmp`, if you want to also use Oracle Solaris Live Upgrade to migrate or patch a ZFS BE or create a ZFS flash archive of this pool.

For example, a system with 12 GB of disk space might be too small for a bootable ZFS environment because 2 GB of disk space is required for each swap and dump device and approximately 6 GB of disk space is required for the ZFS BE that is migrated from the UFS BE.

ZFS Storage Pool Configuration Requirements

Review the following ZFS storage pool configuration requirements:

- The pool that is intended to be the root pool must have an SMI label. This requirement is met if the pool is created with disk slices.
- 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 an Oracle Solaris Live Upgrade migration, you 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 64](#).

- x86: 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 *System Administration Guide: Devices and File Systems*](#).
- Disks that are designated for booting in a ZFS root pool must be limited to 1 TB in size on both SPARC based and x86 based systems.
- Compression can be enabled on the root pool but only after the root pool is installed. No way exists to enable compression on a root pool during 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 or after Solaris Live Upgrade migration to a ZFS root file system. Renaming the root pool might cause an unbootable system.

Installing a ZFS Root File System (Initial Installation)

In this Solaris release, you can perform an initial installation by using the Solaris interactive text installer to create a ZFS storage pool that contains a bootable ZFS root file system. If you have an existing ZFS storage pool that you want to use for your ZFS root file system, then you must use Oracle Solaris Live Upgrade to migrate your existing UFS root file system to a ZFS root file system in an existing ZFS storage pool. For more information, see [“Migrating a UFS Root File System to a ZFS Root File System \(Oracle Solaris Live Upgrade\)” on page 128](#).

If you will be configuring zones after the initial installation of a ZFS root file system and you plan on patching or upgrading the system, see [“Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones \(Solaris 10 10/08\)” on page 134](#) or [“Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones \(at Least Solaris 10 5/09\)” on page 139](#).

If you already have ZFS storage pools on the system, they are acknowledged by the following message. However, these pools remain untouched, unless you select the disks in the existing pools to create the new storage pool.

There are existing ZFS pools available on this system. However, they can only be upgraded using the Live Upgrade tools. The following screens will only allow you to install a ZFS root system, not upgrade one.



Caution – Existing pools will be destroyed if any of their disks are selected for the new pool.

Before you begin the initial installation to create a ZFS storage pool, see [“Oracle Solaris Installation and Oracle Solaris Live Upgrade Requirements for ZFS Support”](#) on page 113.

EXAMPLE 5-1 Initial Installation of a Bootable ZFS Root File System

The Solaris interactive text installation process is basically the same as in previous Solaris releases, except that you are prompted to create a UFS or a ZFS root file system. UFS is the still the default file system in this release. If you select a ZFS root file system, you are prompted to create a ZFS storage pool. The steps for installing a ZFS root file system follow:

1. Select the Solaris interactive installation method because a Solaris Flash installation is not available to create a bootable ZFS root file system. However, you can create a ZFS flash archive to be used during a JumpStart installation. For more information, see [“Installing a ZFS Root File System \(Oracle Solaris Flash Archive Installation\)”](#) on page 122.

Starting in the Solaris 10 10/08 release, you can migrate a UFS root file system to a ZFS root file system as long as at least the Solaris 10 10/08 release is already installed. For more information about migrating to a ZFS root file system, see [“Migrating a UFS Root File System to a ZFS Root File System \(Oracle Solaris Live Upgrade\)”](#) on page 128.

2. To create a ZFS root file system, select the ZFS option. For example:

```
Choose Filesystem Type
```

```
Select the filesystem to use for your Solaris installation
```

```
[ ] UFS
[X] ZFS
```

3. After you select the software to be installed, you are prompted to select the disks to create your ZFS storage pool. This screen is similar as in previous Solaris releases.

```
Select Disks
```

```
On this screen you must select the disks for installing Solaris software.
Start by looking at the Suggested Minimum field; this value is the
approximate space needed to install the software you've selected. For ZFS,
multiple disks will be configured as mirrors, so the disk you choose, or the
slice within the disk must exceed the Suggested Minimum value.
```

```
NOTE: ** denotes current boot disk
```

Disk Device	Available Space
[X] c1t0d0	69994 MB (F4 to edit)
[] c1t1d0	69994 MB
[-] c1t2d0	0 MB

EXAMPLE 5-1 Initial Installation of a Bootable ZFS Root File System (Continued)

```

[-]    c1t3d0                                0 MB

Maximum Root Size: 69994 MB
Suggested Minimum: 8279 MB

```

You can select the disk or disks to be used for your ZFS root pool. If you select two disks, a mirrored two-disk configuration is set up for your root pool. Either a two-disk or a three-disk mirrored pool is optimal. If you have eight disks and you select all of them, those eight disks are used for the root pool as one big mirror. This configuration is not optimal. Another option is to create a mirrored root pool after the initial installation is complete. A RAID-Z pool configuration for the root pool is not supported. For more information about configuring ZFS storage pools, see [“Replication Features of a ZFS Storage Pool”](#) on page 60.

- To select two disks to create a mirrored root pool, use the cursor control keys to select the second disk. In the following example, both `c1t1d0` and `c1t2d0` are selected as the root pool disks. Both disks must have an SMI label and a slice 0. If the disks are not labeled with an SMI label or they don't contain slices, then you must exit the installation program, use the `format` utility to relabel and repartition the disks, and then restart the installation program.

Select Disks

On this screen you must select the disks for installing Solaris software. Start by looking at the Suggested Minimum field; this value is the approximate space needed to install the software you've selected. For ZFS, multiple disks will be configured as mirrors, so the disk you choose, or the slice within the disk must exceed the Suggested Minimum value.
NOTE: ** denotes current boot disk

Disk Device	Available Space
[X] c1t0d0	69994 MB
[X] c1t1d0	69994 MB (F4 to edit)
[-] c1t2d0	0 MB
[-] c1t3d0	0 MB

Maximum Root Size: 69994 MB
Suggested Minimum: 8279 MB

If the Available Space column identifies 0 MB, the disk most likely has an EFI label. If you want to use a disk with an EFI label, you will need to exit the installation program, relabel the disk with an SMI label by using the `format -e` command, then restart the installation program.

If you do not create a mirrored root pool during installation, you can easily create one after the installation. For information, see [“How to Create a Mirrored Root Pool \(Post Installation\)”](#) on page 121.

- After you have selected a disk or disks for your ZFS storage pool, a screen similar to the following is displayed:

Configure ZFS Settings

EXAMPLE 5-1 Initial Installation of a Bootable ZFS Root File System *(Continued)*

Specify the name of the pool to be created from the disk(s) you have chosen. Also specify the name of the dataset to be created within the pool that is to be used as the root directory for the filesystem.

```

          ZFS Pool Name: rpool
    ZFS Root Dataset Name: s10s_u9wos_08
    ZFS Pool Size (in MB): 69995
    Size of Swap Area (in MB): 2048
    Size of Dump Area (in MB): 1536
    (Pool size must be between 6231 MB and 69995 MB)

          [X] Keep / and /var combined
          [ ] Put /var on a separate dataset

```

From this screen, you can change the name of the ZFS pool, the dataset name, the pool size, and the swap and dump device sizes by moving the cursor control keys through the entries and replacing the default value with new values. Or, you can accept the default values. In addition, you can modify how the /var file system is created and mounted.

In this example, the root dataset name is changed to zfsBE.

```

          ZFS Pool Name: rpool
    ZFS Root Dataset Name: zfsBE
    ZFS Pool Size (in MB): 69995
    Size of Swap Area (in MB): 2048
    Size of Dump Area (in MB): 1536
    (Pool size must be between 6231 MB and 69995 MB)

          [X] Keep / and /var combined
          [ ] Put /var on a separate dataset

```

6. You can change the installation profile at this final installation screen. For example:

Profile

The information shown below is your profile for installing Solaris software. It reflects the choices you've made on previous screens.

```

=====
          Installation Option: Initial
          Boot Device: c1t0d0
    Root File System Type: ZFS
          Client Services: None

          Regions: North America
          System Locale: C ( C )

          Software: Solaris 10, Entire Distribution
          Pool Name: rpool
    Boot Environment Name: zfsBE
          Pool Size: 69995 MB
          Devices in Pool: c1t0d0
                          c1t1d0

```

7. After the installation is completed, review the resulting ZFS storage pool and file system information. For example:

EXAMPLE 5-1 Initial Installation of a Bootable ZFS Root File System (Continued)

```
# 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
        clt0d0s0    ONLINE    0     0     0
        clt1d0s0    ONLINE    0     0     0

errors: No known data errors
# zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rpool                                8.03G 58.9G   96K    /rpool
rpool/ROOT                           4.47G 58.9G   21K    legacy
rpool/ROOT/zfsBE                      4.47G 58.9G  4.47G    /
rpool/dump                            1.50G 58.9G  1.50G    -
rpool/export                          44K   58.9G   23K    /export
rpool/export/home                     21K   58.9G   21K    /export/home
rpool/swap                            2.06G 61.0G   16K    -
```

The sample `zfs list` output identifies the root pool components, such as the `rpool/ROOT` directory, which is not accessible by default.

- To create another ZFS boot environment (BE) in the same storage pool, you can use the `lucreate` command. In the following example, a new BE named `zfs2BE` is created. The current BE is named `zfsBE`, as shown in the `zfs list` output. However, the current BE is not acknowledged in the `lustatus` output until the new BE is created.

```
# lustatus
ERROR: No boot environments are configured on this system
ERROR: cannot determine list of all boot environment names
```

If you create a new ZFS BE in the same pool, use syntax similar to the following:

```
# lucreate -n zfs2BE
INFORMATION: The current boot environment is not named - assigning name <zfsBE>.
Current boot environment is named <zfsBE>.
Creating initial configuration for primary boot environment <zfsBE>.
The device </dev/dsk/clt0d0s0> is not a root device for any boot environment; cannot get BE ID.
PBE configuration successful: PBE name <zfsBE> PBE Boot Device </dev/dsk/clt0d0s0>.
Comparing source boot environment <zfsBE> file systems with the file
system(s) you specified for the new boot environment. Determining which
file systems should be in the new boot environment.
Updating boot environment description database on all BEs.
Updating system configuration files.
Creating configuration for boot environment <zfs2BE>.
Source boot environment is <zfsBE>.
Creating boot environment <zfs2BE>.
Cloning file systems from boot environment <zfsBE> to create boot environment <zfs2BE>.
Creating snapshot for <rpool/ROOT/zfsBE> on <rpool/ROOT/zfsBE@zfs2BE>.
Creating clone for <rpool/ROOT/zfsBE@zfs2BE> on <rpool/ROOT/zfs2BE>.
Setting canmount=noauto for </> in zone <global> on <rpool/ROOT/zfs2BE>.
```

EXAMPLE 5-1 Initial Installation of a Bootable ZFS Root File System (Continued)

Population of boot environment <zfs2BE> successful.
 Creation of boot environment <zfs2BE> successful.

Creating a ZFS BE within the same pool uses ZFS clone and snapshot features to instantly create the BE. For more details about using Oracle Solaris Live Upgrade for a ZFS root migration, see [“Migrating a UFS Root File System to a ZFS Root File System \(Oracle Solaris Live Upgrade\)”](#) on page 128.

9. Next, verify the new boot environments. For example:

```
# lustatus
Boot Environment      Is      Active Active   Can   Copy
Name                 Complete Now    On Reboot Delete Status
-----
zfsBE                 yes     yes   yes     no    -
zfs2BE                yes     no    no      yes   -
# zfs list
NAME                 USED  AVAIL  REFER  MOUNTPOINT
rpool                8.03G 58.9G  97K    /rpool
rpool/ROOT           4.47G 58.9G  21K    legacy
rpool/ROOT/zfs2BE    116K  58.9G  4.47G  /
rpool/ROOT/zfsBE     4.47G 58.9G  4.47G  /
rpool/ROOT/zfsBE@zfs2BE 75.5K -      4.47G  -
rpool/dump           1.50G 58.9G  1.50G  -
rpool/export         44K   58.9G  23K    /export
rpool/export/home    21K   58.9G  21K    /export/home
rpool/swap           2.06G 61.0G  16K    -
```

10. To boot from an alternate BE, use the `luactivate` command. After you activate the BE on a SPARC based system, use the `boot -L` command to identify the available BEs when the boot device contains a ZFS storage pool. When booting from an x86 based system, identify the BE to be booted from the GRUB menu.

For example, on a SPARC based system, use the `boot -L` command to display a list of available BEs. To boot from the new BE, `zfs2BE`, select option 2. Then, type the displayed `boot -Z` command.

```
ok boot -L
Executing last command: boot -L
Boot device: /pci@7c0/pci@0/pci@1/pci@0,2/LSILogic,sas@2/disk@0 File and args: -L
1 zfsBE
2 zfs2BE
Select environment to boot: [ 1 - 2 ]: 2

To boot the selected entry, invoke:
boot [<root-device>] -Z rpool/ROOT/zfs2BE
ok boot -Z rpool/ROOT/zfs2BE
```

For more information about booting a ZFS file system, see [“Booting From a ZFS Root File System”](#) on page 152.

▼ How to Create a Mirrored Root Pool (Post Installation)

If you did not create a mirrored ZFS root pool during installation, you can easily create one after the installation.

For information about replacing a disk in root pool, see [“How to Replace a Disk in the ZFS Root Pool” on page 159](#).

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
        c1t0d0s0     ONLINE            0     0     0

errors: No known data errors
```

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

```
# zpool attach rpool c1t0d0s0 c1t1d0s0
Please be sure to invoke installboot(1M) to make 'c1t1d0s0' bootable.
Make sure to wait until resilver is done before rebooting.
```

3 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.
        scrub: resilver in progress for 0h1m, 24.26% done, 0h3m to go
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 3.18G resilvered

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:

```
scrub: resilver completed after 0h10m with 0 errors on Thu Mar 11 11:27:22 2010
```

4 Apply boot blocks to the second disk after resilvering is complete.

```
sparc# installboot -F zfs /usr/platform/'uname -i'/lib/fs/zfs/bootblk /dev/rdisk/c1t1d0s0  
  
x86# installgrub /boot/grub/stage1 /boot/grub/stage2 /dev/rdisk/c1t1d0s0
```

5 Verify that you can boot successfully from the second disk.**6 Set up the system to boot automatically from the new disk, either by using the `eeprom` command, the `setenv` command from the SPARC boot PROM. Or, reconfigure the PC BIOS.**

Installing a ZFS Root File System (Oracle Solaris Flash Archive Installation)

Starting in the Solaris 10 10/09 release, you can create a flash archive on a system that is running a UFS root file system or a ZFS root file system. A flash archive of a ZFS root pool contains the entire pool hierarchy, except for the swap and dump volumes, and any excluded datasets. The swap and dump volumes are created when the flash archive is installed. You can use the flash archive installation method as follows:

- Generate a flash archive that can be used to install and boot a system with a ZFS root file system.
- Perform a JumpStart installation of a system by using a ZFS flash archive. Creating a ZFS flash archive clones an entire root pool, not individual boot environments. Individual datasets within the pool can be excluded by using the `flarcreate` and `flar` commands' `-D` option.

Review the following limitations before you consider installing a system with a ZFS flash archive:

- Only a JumpStart installation of a ZFS flash archive is supported. You cannot use the interactive installation option of a flash archive to install a system with a ZFS root file system. Nor can you use a flash archive to install a ZFS BE with Oracle Solaris Live Upgrade.
- You can only install a flash archive on a system that has the same architecture as the system on which you created the ZFS flash archive. For example, an archive that is created on a sun4u system cannot be installed on a sun4v system.
- Only a full initial installation of a ZFS flash archive is supported. You cannot install differential flash archive of a ZFS root file system nor can you install a hybrid UFS/ZFS archive.
- Existing UFS flash archives can still only be used to install a UFS root file system. The ZFS Flash archive can only be used to install a ZFS root file system.

- Although the entire root pool, minus any explicitly excluded datasets, is archived and installed, only the ZFS BE that is booted when the archive is created is usable after the flash archive is installed. However, pools that are archived with the `flarcreate` or the `flar` commands' `-R rootdir` option can be used to archive a root pool other than the one that is currently booted.
- A ZFS root pool name that is created with a flash archive must match the master root pool name. The root pool name that is used to create the flash archive is the name that is assigned to the new pool created. Changing the pool name is not supported.
- The `flarcreate` and `flar` command options used to include and exclude individual files are not supported in a ZFS flash archive. You can only exclude entire datasets from a ZFS flash archive.
- The `flar info` command is not supported for a ZFS flash archive. For example:

```
# flar info -l zfs10u8flar
ERROR: archive content listing not supported for zfs archives.
```

After a master system is installed with or upgraded to at least the Solaris 10 10/09 release, you can create a ZFS flash archive to be used to install a target system. The basic process follows:

- Install or upgrade to at least the Solaris 10 10/09 release on the master system. Add any customizations that you want.
- Create the ZFS flash archive with the `flarcreate` command on the master system. All datasets in the root pool, except for the swap and dump volumes, are included in the ZFS flash archive.
- Create a JumpStart profile to include the flash archive information on the installation server.
- Install the ZFS flash archive on the target system.

The following archive options are supported for installing a ZFS root pool with a flash archive:

- Use the `flarcreate` or `flar` command to create a flash archive from the specified ZFS root pool. If not specified, a flash archive of the default root pool is created.
- Use `flarcreate -D dataset` to exclude the specified datasets from the flash archive. This option can be used multiple times to exclude multiple datasets.

After a ZFS flash archive is installed, the system is configured as follows:

- The entire dataset hierarchy that existed on the system where the flash archive was created is recreated on the target system, minus any datasets that were specifically excluded at the time of archive creation. The swap and dump volumes are not included in the flash archive.
- The root pool has the same name as the pool that was used to create the archive.
- The boot environment that was active when the flash archive was created is the active and default BE on the deployed systems.

EXAMPLE 5-2 Installing a System with a ZFS Flash Archive

After the master system is installed or upgraded to at least the Solaris 10 10/09 release, create a flash archive of the ZFS root pool. For example:

```
# flarcreate -n zfsBE zfs10upflar
Full Flash
Checking integrity...
Integrity OK.
Running precreation scripts...
Precreation scripts done.
Determining the size of the archive...
The archive will be approximately 4.94GB.
Creating the archive...
Archive creation complete.
Running postcreation scripts...
Postcreation scripts done.

Running pre-exit scripts...
Pre-exit scripts done.
```

On the system that will be used as the installation server, create a JumpStart profile as you would to install any system. For example, the following profile is used to install the `zfs10upflar` archive.

```
install_type flash_install
archive_location nfs system:/export/jump/zfs10upflar
partitioning explicit
pool pool auto auto auto mirror c0t1d0s0 c0t0d0s0
```

Installing a ZFS Root File System (Oracle Solaris JumpStart Installation)

You can create a JumpStart profile to install a ZFS root file system or a UFS root file system.

A ZFS specific profile must contain the new `pool` keyword. The `pool` keyword installs a new root pool, and a new boot environment is created by default. You can provide the name of the boot environment as well as create a separate `/var` dataset with the `bootenv` `installbe` keywords and the `bename` and `dataset` options.

For general information about using JumpStart features, see *Oracle Solaris 10 9/10 Installation Guide: Custom JumpStart and Advanced Installations*.

If you will be configuring zones after the JumpStart installation of a ZFS root file system and you plan on patching or upgrading the system, see “Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (Solaris 10 10/08)” on page 134 or “Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (at Least Solaris 10 5/09)” on page 139.

JumpStart Keywords for ZFS

The following keywords are permitted in a ZFS specific profile:

auto Automatically specifies the size of the slices for the pool, swap volume, or dump volume. The size of the disk is checked to verify that the minimum size can be accommodated. If the minimum size can be accommodated, the largest possible pool size is allocated, given the constraints, such as the size of the disks, preserved slices, and so on.

For example, if you specify `c0t0d0s0`, the root pool slice is created as large as possible if you specify either the `all` or `auto` keywords. Or, you can specify a particular size for the slice, swap volume, or dump volume.

The `auto` keyword works similarly to the `all` keyword when used with a ZFS root pool because pools don't have unused disk space.

bootenv Identifies the boot environment characteristics.

Use the following `bootenv` keyword syntax to create a bootable ZFS root environment:

```
bootenv installbe bename BE-name [dataset mount-point]
```

installbe Creates a new BE that is identified by the `bename` option and `BE-name` entry and installs it.

bename *BE-name* Identifies the `BE-name` to install.

If `bename` is not used with the `pool` keyword, then a default BE is created.

dataset *mount-point* Use the optional `dataset` keyword to identify a `/var` dataset that is separate from the root dataset. The `mount-point` value is currently limited to `/var`. For example, a `bootenv` syntax line for a separate `/var` dataset would be similar to the following:

```
bootenv installbe bename zfsroot dataset /var
```

pool Defines the new root pool to be created. The following keyword syntax must be provided:

```
pool poolname poolsize swapsize dumpsizes vdevlist
```

poolname Identifies the name of the pool to be created. The pool is created with the specified `pool size` and with the specified physical devices (`vdevs`). The `poolname` value should not identify the name of an existing pool or the existing pool is overwritten.

<i>poolsize</i>	Specifies the size of the pool to be created. The value can be <code>auto</code> or <code>existing</code> . The <code>auto</code> value allocates the largest possible pool size, given the constraints, such as size of the disks, preserved slices, and so on. The <code>existing</code> value means the boundaries of existing slices by that name are preserved and not overwritten. The size is assumed to be in MB, unless specified by <code>g</code> (GB).
<i>swapsize</i>	Specifies the size of the swap volume to be created. The <code>auto</code> value means that the default swap size is used. You can specify a size with a <i>size</i> value. The size is in MB, unless specified by <code>g</code> (GB).
<i>dumpsizes</i>	Specifies the size of the dump volume to be created. The <code>auto</code> value means that the default swap size is used. You can specify a size with a <i>size</i> value. The size is assumed to be in MB, unless specified by <code>g</code> (GB).
<i>vdevlist</i>	Specifies one or more devices that are used to create the pool. The format of <i>vdevlist</i> is the same as the format of the <code>zpool create</code> command. At this time, only mirrored configurations are supported when multiple devices are specified. Devices in <i>vdevlist</i> must be slices for the root pool. The <code>any</code> value means that the installation software selects a suitable device.

You can mirror as many disks as you like, but the size of the pool that is created is determined by the smallest of the specified disks. For more information about creating mirrored storage pools, see [“Mirrored Storage Pool Configuration” on page 61](#).

JumpStart Profile Examples for ZFS

This section provides examples of ZFS specific JumpStart profiles.

The following profile performs an initial installation specified with `install_type initial_install` in a new pool, identified with `pool newpool`, whose size is automatically sized with the `auto` keyword to the size of the specified disks. The swap area and dump device are automatically sized with the `auto` keyword in a mirrored configuration of disks (with the `mirror` keyword and disks specified as `c0t0d0s0` and `c0t1d0s0`). Boot environment characteristics are set with the `bootenv` keyword to install a new BE with the keyword `installbe` and a `bename` named `s10-xx` is created.

```
install_type initial_install
pool newpool auto auto auto mirror c0t0d0s0 c0t1d0s0
bootenv installbe bename s10-xx
```

The following profile performs an initial installation with the keyword `install_type initial_install` of the `SUNWCall` metacluster in a new pool called `newpool`, which is 80 GBs in size. This pool is created with a 2-GB swap volume and a 2-GB dump volume, in a mirrored configuration of any two available devices that are large enough to create an 80-GB pool. If two such devices aren't available, the installation fails. Boot environment characteristics are set with the `bootenv` keyword to install a new BE with the keyword `installbe` and a `bename` named `s10-xx` is created.

```
install_type initial_install
cluster SUNWCall
pool newpool 80g 2g 2g mirror any any
bootenv installbe bename s10-xx
```

JumpStart installation syntax enables you to preserve or create a UFS file system on a disk that also includes a ZFS root pool. This configuration is not recommended for production systems, but could be used for transition or migration needs on a small system, such as a laptop.

JumpStart Issues for ZFS

Consider the following issues before starting a JumpStart installation of a bootable ZFS root file system:

- You cannot use an existing ZFS storage pool for a JumpStart installation to create a bootable ZFS root file system. You must create a new ZFS storage pool with syntax similar to the following:

```
pool rpool 20G 4G 4G c0t0d0s0
```

- You must create your pool with disk slices rather than with whole disks as described in [“Oracle Solaris Installation and Oracle Solaris Live Upgrade Requirements for ZFS Support” on page 113](#). For example, the bold syntax in the following example is not acceptable:

```
install_type initial_install
cluster SUNWCall
pool rpool all auto auto mirror c0t0d0 c0t1d0
bootenv installbe bename newBE
```

The bold syntax in the following example is acceptable:

```
install_type initial_install
cluster SUNWCall
pool rpool all auto auto mirror c0t0d0s0 c0t1d0s0
bootenv installbe bename newBE
```

Migrating a UFS Root File System to a ZFS Root File System (Oracle Solaris Live Upgrade)

Oracle Solaris Live Upgrade features related to UFS components are still available, and they work as in previous Solaris releases.

The following features are also available:

- When you migrate your UFS root file system to a ZFS root file system, you must designate an existing ZFS storage pool with the `-p` option.
- If the UFS root file system has components on different slices, they are migrated to the ZFS root pool.
- You can migrate a system with zones but the supported configurations are limited in the Solaris 10 10/08 release. More zone configurations are supported starting in the Solaris 10 5/09 release. For more information, see the following sections:
 - [“Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones \(Solaris 10 10/08\)”](#) on page 134
 - [“Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones \(at Least Solaris 10 5/09\)”](#) on page 139

If you are migrating a system without zones, see [“Using Oracle Solaris Live Upgrade to Migrate to a ZFS Root File System \(Without Zones\)”](#) on page 129.

- Oracle Solaris Live Upgrade can use the ZFS snapshot and clone features when you create a new ZFS BE in the same pool. So, BE creation is much faster than previous Solaris releases.

For detailed information about Oracle Solaris installation and Oracle Solaris Live Upgrade features, see the [Oracle Solaris 10 9/10 Installation Guide: Solaris Live Upgrade and Upgrade Planning](#).

The basic process for migrating a UFS root file system to a ZFS root file system follows:

- Install the Solaris 10 10/08, Solaris 10 5/09, Solaris 10 10/09, or Oracle Solaris 10 9/10 release or use the standard upgrade program to upgrade from a previous Solaris 10 release on any supported SPARC based or x86 based system.
- When you are running at least the Solaris 10 10/08 release, create a ZFS storage pool for your ZFS root file system.
- Use Oracle Solaris Live Upgrade to migrate your UFS root file system to a ZFS root file system.
- Activate your ZFS BE with the `luactivate` command.

For information about ZFS and Oracle Solaris Live Upgrade requirements, see [“Oracle Solaris Installation and Oracle Solaris Live Upgrade Requirements for ZFS Support”](#) on page 113.

ZFS Migration Issues With Oracle Solaris Live Upgrade

Review the following issues before you use Oracle Solaris Live Upgrade to migrate your UFS root file system to a ZFS root file system:

- The Oracle Solaris installation GUI's standard upgrade option is not available for migrating from a UFS to a ZFS root file system. To migrate from a UFS file system, you must use Oracle Solaris Live Upgrade.
- You must create the ZFS storage pool that will be used for booting before the Oracle Solaris Live Upgrade operation. In addition, due to current boot limitations, the ZFS root pool must be created with slices instead of whole disks. For example:

```
# zpool create rpool mirror c1t0d0s0 c1t1d0s0
```

Before you create the new pool, ensure that the disks to be used in the pool have an SMI (VTOC) label instead of an EFI label. If the disk is relabeled with an SMI label, ensure that the labeling process did not change the partitioning scheme. In most cases, all of the disk's capacity should be in the slices that are intended for the root pool.

- You cannot use Oracle Solaris Live Upgrade to create a UFS BE from a ZFS BE. If you migrate your UFS BE to a ZFS BE and you retain your UFS BE, you can boot from either your UFS BE or your ZFS BE.
- Do not rename your ZFS BEs with the `zfs rename` command because Oracle Solaris Live Upgrade feature cannot detect the name change. Subsequent commands, such as `ludelete`, will fail. In fact, do not rename your ZFS pools or file systems if you have existing BEs that you want to continue to use.
- When creating an alternative BE that is a clone of the primary BE, you cannot use the `-f`, `-x`, `-y`, `-Y`, and `-z` options to include or exclude files from the primary BE. You can still use the inclusion and exclusion option set in the following cases:

```
UFS -> UFS
UFS -> ZFS
ZFS -> ZFS (different pool)
```

- Although you can use Oracle Solaris Live Upgrade to upgrade your UFS root file system to a ZFS root file system, you cannot use Oracle Solaris Live Upgrade to upgrade non-root or shared file systems.
- You cannot use the `lu` command to create or migrate a ZFS root file system.

Using Oracle Solaris Live Upgrade to Migrate to a ZFS Root File System (Without Zones)

The following examples show how to migrate a UFS root file system to a ZFS root file system.

If you are migrating or updating a system with zones, see the following sections:

- “Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (Solaris 10 10/08)” on page 134
- “Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (at Least Solaris 10 5/09)” on page 139

EXAMPLE 5-3 Using Oracle Solaris Live Upgrade to Migrate a UFS Root File System to a ZFS Root File System

The following example shows how to create a BE of a ZFS root file system from a UFS root file system. The current BE, `ufsBE`, which contains a UFS root file system, is identified by the `-c` option. If you do not include the optional `-c` option, the current BE name defaults to the device name. The new BE, `zfsBE`, is identified by the `-n` option. A ZFS storage pool must exist before the `lucreate` operation.

The ZFS storage pool must be created with slices rather than with whole disks to be upgradeable and bootable. Before you create the new pool, ensure that the disks to be used in the pool have an SMI (VTOC) label instead of an EFI label. If the disk is relabeled with an SMI label, ensure that the labeling process did not change the partitioning scheme. In most cases, all of the disk's capacity should be in the slice that is intended for the root pool.

```
# zpool create rpool mirror clt2d0s0 c2t1d0s0
# lucreate -c ufsBE -n zfsBE -p rpool
Analyzing system configuration.
No name for current boot environment.
Current boot environment is named <ufsBE>.
Creating initial configuration for primary boot environment <ufsBE>.
The device </dev/dsk/clt0d0s0> is not a root device for any boot environment; cannot get BE ID.
PBE configuration successful: PBE name <ufsBE> PBE Boot Device </dev/dsk/clt0d0s0>.
Comparing source boot environment <ufsBE> file systems with the file
system(s) you specified for the new boot environment. Determining which
file systems should be in the new boot environment.
Updating boot environment description database on all BEs.
Updating system configuration files.
The device </dev/dsk/clt2d0s0> is not a root device for any boot environment; cannot get BE ID.
Creating configuration for boot environment <zfsBE>.
Source boot environment is <ufsBE>.
Creating boot environment <zfsBE>.
Creating file systems on boot environment <zfsBE>.
Creating <zfs> file system for </> in zone <global> on <rpool/ROOT/zfsBE>.
Populating file systems on boot environment <zfsBE>.
Checking selection integrity.
Integrity check OK.
Populating contents of mount point </>.
Copying.
Creating shared file system mount points.
Creating compare databases for boot environment <zfsBE>.
Creating compare database for file system </rpool/ROOT>.
Creating compare database for file system </>.
Updating compare databases on boot environment <zfsBE>.
Making boot environment <zfsBE> bootable.
Creating boot_archive for /.alt.tmp.b-qD.mnt
updating /.alt.tmp.b-qD.mnt/platform/sun4u/boot_archive
```

EXAMPLE 5-3 Using Oracle Solaris Live Upgrade to Migrate a UFS Root File System to a ZFS Root File System *(Continued)*

Population of boot environment <zfsBE> successful.
 Creation of boot environment <zfsBE> successful.

After the `lucreate` operation completes, use the `lustatus` command to view the BE status. For example:

```
# lustatus
Boot Environment      Is      Active Active   Can   Copy
Name                 Complete Now    On Reboot Delete Status
-----
ufsBE                 yes     yes   yes     no    -
zfsBE                 yes     no    no      yes   -
```

Then, review the list of ZFS components. For example:

```
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
rpool               7.17G 59.8G 95.5K  /rpool
rpool/ROOT          4.66G 59.8G 21K   /rpool/ROOT
rpool/ROOT/zfsBE   4.66G 59.8G 4.66G /
rpool/dump          2G    61.8G 16K   -
rpool/swap          517M  60.3G 16K   -
```

Next, use the `luactivate` command to activate the new ZFS BE. For example:

```
# luactivate zfsBE
A Live Upgrade Sync operation will be performed on startup of boot environment <zfsBE>.
```

```
*****
```

```
The target boot environment has been activated. It will be used when you
reboot. NOTE: You MUST NOT USE the reboot, halt, or uadmin commands. You
MUST USE either the init or the shutdown command when you reboot. If you
do not use either init or shutdown, the system will not boot using the
target BE.
```

```
*****
```

```
.
.
.
Modifying boot archive service
Activation of boot environment <zfsBE> successful.
```

Next, reboot the system to the ZFS BE.

```
# init 6
```

Confirm that the ZFS BE is active.

```
# lustatus
Boot Environment      Is      Active Active   Can   Copy
```

EXAMPLE 5-3 Using Oracle Solaris Live Upgrade to Migrate a UFS Root File System to a ZFS Root File System (Continued)

Name	Complete	Now	On Reboot	Delete	Status
ufsBE	yes	no	no	yes	-
zfsBE	yes	yes	yes	no	-

If you switch back to the UFS BE, you must re-import any ZFS storage pools that were created while the ZFS BE was booted because they are not automatically available in the UFS BE.

If the UFS BE is no longer required, you can remove it with the `ludelete` command.

EXAMPLE 5-4 Using Oracle Solaris Live Upgrade to Create a ZFS BE From a ZFS BE

Creating a ZFS BE from a ZFS BE in the same pool is very quick because this operation uses ZFS snapshot and clone features. If the current BE resides on the same ZFS pool, the `-p` option is omitted.

If you have multiple ZFS BEs, do the following to select which BE to boot from:

- SPARC: You can use the `boot -L` command to identify the available BEs and select a BE from which to boot by using the `boot -Z` command.
- x86: You can select a BE from the GRUB menu.

For more information, see [Example 5-9](#).

```
# lucreate -n zfs2BE
Analyzing system configuration.
No name for current boot environment.
INFORMATION: The current boot environment is not named - assigning name <zfsBE>.
Current boot environment is named <zfsBE>.
Creating initial configuration for primary boot environment <zfsBE>.
The device </dev/dsk/clt0d0s0> is not a root device for any boot environment; cannot get BE ID.
PBE configuration successful: PBE name <zfsBE> PBE Boot Device </dev/dsk/clt0d0s0>.
Comparing source boot environment <zfsBE> file systems with the file
system(s) you specified for the new boot environment. Determining which
file systems should be in the new boot environment.
Updating boot environment description database on all BEs.
Updating system configuration files.
Creating configuration for boot environment <zfs2BE>.
Source boot environment is <zfsBE>.
Creating boot environment <zfs2BE>.
Cloning file systems from boot environment <zfsBE> to create boot environment <zfs2BE>.
Creating snapshot for <rpool/ROOT/zfsBE> on <rpool/ROOT/zfsBE@zfs2BE>.
Creating clone for <rpool/ROOT/zfsBE@zfs2BE> on <rpool/ROOT/zfs2BE>.
Setting canmount=noauto for </> in zone <global> on <rpool/ROOT/zfs2BE>.
Population of boot environment <zfs2BE> successful.
Creation of boot environment <zfs2BE> successful.
```

EXAMPLE 5-5 Upgrading Your ZFS BE (luupgrade)

You can upgrade your ZFS BE with additional packages or patches.

The basic process follows:

- Create an alternate BE with the `lucreate` command.
- Activate and boot from the alternate BE.
- Upgrade your primary ZFS BE with the `luupgrade` command to add packages or patches.

```
# lustatus
Boot Environment      Is      Active Active   Can   Copy
Name                  Complete Now    On Reboot Delete Status
-----
zfsBE                  yes     no     no       yes   -
zfs2BE                 yes     yes    yes      no    -
# luupgrade -p -n zfsBE -s /net/system/export/s10up/Solaris_10/Product SUNWchxge
Validating the contents of the media </net/install/export/s10up/Solaris_10/Product>.
Mounting the BE <zfsBE>.
Adding packages to the BE <zfsBE>.

Processing package instance <SUNWchxge> from </net/install/export/s10up/Solaris_10/Product>

Chelsio N110 10GE NIC Driver(sparc) 11.10.0,REV=2006.02.15.20.41
Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.

This appears to be an attempt to install the same architecture and
version of a package which is already installed. This installation
will attempt to overwrite this package.

Using </a> as the package base directory.
## Processing package information.
## Processing system information.
   4 package pathnames are already properly installed.
## Verifying package dependencies.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
## Checking for setuid/setgid programs.

This package contains scripts which will be executed with super-user
permission during the process of installing this package.

Do you want to continue with the installation of <SUNWchxge> [y,n,?] y
Installing Chelsio N110 10GE NIC Driver as <SUNWchxge>

## Installing part 1 of 1.
## Executing postinstall script.

Installation of <SUNWchxge> was successful.
Unmounting the BE <zfsBE>.
The package add to the BE <zfsBE> completed.
```

Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (Solaris 10 10/08)

You can use Oracle Solaris Live Upgrade to migrate a system with zones, but the supported configurations are limited in the Solaris 10 10/08 release. If you are installing or upgrading to at least the Solaris 10 5/09 release, more zone configurations are supported. For more information, see [“Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones \(at Least Solaris 10 5/09\)”](#) on page 139.

This section describes how to configure and install a system with zones so that it can be upgraded and patched with Oracle Solaris Live Upgrade. If you are migrating to a ZFS root file system without zones, see [“Using Oracle Solaris Live Upgrade to Migrate to a ZFS Root File System \(Without Zones\)”](#) on page 129.

If you are migrating a system with zones or if you are configuring a system with zones in the Solaris 10 10/08 release, review the following procedures:

- [“How to Migrate a UFS Root File System With Zone Roots on UFS to a ZFS Root File System \(Solaris 10 10/08\)”](#) on page 134
- [“How to Configure a ZFS Root File System With Zone Roots on ZFS \(Solaris 10 10/08\)”](#) on page 136
- [“How to Upgrade or Patch a ZFS Root File System With Zone Roots on ZFS \(Solaris 10 10/08\)”](#) on page 137
- [“Resolving ZFS Mount-Point Problems That Prevent Successful Booting \(Solaris 10 10/08\)”](#) on page 156

Follow these recommended procedures to set up zones on a system with a ZFS root file system to ensure that you can use Oracle Solaris Live Upgrade on that system.

▼ How to Migrate a UFS Root File System With Zone Roots on UFS to a ZFS Root File System (Solaris 10 10/08)

This procedure explains how to migrate a UFS root file system with zones installed to a ZFS root file system and ZFS zone root configuration that can be upgraded or patched.

In the steps that follow the example pool name is `rpool`, and the example name of the active boot environment is `s10BE*`.

1 Upgrade the system to the Solaris 10 10/08 release if it is running a previous Solaris 10 release.

For more information about upgrading a system is running the Solaris 10 release, see [Oracle Solaris 10 9/10 Installation Guide: Solaris Live Upgrade and Upgrade Planning](#).

2 Create the root pool.

```
# zpool create rpool mirror c0t1d0 c1t1d0
```

For information about the root pool requirements, see “Oracle Solaris Installation and Oracle Solaris Live Upgrade Requirements for ZFS Support” on page 113.

3 Confirm that the zones from the UFS environment are booted.

4 Create the new ZFS boot environment.

```
# lucreate -n s10BE2 -p rpool
```

This command establishes datasets in the root pool for the new boot environment and copies the current boot environment (including the zones) to those datasets.

5 Activate the new ZFS boot environment.

```
# luactivate s10BE2
```

Now, the system is running a ZFS root file system, but the zone roots on UFS are still in the UFS root file system. The next steps are required to fully migrate the UFS zones to a supported ZFS configuration.

6 Reboot the system.

```
# init 6
```

7 Migrate the zones to a ZFS BE.

a. Boot the zones.

b. Create another ZFS BE within the pool.

```
# lucreate s10BE3
```

c. Activate the new boot environment.

```
# luactivate s10BE3
```

d. Reboot the system.

```
# init 6
```

This step verifies that the ZFS BE and the zones are booted.

8 Resolve any potential mount-point problems.

Due to a bug in Oracle Solaris Live Upgrade, the inactive boot environment might fail to boot because a ZFS dataset or a zone's ZFS dataset in the boot environment has an invalid mount point.

a. Review the `zfs list` output.

Look for incorrect temporary mount points. For example:

```
# zfs list -r -o name,mountpoint rpool/ROOT/s10u6
```

NAME	MOUNTPOINT
rpool/ROOT/s10u6	/.alt.tmp.b-VP.mnt/
rpool/ROOT/s10u6/zones	/.alt.tmp.b-VP.mnt//zones
rpool/ROOT/s10u6/zones/zonerootA	/.alt.tmp.b-VP.mnt/zones/zonerootA

The mount point for the root ZFS BE (rpool/ROOT/s10u6) should be /.

b. Reset the mount points for the ZFS BE and its datasets.

For example:

```
# zfs inherit -r mountpoint rpool/ROOT/s10u6
# zfs set mountpoint=/ rpool/ROOT/s10u6
```

c. Reboot the system.

When the option to boot a specific boot environment is presented, either in the GRUB menu or at the OpenBoot PROM prompt, select the boot environment whose mount points were just corrected.

▼ How to Configure a ZFS Root File System With Zone Roots on ZFS (Solaris 10 10/08)

This procedure explains how to set up a ZFS root file system and ZFS zone root configuration that can be upgraded or patched. In this configuration, the ZFS zone roots are created as ZFS datasets.

In the steps that follow the example pool name is rpool and the example name of the active boot environment is s10BE. The name for the zones dataset can be any legal dataset name. In the following example, the zones dataset name is zones.

1 Install the system with a ZFS root, either by using the Solaris interactive text installer or the Solaris JumpStart installation method.

For information about installing a ZFS root file system by using the initial installation method or the Solaris JumpStart method, see [“Installing a ZFS Root File System \(Initial Installation\)” on page 115](#) or [“Installing a ZFS Root File System \(Oracle Solaris JumpStart Installation\)” on page 124](#).

2 Boot the system from the newly created root pool.

3 Create a dataset for grouping the zone roots.

For example:

```
# zfs create -o canmount=noauto rpool/ROOT/s10BE/zones
```

Setting the noauto value for the canmount property prevents the dataset from being mounted other than by the explicit action of Oracle Solaris Live Upgrade and system startup code.

4 Mount the newly created zones dataset.

```
# zfs mount rpool/ROOT/s10BE/zones
```

The dataset is mounted at /zones.

5 Create and mount a dataset for each zone root.

```
# zfs create -o canmount=noauto rpool/ROOT/s10BE/zones/zonerootA
# zfs mount rpool/ROOT/s10BE/zones/zonerootA
```

6 Set the appropriate permissions on the zone root directory.

```
# chmod 700 /zones/zonerootA
```

7 Configure the zone, setting the zone path as follows:

```
# zonecfg -z zoneA
zoneA: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:zoneA> create
zonecfg:zoneA> set zonepath=/zones/zonerootA
```

You can enable the zones to boot automatically when the system is booted by using the following syntax:

```
zonecfg:zoneA> set autoboot=true
```

8 Install the zone.

```
# zoneadm -z zoneA install
```

9 Boot the zone.

```
# zoneadm -z zoneA boot
```

▼ How to Upgrade or Patch a ZFS Root File System With Zone Roots on ZFS (Solaris 10/08)

Use this procedure when you need to upgrade or patch a ZFS root file system with zone roots on ZFS. These updates can either be a system upgrade or the application of patches.

In the steps that follow, newBE is the example name of the boot environment that is upgraded or patched.

1 Create the boot environment to upgrade or patch.

```
# lucreate -n newBE
```

The existing boot environment, including all the zones, is cloned. A dataset is created for each dataset in the original boot environment. The new datasets are created in the same pool as the current root pool.

2 Select one of the following to upgrade the system or apply patches to the new boot environment:

- Upgrade the system.

```
# luupgrade -u -n newBE -s /net/install/export/s10u7/latest
```

where the `-s` option specifies the location of the Solaris installation medium.

- Apply patches to the new boot environment.

```
# luupgrade -t -n newBE -t -s /patchdir 139147-02 157347-14
```

3 Activate the new boot environment.

```
# luactivate newBE
```

4 Boot from the newly activated boot environment.

```
# init 6
```

5 Resolve any potential mount-point problems.

Due to a bug in Oracle Solaris Live Upgrade feature, the inactive boot environment might fail to boot because a ZFS dataset or a zone's ZFS dataset in the boot environment has an invalid mount point.

a. Review the `zfs list` output.

Look for incorrect temporary mount points. For example:

```
# zfs list -r -o name,mountpoint rpool/ROOT/newBE
```

NAME	MOUNTPOINT
rpool/ROOT/newBE	/.alt.tmp.b-VP.mnt/
rpool/ROOT/newBE/zones	/.alt.tmp.b-VP.mnt/zones
rpool/ROOT/newBE/zones/zonerootA	/.alt.tmp.b-VP.mnt/zones/zonerootA

The mount point for the root ZFS BE (`rpool/ROOT/newBE`) should be `/`.

b. Reset the mount points for the ZFS BE and its datasets.

For example:

```
# zfs inherit -r mountpoint rpool/ROOT/newBE
# zfs set mountpoint=/ rpool/ROOT/newBE
```

c. Reboot the system.

When the option to boot a specific boot environment is presented, either in the GRUB menu or at the OpenBoot PROM prompt, select the boot environment whose mount points were just corrected.

Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (at Least Solaris 10 5/09)

You can use the Oracle Solaris Live Upgrade feature to migrate or upgrade a system with zones starting in the Solaris 10 10/08 release. Additional sparse—root and whole—root zone configurations are supported by Live Upgrade starting in the Solaris 10 5/09 release.

This section describes how to configure a system with zones so that it can be upgraded and patched with Oracle Solaris Live Upgrade starting in the Solaris 10 5/09 release. If you are migrating to a ZFS root file system without zones, see [“Using Oracle Solaris Live Upgrade to Migrate to a ZFS Root File System \(Without Zones\)”](#) on page 129.

Consider the following points when using Oracle Solaris Live Upgrade with ZFS and zones starting in at least the Solaris 10 5/09 release:

- To use Oracle Solaris Live Upgrade with zone configurations that are supported starting in at least the Solaris 10 5/09 release, you must first upgrade your system to at least the Solaris 10 5/09 release by using the standard upgrade program.
- Then, with Oracle Solaris Live Upgrade, you can either migrate your UFS root file system with zone roots to a ZFS root file system or you can upgrade or patch your ZFS root file system and zone roots.
- You cannot directly migrate unsupported zone configurations from a previous Solaris 10 release to at least the Solaris 10 5/09 release.

If you are migrating or configuring a system with zones starting in the Solaris 10 5/09 release, review the following information:

- [“Supported ZFS with Zone Root Configuration Information \(at Least Solaris 10 5/09\)”](#) on page 139
- [“How to Create a ZFS BE With a ZFS Root File System and a Zone Root \(at Least Solaris 10 5/09\)”](#) on page 141
- [“How to Upgrade or Patch a ZFS Root File System With Zone Roots \(at Least Solaris 10 5/09\)”](#) on page 142
- [“How to Migrate a UFS Root File System With a Zone Root to a ZFS Root File System \(at Least Solaris 10 5/09\)”](#) on page 146

Supported ZFS with Zone Root Configuration Information (at Least Solaris 10 5/09)

Review the supported zone configurations before using Oracle Solaris Live Upgrade to migrate or upgrade a system with zones.

- **Migrate a UFS root file system to a ZFS root file system** – The following configurations of zone roots are supported:

- In a directory in the UFS root file system
- In a subdirectory of a mount point in the UFS root file system
- UFS root file system with a zone root in a UFS root file system directory or in a subdirectory of a UFS root file system mount point and a ZFS non-root pool with a zone root

The following UFS/zone configuration is not supported: UFS root file system that has a zone root as a mount point.

- **Migrate or upgrade a ZFS root file system** – The following configurations of zone roots are supported:
 - In a dataset in the ZFS root pool. In some cases, if a dataset for the zone root is not provided before the Oracle Solaris Live Upgrade operation, a dataset for the zone root (zoneds) will be created by Oracle Solaris Live Upgrade.
 - In a subdirectory of the ZFS root file system
 - In a dataset outside of the ZFS root file system
 - In a subdirectory of a dataset outside of the ZFS root file system
 - In a dataset in a non root pool. In the following example, `zonepool/zones` is a dataset that contains the zone roots, and `rpool` contains the ZFS BE:

```
zonepool
zonepool/zones
zonepool/zones/myzone
rpool
rpool/ROOT
rpool/ROOT/myBE
```

Oracle Solaris Live Upgrade snapshots and clones the zones in `zonepool` and the `rpool` BE if you use this syntax:

```
# lucreate -n newBE
```

The `newBE` BE in `rpool/ROOT/newBE` is created. When activated, `newBE` provides access to the `zonepool` components.

In the proceeding example, if `/zonepool/zones` was a subdirectory and not a separate dataset, then Live Upgrade would migrate it as components of the root pool, `rpool`.

- **Zones Migration or upgrade information with zones for both UFS and ZFS** – Review the following considerations that might affect a migration or an upgrade of either a UFS and ZFS environment:
 - If you configured your zones as described in [“Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones \(Solaris 10 10/08\)”](#) on page 134 in the Solaris 10 10/08 release and have upgraded to at least the Solaris 10 5/09, you should be able to migrate to a ZFS root file system or use Solaris Live Upgrade to upgrade to at least the Solaris 10 5/09 release.

- Do not create zone roots in nested directories, for example, zones/zone1 and zones/zone1/zone2. Otherwise, mounting might fail at boot time.

▼ How to Create a ZFS BE With a ZFS Root File System and a Zone Root (at Least Solaris 10 5/09)

Use this procedure after you have performed an initial installation of at least the Solaris 10 5/09 release to create a ZFS root file system. Also use this procedure after you have used the luupgrade feature to upgrade a ZFS root file system to at least the Solaris 10 5/09 release. A ZFS BE that is created using this procedure can then be upgraded or patched.

In the steps that follow, the example Oracle Solaris 10 9/10 system has a ZFS root file system and a zone root dataset in /rpool/zones. A ZFS BE named zfs2BE is created and can then be upgraded or patched.

1 Review the existing ZFS file systems.

```
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
rpool                7.26G 59.7G  98K    /rpool
rpool/ROOT           4.64G 59.7G  21K    legacy
rpool/ROOT/zfsBE     4.64G 59.7G  4.64G  /
rpool/dump           1.00G 59.7G  1.00G  -
rpool/export         44K   59.7G  23K    /export
rpool/export/home    21K   59.7G  21K    /export/home
rpool/swap           1G    60.7G  16K    -
rpool/zones          633M  59.7G  633M   /rpool/zones
```

2 Ensure that the zones are installed and booted.

```
# zoneadm list -cv
ID  NAME      STATUS  PATH                BRAND  IP
0   global    running /                    native shared
2   zfszone   running /rpool/zones        native shared
```

3 Create the ZFS BE.

```
# lucreate -n zfs2BE
Analyzing system configuration.
No name for current boot environment.
INFORMATION: The current boot environment is not named - assigning name <zfsBE>.
Current boot environment is named <zfsBE>.
Creating initial configuration for primary boot environment <zfsBE>.
The device </dev/dsk/clt0d0s0> is not a root device for any boot environment; cannot get BE ID.
PBE configuration successful: PBE name <zfsBE> PBE Boot Device </dev/dsk/clt0d0s0>.
Comparing source boot environment <zfsBE> file systems with the file
file system(s) you specified for the new boot environment. Determining which
file systems should be in the new boot environment.
Updating boot environment description database on all BEs.
Updating system configuration files.
Creating configuration for boot environment <zfs2BE>.
Source boot environment is <zfsBE>.
Creating boot environment <zfs2BE>.
```

Cloning file systems from boot environment <zfsBE> to create boot environment <zfs2BE>.
 Creating snapshot for <rpool/ROOT/zfsBE> on <rpool/ROOT/zfsBE@zfs2BE>.
 Creating clone for <rpool/ROOT/zfsBE@zfs2BE> on <rpool/ROOT/zfs2BE>.
 Setting canmount=noauto for </> in zone <global> on <rpool/ROOT/zfs2BE>.
 Population of boot environment <zfs2BE> successful.
 Creation of boot environment <zfs2BE> successful.

4 Activate the ZFS BE.

```
# lustatus
Boot Environment      Is      Active Active   Can      Copy
Name                 Complete Now    On Reboot Delete Status
-----
zfsBE                 yes     yes   yes     no       -
zfs2BE                yes     no    no      yes      -
# luactivate zfs2BE
A Live Upgrade Sync operation will be performed on startup of boot environment <zfs2BE>.
.
.
.
# init 6
```

5 Confirm that the ZFS file systems and zones are created in the new BE.

```
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
rpool                7.38G 59.6G  98K    /rpool
rpool/ROOT           4.72G 59.6G  21K    legacy
rpool/ROOT/zfs2BE   4.72G 59.6G  4.64G  /
rpool/ROOT/zfs2BE@zfs2BE 74.0M -      4.64G  -
rpool/ROOT/zfsBE    5.45M 59.6G  4.64G  /.alt.zfsBE
rpool/dump           1.00G 59.6G  1.00G  -
rpool/export         44K   59.6G  23K    /export
rpool/export/home    21K   59.6G  21K    /export/home
rpool/swap           1G    60.6G  16K    -
rpool/zones          17.2M 59.6G  633M   /rpool/zones
rpool/zones-zfsBE    653M  59.6G  633M   /rpool/zones-zfsBE
rpool/zones-zfsBE@zfs2BE 19.9M -      633M   -
# zoneadm list -cv
ID NAME                STATUS  PATH                BRAND  IP
  0 global              running /                    native shared
- zfszone              installed /rpool/zones        native shared
```

▼ How to Upgrade or Patch a ZFS Root File System With Zone Roots (at Least Solaris 10 5/09)

Use this procedure when you need to upgrade or patch a ZFS root file system with zone roots in at least the Solaris 10 5/09 release. These updates can either be a system upgrade or the application of patches.

In the steps that follow, `zfs2BE`, is the example name of the boot environment that is upgraded or patched.

1 Review the existing ZFS file systems.

```
# zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rpool                               7.38G  59.6G  100K   /rpool
rpool/ROOT                          4.72G  59.6G   21K   legacy
rpool/ROOT/zfs2BE                   4.72G  59.6G  4.64G  /
rpool/ROOT/zfs2BE@zfs2BE            75.0M   -    4.64G  -
rpool/ROOT/zfsBE                    5.46M  59.6G  4.64G  /
rpool/dump                          1.00G  59.6G  1.00G  -
rpool/export                        44K    59.6G   23K   /export
rpool/export/home                   21K    59.6G   21K   /export/home
rpool/swap                          1G     60.6G   16K   -
rpool/zones                         22.9M  59.6G  637M   /rpool/zones
rpool/zones-zfsBE                   653M   59.6G  633M   /rpool/zones-zfsBE
rpool/zones-zfsBE@zfs2BE           20.0M   -    633M   -
```

2 Ensure that the zones are installed and booted.

```
# zoneadm list -cv
ID NAME          STATUS  PATH                BRAND  IP
0  global         running /                    native shared
5  zfszone        running /rpool/zones        native shared
```

3 Create the ZFS BE to upgrade or patch.

```
# lucreate -n zfs2BE
Analyzing system configuration.
Comparing source boot environment <zfsBE> file systems with the file
system(s) you specified for the new boot environment. Determining which
file systems should be in the new boot environment.
Updating boot environment description database on all BEs.
Updating system configuration files.
Creating configuration for boot environment <zfs2BE>.
Source boot environment is <zfsBE>.
Creating boot environment <zfs2BE>.
Cloning file systems from boot environment <zfsBE> to create boot environment <zfs2BE>.
Creating snapshot for <rpool/ROOT/zfsBE> on <rpool/ROOT/zfsBE@zfs2BE>.
Creating clone for <rpool/ROOT/zfsBE@zfs2BE> on <rpool/ROOT/zfs2BE>.
Setting canmount=noauto for </> in zone <global> on <rpool/ROOT/zfs2BE>.
Creating snapshot for <rpool/zones> on <rpool/zones@zfs10092BE>.
Creating clone for <rpool/zones@zfs2BE> on <rpool/zones-zfs2BE>.
Population of boot environment <zfs2BE> successful.
Creation of boot environment <zfs2BE> successful.
```

4 Select one of the following to upgrade the system or apply patches to the new boot environment:

- Upgrade the system.

```
# luupgrade -u -n zfs2BE -s /net/install/export/s10up/latest
```

where the `-s` option specifies the location of the Solaris installation medium.

This process can take a very long time.

For a complete example of the `luupgrade` process, see [Example 5–6](#).

- Apply patches to the new boot environment.

```
# luupgrade -t -n zfs2BE -t -s /patchdir patch-id-02 patch-id-04
```

5 Activate the new boot environment.

```
# lustatus
```

Boot Environment Name	Is Complete	Active Now	Active On Reboot	Can Delete	Copy Status
zfsBE	yes	yes	yes	no	-
zfs2BE	yes	no	no	yes	-

```
# luactivate zfs2BE
```

```
A Live Upgrade Sync operation will be performed on startup of boot environment <zfs2BE>.
```

```
.  
.
.
```

6 Boot from the newly activated boot environment.

```
# init 6
```

Example 5-6 Upgrading a ZFS Root File System With a Zone Root to a Oracle Solaris 10 9/10 ZFS Root File System

In this example, a ZFS BE (zfsBE), which was created on a Solaris 10 10/09 system with a ZFS root file system and zone root in a non root pool, is upgraded to the Oracle Solaris 10 9/10 release. This process can take a long time. Then, the upgraded BE (zfs2BE) is activated. Ensure that the zones are installed and booted before attempting the upgrade.

In this example, the zonepool pool, the /zonepool/zones dataset, and the zfszone zone are created as follows:

```
# zpool create zonepool mirror c2t1d0 c2t5d0
# zfs create zonepool/zones
# chmod 700 zonepool/zones
# zonecfg -z zfszone
zfszone: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:zfszone> create
zonecfg:zfszone> set zonepath=/zonepool/zones
zonecfg:zfszone> verify
zonecfg:zfszone> exit
# zoneadm -z zfszone install
cannot create ZFS dataset zonepool/zones: dataset already exists
Preparing to install zone <zfszone>.
Creating list of files to copy from the global zone.
Copying <8960> files to the zone.
.  
.
.
```



```

# zoneadm list -cv
ID NAME           STATUS  PATH                               BRAND  IP
0  global          running /                                   native shared
2  zfszone         running /zonepool/zones                 native shared

# lucreate -n zfsBE
.
.
.
# luupgrade -u -n zfsBE -s /net/install/export/s10up/latest
40410 blocks
miniroot filesystem is <lofs>
Mounting miniroot at </net/system/export/s10up/latest/Solaris_10/Tools/Boot>
Validating the contents of the media </net/system/export/s10up/latest>.
The media is a standard Solaris media.
The media contains an operating system upgrade image.
The media contains <Solaris> version <10>.
Constructing upgrade profile to use.
Locating the operating system upgrade program.
Checking for existence of previously scheduled Live Upgrade requests.
Creating upgrade profile for BE <zfsBE>.
Determining packages to install or upgrade for BE <zfsBE>.
Performing the operating system upgrade of the BE <zfsBE>.
CAUTION: Interrupting this process may leave the boot environment unstable
or unbootable.
Upgrading Solaris: 100% completed
Installation of the packages from this media is complete.
Updating package information on boot environment <zfsBE>.
Package information successfully updated on boot environment <zfsBE>.
Adding operating system patches to the BE <zfsBE>.
The operating system patch installation is complete.
INFORMATION: The file </var/sadm/system/logs/upgrade_log> on boot
environment <zfsBE> contains a log of the upgrade operation.
INFORMATION: The file </var/sadm/system/data/upgrade_cleanup> on boot
environment <zfsBE> contains a log of cleanup operations required.
INFORMATION: Review the files listed above. Remember that all of the files
are located on boot environment <zfsBE>. Before you activate boot
environment <zfsBE>, determine if any additional system maintenance is
required or if additional media of the software distribution must be
installed.
The Solaris upgrade of the boot environment <zfsBE> is complete.
Installing failsafe
Failsafe install is complete.
# luactivate zfsBE
# init 6
# lustatus
Boot Environment      Is      Active Active   Can   Copy
Name                 Complete Now    On Reboot Delete Status
-----
zfsBE                 yes     no     no     yes   -
zfs2BE                yes     yes    yes    no    -
# zoneadm list -cv
ID NAME           STATUS  PATH                               BRAND  IP
0  global          running /                                   native shared
-  zfszone         installed /zonepool/zones                 native shared

```

▼ How to Migrate a UFS Root File System With a Zone Root to a ZFS Root File System (at Least Solaris 10 5/09)

Use this procedure to migrate a system with a UFS root file system and a zone root to at least the Solaris 10 5/09 release. Then, use Oracle Solaris Live Upgrade to create a ZFS BE.

In the steps that follow, the example UFS BE name is `c0t1d0s0`, the UFS zone root is `zonepool/zfszone`, and the ZFS root BE is `zfsBE`.

1 Upgrade the system to at least the Solaris 10 5/09 release if it is running a previous Solaris 10 release.

For information about upgrading a system that is running the Solaris 10 release, see [Oracle Solaris 10 9/10 Installation Guide: Solaris Live Upgrade and Upgrade Planning](#).

2 Create the root pool.

For information about the root pool requirements, see “Oracle Solaris Installation and Oracle Solaris Live Upgrade Requirements for ZFS Support” on page 113.

3 Confirm that the zones from the UFS environment are booted.

```
# zoneadm list -cv
ID NAME          STATUS  PATH                                BRAND  IP
0  global         running /                                     native shared
2  zfszone        running /zonepool/zones                  native shared
```

4 Create the new ZFS boot environment.

```
# lucreate -c c1t1d0s0 -n zfsBE -p rpool
```

This command establishes datasets in the root pool for the new boot environment and copies the current boot environment (including the zones) to those datasets.

5 Activate the new ZFS boot environment.

```
# lustatus
Boot Environment      Is      Active Active   Can   Copy
Name                  Complete Now    On Reboot Delete Status
-----
c1t1d0s0              yes     no     no      yes   -
zfsBE                 yes     yes    yes     no    -      #
luactivate zfsBE
A Live Upgrade Sync operation will be performed on startup of boot environment <zfsBE>.
.
.
.
```

6 Reboot the system.

```
# init 6
```

7 Confirm that the ZFS file systems and zones are created in the new BE.

```
# zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rpool                                6.17G  60.8G   98K    /rpool
rpool/ROOT                            4.67G  60.8G   21K    /rpool/ROOT
rpool/ROOT/zfsBE                       4.67G  60.8G  4.67G    /
rpool/dump                             1.00G  60.8G  1.00G    -
rpool/swap                             517M   61.3G   16K    -
zonepool                              634M   7.62G   24K    /zonepool
zonepool/zones                         270K   7.62G  633M    /zonepool/zones
zonepool/zones-clt1d0s0                 634M   7.62G  633M    /zonepool/zones-clt1d0s0
zonepool/zones-clt1d0s0@zfsBE           262K    -     633M    -

# zoneadm list -cv
ID NAME                STATUS  PATH                                BRAND  IP
  0 global              running /                                    native shared
  - zfszone             installed /zonepool/zones                    native shared
```

Example 5-7 Migrating a UFS Root File System With a Zone Root to a ZFS Root File System

In this example, a Oracle Solaris 10 9/10 system with a UFS root file system and a zone root (/uzone/ufszone), as well as a ZFS non-root pool (pool) and a zone root (/pool/zfszone), is migrated to a ZFS root file system. Ensure that the ZFS root pool is created and that the zones are installed and booted before attempting the migration.

```
# zoneadm list -cv
ID NAME                STATUS  PATH                                BRAND  IP
  0 global              running /                                    native shared
  2 ufszone             running /uzone/ufszone                       native shared
  3 zfszone             running /pool/zones/zfszone                    native shared
```

```
# lucreate -c ufsBE -n zfsBE -p rpool
```

```
Analyzing system configuration.
```

```
No name for current boot environment.
```

```
Current boot environment is named <zfsBE>.
```

```
Creating initial configuration for primary boot environment <zfsBE>.
```

```
The device </dev/dsk/clt0d0s0> is not a root device for any boot environment; cannot get BE ID.
```

```
PBE configuration successful: PBE name <ufsBE> PBE Boot Device </dev/dsk/clt0d0s0>.
```

```
Comparing source boot environment <ufsBE> file systems with the file system(s) you specified for the new boot environment. Determining which file systems should be in the new boot environment.
```

```
Updating boot environment description database on all BEs.
```

```
Updating system configuration files.
```

```
The device </dev/dsk/clt1d0s0> is not a root device for any boot environment; cannot get BE ID.
```

```
Creating configuration for boot environment <zfsBE>.
```

```
Source boot environment is <ufsBE>.
```

```
Creating boot environment <zfsBE>.
```

```
Creating file systems on boot environment <zfsBE>.
```

```
Creating <zfs> file system for </> in zone <global> on <rpool/ROOT/zfsBE>.
```

```
Populating file systems on boot environment <zfsBE>.
```

```
Checking selection integrity.
```

```
Integrity check OK.
```

```
Populating contents of mount point </>.
```

```
Copying.
```

```
Creating shared file system mount points.
```

```

Copying root of zone <ufszone> to </.alt.tmp.b-EYd.mnt/uzone/ufszone>.
Creating snapshot for <pool/zones/zfszone> on <pool/zones/zfszone@zfsBE>.
Creating clone for <pool/zones/zfszone@zfsBE> on <pool/zones/zfszone-zfsBE>.
Creating compare databases for boot environment <zfsBE>.
Creating compare database for file system </rpool/ROOT>.
Creating compare database for file system </>.
Updating compare databases on boot environment <zfsBE>.
Making boot environment <zfsBE> bootable.
Creating boot_archive for /.alt.tmp.b-DLd.mnt
updating /.alt.tmp.b-DLd.mnt/platform/sun4u/boot_archive
Population of boot environment <zfsBE> successful.
Creation of boot environment <zfsBE> successful.

```

lustatus

Boot Environment Name	Is Complete	Active Now	Active On Reboot	Can Delete	Copy Status
ufsBE	yes	yes	yes	no	-
zfsBE	yes	no	no	yes	-

luactivate zfsBE

```

.
.
.

```

init 6

```

.
.
.

```

zfs list

NAME	USED	AVAIL	REFER	MOUNTPOINT
pool	628M	66.3G	19K	/pool
pool/zones	628M	66.3G	20K	/pool/zones
pool/zones/zfszone	75.5K	66.3G	627M	/pool/zones/zfszone
pool/zones/zfszone-ufsBE	628M	66.3G	627M	/pool/zones/zfszone-ufsBE
pool/zones/zfszone-ufsBE@zfsBE	98K	-	627M	-
rpool	7.76G	59.2G	95K	/rpool
rpool/ROOT	5.25G	59.2G	18K	/rpool/ROOT
rpool/ROOT/zfsBE	5.25G	59.2G	5.25G	/
rpool/dump	2.00G	59.2G	2.00G	-
rpool/swap	517M	59.7G	16K	-

zoneadm list -cv

ID	NAME	STATUS	PATH	BRAND	IP
0	global	running	/	native	shared
-	ufszone	installed	/uzone/ufszone	native	shared
-	zfszone	installed	/pool/zones/zfszone	native	shared

ZFS Support for Swap and Dump Devices

During an initial Solaris OS installation or after performing an Oracle Solaris Live Upgrade migration from a UFS file system, 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 256,1          16 4194288 4194288

```

During an initial Solaris OS installation or a Oracle Solaris Live Upgrade from a UFS file system, 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 setup automatically at installation time. For example:

```
# dumpadm
  Dump content: kernel pages
  Dump device: /dev/zvol/dsk/rpool/dump (dedicated)
Savecore directory: /var/crash/t2000
  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 [“Oracle Solaris Installation and Oracle Solaris Live Upgrade Requirements for ZFS Support” on page 113](#).

Both the swap volume size and the dump volume size can be adjusted during and after installation. For more information, see [“Adjusting the Sizes of Your ZFS Swap Device and Dump Device” on page 149](#).

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

- Separate ZFS volumes must be used for the swap area and the dump device.
- 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 or upgraded, use the `swap` and `dumpadm` commands as in previous Solaris releases. For more information, see [Chapter 20, “Configuring Additional Swap Space \(Tasks\),” in *System Administration Guide: Devices and File Systems*](#) and [Chapter 17, “Managing System Crash Information \(Tasks\),” in *System Administration Guide: Advanced Administration*](#).

See the following sections for more information:

- [“Adjusting the Sizes of Your ZFS Swap Device and Dump Device” on page 149](#)
- [“Troubleshooting ZFS Dump Device Issues” on page 151](#)

Adjusting the Sizes of Your ZFS Swap Device and Dump Device

Because of the differences in the way a ZFS root installation determines the size of swap and dump devices, you might need to adjust their size before, during, or after installation.

- You can adjust the size of your swap and dump volumes during an initial installation. For more information, see [Example 5-1](#).
- You can create and size your swap and dump volumes before you perform an Oracle Solaris Live Upgrade operation. For example:

1. Create your storage pool.

```
# zpool create rpool mirror c0t0d0s0 c0t1d0s0
```

2. Create your dump device.

```
# zfs create -V 2G rpool/dump
```

3. Enable the dump device.

```
# dumpadm -d /dev/zvol/dsk/rpool/dump
Dump content: kernel pages
Dump device: /dev/zvol/dsk/rpool/dump (dedicated)
Savecore directory: /var/crash/t2000
Savecore enabled: yes
Save compressed: on
```

4. Select one of the following to create your swap area:

- SPARC: Create your swap area. Set the block size to 8 KB.

```
# zfs create -V 2G -b 8k rpool/swap
```

- x86: Create your swap area. Set the block size to 4 KB.

```
# zfs create -V 2G -b 4k rpool/swap
```

5. You must enable the swap volume when a new swap device is added or changed.
6. Add an entry for the swap volume to the `/etc/vfstab` file.

Oracle Solaris Live Upgrade does not resize existing 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 but until CR 6765386 is integrated, it is best to remove the swap device first. Then, recreate it. For example:

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

For information about removing a swap device on an active system, see this site:

http://www.solarisinternals.com/wiki/index.php/ZFS_Troubleshooting_Guide

- You can adjust the size of the swap and dump volumes in a JumpStart profile by using profile syntax similar to the following:

```
install_type initial_install
cluster SUNWCxall
pool rpool 16g 2g 2g c0t0d0s0
```

In this profile, two 2g entries set the size of the swap volume and dump volume as 2 GB each.

- If you need more swap space on a system that is already installed, just add another swap volume. For example:

```
# zfs create -V 2G rpool/swap2
```

Then, 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
```

Finally, add an entry for the second 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 volume 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 volume if the default dump volume size is too small. For example, on a large-memory system, the dump volume size is increased to 40 GB as follows:

```
# zfs set volsize=40G rpool/dump
```

Resizing a large dump volume 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/t2000
    Savecore enabled: yes
```

- 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 about sizing the swap and dump devices, see “[Planning for Swap Space](#)” in *System Administration Guide: 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 use the new style of booting with a boot archive, which is a file system image that contains the files required for booting. When a system is booted 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.

When a system is booted for installation, a RAM disk is used for the root file system during the entire installation process.

Booting from a ZFS file system differs from booting from a UFS file system because with ZFS, the boot device specifier identifies a storage pool, not a single root file system. A storage pool can contain multiple *bootable datasets* or 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 dataset selected for booting is identified by the pool's `boot fs` property. This default selection can be overridden by specifying an alternate bootable dataset in the `boot -Z` command.

Booting From an Alternate Disk in a Mirrored ZFS Root Pool

You can create a mirrored ZFS root pool when the system is installed, or you can attach a disk to create a mirrored ZFS root pool after installation. For more information see:

- “[Installing a ZFS Root File System \(Initial Installation\)](#)” on page 115
- “[How to Create a Mirrored Root Pool \(Post Installation\)](#)” on page 121

Review the following known issues regarding mirrored ZFS root pools:

- CR 6668666 – You must install the boot information on the additionally attached disks by using the `installboot` or `installgrub` commands to enable booting on the other disks in the mirror. If you create a mirrored ZFS root pool with the initial installation method, then this step is unnecessary. For example, if `c0t1d0s0` was the second disk added to the mirror, then the `installboot` or `installgrub` command syntax would be as follows:

- SPARC:

```
sparc# installboot -F zfs /usr/platform/'uname -i'/lib/fs/zfs/bootblk /dev/rdisk/c0t1d0s0
```

- x86:

```
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 the following 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

- SPARC: Enter the alternate disk at the ok prompt.

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

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@0,0:a'
```

- x86: Select an alternate disk in the mirrored ZFS root pool from the appropriate BIOS menu.

Then, use syntax similar to the following to confirm that you are booted from the alternate disk:

```
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'
```

SPARC: Booting From a ZFS Root File System

On a SPARC based system with multiple ZFS BEs, you can boot from any BE by using the `luactivate` command.

During the Solaris OS installation and Oracle Solaris Live Upgrade process, the ZFS root file system is automatically designated with the `bootfs` property.

Multiple bootable datasets can exist within a pool. By default, the bootable dataset 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 dataset 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 or migrated to a ZFS root file system, an entry similar to the following is added to the `menu.lst` file:

```
title zfsBE
bootfs rpool/ROOT/zfsBE
title zfs2BE
bootfs rpool/ROOT/zfs2BE
```

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

On a SPARC based system, two new boot options are available:

- After the BE is activated, you can use the `boot -L` command to display a list of bootable datasets within a ZFS pool. Then, you can select one of the bootable datasets in the list. Detailed instructions for booting that dataset are displayed. You can boot the selected dataset by following the instructions.
- You can use the `boot -Z dataset` command to boot a specific ZFS dataset.

EXAMPLE 5-8 SPARC: 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 `luactivate` command to specify a default BE.

For example, the following ZFS BEs are available as described by the `lustatus` output:

```
# lustatus
Boot Environment      Is      Active Active   Can   Copy
Name                  Complete Now    On Reboot Delete Status
-----
zfsBE                  yes     no     no      yes   -
zfs2BE                 yes     yes    yes     no    -
```

If you have multiple ZFS BEs on your SPARC based system, you can use the `boot -L` command to boot from a BE that is different from the default BE. However, a BE that is booted from a `boot -L` session is not reset as the default BE nor is the `bootfs` property updated. If you want to make the BE booted from a `boot -L` session the default BE, then you must activate it with the `luactivate` command.

For example:

EXAMPLE 5-8 SPARC: Booting From a Specific ZFS Boot Environment (Continued)

```

ok boot -L
Rebooting with command: boot -L
Boot device: /pci@7c0/pci@0/pci@1/pci@0,2/LSILogic,sas@2/disk@0 File and args: -L

1 zfsBE
2 zfs2BE
Select environment to boot: [ 1 - 2 ]: 1
To boot the selected entry, invoke:
boot [<root-device>] -Z rpool/ROOT/zfsBE

Program terminated
ok boot -Z rpool/ROOT/zfsBE

```

EXAMPLE 5-9 SPARC: Booting a ZFS File System in Failsafe Mode

On a SPARC based system, you can boot from the failsafe archive located in `/platform/uname-i/failsafe` as follows:

```
ok boot -F failsafe
```

To boot a failsafe archive from a particular ZFS bootable dataset, use syntax similar to the following:

```
ok boot -Z rpool/ROOT/zfsBE -F failsafe
```

x86: Booting From a ZFS Root File System

The following entries are added to the `/pool-name/boot/grub/menu.lst` file during the Solaris OS installation process or Oracle Solaris Live Upgrade operation to boot ZFS automatically:

```

title Solaris 10 9/10 X86
findroot (rootfs0,0,a)
kernel$ /platform/i86pc/multiboot -B $ZFS-BOOTFS
module /platform/i86pc/boot_archive
title Solaris failsafe
findroot (rootfs0,0,a)
kernel /boot/multiboot kernel/unix -s -B console=ttya
module /boot/x86.miniroot-safe

```

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 dataset, the following option is added:

```
-B $ZFS-BOOTFS
```

EXAMPLE 5-10 x86: Booting a ZFS File System

When a system boots from a ZFS file system, the root device is specified by the boot `-B $ZFS-BOOTFS` parameter on either the kernel or module line in the GRUB menu entry. This parameter value, similar to all parameters specified by the `-B` option, is passed by GRUB to the kernel. For example:

```
title Solaris 10 9/10 X86
findroot (rootfs0,0,a)
kernel$ /platform/i86pc/multiboot -B $ZFS-BOOTFS
module /platform/i86pc/boot_archive
title Solaris failsafe
findroot (rootfs0,0,a)
kernel /boot/multiboot kernel/unix -s -B console=ttya
module /boot/x86.miniroot-safe
```

EXAMPLE 5-11 x86: Booting a ZFS File System in Failsafe Mode

The x86 failsafe archive is `/boot/x86.miniroot-safe` and can be booted by selecting the Solaris failsafe entry from the GRUB menu. For example:

```
title Solaris failsafe
findroot (rootfs0,0,a)
kernel /boot/multiboot kernel/unix -s -B console=ttya
module /boot/x86.miniroot-safe
```

Resolving ZFS Mount-Point Problems That Prevent Successful Booting (Solaris 10 10/08)

The best way to change the active boot environment is to use the `luactivate` command. If booting the active environment fails due to a bad patch or a configuration error, the only way to boot from a different environment is to select that environment at boot time. You can select an alternate BE from the GRUB menu on an x86 based system or by booting it explicitly from the PROM on a SPARC based system.

Due to a bug in Oracle Solaris Live Upgrade in the Solaris 10 10/08 release, the inactive boot environment might fail to boot because a ZFS dataset or a zone's ZFS dataset in the boot environment has an invalid mount point. The same bug also prevents the BE from mounting if it has a separate `/var` dataset.

If a zone dataset has an invalid mount point, the mount point can be corrected by performing the following steps.

▼ How to Resolve ZFS Mount-Point Problems

- 1 Boot the system from a failsafe archive.

- 2 Import the pool.

For example:

```
# zpool import rpool
```

- 3 Look for incorrect temporary mount points.

For example:

```
# zfs list -r -o name,mountpoint rpool/ROOT/s10u6
```

NAME	MOUNTPOINT
rpool/ROOT/s10u6	/.alt.tmp.b-VP.mnt/
rpool/ROOT/s10u6/zones	/.alt.tmp.b-VP.mnt//zones
rpool/ROOT/s10u6/zones/zonerootA	/.alt.tmp.b-VP.mnt/zones/zonerootA

The mount point for the root BE (rpool/ROOT/s10u6) should be /.

If the boot is failing because of /var mounting problems, look for a similar incorrect temporary mount point for the /var dataset.

- 4 Reset the mount points for the ZFS BE and its datasets.

For example:

```
# zfs inherit -r mountpoint rpool/ROOT/s10u6
# zfs set mountpoint=/ rpool/ROOT/s10u6
```

- 5 Reboot the system.

When the option to boot a specific boot environment is presented, either in the GRUB menu or at the OpenBoot PROM prompt, select the boot environment whose mount points were just corrected.

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.

You will need to boot failsafe mode or boot from alternate media, depending on the severity of the error. In general, you can boot failsafe mode to recover a lost or unknown root password.

- [“How to Boot ZFS Failsafe Mode” on page 158](#)
- [“How to Boot ZFS From Alternate Media” on page 158](#)

If you need to recover a root pool or root pool snapshot, see [“Recovering the ZFS Root Pool or Root Pool Snapshots”](#) on page 159.

▼ How to Boot ZFS Failsafe Mode

1 Boot failsafe mode.

On a SPARC system:

```
ok boot -F failsafe
```

On an x86 system, select failsafe mode from the GRUB prompt.

2 Mount the ZFS BE on /a when prompted:

```
.
.
.
ROOT/zfsBE was found on rpool.
Do you wish to have it mounted read-write on /a? [y,n,?] y
mounting rpool on /a
Starting shell.
```

3 Change to the /a/etc directory.

```
# cd /a/etc
```

4 If necessary, set the TERM type.

```
# TERM=vt100
# export TERM
```

5 Correct the passwd or shadow file.

```
# vi shadow
```

6 Reboot the system.

```
# init 6
```

▼ How to Boot ZFS From Alternate Media

If a problem prevents the system from booting successfully or some other severe problem occurs, you will need to boot from a network install server or from a Solaris installation CD, import the root pool, mount the ZFS BE, and attempt to resolve the issue.

1 Boot from an installation CD or from the network.

■ SPARC:

```
ok boot cdrom -s
ok boot net -s
```

If you don't use the -s option, you will need to exit the installation program.

- x86: Select the network boot or boot from local CD option.
- 2 **Import the root pool and specify an alternate mount point. For example:**

```
# zpool import -R /a rpool
```
 - 3 **Mount the ZFS BE. For example:**

```
# zfs mount rpool/ROOT/zfsBE
```
 - 4 **Access the ZFS BE contents from the /a directory.**

```
# cd /a
```
 - 5 **Reboot the system.**

```
# init 6
```

Recovering the ZFS Root Pool or Root Pool Snapshots

The following sections describe how to perform the following tasks:

- [“How to Replace a Disk in the ZFS Root Pool” on page 159](#)
- [“How to Create Root Pool Snapshots” on page 161](#)
- [“How to Recreate a ZFS Root Pool and Restore Root Pool Snapshots” on page 162](#)
- [“How to Roll Back Root Pool Snapshots From a Failsafe Boot” on page 164](#)

▼ How to Replace a Disk in the 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 a smaller disk with a larger disk.
- A root pool disk is failing. In a non-redundant pool, if the disk is failing such that the system won't boot, you must 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 can attempt a disk replacement without booting 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 procedure in this section for an example of attaching an additional disk and detaching a root pool disk.

Some hardware requires that you take a disk offline and unconfigure it before attempting the `zpool replace` operation to replace a failed disk. For example:

```
# zpool offline rpool c1t0d0s0
# cfgadm -c unconfigure c1::disk/c1t0d0
<Physically remove failed disk c1t0d0>
```

```
<Physically insert replacement disk c1t0d0>
# cfdadm -c configure c1::disk/c1t0d0
# 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/c1t9d0s0
```

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

You must identify the boot device pathnames of the current disk and the new disk so that you can test booting from the replacement disk and also manually boot from the existing disk, if the replacement disk fails. In the example in the following procedure, the path name for current root pool disk (c1t10d0s0) is:

```
/pci@8,700000/pci@3/scsi@5/sd@a,0
```

The path name for the replacement boot disk (c1t9d0s0) is:

```
/pci@8,700000/pci@3/scsi@5/sd@9,0
```

1 Physically connect the replacement (or new) disk.

2 Confirm that the new disk has an SMI label and a slice 0.

For information about relabeling a disk that is intended for the root pool, see the following site:

http://www.solarisinternals.com/wiki/index.php/ZFS_Troubleshooting_Guide

3 Attach the new disk to the root pool.

For example:

```
# zpool attach rpool c1t10d0s0 c1t9d0s0
```

4 Confirm the root pool status.

For example:

```
# 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.
        scrub: resilver in progress, 25.47% done, 0h4m to go
config:
```

NAME	STATE	READ	WRITE	CKSUM
rpool	ONLINE	0	0	0
mirror-0	ONLINE	0	0	0
c1t10d0s0	ONLINE	0	0	0
c1t9d0s0	ONLINE	0	0	0


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

5 After the resilvering is completed, apply the boot blocks to the new disk.

Using syntax similar to the following:

- SPARC:

```
# installboot -F zfs /usr/platform/'uname -i'/lib/fs/zfs/bootblk /dev/rdisk/c1t9d0s0
```

- x86:

```
# installgrub /boot/grub/stage1 /boot/grub/stage2 /dev/rdisk/c1t9d0s0
```

6 Verify that you can boot from the new disk.

For example, on a SPARC based system, you would use syntax similar to the following:

```
ok boot /pci@8,700000/pci@3/scsi@5/sd@9,0
```

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

For example:

```
# zpool detach rpool c1t10d0s0
```

8 Set up the system to boot automatically from the new disk, either by using the `eeprom` command, the `setenv` command from the SPARC boot PROM, or reconfigure the PC BIOS.

▼ How to Create Root Pool Snapshots

You can create root pool snapshots for recovery purposes. The best way to create root pool snapshots is to perform a recursive snapshot of the root pool.

The following procedure creates a recursive root pool snapshot and stores the snapshot as a file in a pool on a remote system. If a root pool fails, the remote dataset can be mounted by using NFS and the snapshot file can be received into the recreated pool. You can instead store root pool snapshots as the actual snapshots in a pool on a remote system. Sending and receiving the snapshots from a remote system is a bit more complicated because you must configure `ssh` or use `rsh` while the system to be repaired is booted from the Solaris OS miniroot.

For information about remotely storing and recovering root pool snapshots, for the most up-to-date information about root pool recovery, go to this site:

http://www.solarisinternals.com/wiki/index.php/ZFS_Troubleshooting_Guide

Validating remotely stored snapshots as files or snapshots is an important step in root pool recovery. With either method, snapshots should be recreated on a routine basis, such as when the pool configuration changes or when the Solaris OS is upgraded.

In the following procedure, the system is booted from the zfsBE boot environment.

1 Create a pool and file system on a remote system to store the snapshots.

For example:

```
remote# zfs create rpool/snaps
```

2 Share the file system with the local system.

For example:

```
remote# zfs set sharenfs='rw=local-system,root=local-system' rpool/snaps
# share
-@rpool/snaps /rpool/snaps sec=sys,rw=local-system,root=local-system ""
```

3 Create a recursive snapshot of the root pool.

```
local# zfs snapshot -r rpool@0804
local# zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rpool                               6.17G 60.8G  98K    /rpool
rpool@0804                           0      -     98K    -
rpool/ROOT                           4.67G 60.8G  21K    /rpool/ROOT
rpool/ROOT@0804                       0      -     21K    -
rpool/ROOT/zfsBE                     4.67G 60.8G  4.67G  /
rpool/ROOT/zfsBE@0804                 386K   -     4.67G  -
rpool/dump                           1.00G 60.8G  1.00G  -
rpool/dump@0804                       0      -     1.00G  -
rpool/swap                           517M  61.3G  16K    -
rpool/swap@0804                       0      -     16K    -
```

4 Send the root pool snapshots to the remote system.

For example:

```
local# zfs send -Rv rpool@0804 > /net/remote-system/rpool/snaps/rpool.0804
sending from @ to rpool@0804
sending from @ to rpool/swap@0804
sending from @ to rpool/ROOT@0804
sending from @ to rpool/ROOT/zfsBE@0804
sending from @ to rpool/dump@0804
```

▼ How to Recreate a ZFS Root Pool and Restore Root Pool Snapshots

In this procedure, assume the following conditions:

- The ZFS root pool cannot be recovered.
- The ZFS root pool snapshots are stored on a remote system and are shared over NFS.

All the steps are performed on the local system.

1 Boot from a CD/DVD or the network.

- SPARC: Select one of the following boot methods:

```
ok boot net -s
ok boot cdrom -s
```

If you don't use -s option, you'll need to exit the installation program.

- x86: Select the option for booting from the DVD or the network. Then, exit the installation program.

2 Mount the remote snapshot dataset.

For example:

```
# mount -F nfs remote-system:/rpool/snaps /mnt
```

If your network services are not configured, you might need to specify the *remote-system's* IP address.

3 If the root pool disk is replaced and does not contain a disk label that is usable by ZFS, you must relabel the disk.

For more information about relabeling the disk, go to the following site:

http://www.solarisinternals.com/wiki/index.php/ZFS_Troubleshooting_Guide

4 Recreate the root pool.

For example:

```
# zpool create -f -o failmode=continue -R /a -m legacy -o cachefile=
/etc/zfs/zpool.cache rpool c1t1d0s0
```

5 Restore the root pool snapshots.

This step might take some time. For example:

```
# cat /mnt/rpool.0804 | zfs receive -Fdu rpool
```

Using the -u option means that the restored archive is not mounted when the zfs receive operation completes.

6 Verify that the root pool datasets are restored.

For example:

```
# zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rpool                                6.17G  60.8G   98K    /a/rpool
rpool@0804                            0      -     98K    -
rpool/ROOT                            4.67G  60.8G   21K    /legacy
rpool/ROOT@0804                        0      -     21K    -
rpool/ROOT/zfsBE                       4.67G  60.8G  4.67G   /a
rpool/ROOT/zfsBE@0804                  398K   -   4.67G   -
rpool/dump                             1.00G  60.8G  1.00G   -
```

```

rpool/dump@0804          0      -  1.00G  -
rpool/swap              517M  61.3G   16K   -
rpool/swap@0804         0      -   16K   -

```

7 Set the boot fs property on the root pool BE.

For example:

```
# zpool set bootfs=rpool/ROOT/zfsBE rpool
```

8 Install the boot blocks on the new disk.

SPARC:

```
# installboot -F zfs /usr/platform/'uname -i'/lib/fs/zfs/bootblk /dev/rdisk/c1t1d0s0
x86:
```

```
# installgrub /boot/grub/stage1 /boot/grub/stage2 /dev/rdisk/c1t1d0s0
```

9 Reboot the system.

```
# init 6
```

▼ How to Roll Back Root Pool Snapshots From a Failsafe Boot

This procedure assumes that existing root pool snapshots are available. In the example, they are available on the local system.

```

# zfs snapshot -r rpool@0804
# zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rpool                                6.17G  60.8G   98K    /rpool
rpool@0804                            0      -    98K    -
rpool/ROOT                            4.67G  60.8G   21K    /rpool/ROOT
rpool/ROOT@0804                        0      -    21K    -
rpool/ROOT/zfsBE                       4.67G  60.8G  4.67G    /
rpool/ROOT/zfsBE@0804                  398K   -  4.67G    -
rpool/dump                             1.00G  60.8G  1.00G    -
rpool/dump@0804                         0      -  1.00G    -
rpool/swap                             517M  61.3G   16K    -
rpool/swap@0804                         0      -   16K    -

```

1 Shut down the system and boot failsafe mode.

```

ok boot -F failsafe
ROOT/zfsBE was found on rpool.
Do you wish to have it mounted read-write on /a? [y,n,?] y
mounting rpool on /a

```

```
Starting shell.
```

2 Roll back each root pool snapshot.

```
# zfs rollback rpool@0804  
# zfs rollback rpool/ROOT@0804  
# zfs rollback rpool/ROOT/zfsBE@0804
```

3 Reboot to multiuser mode.

```
# init 6
```


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 167
- “Creating, Destroying, and Renaming ZFS File Systems” on page 168
- “Introducing ZFS Properties” on page 171
- “Querying ZFS File System Information” on page 183
- “Managing ZFS Properties” on page 185
- “Mounting and Sharing ZFS File Systems” on page 190
- “Setting ZFS Quotas and Reservations” on page 196

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 257.](#)

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 168
- “Destroying a ZFS File System” on page 169
- “Renaming a ZFS File System” on page 170

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 45.

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

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

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 `bonwick` file system is mounted at `/tank/home/bonwick`. For more information about automatically managed mount points, see “[Managing ZFS Mount Points](#)” on page 190.

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 171](#).

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 191](#).

In the following example, the `tabriz` file system is destroyed:

```
# zfs destroy tank/home/tabriz
```



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/ahrens
cannot unmount 'tank/home/ahrens': Device busy
```

```
# zfs destroy -f tank/home/ahrens
```

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/billm
tank/ws/bonwick
tank/ws/maybee
```

```
# 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/schrock
cannot destroy 'tank/home/schrock': filesystem has dependent clones
use '-R' to destroy the following datasets:
tank/clones/schrock-clone
```

```
# zfs destroy -R tank/home/schrock
```



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 `kustarz` to `kustarz_old`:

```
# zfs rename tank/home/kustarz tank/home/kustarz_old
```

The following example shows how to use `zfs rename` to relocate a file system:

```
# zfs rename tank/home/maybee tank/ws/maybee
```

In this example, the `maybee` 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, `rename` operation fails.

For more information about quotas, see [“Setting ZFS Quotas and Reservations”](#) on page 196.

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 206.

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 178
- “Settable ZFS Native Properties” on page 179
- “ZFS User Properties” on page 182

Properties are divided into two types, native properties and user-defined properties. Native properties either export 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 182.

Most settable properties are also inheritable. An inheritable property is a property that, when set on a parent dataset, 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 185.
<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 182.

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 224.

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
<code>aclmode</code>	String	<code>groupmask</code>	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 224.
<code>atime</code>	Boolean	<code>on</code>	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 dataset 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 54.</p>
<code>canmount</code>	Boolean	<code>on</code>	<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 dataset can only be mounted and unmounted explicitly. The dataset is not mounted automatically when the dataset 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 “<code>canmount</code> Property” on page 180.</p>
<code>checksum</code>	String	<code>on</code>	Controls the checksum used to verify data integrity. The default value is <code>on</code> , which automatically selects an appropriate algorithm, currently <code>fletcher4</code> . The values are <code>on</code> , <code>off</code> , <code>fletcher2</code> , <code>fletcher4</code> , and <code>sha256</code> . A value of <code>off</code> disables integrity checking on user data. A value of <code>off</code> is not recommended.

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
<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>
<code>copies</code>	Number	1	<p>Sets the number of copies of user data per file system. Available values are 1, 2, or 3. These copies are in addition to any pool-level redundancy. Disk space used by multiple copies of user data is charged to the corresponding file and dataset, and counts against quotas and reservations. In addition, the <code>used</code> property is updated when multiple copies are enabled. Consider setting this property when the file system is created because changing this property on an existing file system only affects newly written data.</p>
<code>creation</code>	String	N/A	<p>Read-only property that identifies the date and time that a dataset was created.</p>
<code>devices</code>	Boolean	<code>on</code>	<p>Controls whether device files in a file system can be opened.</p>
<code>exec</code>	Boolean	<code>on</code>	<p>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.</p>
<code>mounted</code>	Boolean	N/A	<p>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>.</p>

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
mountpoint	String	N/A	<p>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.</p> <p>For more information about using this property, see “Managing ZFS Mount Points” on page 190.</p>
primarycache	String	all	<p>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.</p>
origin	String	N/A	<p>Read-only property for cloned file systems or volumes that identifies the snapshot from which the clone was created. The origin cannot be destroyed (even with the <code>-r</code> or <code>-f</code> option) as long as a clone exists.</p> <p>Non-cloned file systems have an origin of <code>none</code>.</p>
quota	Number (or none)	none	<p>Limits the amount of disk space a dataset and its descendents can consume. This property enforces a hard limit on the amount of disk space used, including all space consumed by descendents, such as file systems and snapshots. Setting a quota on a descendent of a dataset 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 <code>volsize</code> property acts as an implicit quota.</p> <p>For information about setting quotas, see “Setting Quotas on ZFS File Systems” on page 196.</p>
readonly	Boolean	off	<p>Controls whether a dataset can be modified. When set to <code>on</code>, no modifications can be made.</p> <p>The property abbreviation is <code>rdonly</code>.</p>
recordsize	Number	128K	<p>Specifies a suggested block size for files in a file system.</p> <p>The property abbreviation is <code>recsize</code>. For a detailed description, see “recordsize Property” on page 181.</p>

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
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>refreserv</code>.</p>
reservation	Number (or none)	none	<p>Sets the minimum amount of disk space guaranteed to a dataset and its descendents. When the amount of disk space used is below this value, the dataset is treated as if it were using the amount of space specified by its reservation. Reservations are accounted for in the parent dataset's disk space used, and count against the parent dataset'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 200.</p>

TABLE 6-1 ZFS Native Property Descriptions (Continued)

Property Name	Type	Default Value	Description
secondarycache	String	all	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.
setuid	Boolean	on	Controls whether the <code>setuid</code> bit is honored in a file system.
shareiscsi	String	off	Controls whether a ZFS volume is shared as an iSCSI target. The property values are <code>on</code> , <code>off</code> , and <code>type=disk</code> . You might want to set <code>shareiscsi=on</code> for a file system so that all ZFS volumes within the file system are shared by default. However, setting this property on a file system has no direct effect.
sharenfs	String	off	Controls whether a file system is available over NFS and what options are used. If set to <code>on</code> , the <code>zfs share</code> command is invoked with no options. Otherwise, the <code>zfs share</code> command is invoked with options equivalent to the contents of this property. If set to <code>off</code> , the file system is managed by using the legacy <code>share</code> and <code>unshare</code> commands and the <code>dfstab</code> file. For more information about sharing ZFS file systems, see “Sharing and Unsharing ZFS File Systems” on page 194 .
snapdir	String	hidden	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 203 .
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	Read-only property that identifies the amount of disk space consumed by a dataset and all its descendents. For a detailed description, see “The used Property” on page 179 .
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 refreservation reservations. The property abbreviation is usedds.
usedbyrefreservation	Number	off	Read-only property that identifies the amount of disk space that is used by a refreservation set on a dataset, which would be freed if the refreservation was removed. The property abbreviation is usedrefreserv.
usedbysnapshots	Number	off	Read-only property that identifies the amount of disk space that is consumed by snapshots of a dataset. In particular, it is the amount of disk space that would be freed if all of this dataset's snapshots were destroyed. Note that this value is not simply the sum of the snapshots' used properties, because space can be shared by multiple snapshots. The property abbreviation is usedsnap.
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.
volsize	Number	N/A	For volumes, specifies the logical size of the volume. For a detailed description, see “ volsize Property ” on page 181.
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
zoned	Boolean	N/A	Indicates whether a dataset 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 259.
xattr	Boolean	on	Indicates whether extended attributes are enabled (on) or disabled (off) 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
- mounted
- origin
- referenced
- type
- used

For detailed information, see “The used Property” on page 179.

- 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 54.

The used Property

The used property is a read-only property that identifies the amount of disk space consumed by this dataset and all its descendents. This value is checked against the dataset's quota and reservation. The disk space used does not include the dataset's reservation, but does consider the reservation of any descendent datasets. The amount of disk space that a dataset consumes from its parent, as well as the amount of disk space that is freed if the dataset is recursively destroyed, is the greater of its space used and its reservation.

When snapshots are created, their disk space is initially shared between the snapshot and the file system, and possibly with previous snapshots. As the file system changes, disk space that was previously shared becomes unique to the snapshot and is counted in the snapshot's space used. The disk space that is used by a snapshot accounts for its unique data. Additionally, deleting snapshots can increase the amount of disk space unique to (and used by) other snapshots. For more information about snapshots and space issues, see [“Out of Space Behavior” on page 54](#).

The amount of disk space used, available, and referenced does not include pending changes. Pending changes are generally accounted for within a few seconds. Committing a change to a disk using the `fsync(3c)` or `O_SYNC` function does not necessarily guarantee that the disk space usage information will be updated immediately.

The `usedbychildren`, `usedbydataset`, `usedbyrefreservation`, and `usedbysnapshots` property information can be displayed with the `zfs list -o space` command. These properties identify the used property into disk space that is consumed by descendents. For more information, see [Table 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 185](#) or by using the `zfs create` command as described in [“Creating a ZFS File System” on page 168](#). 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 196](#).

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 224](#).
- `aclmode`
For a detailed description, see [“ACL Properties” on page 224](#).

- `atime`
- `canmount`
- `checksum`
- `compression`
- `copies`
- `devices`
- `exec`
- `mountpoint`
- `primarycache`
- `quota`
- `readonly`
- `recordsize`

For a detailed description, see “[recordsize Property](#)” on page 181.

- `refquota`
- `refreservation`
- `reservation`
- `secondarycache`
- `shareiscsi`
- `sharenfs`
- `setuid`
- `snapdir`
- `version`
- `volsize`

For a detailed description, see “[volsize Property](#)” on page 181.

- `volblocksize`
- `zoned`
- `xattr`

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 dataset 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 dataset can only be mounted explicitly, not automatically. This value setting is used by the Oracle Solaris upgrade software so that only those datasets belonging to the active boot environment are mounted at boot time.

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`.

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 257](#).

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.`, `sun.` 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 171](#).

For example:

```
# zfs list
NAME          USED  AVAIL  REFER  MOUNTPOINT
pool          476K  16.5G  21K    /pool
pool/clone    18K   16.5G  18K    /pool/clone
pool/home     296K  16.5G  19K    /pool/home
pool/home/marks 277K  16.5G  277K   /pool/home/marks
pool/home/marks@snap 0      -      277K   -
pool/test     18K   16.5G  18K    /test
```

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 -r pool/home/marks
NAME          USED  AVAIL  REFER  MOUNTPOINT
pool/home/marks 277K  16.5G  277K   /pool/home/marks
pool/home/marks@snap 0      -      277K   -
```

You can use the `zfs list` command with the mount point of a file system. For example:

```
# zfs list /pool/home/marks
NAME          USED  AVAIL  REFER  MOUNTPOINT
pool/home/marks 277K  16.5G  277K   /pool/home/marks
```

The following example shows how to display basic information about `tank/home/chua` and all of its descendent datasets:

```
# zfs list -r tank/home/chua
NAME                                USED  AVAIL  REFER  MOUNTPOINT
tank/home/chua                     26.0K  4.81G  10.0K  /tank/home/chua
tank/home/chua/projects             16K   4.81G   9.0K   /tank/home/chua/projects
tank/home/chua/projects/fs1         8K   4.81G    8K   /tank/home/chua/projects/fs1
tank/home/chua/projects/fs2         8K   4.81G    8K   /tank/home/chua/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 171](#). 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 -o name,sharenfs,mountpoint
NAME                                SHARENFS  MOUNTPOINT
tank                                off        /tank
tank/home                           on        /tank/home
tank/home/ahrens                     on        /tank/home/ahrens
tank/home/bonwick                    on        /tank/home/bonwick
tank/home/chua                       on        /tank/home/chua
tank/home/eschrock                   on        legacy
tank/home/moore                      on        /tank/home/moore
tank/home/tabriz                     ro        /tank/home/tabriz
```

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 -t filesystem -o name,used
NAME                USED
pool                 476K
pool/clone           18K
pool/home            296K
pool/home/marks     277K
pool/test            18K
```

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 -H -o name
pool
pool/clone
pool/home
pool/home/marks
pool/home/marks@snap
pool/test
```

Managing ZFS Properties

Dataset properties are managed through the `zfs` command's `set`, `inherit`, and `get` subcommands.

- [“Setting ZFS Properties” on page 185](#)
- [“Inheriting ZFS Properties” on page 186](#)
- [“Querying ZFS Properties” on page 187](#)

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 179](#).

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 order of magnitude): BKMGTPEZ. Any of these suffixes can be followed by an optional `b`, indicating bytes, with the exception of the `B` suffix, which already indicates bytes. The following four invocations of `zfs set` are equivalent numeric expressions that set the `quota` property to be set to the value of 50 GB on the `tank/home/marks` file system:

```
# zfs set quota=50G tank/home/marks
# zfs set quota=50g tank/home/marks
# zfs set quota=50GB tank/home/marks
# zfs set quota=50gb tank/home/marks
```

The values of non-numeric properties are case-sensitive and must be in lowercase letters, with the exception of `mountpoint` and `sharenfs`. The values of these properties 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 dataset, unless a quota or reservation is explicitly set on the descendent dataset. 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 dataset.

The following example uses the `zfs set` command to turn on compression for the `tank/home/bonwick` 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/bonwick
# zfs get -r compression tank
NAME          PROPERTY      VALUE          SOURCE
tank          compression   off            default
tank/home     compression   off            default
tank/home/bonwick compression   on             local
# zfs inherit compression tank/home/bonwick
# zfs get -r compression tank
NAME          PROPERTY      VALUE          SOURCE
tank          compression   off            default
tank/home     compression   off            default
tank/home/bonwick 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 datasets.

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 183. 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 193.
-(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     Tue Jun 29 11:44 2010 -
tank/home     used         21K           -
tank/home     available    66.9G        -
tank/home     referenced   21K           -
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     aclmode      groupmask     default
tank/home     aclinherit   restricted     default
tank/home     canmount     on            default
tank/home     shareiscsi   off           default
tank/home     xattr        on            default
tank/home     copies       1             default
tank/home     version      4             -
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 21K          -
tank/home     usedbychildren 0             -
tank/home     usedbyrefreservation 0           -
tank/home     logbias      latency       default
```

Note – The `casesensitivity`, `nbmand`, `normalization`, `sharesmb`, `utf8only`, and `vscan` properties are not fully operational in the Oracle Solaris 10 release because the Oracle Solaris SMB service is not supported in the Oracle Solaris 10 release.

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 `pool`.

```
# zfs get -s local all pool
NAME                PROPERTY          VALUE            SOURCE
pool                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 dataset. In the following example, all temporary properties on all datasets within `tank` are recursively displayed:

```
# zfs get -r -s temporary all tank
NAME                PROPERTY          VALUE            SOURCE
tank/home           atime             off             temporary
tank/home/bonwick   atime             off             temporary
tank/home/marks     atime             off             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/bonwick   atime             off             local
tank/home/marks     quota             50G            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 171 section.
- A comma-separated list of literal fields, name, value, property, and source, to be output followed by a space and an argument, which is a comma-separated list of properties.

The following example shows how to retrieve a single value by using the `-H` and `-o` options of `zfs get`:

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

The `-p` option reports numeric values as their exact values. For example, 1 MB would be reported as 1000000. This option can be used as follows:

```
# zfs get -H -o value -p used tank/home
182983742
```

You can use the `-r` option, along with any of the preceding options, to recursively retrieve the requested values for all descendents. The following example uses the `-H`, `-o`, and `-r` options to retrieve the dataset name and the value of the `used` property for `export/home` and its descendents, while omitting the header output:

```
# zfs get -H -o name,value -r used export/home
export/home          5.57G
export/home/marks    1.43G
export/home/maybee   2.15G
```

Mounting and Sharing ZFS File Systems

This section describes how mount points and shared file systems are managed in ZFS.

- [“Managing ZFS Mount Points” on page 190](#)
- [“Mounting ZFS File Systems” on page 192](#)
- [“Using Temporary Mount Properties” on page 193](#)
- [“Unmounting ZFS File Systems” on page 194](#)
- [“Sharing and Unsharing ZFS File Systems” on page 194](#)

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 dataset at creation time by using `zpool create's -m` option. For more information about creating pools, see [“Creating a ZFS Storage Pool” on page 63](#).

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 when the `zfs mount -a` command is invoked, 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 [“canmount Property” on page 180](#).

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 191.

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 dataset whose `mountpoint` property is not `legacy` is managed by ZFS. In the following example, a dataset 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 datasets of this type. The following examples show how to set up and manage a ZFS dataset in legacy mode:

```
# zfs set mountpoint=legacy tank/home/eschrock
# 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/eschrock -          /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 42](#).

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
tank                /tank
tank/home           /tank/home
tank/home/bonwick   /tank/home/bonwick
tank/ws             /tank/ws
```

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. To force a mount on top of a nonempty directory, you must use the `-O` option. For example:

```
# zfs mount tank/home/lalt
cannot mount '/export/home/lalt': directory is not empty
use legacy mountpoint to allow this behavior, or use the -O flag
# zfs mount -O tank/home/lalt
```

Legacy mount points must be managed through legacy tools. An attempt to use ZFS tools results in an error. For example:

```
# zfs mount pool/home/billm
cannot mount 'pool/home/billm': 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 dataset. 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
devices	devices/nodevices
exec	exec/noexec
nbmand	nbmand/nonbmand
readonly	ro/rw
setuid	setuid/nosetuid
xattr	xattr/noaxttr

The mount option `nosuid` is an alias for `nodevices`, `nosetuid`.

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 dataset 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/perrin` file system. The file system is assumed to be unmounted.

```
# zfs mount -o ro tank/home/perrin
```

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 tank/home/perrin
# zfs get atime tank/home/perrin
NAME                PROPERTY    VALUE    SOURCE
tank/home/perrin    atime      off      temporary
```

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 tank/home/tabriz
```

In the following example, the file system is unmounted by its mount point:

```
# zfs unmount /export/home/tabriz
```

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/eschrock
cannot unmount '/export/home/eschrock': Device busy
# zfs unmount -f tank/home/eschrock
```

To provide for backward compatibility, the legacy `umount` command can be used to unmount ZFS file systems. For example:

```
# umount /export/home/bob
```

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

Sharing and Unsharing ZFS File Systems

ZFS can automatically share file systems by setting the `sharenfs` property. Using this property, you do not have to modify the `/etc/dfs/dfstab` file when a new file system is shared. The `sharenfs` property is a comma-separated list of options to pass to the `share` command. The value `on` is an alias for the default share options, which provides `read/write` permissions to anyone. The value `off` indicates that the file system is not managed by ZFS and can be shared through traditional means, such as the `/etc/dfs/dfstab` file. All file systems whose `sharenfs` property is not `off` are shared during boot.

Controlling Share Semantics

By default, all file systems are unshared. To share a new file system, use `zfs set` syntax similar to the following:

```
# zfs set sharenfs=on tank/home/eschrock
```

The `sharenfs` property is inherited, and file systems are automatically shared on creation if their inherited property is not `off`. For example:

```
# zfs set sharenfs=on tank/home
# zfs create tank/home/bricker
# zfs create tank/home/tabriz
# zfs set sharenfs=ro tank/home/tabriz
```

Both `tank/home/bricker` and `tank/home/tabriz` are initially shared as writable because they inherit the `sharenfs` property from `tank/home`. After the property is set to `ro` (read only), `tank/home/tabriz` is shared as read-only regardless of the `sharenfs` property that is set for `tank/home`.

Unsharing ZFS File Systems

Although most file systems are automatically shared or unshared during boot, creation, and destruction, file systems sometimes need to be explicitly unshared. To do so, use the `zfs unshare` command. For example:

```
# zfs unshare tank/home/tabriz
```

This command unshares the `tank/home/tabriz` file system. To unshare all ZFS file systems on the system, you need to use the `-a` option.

```
# zfs unshare -a
```

Sharing ZFS File Systems

Most of the time, the automatic behavior of ZFS with respect to sharing file system on boot and creation is sufficient for normal operations. If, for some reason, you unshare a file system, you can share it again by using the `zfs share` command. For example:

```
# zfs share tank/home/tabriz
```

You can also share all ZFS file systems on the system by using the `-a` option.

```
# zfs share -a
```

Legacy Share Behavior

If the `sharenfs` property is set to `off`, then ZFS does not attempt to share or unshare the file system at any time. This value enables you to administer file system sharing through traditional means, such as the `/etc/dfs/dfstab` file.

Unlike the legacy `mount` command, the legacy `share` and `unshare` commands can still function on ZFS file systems. As a result, you can manually share a file system with options that differ from the options of the `sharenfs` property. This administrative model is discouraged. Choose to manage NFS shares either completely through ZFS or completely through the `/etc/dfs/dfstab` file. The ZFS administrative model is designed to be simpler and less work than the traditional model.

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 dataset on which they are set and all descendents of that dataset.

That is, if a quota is set on the `tank/home` dataset, 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 dataset 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 datasets and their descendents.
- The `refquota` and `refreservation` properties are appropriate for managing disk space consumed by datasets.
- 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 196](#) and [“Setting Reservations on ZFS File Systems” on page 200](#).

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/bonwick`:

```
# zfs set quota=10G tank/home/bonwick
# zfs get quota tank/home/bonwick
NAME                PROPERTY      VALUE          SOURCE
tank/home/bonwick  quota        10.0G         local
```

Quotas also affect the output of the `zfs list` and `df` commands. For example:

```
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
tank/home           16.5K 33.5G  8.50K  /export/home
tank/home/bonwick  15.0K 10.0G  8.50K  /export/home/bonwick
tank/home/bonwick/ws 6.50K 10.0G  8.50K  /export/home/bonwick/ws
# df -h /export/home/bonwick
Filesystem          size  used  avail  capacity  Mounted on
tank/home/bonwick  10G   8K   10G   1%        /export/home/bonwick
```

Note that although `tank/home` has 33.5 GB of disk space available, `tank/home/bonwick` and `tank/home/bonwick/ws` each have only 10 GB of disk space available, due to the quota on `tank/home/bonwick`.

You cannot set a quota to an amount less than is currently being used by a dataset. For example:

```
# zfs set quota=10K tank/home/bonwick
cannot set quota for 'tank/home/bonwick': size is less than current used or reserved space
```

You can set a `refquota` on a dataset that limits the amount of disk space that the dataset can consume. This hard limit does not include disk space that is consumed by descendents. For example:

```
# zfs set refquota=10g students/studentA
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
profs               106K 33.2G  18K   /profs
students            57.7M 33.2G  19K   /students
students/studentA  57.5M 9.94G  57.5M /students/studentA
# zfs snapshot students/studentA@today
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
profs               106K 33.2G  18K   /profs
students            57.7M 33.2G  19K   /students
students/studentA  57.5M 9.94G  57.5M /students/studentA
students/studentA@today  0    -    57.5M -
```

For additional convenience, you can set another quota on a dataset to help manage the disk space that is consumed by snapshots. For example:

```
# zfs set quota=20g students/studentA
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
profs               106K 33.2G  18K   /profs
students            57.7M 33.2G  19K   /students
students/studentA  57.5M 9.94G  57.5M /students/studentA
students/studentA@today  0    -    57.5M -
```

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@staff=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@staff students/labstaff
NAME          PROPERTY  VALUE      SOURCE
students/labstaff  groupquota@staff  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     227M  none
POSIX User  student1 455M  10G
# zfs groupspace students/labstaff
TYPE      NAME      USED  QUOTA
POSIX Group  root     217M  none
POSIX Group  staff    217M  20G
```

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  455M        local
# zfs get groupused@staff students/labstaff
NAME          PROPERTY  VALUE      SOURCE
students/labstaff  groupused@staff  217M        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@user1=none students/compsci
# zfs set groupquota@staff=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      227M  none
POSIX User student1 455M  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      227M  none
POSIX User student1 455M  10G
# quota student1
# quota -v student1
Disk quotas for student1 (uid 201):
Filesystem      usage  quota  limit      timeleft  files  quota  limit      timeleft
/students/compsci
                466029 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/moore
# zfs get reservation tank/home/moore
NAME                PROPERTY    VALUE    SOURCE
tank/home/moore     reservation 5G       local
```

Reservations can affect the output of the `zfs list` command. For example:

```
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
tank/home           5.00G 33.5G  8.50K  /export/home
tank/home/moore     15.0K 33.5G  8.50K  /export/home/moore
```

Note that `tank/home` is using 5 GB of disk space, although the total amount of space referred to by `tank/home` and its descendents is much less than 5 GB. The used space reflects the space reserved for `tank/home/moore`. Reservations are considered in the used disk space calculation of the parent dataset 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/moore
# zfs set reservation=5G tank/home/moore
# zfs get reservation tank/home/moore
NAME                PROPERTY    VALUE    SOURCE
tank/home/moore     reservation 5.00G    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.

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 203
- “Creating and Destroying ZFS Snapshots” on page 204
- “Displaying and Accessing ZFS Snapshots” on page 207
- “Rolling Back a ZFS Snapshot” on page 208
- “Overview of ZFS Clones” on page 209
- “Creating a ZFS Clone” on page 210
- “Destroying a ZFS Clone” on page 210
- “Replacing a ZFS File System With a ZFS Clone” on page 210
- “Sending and Receiving ZFS Data” on page 211

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 211](#).

- [“Creating and Destroying ZFS Snapshots” on page 204](#)
- [“Displaying and Accessing ZFS Snapshots” on page 207](#)
- [“Rolling Back a ZFS Snapshot” on page 208](#)

Creating and Destroying ZFS Snapshots

Snapshots are created by using the `zfs snapshot` 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 45](#).

In the following example, a snapshot of `tank/home/ahrens` that is named `friday` is created.

```
# zfs snapshot tank/home/ahrens@friday
```

You can create snapshots for all descendent file systems by using the `-r` option. For example:

```
# zfs snapshot -r tank/home@now
# zfs list -t snapshot
NAME                                USED  AVAIL  REFER  MOUNTPOINT
rpool/ROOT/zfs2BE@zfs2BE           78.3M  -      4.53G  -
tank/home@now                       0      -      26K    -
tank/home/ahrens@now                0      -      259M   -
tank/home/anne@now                  0      -      156M   -
tank/home/bob@now                    0      -      156M   -
tank/home/cindys@now                 0      -      104M   -
```

Snapshots have no modifiable properties. Nor can dataset properties be applied to a snapshot. For example:

```
# zfs set compression=on tank/home/ahrens@now
cannot set compression property for 'tank/home/ahrens@now': snapshot
properties cannot be modified
```

Snapshots are destroyed by using the `zfs destroy` command. For example:

```
# zfs destroy tank/home/ahrens@now
```

A dataset cannot be destroyed if snapshots of the dataset exist. For example:

```
# zfs destroy tank/home/ahrens
cannot destroy 'tank/home/ahrens': filesystem has children
use '-r' to destroy the following datasets:
```

```
tank/home/ahrens@tuesday
tank/home/ahrens@wednesday
tank/home/ahrens@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 169](#).

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 one whenever a hold is put on a snapshot and decreases by one whenever a hold is released.

In the previous Solaris release, a snapshot could only be destroyed by using the `zfs destroy` command if it had no clones. In this 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/cindys/snap@1`.

```
# zfs hold keep tank/home/cindys@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/cindys@snap1
cannot destroy 'tank/home/cindys@snap1': dataset is busy
```

If you want to destroy a held snapshot, use the `-d` option. For example:

```
# zfs destroy -d tank/home/cindys@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   Thu Jul 15 11:25:39 2010
```

```
# zfs holds -r tank/home@now
NAME      TAG      TIMESTAMP
tank/home/cindys@now  keep   Thu Jul 15 11:25:39 2010
tank/home/mark@now    keep   Thu Jul 15 11:25:39 2010
tank/home@now         keep   Thu Jul 15 11:25:39 2010
```

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/cindys@083006 tank/home/cindys@today
```

In addition, the following shortcut syntax is equivalent to the preceding syntax:

```
# zfs rename tank/home/cindys@083006 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/cindys@today pool/home/cindys@Saturday
cannot rename to 'pool/home/cindys@today': snapshots must be part of same dataset
```

You can recursively rename snapshots by using the `zfs rename -r` command. For example:

```
# zfs list
NAME      USED  AVAIL  REFER  MOUNTPOINT
users    270K  16.5G   22K    /users
users/home  76K  16.5G   22K    /users/home
```

```

users/home@yesterday          0      -    22K  -
users/home/markm             18K   16.5G  18K  /users/home/markm
users/home/markm@yesterday   0      -    18K  -
users/home/marks             18K   16.5G  18K  /users/home/marks
users/home/marks@yesterday   0      -    18K  -
users/home/neil              18K   16.5G  18K  /users/home/neil
users/home/neil@yesterday    0      -    18K  -
# zfs rename -r users/home@yesterday @2daysago
# zfs list -r users/home
NAME                               USED  AVAIL  REFER  MOUNTPOINT
users/home                         76K   16.5G  22K    /users/home
users/home@2daysago               0      -    22K    -
users/home/markm                  18K   16.5G  18K    /users/home/markm
users/home/markm@2daysago         0      -    18K    -
users/home/marks                   18K   16.5G  18K    /users/home/marks
users/home/marks@2daysago         0      -    18K    -
users/home/neil                   18K   16.5G  18K    /users/home/neil
users/home/neil@2daysago          0      -    18K    -

```

Displaying and Accessing ZFS Snapshots

You can enable or disable the display of snapshot listings in the `zfs list` output by using the `listsnapshots` pool property. This property is enabled by default.

If you disable this property, you can 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 on          default
# zpool set listsnapshots=off tank
# zpool get listsnapshots tank
NAME PROPERTY      VALUE      SOURCE
tank listsnapshots off          local

```

Snapshots of file systems are accessible in the `.zfs/snapshot` directory within the root of the file system. For example, if `tank/home/ahrens` is mounted on `/home/ahrens`, then the `tank/home/ahrens@thursday` snapshot data is accessible in the `/home/ahrens/.zfs/snapshot/thursday` directory.

```

# ls /tank/home/ahrens/.zfs/snapshot
tuesday wednesday thursday

```

You can list snapshots as follows:

```

# zfs list -t snapshot
NAME                               USED  AVAIL  REFER  MOUNTPOINT
pool/home/anne@monday              0      -    780K  -
pool/home/bob@monday               0      -    1.01M  -
tank/home/ahrens@tuesday           8.50K  -    780K  -
tank/home/ahrens@wednesday         8.50K  -    1.01M  -
tank/home/ahrens@thursday          0      -    1.77M  -
tank/home/cindys@today             8.50K  -    524K  -

```

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@now                       Wed Jun 30 16:16 2010
tank/home/ahrens@now                Wed Jun 30 16:16 2010
tank/home/anne@now                  Wed Jun 30 16:16 2010
tank/home/bob@now                   Wed Jun 30 16:16 2010
tank/home/cindys@now                Wed Jun 30 16:16 2010
```

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
# zfs list -ro space tank/home
NAME                                AVAIL    USED    USED SNAP    USED DS    USED REFRESERV    USED CHILD
tank/home                            66.3G    675M    0            26K        0                675M
tank/home@now                          -         0        -            -          -                 -
tank/home/ahrens                       66.3G    259M    0            259M       0                 0
tank/home/ahrens@now                    -         0        -            -          -                 -
tank/home/anne                          66.3G    156M    0            156M       0                 0
tank/home/anne@now                       -         0        -            -          -                 -
tank/home/bob                           66.3G    156M    0            156M       0                 0
tank/home/bob@now                        -         0        -            -          -                 -
tank/home/cindys                        66.3G    104M    0            104M       0                 0
tank/home/cindys@now                     -         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/ahrens` file system is rolled back to the tuesday snapshot:

```
# zfs rollback tank/home/ahrens@tuesday
cannot rollback to 'tank/home/ahrens@tuesday': more recent snapshots exist
use '-r' to force deletion of the following snapshots:
tank/home/ahrens@wednesday
tank/home/ahrens@thursday
# zfs rollback -r tank/home/ahrens@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/ahrens
NAME                CREATION
tank/home/ahrens@now Wed Jun 30 16:16 2010
```

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 dataset 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 185](#).

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 210](#)
- [“Destroying a ZFS Clone” on page 210](#)

- [“Replacing a ZFS File System With a ZFS Clone” on page 210](#)

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/ahrens/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/ahrens/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 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/ahrens/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 210](#).

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.

- [“Sending a ZFS Snapshot” on page 212](#)
- [“Receiving a ZFS Snapshot” on page 213](#)
- [“Sending and Receiving Complex ZFS Snapshot Streams” on page 214](#)
- [“Saving ZFS Data With Other Backup Products” on page 212](#)

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 203.
- **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 Solaris 10 Release Notes or the ZFS FAQ, available here:

<http://hub.opensolaris.org/bin/view/Community+Group+zfs/faq/#backupsoftware>

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/data@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:

```
host1# zfs send tank/dana@snap1 | ssh host2 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:

```
host1# zfs send -i tank/dana@snap1 tank/dana@snap2 | ssh host2 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:

```
host1# zfs send -i snap1 tank/dana@snap2 > ssh host2 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 snapshot1:

```
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:

```
host2# 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:

```
host1# zfs send -i tank/dana@snap2 tank/dana@snap3 | ssh host2 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:

```
host1# zfs send -i tank/dana@snap4 tank/dana@snap5 | ssh host2 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
```

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.
- 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
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	-

EXAMPLE 7-1 Sending and Receiving Complex ZFS Snapshot Streams (Continued)

```
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/fsclonesnap-I
# zfs destroy pool/clone@snapA
# zfs destroy pool/clone
# zfs receive -F pool/clone < /snaps/fsclonesnap-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` dataset and its descendents, and to send the replicated stream to another pool, `users2`.

```
# zfs create users2 mirror c0t1d0 c1t1d0
# zfs receive -F -d users2 < /snaps/users-R
```


EXAMPLE 7-1 Sending and Receiving Complex ZFS Snapshot Streams *(Continued)*

```
# 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 to Protect Oracle Solaris ZFS Files

This chapter provides information about using access control lists (ACLs) to protect your ZFS files. ACLs provide more granular permissions than the standard UNIX permissions.

The following sections are provided in this chapter:

- “New Solaris ACL Model” on page 219
- “Setting ACLs on ZFS Files” on page 225
- “Setting and Displaying ACLs on ZFS Files in Verbose Format” on page 228
- “Setting and Displaying ACLs on ZFS Files in Compact Format” on page 239

New Solaris ACL Model

Previous versions of the Solaris OS 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 follow:

- The new ACL model is based on the NFSv4 specification and is similar to NT-style ACLs.
- The new model provides a much more granular set of access privileges. For more information, see [Table 8-2](#).
- ACLs are set and displayed with the `chmod` and `ls` commands rather than the `setfacl` and `getfacl` commands.
- The new model provides richer inheritance semantics for designating how access privileges are applied from a directory to subdirectories, and so on. For more information, see “[ACL Inheritance](#)” on page 223.

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 based ACLs use a single entry to define which permissions are allowed and which 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 the permissions that weren't defined in that ACE are allowed or denied.

Translation between NFSv4 ACLs and POSIX-draft ACLs is as follows:

- If you use an ACL-aware utility, such as the `cp`, `mv`, `tar`, `cpio`, or `rcp` command, to transfer UFS files with ACLs to a ZFS file system, the POSIX-draft ACLs are translated into the equivalent NFSv4 ACLs.
- Some NFSv4 ACLs are translated into POSIX-draft ACLs. You see a message similar to the following if an NFSv4 ACL isn't translated into 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-draft ACLs, they are translated into NFSv4 ACLs.
- If you attempt to set an NFSv4 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-draft ACL on a ZFS file, you 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 212](#).

Syntax Descriptions for Setting ACLs

Two basic ACL formats follow:

Syntax for Setting Trivial ACLs

An ACL is *trivial* in that it only represents the traditional UNIX owner/group/other entries.

```
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 8-3](#).

deny | allow

Identifies whether the access permissions are granted or denied.

In the following example, the *ACL-entry-ID* value is not relevant:

```
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
```

In this example, the **2**, known as the *index-ID* designation, 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** in the `chmod` command syntax, 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. This entry 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. This entry 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	Placeholder. Not currently implemented.
delete	d	Permission to delete a file.
delete_child	D	Permission to delete a file or a directory within a directory.
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 (<code>ls</code>).
read_attributes	a	Permission to read the basic attributes (non-ACLs) of a file. Think of basic attributes as the <code>stat</code> -level attributes. Allowing this access mask bit means that the entity can execute <code>ls(1)</code> and <code>stat(2)</code> .
read_data	r	Permission to read the contents of a file.
read_xattr	R	Permission to read the extended attributes of a file or to perform a lookup in the file's extended attributes directory.
synchronize	s	Placeholder. 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 time stamps associated with a file or directory to an arbitrary value.
write_acl	C	Permission to write the ACL or 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> command on the file. Permission to take ownership of a file or permission to change the group ownership of a 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.

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 permissions on the parent directory.

By default, ACLs are not propagated. If you set a non-trivial ACL on a directory, it is not inherited by 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 8-3 ACL Inheritance Flags

Inheritance Flag	Compact Inheritance Flag	Description
<code>file_inherit</code>	<code>f</code>	Inherit the ACL from the parent directory but only applies to the directory's files.
<code>dir_inherit</code>	<code>d</code>	Inherit the ACL from the parent directory but only applies to the directory's subdirectories.
<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.

In addition, you can set a default ACL inheritance policy on a 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

A ZFS file system has two properties related to ACLs.

- `aclinherit` – This property determines 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 new file or directory is equal to the permissions 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 the property value is set to `passthrough`, files are created with permissions determined by the inheritable ACEs. If no inheritable ACEs exist that affect the permissions, then the permissions are set in accordance to the requested permissions from the application.

- `passthrough-x` – This property value has the same semantics as `passthrough`, except that when `passthrough-x` is enabled, files are created with the execute (x) permission, but only if the execute permission is set in the file creation mode and in an inheritable ACE that affects the mode.

The default value for the `aclinherit` property is `restricted`.

- `aclmode` – This property modifies ACL behavior when a file is initially created or whenever a file or directory's permissions are modified by the `chmod` command. Values include the following:
 - `discard` – All ACL entries are removed except for the entries needed to define the mode of the file or directory.
 - `groupmask` – User or group ACL permissions are reduced so that they are no greater than the group permissions, unless it is a user entry that has the same UID as the owner of the file or directory. Then, the ACL permissions are reduced so that they are no greater than the owner permissions.
 - `passthrough` – During a `chmod` operation, ACEs other than `owner@`, `group@`, or `everyone@` are not modified in any way. ACEs with `owner@`, `group@`, or `everyone@` are disabled to set the file mode as requested by the `chmod` operation.

The default value for the `aclmode` property is `groupmask`.

Setting ACLs on ZFS Files

As implemented with ZFS, ACLs are composed 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.

If you change the permissions of the file, a 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 a group or to 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 follow:

- 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.
- After 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 a file is granted the `write_acl` permission unconditionally, even if the permission is explicitly denied. Otherwise, any permission left unspecified is denied.

In cases of deny permissions or when an access permission for a file 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 to be inherited by the directory's children, you have to use the ACL inheritance flags. For more information, see [Table 8–3](#) and [“Setting ACL Inheritance on ZFS Files in Verbose Format” on page 233](#).

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 May 20 14:09 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
```

Note that each user category (owner@, group@, everyone@) has two ACL entries in this example. One entry is for deny permissions, and one entry is for allow permissions.

A description of this file ACL follows:

- 0:owner@ The owner is denied execute permission on the file (execute:deny).
- 1:owner@ The owner can read and modify the contents of the file (read_data/write_data/append_data). The owner can also modify the file's attributes such as time stamps, extended attributes, and ACLs (write_xattr/write_attributes /write_acl). In addition, the owner can modify the ownership of the file (write_owner:allow).
- 2:group@ The group is denied modify and execute permissions on the file (write_data/append_data/execute:deny).
- 3:group@ The group is granted read permissions to the file (read_data:allow).
- 4:everyone@ Everyone who is not the owner of the file or a member of the owning group of the file is denied permission to execute or modify the contents of the file and to modify any attributes of the file (write_data/append_data/write_xattr/execute/write_attributes/write_acl/write_owner:deny).
- 5:everyone@ Everyone who is not the owner of the file or a member of the owning group of the file is granted read permissions on the file and the file's attributes

(read_data/read_xattr/read_attributes/read_acl/synchronize:allow). The synchronize access permission is not currently implemented.

When you create a new directory and depending on the umask value, a default directory ACL, similar to the following is applied:

```
$ ls -dv dir.1
drwxr-xr-x 2 root    root          2 May 20 14:11 dir.1
0:owner@::deny
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
  /append_data/write_xattr/execute/write_attributes/write_acl
  /write_owner:allow
2:group@:add_file/write_data/add_subdirectory/append_data:deny
3:group@:list_directory/read_data/execute:allow
4:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
  /write_attributes/write_acl/write_owner:deny
5:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
  /read_acl/synchronize:allow
```

A description of this directory ACL follows:

- 0: owner@ The owner deny list is empty for the directory (: :deny).
- 1: owner@ The owner can read and modify the directory contents (list_directory/read_data/add_file/write_data/add_subdirectory/append_data), search the contents (execute), and modify the directory's attributes such as time stamps, extended attributes, and ACLs (write_xattr/write_attributes/write_acl). In addition, the owner can modify the ownership of the directory (write_owner:allow).
- 2: group@ The group cannot add to or modify the directory contents (add_file/write_data/add_subdirectory/append_data :deny).
- 3: group@ The group can list and read the directory contents. In addition, the group has execute permission to search the directory contents (list_directory/read_data/execute:allow).
- 4: everyone@ Everyone who is not the owner of the file or a member of the owning group of the file is denied permission to add to or modify the contents of the directory (add_file/write_data/add_subdirectory/append_data). In addition, the permission to modify any attributes of the directory is denied (write_xattr/write_attributes/write_acl/write_owner:deny).
- 5: everyone@ Everyone who is not the owner of the file or a member of the owning group of the file is granted read and execute permissions on 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 220.

- 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      206663 May 20 14:09 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
```

For information about using the compact ACL format, see [“Setting and Displaying ACLs on ZFS Files in Compact Format”](#) on page 239.

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      206663 May 20 15:03 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
```

In the following example, `write_data` permissions are granted for `group@`:

```
# chmod A2=group@:append_data/execute:deny file.1
# chmod A3=group@:read_data/write_data:allow file.1
# ls -v file.1
-rw-rw-r-- 1 root      root      206663 May 20 15:03 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@:append_data/execute:deny
3:group@:read_data/write_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
```

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      206663 May 20 15:03 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
```

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 user `gozer` on the `test.dir` directory:

EXAMPLE 8-2 Setting Non-Trivial ACLs on ZFS Files (Continued)

```
# chmod A+user:gozer:read_data/execute:allow test.dir
# ls -dv test.dir
drwxr-xr-x+ 2 root      root          2 May 20 15:09 test.dir
 0:user:gozer:list_directory/read_data/execute:allow
 1:owner@::deny
 2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
  /append_data/write_xattr/execute/write_attributes/write_acl
  /write_owner:allow
 3:group@:add_file/write_data/add_subdirectory/append_data:deny
 4:group@:list_directory/read_data/execute:allow
 5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
  /write_attributes/write_acl/write_owner:deny
 6: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 May 20 15:09 test.dir
 0:owner@::deny
 1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
  /append_data/write_xattr/execute/write_attributes/write_acl
  /write_owner:allow
 2:group@:add_file/write_data/add_subdirectory/append_data:deny
 3:group@:list_directory/read_data/execute:allow
 4:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
  /write_attributes/write_acl/write_owner:deny
 5:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
  /read_acl/synchronize:allow
```

EXAMPLE 8-3 ACL Interaction With Permissions on ZFS Files

These examples illustrate the interaction between setting ACLs and then changing the file or directory's permissions.

In the following example, a trivial ACL exists on file .2:

```
# ls -v file.2
-rw-r--r-- 1 root      root          3103 May 20 15:23 file.2
 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
```

In the following example, ACL allow permissions are removed from everyone@:

EXAMPLE 8-3 ACL Interaction With Permissions on ZFS Files (Continued)

```
# chmod A5- file.2
# ls -v file.2
-rw-r-----+ 1 root    root          3103 May 20 15:23 file.2
  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
```

In this output, the file's permissions are reset from 644 to 640. Read permissions for everyone@ are effectively removed from the file's permissions 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          6986 May 20 15:25 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. Because no owner@ or group@ ACL entry exists to override the permissions for owner and group, the permissions are set to 666.

In the following example, the existing ACL is replaced with read permissions for user gozer:

```
# chmod A=user:gozer:read_data:allow file.3
# ls -v file.3
-----+ 1 root    root          6986 May 20 15:25 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          6986 May 20 15:25 file.3
  0:user:gozer::deny
  1:user:gozer:read_data:allow
  2:owner@:execute:deny
  3:owner@:read_data/write_data/append_data/write_xattr/write_attributes
    /write_acl/write_owner:allow
```

EXAMPLE 8-3 ACL Interaction With Permissions on ZFS Files (Continued)

```

4:group@:write_data/append_data:deny
5:group@:read_data/execute:allow
6:everyone@:write_data/append_data/write_xattr/write_attributes
  /write_acl/write_owner:deny
7: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, thus restoring the trivial ACLs on the 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 May 20 15:32 test5.dir
0:user:lp:read_data:file_inherit:deny
1:user:gozer:read_data:file_inherit:deny
2:owner@::deny
3:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
  /append_data/write_xattr/execute/write_attributes/write_acl
  /write_owner:allow
4:group@:add_file/write_data/add_subdirectory/append_data:deny
5:group@:list_directory/read_data/execute:allow
6:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
  /write_attributes/write_acl/write_owner:deny
7: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 six default values for `owner@`, `group@`, and `everyone@`.

```

# chmod A- test5.dir
# ls -dv test5.dir
drwxr-xr-x 2 root      root          2 May 20 15:32 test5.dir
0:owner@::deny
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
  /append_data/write_xattr/execute/write_attributes/write_acl
  /write_owner:allow
2:group@:add_file/write_data/add_subdirectory/append_data:deny
3:group@:list_directory/read_data/execute:allow
4:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
  /write_attributes/write_acl/write_owner:deny
5:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
  /read_acl/synchronize:allow

```


Setting ACL Inheritance on ZFS Files in Verbose Format

You can specify whether and how ACLs are 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.

In addition, two ACL properties can be set globally on file systems: `aclinherit` and `aclmode`. By default, `aclinherit` is set to `restricted` and `aclmode` is set to `groupmask`.

For more information, see [“ACL Inheritance” on page 223](#).

EXAMPLE 8-5 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 the `test.dir` directory:

```
# chmod A+user:gozer:read_data/write_data/execute:allow test.dir
# ls -dv test.dir
drwxr-xr-x+ 2 root    root          2 May 20 15:41 test.dir
0:user:gozer:list_directory/read_data/add_file/write_data/execute:allow
1:owner@::deny
2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
  /append_data/write_xattr/execute/write_attributes/write_acl
  /write_owner:allow
3:group@:add_file/write_data/add_subdirectory/append_data:deny
4:group@:list_directory/read_data/execute:allow
5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
  /write_attributes/write_acl/write_owner:deny
6: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 the subdirectory if the permissions on the subdirectory granted him access as the file owner, group member, or `everyone@`. For example:

```
# mkdir test.dir/sub.dir
# ls -dv test.dir/sub.dir
drwxr-xr-x 2 root    root          2 May 20 15:42 test.dir/sub.dir
0:owner@::deny
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
  /append_data/write_xattr/execute/write_attributes/write_acl
  /write_owner:allow
2:group@:add_file/write_data/add_subdirectory/append_data:deny
3:group@:list_directory/read_data/execute:allow
4:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
  /write_attributes/write_acl/write_owner:deny
5:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
  /read_acl/synchronize:allow
```

EXAMPLE 8-6 Granting ACL Inheritance on Files and Directories

The following examples identify the file and directory ACEs that are applied when the `file_inherit` flag is set.

In this 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 May 20 15:50 test2.dir
 0:user:gozer:read_data/write_data:file_inherit:allow
 1:owner@::deny
 2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
  /append_data/write_xattr/execute/write_attributes/write_acl
  /write_owner:allow
 3:group@:add_file/write_data/add_subdirectory/append_data:deny
 4:group@:list_directory/read_data/execute:allow
 5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
  /write_attributes/write_acl/write_owner:deny
 6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
  /read_acl/synchronize:allow
```

In this 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 that 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 May 20 15:51 test2.dir/file.2
 0:user:gozer:write_data:deny
 1:user:gozer:read_data/write_data:allow
 2:owner@:execute:deny
 3:owner@:read_data/write_data/append_data/write_xattr/write_attributes
  /write_acl/write_owner:allow
 4:group@:write_data/append_data/execute:deny
 5:group@:read_data:allow
 6:everyone@:write_data/append_data/write_xattr/execute/write_attributes
  /write_acl/write_owner:deny
 7:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
  :allow
```

Because the `aclmode` property for this file is set to the default value, `groupmask`, user `gozer` does not have `write_data` permission on `file.2` because the file's group permission does not allow it.

When using the `inherit_only` permission, which is applied when the `file_inherit` or `dir_inherit` flag is 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 a member of the file's group owner. For example:

EXAMPLE 8-6 Granting ACL Inheritance on Files and Directories (Continued)

```
# mkdir test2.dir/subdir.2
# ls -dv test2.dir/subdir.2
drwxr-xr-x+ 2 root    root          2 May 20 15:52 test2.dir/subdir.2
0:user:gozer:list_directory/read_data/add_file/write_data:file_inherit
  /inherit_only:allow
1:owner@::deny
2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
  /append_data/write_xattr/execute/write_attributes/write_acl
  /write_owner:allow
3:group@:add_file/write_data/add_subdirectory/append_data:deny
4:group@:list_directory/read_data/execute:allow
5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
  /write_attributes/write_acl/write_owner:deny
6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
  /read_acl/synchronize:allow
```

The following examples identify the file and directory ACLs that are applied when both the `file_inherit` and `dir_inherit` flags are set.

In this 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
# ls -dv test3.dir
drwxr-xr-x+ 2 root    root          2 May 20 15:53 test3.dir
0:user:gozer:list_directory/read_data/add_file/write_data/execute
  :file_inherit/dir_inherit:allow
1:owner@::deny
2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
  /append_data/write_xattr/execute/write_attributes/write_acl
  /write_owner:allow
3:group@:add_file/write_data/add_subdirectory/append_data:deny
4:group@:list_directory/read_data/execute:allow
5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
  /write_attributes/write_acl/write_owner:deny
6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
  /read_acl/synchronize:allow

# touch test3.dir/file.3
# ls -v test3.dir/file.3
-rw-r--r--+ 1 root    root          0 May 20 15:58 test3.dir/file.3
0:user:gozer:write_data/execute:deny
1:user:gozer:read_data/write_data/execute:allow
2:owner@:execute:deny
3:owner@:read_data/write_data/append_data/write_xattr/write_attributes
  /write_acl/write_owner:allow
4:group@:write_data/append_data/execute:deny
5:group@:read_data:allow
6:everyone@:write_data/append_data/write_xattr/execute/write_attributes
  /write_acl/write_owner:deny
7:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
  :allow
```

EXAMPLE 8-6 Granting ACL Inheritance on Files and Directories (Continued)

```

# mkdir test3.dir/subdir.1
# ls -dv test3.dir/subdir.1
drwxr-xr-x+ 2 root    root          2 May 20 15:59 test3.dir/subdir.1
 0:user:gozer:list_directory/read_data/add_file/write_data/execute
   :file_inherit/dir_inherit/inherit_only:allow
 1:user:gozer:add_file/write_data:deny
 2:user:gozer:list_directory/read_data/add_file/write_data/execute:allow
 3:owner@::deny
 4:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
   /append_data/write_xattr/execute/write_attributes/write_acl
   /write_owner:allow
 5:group@:add_file/write_data/add_subdirectory/append_data:deny
 6:group@:list_directory/read_data/execute:allow
 7:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
   /write_attributes/write_acl/write_owner:deny
 8:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
   /read_acl/synchronize:allow

```

In these examples, because the permissions 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 this example, user `gozer` is granted read, write, and execute permissions that are inherited for newly created files. However, they 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 May 20 16:02 test4.dir
 0:user:gozer:list_directory/read_data/add_file/write_data/execute
   :file_inherit/no_propagate:allow
 1:owner@::deny
 2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
   /append_data/write_xattr/execute/write_attributes/write_acl
   /write_owner:allow
 3:group@:add_file/write_data/add_subdirectory/append_data:deny
 4:group@:list_directory/read_data/execute:allow
 5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
   /write_attributes/write_acl/write_owner:deny
 6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
   /read_acl/synchronize:allow

```

As the following example shows, when a new subdirectory is created, user `gozer`'s `read_data/write_data/execute` permissions 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 May 20 16:03 test4.dir/sub4.dir
 0:owner@::deny
 1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory

```

EXAMPLE 8-6 Granting ACL Inheritance on Files and Directories (Continued)

```

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

```

As the following example shows, user gozer's read_data/write_data/execute permissions for files are propagated to the newly created file:

```

# touch test4.dir/file.4
# ls -v test4.dir/file.4
-rw-r--r--+ 1 root    root          0 May 20 16:04 test4.dir/file.4
  0:user:gozer:write_data/execute:deny
  1:user:gozer:read_data/write_data/execute:allow
  2:owner@:execute:deny
  3:owner@:read_data/write_data/append_data/write_xattr/write_attributes
    /write_acl/write_owner:allow
  4:group@:write_data/append_data/execute:deny
  5:group@:read_data:allow
  6:everyone@:write_data/append_data/write_xattr/execute/write_attributes
    /write_acl/write_owner:deny
  7:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
    :allow

```

EXAMPLE 8-7 ACL Inheritance With aclmode Property Set to passthrough

As the following example shows, when the aclmode property on the tank/cindys file system is set to passthrough, user gozer inherits the ACL applied on the test4.dir directory for the newly created file.4:

```

# zfs set aclmode=passthrough tank/cindys
# touch test4.dir/file.4
# ls -v test4.dir/file.4
-rw-r--r--+ 1 root    root          0 May 20 16:08 test4.dir/file.4
  0:user:gozer:write_data/execute:deny
  1:user:gozer:read_data/write_data/execute:allow
  2:owner@:execute:deny
  3:owner@:read_data/write_data/append_data/write_xattr/write_attributes
    /write_acl/write_owner:allow
  4:group@:write_data/append_data/execute:deny
  5:group@:read_data:allow
  6:everyone@:write_data/append_data/write_xattr/execute/write_attributes
    /write_acl/write_owner:deny
  7:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
    :allow

```

This output shows that the read_data/write_data/execute:allow:file_inherit/dir_inherit ACL that was set on the parent directory, test4.dir, is passed through to user gozer.

EXAMPLE 8-8 ACL Inheritance With `aclmode` Property Set to `discard`

If the `aclmode` property on a file system is set to `discard`, then ACLs can potentially be discarded when the permissions on a directory change. For example:

```
# zfs set aclmode=discard tank/cindys
# 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 May 20 16:09 test5.dir
 0:user:gozer:list_directory/read_data/add_file/write_data/execute
   :dir_inherit:allow
 1:owner@::deny
 2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
   /append_data/write_xattr/execute/write_attributes/write_acl
   /write_owner:allow
 3:group@:add_file/write_data/add_subdirectory/append_data:deny
 4:group@:list_directory/read_data/execute:allow
 5:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
   /write_attributes/write_acl/write_owner:deny
 6:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
   /read_acl/synchronize:allow
```

If, at a later time, you decide to tighten the permissions 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 May 20 16:09 test5.dir
 0:owner@::deny
 1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
   /append_data/write_xattr/execute/write_attributes/write_acl
   /write_owner:allow
 2:group@:add_file/write_data/add_subdirectory/append_data/execute:deny
 3:group@:list_directory/read_data:allow
 4:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
   /execute/write_attributes/write_acl/write_owner:deny
 5:everyone@:list_directory/read_data/read_xattr/read_attributes/read_acl
   /synchronize:allow
```

EXAMPLE 8-9 ACL Inheritance With the `aclinherit` Property 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 shows how you can specify two ACEs in the same `chmod` command.

```
# zfs set aclinherit=noallow tank/cindys
# 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 May 20 16:11 test6.dir
 0:user:gozer:read_data:file_inherit:deny
 1:user:lp:read_data:file_inherit:allow
 2:owner@::deny
 3:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
```

EXAMPLE 8-9 ACL Inheritance With the `aclinherit` Property Set to `noallow` (Continued)

```

    /append_data/write_xattr/execute/write_attributes/write_acl
    /write_owner:allow
4:group@:add_file/write_data/add_subdirectory/append_data:deny
5:group@:list_directory/read_data/execute:allow
6:everyone@:add_file/write_data/add_subdirectory/append_data/write_xattr
  /write_attributes/write_acl/write_owner:deny
7:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
  /read_acl/synchronize:allow

```

As the following example shows, when a new file is created, the ACL that allows `read_data` permission is discarded.

```

# touch test6.dir/file.6
# ls -v test6.dir/file.6
-rw-r--r--  1 root   root           0 May 20 16:13 test6.dir/file.6
  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

```

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 8-3](#).

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           206663 Jun 17 10:07 file.1
  owner@:--x-----:-----:deny
  owner@:rw-p---A-W-Co-:-----:allow
  group@:-wpx-----:-----:deny
  group@:r-----:-----:allow
  everyone@:-wpx---A-W-Co-:-----:deny
  everyone@:r-----a-R-c--s:-----:allow

```

This compact ACL output is described as follows:

`owner@` The owner is denied execute permissions on the file (`x=execute`).

`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 time stamps, extended attributes, and ACLs (A=write_xattr, W=write_attributes, and C=write_acl). In addition, the owner can modify the ownership of the file (o=write_owner).
group@	The group is denied modify and execute permissions on the file (write_data, p=append_data, and x=execute).
group@	The group is granted read permissions on the file (r=read_data).
everyone@	Everyone who is not the owner of the file or a member of the owning group of the file is denied permission to execute or modify the contents of the file, and to modify any attributes of the file (w=write_data, x=execute, p=append_data, A=write_xattr, W=write_attributes, C=write_acl, and o=write_owner).
everyone@	Everyone who is not the owner of the file or a member of the owning group of the file on the file and the file's attributes (r=read_data, a=append_data, R=read_xattr, c=read_acl, and s=synchronize). The synchronize access permission is not currently implemented.

The compact ACL format provides the following advantages over the verbose ACL format:

- Permissions can be specified as positional arguments to the `chmod` command.
- The hyphen (-) characters, which identify no permissions, can be removed. 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 228](#).

EXAMPLE 8-10 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      206663 Jun 17 10:07 file.1
  owner@:--x-----:-----:deny
  owner@:rw-p---A-W-Co-:-----:allow
  group@:-wxp-----:-----:deny
  group@:r-----:-----:allow
  everyone@:-wxp---A-W-Co-:-----:deny
  everyone@:r-----a-R-c--s:-----:allow
```

In the following example, `read_data/execute` permissions are added for user `gozer` on `file.1`:

```
# chmod A+user:gozer:rx:allow file.1
# ls -V file.1
-rw-r--r--+ 1 root    root      206663 Jun 17 10:07 file.1
  user:gozer:r-x-----:-----:allow
  owner@:--x-----:-----:deny
```


EXAMPLE 8-10 Setting and Displaying ACLs in Compact Format (Continued)

```

owner@:rw-p---A-W-Co:-----:allow
group@:-wxp-----:-----:deny
group@:r-----:-----:allow
everyone@:-wxp---A-W-Co:-----:deny
everyone@:r-----a-R-c--s:-----:allow

```

Another way to add the same permissions for user gozer is to insert a new ACL entry at a specific position, 4, for example. As such, the existing ACLs at positions 4–6 are pushed down. For example:

```

# chmod A4+user:gozer:rx:allow file.1
# ls -V file.1
-rw-r--r--+ 1 root    root        206663 Jun 17 10:16 file.1
owner@:--x-----:-----:deny
owner@:rw-p---A-W-Co:-----:allow
group@:-wxp-----:-----:deny
group@:r-----:-----:allow
user:gozer:r-x-----:-----:allow
everyone@:-wxp---A-W-Co:-----:deny
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.

```

# chmod A+user:gozer:rw:fd:allow dir.2
# ls -dV dir.2
drwxr-xr-x+ 2 root    root            2 Jun 17 10:19 dir.2
user:gozer:rw-----:fd----:allow
owner@:-----:-----:deny
owner@:rwxp---A-W-Co:-----:allow
group@:-w-p-----:-----:deny
group@:r-x-----:-----:allow
everyone@:-w-p---A-W-Co:-----:deny
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 the `dir.2` directory for user gozer to user cindys on `dir.2`, copy and paste the permissions and inheritance flags (`rw-----:fd----:allow`) into your `chmod` command as follows:

```

# chmod A+user:cindys:rw-----:fd----:allow dir.2
# ls -dV dir.2
drwxr-xr-x+ 2 root    root            2 Jun 17 10:19 dir.2
user:cindys:rw-----:fd----:allow
user:gozer:rw-----:fd----:allow
owner@:-----:-----:deny
owner@:rwxp---A-W-Co:-----:allow
group@:-w-p-----:-----:deny
group@:r-x-----:-----:allow
everyone@:-w-p---A-W-Co:-----:deny
everyone@:r-x---a-R-c--s:-----:allow

```

EXAMPLE 8-11 ACL Inheritance With the `aclinherit` Property Set to `passthrough`

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 permissions that are determined by the inheritable ACEs. If no inheritable ACEs exist that affect the permissions, then the permissions are set in accordance to the requested permissions from the application.

The following examples use compact ACL syntax to show how to inherit permissions by setting the `aclinherit` property to `passthrough`.

In this example, an ACL is set on the `test1.dir` directory 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. Additionally, directories inherit six other ACEs that propagate the ACEs to the newly created directories and files.

```
# zfs set aclinherit=passthrough tank/cindys
# pwd
/tank/cindys
# 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 Jun 17 10:37 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 by the newly created files.

```
# cd test1.dir
# touch file.1
# ls -V file.1
-rwxrwx---+ 1 root    root          0 Jun 17 10:38 file.1
      owner@:rwxpdDaARWcCos:-----:allow
      group@:rwxp-----:-----:allow
      everyone@:-----:-----: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 Jun 17 10:39 subdir.1
      owner@:rwxpdDaARWcCos:fdi---:allow
      owner@:rwxpdDaARWcCos:-----:allow
      group@:rwxp-----:fdi---:allow
      group@:rwxp-----:-----:allow
      everyone@:-----:fdi---:allow
      everyone@:-----:-----:allow
```

EXAMPLE 8-11 ACL Inheritance With the `aclinherit` Property Set to `passthrough` (Continued)

The `-di-` and `f-i-` entries are for propagating inheritance and are not considered during access control. In this example, a file is created with a trivial ACL in another directory where inherited ACEs are not present.

```
# cd /tank/cindys
# mkdir test2.dir
# cd test2.dir
# touch file.2
# ls -V file.2
-rw-r--r-- 1 root    root          0 Jun 17 10:40 file.2
  owner@:--x-----:-----:deny
  owner@:rw-p--A-W-Co-:-----:allow
  group@:-w-xp-----:-----:deny
  group@:r-----:-----:allow
  everyone@:-w-xp--A-W-Co-:-----:deny
  everyone@:r-----a-R-c--s:-----:allow
```

EXAMPLE 8-12 ACL Inheritance With the `aclinherit` Property Set to `passthrough-x`

When `aclinherit` property is set to `passthrough-x`, files are created with the execute (`x`) permission for `owner@`, `group@`, or `everyone@`, but only if execute permission is set in file creation mode and in an inheritable ACE that affects the mode.

The following example shows how to inherit the execute permission by setting the `aclinherit` property to `passthrough-x`.

```
# zfs set aclinherit=passthrough-x tank/cindys
```

The following ACL is set on `/tank/cindys/test1.dir` to provide executable ACL inheritance for files for `owner@`, `group@`, and `everyone@`.

```
# chmod A=owner@:rwxpcCosRrWaAdD:fd:allow,group@:rwxp:fd:allow,everyone@::fd:allow test1.dir
# ls -Vd test1.dir
drwxrwx---+ 2 root    root          2 Jun 17 10:41 test1.dir
  owner@:rwxpdDaARWcCos:fd----:allow
  group@:rwxp-----:fd----:allow
  everyone@:-----:fd----:allow
```

A file (`file1`) is created with requested permissions `0666`. The resulting permissions are `0660`. The execution permission was not inherited because the creation mode did not request it.

```
# touch test1.dir/file1
# ls -V test1.dir/file1
-rw-rw----+ 1 root    root          0 Jun 17 10:42 test1.dir/file1
  owner@:rw-pdDaARWcCos:-----:allow
  group@:rw-p-----:-----:allow
  everyone@:-----:-----:allow
```

Next, an executable called `t` is generated by using the `cc` compiler in the `testdir` directory.

EXAMPLE 8-12 ACL Inheritance With the `aclinherit` Property Set to `passthrough-x` (Continued)

```
# cc -o t t.c
# ls -V t
-rwxrwx---+ 1 root      root          7396 Jun 17 10:50 t
                owner@:rwxpdDaARWcCos:-----:allow
                group@:rwxp-----:-----:allow
                everyone@:-----:-----: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.

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 245
- “Delegating ZFS Permissions” on page 246
- “Displaying ZFS Delegated Permissions (Examples)” on page 253
- “Delegating ZFS Permissions (Examples)” on page 250
- “Removing ZFS Delegated Permissions (Examples)” on page 255

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 *System Administration Guide: 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 datasets 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 dataset only or to all descendents of the current dataset.

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.
mount	The permission to mount and unmount a dataset, 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.
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 dataset.	
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
- 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`
- `devices`
- `exec`
- `mountpoint`
- `nbmand`
- `normalization`
- `primarycache`
- `quota`
- `readonly`
- `recordsize`
- `refreservation`
- `reservation`
- `secondarycache`
- `setuid`
- `shareiscsi`
- `sharenfs`
- `sharesmb`
- `snapdir`
- `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 171](#).

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 dataset and not its descendents, unless the `-d` option is also specified. The `-d` option indicates that the permissions are allowed for the descendent datasets and not for this dataset, 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 cindys create,destroy,mount,snapshot tank/cindys
# zfs allow tank/cindys
-----
Local+Descendent permissions on (tank/cindys)
      user cindys create,destroy,mount,snapshot
-----
```

To remove these permissions, you would use the following syntax:

```
# zfs unallow cindys tank/cindys
# zfs allow tank/cindys
```

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 `marks` `create` and `mount` permissions on the `tank` file system, set the permissions first:

```
# chmod A+user:marks:add_subdirectory:fd:allow /tank
```

Then, use the `zfs allow` command to delegate `create`, `destroy`, and `mount` permissions. For example:

```
# zfs allow marks create,destroy,mount tank
```

Now, user `marks` can create his own file systems in the `tank` file system. For example:

```
# su marks
marks$ zfs create tank/marks
marks$ ^D
# su lp
$ zfs create tank/lp
cannot create 'tank/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` 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
# zfs allow -c create,destroy tank
# zfs allow tank
-----
Create time permissions on (tank)
  create,destroy
Local+Descendent permissions on (tank)
  group staff create,mount
-----
# su cindys
cindys% zfs create tank/cindys
cindys% exit
# su marks
marks% zfs create tank/marks/data
marks% exit
cindys% zfs destroy tank/marks/data
cannot destroy 'tank/mark': 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 marks is delegated create, destroy, and mount permissions for the local and descendent file systems. User marks is delegated local permission to snapshot the tank 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 marks snapshot tank
# zfs allow tank
-----
Local permissions on (tank)
    user marks snapshot
Local+Descendent permissions on (tank)
    user marks create,destroy,mount
-----
# su marks
marks$ zfs snapshot tank/@snap1
marks$ zfs snapshot tank/marks@snap1
cannot create snapshot 'mark/marks@snap1': permission denied
```

To delegate user marks permission at the descendent file system level, use the `zfs allow -d` option. For example:

```
# zfs unallow -l marks snapshot tank
# zfs allow -d marks snapshot tank
# zfs allow tank
-----
Descendent permissions on (tank)
    user marks snapshot
Local+Descendent permissions on (tank)
    user marks create,destroy,mount
-----
# su marks
$ zfs snapshot tank@snap2
cannot create snapshot 'tank@snap2': permission denied
$ zfs snapshot tank/marks@snappy
```

Now, user marks can only create a snapshot below the tank 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 file systems are created.

```
# zfs allow staff create,mount tank
# zfs allow -c destroy,snapshot tank
# zfs allow tank
-----
Create time permissions on (tank)
    destroy,snapshot
Local+Descendent permissions on (tank)
    group staff create,mount
-----
```

EXAMPLE 9-4 Defining and Using Complex Delegated Permissions (Continued)

Because user marks is a member of the `staff` group, he can create file systems in `tank`. In addition, user marks can create a snapshot of `tank/marks2` because he has specific permissions to do so. For example:

```
# su marks
$ zfs create tank/marks2
$ zfs allow tank/marks2
-----
Local permissions on (tank/marks2)
    user marks destroy,snapshot
-----
Create time permissions on (tank)
    destroy,snapshot
Local+Descendent permissions on (tank)
    group staff create
    everyone mount
-----
```

But, user marks cannot create a snapshot in `tank/marks` because he doesn't have specific permissions to do so. For example:

```
$ zfs snapshot tank/marks2@snap1
$ zfs snapshot tank/marks@snappp
cannot create snapshot 'tank/marks@snappp': permission denied
```

In this example, user marks 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/marks2
$ ls
$ cd .zfs
$ ls
snapshot
$ cd snapshot
$ ls -l
total 3
drwxr-xr-x  2 marks  staff          2 Dec 15 13:53 snap1
$ pwd
/tank/marks2/.zfs/snapshot
$ mkdir snap2
$ zfs list
NAME                                USED  AVAIL  REFER  MOUNTPOINT
tank                                264K  33.2G  33.5K  /tank
tank/marks                          24.5K  33.2G  24.5K  /tank/marks
tank/marks2                          46K   33.2G  24.5K  /tank/marks2
tank/marks2@snap1                    21.5K  -      24.5K  -
tank/marks2@snap2                     0     -      24.5K  -
$ 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 `cindys`, 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
-----
Permission sets on (tank)
    @myset clone,create,destroy,mount,promote,readonly,snapshot
-----
# zfs allow staff @myset,rename tank
# zfs allow tank
-----
Permission sets on (tank)
    @myset clone,create,destroy,mount,promote,readonly,snapshot
Local+Descendent permissions on (tank)
    group staff @myset,rename
# chmod A+group:staff:add_subdirectory:fd:allow tank
# su cindys
cindys% zfs create tank/data
Cindys% zfs allow tank
-----
Permission sets on (tank)
    @myset clone,create,destroy,mount,promote,readonly,snapshot
Local+Descendent permissions on (tank)
    group staff @myset,rename
-----
cindys% ls -l /tank
total 15
drwxr-xr-x  2 cindys  staff          2 Aug  8 14:10 data
cindys% 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 `cindys` has `create`, `destroy`, `mount`, `snapshot` permissions on the `tank/cindys` file system.

```
# zfs allow tank/cindys
-----
Local+Descendent permissions on (tank/cindys)
  user cindys 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 dataset 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.
- 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
-----
Permission sets on (pool/fred)
  @eng create,destroy,snapshot,mount,clone,promote,rename
  @simple create,mount
Create time permissions on (pool/fred)
  @eng,mountpoint
Local permissions on (pool/fred)
  user tom @eng
  user joe create,destroy,mount
Local+Descendent permissions on (pool/fred)
  user fred @basic,share,rename
```

EXAMPLE 9-7 Displaying Complex Delegated Administration Permissions (Continued)

```

Descendent permissions on (pool/fred)
  user barney @basic
  group staff @basic
-----
Permission sets on (pool)
  @simple create,destroy,mount
Local permissions on (pool)
  group staff @simple
-----

```

Removing ZFS Delegated Permissions (Examples)

You can use the `zfs unallow` command to remove delegated permissions. For example, user `cindys` has `create`, `destroy`, `mount`, and `snapshot` permissions on the `tank/cindys` file system.

```

# zfs allow cindys create,destroy,mount,snapshot tank/cindys
# zfs allow tank/cindys
-----
Local+Descendent permissions on (tank/cindys)
  user cindys create,destroy,mount,snapshot
-----

```

The following `zfs unallow` syntax removes user `cindys`'s `snapshot` permission from the `tank/cindys` file system:

```

# zfs unallow cindys snapshot tank/cindys
# zfs allow tank/cindys
-----
Local+Descendent permissions on (tank/cindys)
  user cindys create,destroy,mount
-----
cindys% zfs create tank/cindys/data
cindys% zfs snapshot tank/cindys@today
cannot create snapshot 'tank/cindys@today': permission denied

```

As another example, user `marks` has the following permissions on the `tank/marks` file system:

```

# zfs allow tank/marks
-----
Local+Descendent permissions on (tank/marks)
  user marks create,destroy,mount
-----

```

The following `zfs unallow` syntax removes all permissions for user `marks` from the `tank/marks` file system:

```
# zfs unallow marks tank/marks
```

The following `zfs unallow` syntax removes a permission set on the `tank` file system.

```
# zfs allow tank
-----
Permission sets on (tank)
  @myset clone,create,destroy,mount,promote,readonly,snapshot
Create time permissions on (tank)
  create,destroy,mount
Local+Descendent permissions on (tank)
  group staff create,mount
-----
# zfs unallow -s @myset tank
$ zfs allow tank
-----
Create time permissions on (tank)
  create,destroy,mount
Local+Descendent permissions on (tank)
  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 257
- “Using ZFS on a Solaris System With Zones Installed” on page 259
- “Using ZFS Alternate Root Pools” on page 265
- “ZFS Rights Profiles” on page 266

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 262.

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/t2000
Savecore enabled: yes
```

If you need to change your swap area or dump device after the system is installed or upgraded, 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 Device and Dump Device”](#) on page 149.

Using a ZFS Volume as a Solaris iSCSI Target

You can easily create a ZFS volume as an iSCSI target by setting the `shareiscsi` property on the volume. For example:

```
# zfs create -V 2g tank/volumes/v2
# zfs set shareiscsi=on tank/volumes/v2
# iscsitadm list target
Target: tank/volumes/v2
  iSCSI Name: iqn.1986-03.com.sun:02:984fe301-c412-ccc1-cc80-cf9a72aa062a
  Connections: 0
```

After the iSCSI target is created, set up the iSCSI initiator. For more information about Solaris iSCSI targets and initiators, see [Chapter 14, “Configuring Oracle Solaris iSCSI Targets and Initiators \(Tasks\)”](#) in *System Administration Guide: Devices and File Systems*.

Note – Solaris iSCSI targets can also be created and managed with the `iscsitadm` command. If you set the `shareiscsi` property on a ZFS volume, do not use the `iscsitadm` command to also create the same target device. Otherwise, you create duplicate target information for the same device.

A ZFS volume as an iSCSI target is managed just like any other ZFS dataset. However, the rename, export, and import operations work a little differently for iSCSI targets.

- When you rename a ZFS volume, the iSCSI target name remains the same. For example:

```
# zfs rename tank/volumes/v2 tank/volumes/v1
# iscsitadm list target
Target: tank/volumes/v1
   iSCSI Name: iqn.1986-03.com.sun:02:984fe301-c412-ccc1-cc80-cf9a72aa062a
   Connections: 0
```

- Exporting a pool that contains a shared ZFS volume causes the target to be removed. Importing a pool that contains a shared ZFS volume causes the target to be shared. For example:

```
# zpool export tank
# iscsitadm list target
# zpool import tank
# iscsitadm list target
Target: tank/volumes/v1
   iSCSI Name: iqn.1986-03.com.sun:02:984fe301-c412-ccc1-cc80-cf9a72aa062a
   Connections: 0
```

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 260](#)
- [“Delegating Datasets to a Non-Global Zone” on page 261](#)
- [“Adding ZFS Volumes to a Non-Global Zone” on page 262](#)
- [“Using ZFS Storage Pools Within a Zone” on page 262](#)
- [“Managing ZFS Properties Within a Zone” on page 263](#)
- [“Understanding the zoned Property” on page 264](#)

For information about configuring zones on a system with a ZFS root file system that will be migrated or patched with Oracle Solaris Live Upgrade, see [“Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones \(Solaris 10 10/08\)” on page 134](#) or [“Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones \(at Least Solaris 10 5/09\)” on page 139](#).

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`.
- Due to CR 6449301, do not add a ZFS dataset to a non-global zone when the non-global zone is configured. Instead, add a ZFS dataset after the zone is installed.
- 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, “Zones,” in *System Administration Guide: Oracle Solaris Containers-Resource Management and Oracle Solaris Zones*](#).
- If you delegate a ZFS file system to a non-global zone, you must remove that file system from the non-global zone before using Oracle Solaris Live Upgrade. Otherwise, Oracle Live Upgrade will fail due to a read-only file system error.

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`.

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, “Zones,” in *System Administration Guide: Oracle Solaris Containers-Resource Management and Oracle Solaris Zones*](#).

Delegating Datasets to a Non-Global Zone

To meet the primary goal of delegating the administration of storage to a zone, ZFS supports adding datasets to a 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> 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. 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 you are using Oracle Solaris Live Upgrade to upgrade your ZFS BE with non-global zones, first remove any delegated datasets. Otherwise, Oracle Solaris Live Upgrade will fail with a read-only file system error. For example:

```
zonecfg:zion>
zonecfg:zion> remove dataset name=tank/zone/zion
zonecfg:zion1> exit
```

For more information about what actions are allowed within zones, see [“Managing ZFS Properties Within a Zone” on page 263](#).

Adding ZFS Volumes to a Non-Global Zone

ZFS volumes cannot be added to a non-global zone by using the `zonecfg` command's `add dataset` subcommand. However, volumes can be added to a zone by using the `zonecfg` command's `add device` subcommand.

In the following example, a ZFS volume is added to a non-global zone by a global zone administrator from the global zone:

```
# zonecfg -z zion
zion: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:zion> create
zonecfg:zion> add device
zonecfg:zion:device> set match=/dev/zvol/dsk/tank/vol
zonecfg:zion:device> end
```

This syntax adds the `tank/vol` volume to the `zion` zone. Note that adding a raw volume to a zone has implicit security risks, even if the volume doesn't correspond to a physical device. In particular, the zone administrator could create malformed file systems that would panic the system when a mount is attempted. For more information about adding devices to zones and the related security risks, see [“Understanding the zoned Property” on page 264](#).

For more information about adding devices to zones, see [Part II, “Zones,” in *System Administration Guide: Oracle Solaris Containers-Resource Management and Oracle Solaris Zones*](#).

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, each dataset would have the following properties.

Dataset	Visible	Writable	Immutable Properties
tank	Yes	No	-
tank/home	No	-	-
tank/data	Yes	No	-
tank/data/matrix	No	-	-
tank/data/zion	Yes	Yes	sharenfs, zoned, quota, reservation
tank/data/zion/home	Yes	Yes	sharenfs, zoned

Note that every parent of `tank/zone/zion` is visible as read-only, all descendents are writable, and datasets that are not part of the parent hierarchy are not visible at all. The zone administrator cannot change the `sharenfs` property because non-global zones cannot act as NFS servers. The zone administrator cannot change the `zoned` property because doing so would expose a security risk as 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.

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
```

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 [System Administration Guide: 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.”](#)

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 267
- “Checking ZFS File System Integrity” on page 269
- “Resolving Problems With ZFS” on page 271
- “Repairing a Damaged ZFS Configuration” on page 276
- “Resolving a Missing Device” on page 276
- “Replacing or Repairing a Damaged Device” on page 278
- “Repairing Damaged Data” on page 287
- “Repairing an Unbootable System” on page 291

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 268
- “Damaged Devices in a ZFS Storage Pool” on page 268
- “Corrupted ZFS Data” on page 268

Note that a single pool can experience all three errors, so a complete repair procedure involves finding and correcting one error, proceeding to the next error, and so on.

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 an administrator accidentally overwrites 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 289](#).

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 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
scrub: scrub completed after 0h7m with 0 errors on Tue Tue Feb  2 12:54:00 2010
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 95](#).

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 285.

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 272
- [“Reviewing zpool status Output”](#) on page 272
- [“System Reporting of ZFS Error Messages”](#) on page 275

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/erick
2010-07-15.12:07:01 zfs set checksum=off tank/erick
```

In this output, note that checksums are disabled for the `tank/erick` file system. This configuration is not recommended.

- Identify the errors through the `fmd` 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 95](#).

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 278](#).

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 285](#).

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.

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 269](#).

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
clt0d0	ONLINE	0	0	0	
clt1d0	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 287](#).

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 events to notify the administrator:

- **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 `cmdump` 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 103](#).

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 `cmd` 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: neo2
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
```

```
scrub: scrub completed after 0h0m with 0 errors on Tue Feb  2 13:15:20 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
```

You can see from this output that the missing `c1t1d0` device is not functioning. If you determine that the device is faulty, replace it.

Then, use the `zpool online` command to bring online the replaced device. For example:

```
# zpool online tank c1t1d0
```

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 278.

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 83.

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 tpool
pool: tpool
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 2 errors on Tue Jul 13 11:08:37 2010
config:
```

NAME	STATE	READ	WRITE	CKSUM
tpool	ONLINE	2	0	0
c1t1d0	ONLINE	2	0	0
c1t3d0	ONLINE	0	0	0

```
errors: Permanent errors have been detected in the following files:
```

```
  /tpool/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 287](#).

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 84](#).

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 280](#)
- [“Devices That Cannot be Replaced” on page 281](#)
- [“Replacing a Device in a ZFS Storage Pool” on page 281](#)
- [“Viewing Resilvering Status” on page 285](#)

Determining If a Device Can Be Replaced

For a device to be replaced, the pool must be in the `ONLINE` state. The device must be part of a redundant configuration, or it must be healthy (in the `ONLINE` state). If the device is part of a redundant configuration, sufficient replicas from which to retrieve good data must exist. 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 289](#).

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 85](#).

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 Oracle's Sun Fire x4500 system. 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:

- 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
```

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
```

EXAMPLE 11-1 Replacing a Device in a ZFS Storage Pool (Continued)

```
# zpool status tank
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

EXAMPLE 11-1 Replacing a Device in a ZFS Storage Pool (Continued)

errors: No known data errors

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

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: <http://www.sun.com/msg/ZFS-8000-K4>
- Physically replace the failed log device.
- Bring the new log device online.
- Clear the pool's error condition.

```
# 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.
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	ONLINE	0	0	0
c0t1d0	ONLINE	0	0	0
c0t4d0	ONLINE	0	0	0

EXAMPLE 11-2 Replacing a Failed Log Device (Continued)

```

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

# 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.
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

```

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.

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: DEGRADED
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.
       scrub: resilver in progress for 0h0m, 22.60% done, 0h1m to go
config:
      NAME                STATE          READ WRITE CKSUM
      tank                DEGRADED      0     0     0
      mirror-0           DEGRADED      0     0     0
      replacing-0        DEGRADED      0     0     0
      c1t0d0              UNAVAIL       0     0     0  cannot open
      c2t0d0              ONLINE        0     0     0  85.0M resilvered
      c1t1d0              ONLINE        0     0     0
```

errors: No known data errors

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 287
- “Repairing a Corrupted File or Directory” on page 288
- “Repairing ZFS Storage Pool-Wide Damage” on page 289

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 269.

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
      ct1t10d0    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:

- 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: FAILED
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	FAILED	0	0	1	corrupted data
c1t1d0	ONLINE	0	0	2	
c1t3d0	ONLINE	0	0	4	

The recovery process as described above 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 above 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'll see the damaged pool messages when you attempt to import the pool.

- If the pool cannot be recovered by the pool recovery method described above, 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 recreate 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.

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 293](#)
- [“ZFS Pool Versions” on page 293](#)
- [“ZFS File System Versions” on page 294](#)

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 \(zfs upgrade\)” on page 30](#) and [“Upgrading ZFS Storage Pools” on page 109](#).

ZFS Pool Versions

The following table provides a list of ZFS pool versions that are available in the Solaris releases.

Version	Solaris 10	Description
1	Solaris 10 6/06	Initial ZFS version
2	Solaris 10 11/06	Ditto blocks (replicated metadata)

Version	Solaris 10	Description
3	Solaris 10 11/06	Hot spares and double parity RAID-Z
4	Solaris 10 8/07	zpool history
5	Solaris 10 10/08	gzip compression algorithm
6	Solaris 10 10/08	bootfs pool property
7	Solaris 10 10/08	Separate intent log devices
8	Solaris 10 10/08	Delegated administration
9	Solaris 10 10/08	refquota and refreservation properties
10	Solaris 10 5/09	Cache devices
11	Solaris 10 10/09	Improved scrub performance
12	Solaris 10 10/09	Snapshot properties
13	Solaris 10 10/09	snapped property
14	Solaris 10 10/09	aclinherit passthrough-x property
15	Solaris 10 10/09	user and group space accounting
16	Solaris 10 9/10	stmf property support
17	Solaris 10 9/10	Triple-parity RAID-Z
18	Solaris 10 9/10	Snapshot user holds
19	Solaris 10 9/10	Log device removal
20	Solaris 10 9/10	Compression using zle (zero-length encoding)
21	Solaris 10 9/10	Reserved
22	Solaris 10 9/10	Received properties

ZFS File System Versions

The following table lists the ZFS file system versions that are available in the Solaris releases.

Version	Solaris 10	Description
1	Solaris 10 6/06	Initial ZFS file system version
2	Solaris 10 10/08	Enhanced directory entries
3	Solaris 10 10/08	Case insensitivity and file system unique identifier (FUID)

Version	Solaris 10	Description
4	Solaris 10 10/09	userquota and groupquota properties

Index

A

accessing

- ZFS snapshot
(example of), 207

ACL model, Solaris, differences between ZFS and traditional file systems, 55

ACL property mode

- `aclinherit`, 171
- `aclmode`, 172

`aclinherit` property, 224

`aclmode` property, 225

ACLs

- access privileges, 222
- ACL inheritance, 223
- ACL inheritance flags, 223
- ACL on ZFS directory
 - detailed description, 227
- ACL on ZFS file
 - detailed description, 226
- ACL property, 224
- `aclinherit` property, 224
- `aclmode` property, 225
- description, 219
- differences from POSIX-draft ACLs, 220
- entry types, 222
- format description, 220
- modifying trivial ACL on ZFS file (verbose format)
(example of), 229
- restoring trivial ACL on ZFS file (verbose format)
(example of), 232
- setting ACL inheritance on ZFS file (verbose format)
(example of), 233

ACLs (*Continued*)

- setting ACLs on ZFS file (compact format)
(example of), 240
- description, 239
- setting ACLs on ZFS file (verbose format)
description, 228
- setting on ZFS files
description, 225

adding

- cache devices (example of), 76
- devices to a ZFS storage pool (`zpool add`)
(example of), 73
- disks to a RAID-Z configuration (example of), 74
- mirrored log device (example of), 75
- ZFS file system to a non-global zone
(example of), 260
- ZFS volume to a non-global zone
(example of), 262

adjusting, sizes of swap and dump devices, 149

`allocated` property, description, 93

alternate root pools

- creating
(example of), 265
- description, 265
- importing
(example of), 265

`altroot` property, description, 93

`atime` property, description, 172

attaching

- devices to ZFS storage pool (`zpool attach`)
(example of), 77

`autoreplace` property, description, 93

available property, description, 172

B

boot blocks, installing with `installboot` and `installgrub`, 153

`bootfs` property, description, 93

booting

root file system, 152

ZFS BE with `boot -L` and `boot -Z` on SPARC systems, 154

C

cache devices

considerations for using, 67

creating a ZFS storage pool with (example of), 67

cache devices, adding, (example of), 76

cache devices, removing, (example of), 76

`cachefile` property, description, 93

`canmount` property

description, 172

detailed description, 181

`capacity` property, description, 93

checking, ZFS data integrity, 269

checksum, definition, 44

checksum property, description, 172

checksummed data, description, 42

clearing

a device in a ZFS storage pool (`zpool clear`)
description, 84

device errors (`zpool clear`)
(example of), 279

clearing a device

ZFS storage pool
(example of), 85

clone, definition, 44

clones

creating (example of), 210

destroying (example of), 210

features, 209

command history, `zpool history`, 35

components of, ZFS storage pool, 57

components of ZFS, naming requirements, 45

`compression` property, description, 173

`compressratio` property, description, 173

controlling, data validation (scrubbing), 269

`copies` property, description, 173

crash dump, saving, 151

creating

a basic ZFS file system (`zpool create`)
(example of), 48

a new pool by splitting a mirrored storage pool
(`zpool split`)
(example of), 79

a ZFS storage pool (`zpool create`)
(example of), 48

alternate root pools
(example of), 265

double-parity RAID-Z storage pool (`zpool create`)
(example of), 65

mirrored ZFS storage pool (`zpool create`)
(example of), 64

single-parity RAID-Z storage pool (`zpool create`)
(example of), 65

triple-parity RAID-Z storage pool (`zpool create`)
(example of), 65

ZFS clone (example of), 210

ZFS file system, 51
(example of), 168
description, 168

ZFS file system hierarchy, 50

ZFS snapshot
(example of), 204

ZFS storage pool
description, 63

ZFS storage pool (`zpool create`)
(example of), 63

ZFS storage pool with cache devices (example
of), 67

ZFS storage pool with log devices (example of), 66

ZFS volume
(example of), 257

creation property, description, 173

D

data

- corrupted, 268
- corruption identified (`zpool status -v`)
 - (example of), 274
- repair, 269
- resilvering
 - description, 270
- scrubbing
 - (example of), 270
- validation (scrubbing), 269

dataset

- definition, 44
- description, 168

dataset types, description, 184

delegated administration, overview, 245

delegating

- dataset to a non-global zone
 - (example of), 261
- permissions (example of), 250

delegating permissions, `zfs allow`, 248

delegating permissions to a group, (example of), 250

delegating permissions to an individual user, (example of), 250

delegation property, description, 93

delegation property, disabling, 246

destroying

- ZFS clone (example of), 210
- ZFS file system
 - (example of), 169
- ZFS file system with dependents
 - (example of), 169
- ZFS snapshot
 - (example of), 205
- ZFS storage pool
 - description, 63
- ZFS storage pool (`zpool destroy`)
 - (example of), 71

detaching

- devices to ZFS storage pool (`zpool detach`)
 - (example of), 79

detecting

- in-use devices
 - (example of), 69

detecting (*Continued*)

- mismatched replication levels
 - (example of), 70

determining

- if a device can be replaced
 - description, 280
- type of device failure
 - description, 278

devices property, description, 173

differences between ZFS and traditional file systems

- file system granularity, 53
- mounting ZFS file systems, 55
- new Solaris ACL model, 55
- out of space behavior, 54
- traditional volume management, 55
- ZFS space accounting, 54

disks, as components of ZFS storage pools, 58

displaying

- command history, 35
- delegated permissions (example of), 253
- detailed ZFS storage pool health status
 - (example of), 102
- health status of storage pools
 - description of, 100
- syslog reporting of ZFS error messages
 - description, 275
- ZFS storage pool health status
 - (example of), 101
- ZFS storage pool I/O statistics
 - description, 98
- ZFS storage pool vdev I/O statistics
 - (example of), 99
- ZFS storage pool-wide I/O statistics
 - (example of), 98

dry run

- ZFS storage pool creation (`zpool create -n`)
 - (example of), 71

dumpadm, enabling a dump device, 151

dynamic striping

- description, 62
- storage pool feature, 62

E

- EFI label
 - description, 58
 - interaction with ZFS, 58
- exec property, description, 173
- exporting
 - ZFS storage pool
 - (example of), 104

F

- failmode property, description, 94
- failure modes
 - corrupted data, 268
 - damaged devices, 268
 - missing (faulted) devices, 268
- failures, 267
- file system, definition, 44
- file system granularity, differences between ZFS and traditional file systems, 53
- file system hierarchy, creating, 50
- files, as components of ZFS storage pools, 60
- free property, description, 94

G

- guid property, description, 94

H

- hardware and software requirements, 47
- health property, description, 94
- hot spares
 - creating
 - (example of), 87
 - description of
 - (example of), 87

I

- identifying
 - storage requirements, 49
 - type of data corruption (`zpool status -v`)
 - (example of), 287
 - ZFS storage pool for import (`zpool import -a`)
 - (example of), 104
- importing
 - alternate root pools
 - (example of), 265
 - ZFS storage pool
 - (example of), 107
 - ZFS storage pool from alternate directories (`zpool import -d`)
 - (example of), 106
- in-use devices
 - detecting
 - (example of), 69
- inheriting
 - ZFS properties (`zfs inherit`)
 - description, 186
- initial installation of ZFS root file system, (example of), 116
- installing
 - ZFS root file system
 - (initial installation), 115
 - features, 112
 - JumpStart installation, 124
 - requirements, 113
- installing boot blocks
 - `installboot` and `installgrp`
 - (example of), 153

J

- JumpStart installation
 - root file system
 - issues, 127
 - profile examples, 126
 - JumpStart profile keywords, ZFS root file system, 125

L

listing

- descendents of ZFS file systems
 - (example of), 184
- types of ZFS file systems
 - (example of), 185
- ZFS file systems
 - (example of), 183
- ZFS file systems (`zfs list`)
 - (example of), 52
- ZFS file systems without header information
 - (example of), 185
- ZFS pool information, 50
- ZFS properties (`zfs list`)
 - (example of), 187
- ZFS properties by source value
 - (example of), 189
- ZFS properties for scripting
 - (example of), 189
- ZFS storage pools
 - (example of), 96
 - description, 95

`listsnapshots` property, description, 94

`luactivate`

- root file system
 - (example of), 131

`lucreate`

- root file system migration
 - (example of), 130
- ZFS BE from a ZFS BE
 - (example of), 132

M

migrating

- UFS root file system to ZFS root file system
 - (Oracle Solaris Live Upgrade), 128
 - issues, 129

migrating ZFS storage pools, description, 103

mirror, definition, 44

mirrored configuration

- conceptual view, 61
- description, 61
- redundancy feature, 61

mirrored log device, adding, (example of), 75

mirrored log devices, creating a ZFS storage pool with

- (example of), 66

mirrored storage pool (`zpool create`), (example of), 64

mismatched replication levels

- detecting
 - (example of), 70

modifying

- trivial ACL on ZFS file (verbose format)
 - (example of), 229

mount point, default for ZFS storage pools, 71

mount points

- automatic, 190
- legacy, 190
- managing ZFS
 - description, 190

mounted property, description, 173

mounting

- ZFS file systems
 - (example of), 192

mounting ZFS file systems, differences between ZFS and traditional file systems, 55

mountpoint, default for ZFS file system, 168

mountpoint property, description, 174

N

naming requirements, ZFS components, 45

NFSv4 ACLs

- ACL inheritance, 223
- ACL inheritance flags, 223
- ACL property, 224
- differences from POSIX-draft ACLs, 220
- format description, 220
- model

description, 219

notifying

- ZFS of reattached device (`zpool online`)
 - (example of), 277

- O**
- offlining a device (zpool offline)
 - ZFS storage pool
 - (example of), 83
 - onlining a device
 - ZFS storage pool (zpool online)
 - (example of), 84
 - onlining and offlining devices
 - ZFS storage pool
 - description, 82
 - Oracle Solaris Live Upgrade
 - for root file system migration, 128
 - root file system migration
 - (example of), 130
 - root file system migration issues, 129
 - origin property, description, 174
 - out of space behavior, differences between ZFS and traditional file systems, 54
- P**
- permission sets, defined, 245
 - pool, definition, 44
 - pooled storage, description, 41
 - POSIX-draft ACLs, description, 220
 - primarycache property, description, 174
 - properties of ZFS
 - description, 171
 - description of heritable properties, 171
- Q**
- quota property, description, 174
 - quotas and reservations, description, 196
- R**
- RAID-Z, definition, 45
 - RAID-Z configuration
 - (example of), 65
 - conceptual view, 61
 - double-parity, description, 61
 - RAID-Z configuration (*Continued*)
 - redundancy feature, 61
 - single-parity, description, 61
 - RAID-Z configuration, adding disks to, (example of), 74
 - read-only properties of ZFS
 - available, 172
 - compression, 173
 - creation, 173
 - description, 178
 - mounted, 173
 - origin, 174
 - referenced, 175
 - type, 176
 - used, 176
 - usedbychildren, 176
 - usedbydataset, 177
 - usedbyrefreservation, 177
 - usedbysnapshots, 177
 - read-only property, description, 174
 - receiving
 - ZFS file system data (zfs receive)
 - (example of), 213
 - recordsize property
 - description, 174
 - detailed description, 181
 - recovering
 - destroyed ZFS storage pool
 - (example of), 108
 - referenced property, description, 175
 - refquota property, description, 175
 - refreservation property, description, 175
 - removing, cache devices (example of), 76
 - removing permissions, zfs unallow, 249
 - renaming
 - ZFS file system
 - (example of), 170
 - ZFS snapshot
 - (example of), 206
 - repairing
 - a damaged ZFS configuration
 - description, 276
 - an unbootable system
 - description, 291

- repairing (*Continued*)
 - pool-wide damage
 - description, 290
 - repairing a corrupted file or directory
 - description, 288
 - replacing
 - a device (zpool replace)
 - (example of), 85, 281, 285
 - a missing device
 - (example of), 276
 - replication features of ZFS, mirrored or RAID-Z, 60
 - requirements, for installation and Oracle Solaris Live Upgrade, 113
 - reservation property, description, 175
 - resilvering, definition, 45
 - resilvering and data scrubbing, description, 270
 - restoring
 - trivial ACL on ZFS file (verbose format)
 - (example of), 232
 - rights profiles, for management of ZFS file systems and storage pools, 266
 - rolling back
 - ZFS snapshot
 - (example of), 209
- S**
- savecore, saving crash dumps, 151
 - saving
 - crash dumps
 - savecore, 151
 - ZFS file system data (zfs send)
 - (example of), 212
 - scripting
 - ZFS storage pool output
 - (example of), 96
 - scrubbing
 - (example of), 270
 - data validation, 269
 - secondarycache property, description, 176
 - self-healing data, description, 62
 - sending and receiving
 - ZFS file system data
 - description, 211
 - separate log devices, considerations for using, 31
 - settable properties of ZFS
 - aclinherit, 171
 - aclmode, 172
 - atime, 172
 - canmount, 172
 - detailed description, 181
 - checksum, 172
 - compression, 173
 - copies, 173
 - description, 179
 - devices, 173
 - exec, 173
 - mountpoint, 174
 - primarycache, 174
 - quota, 174
 - read-only, 174
 - recordsize, 174
 - detailed description, 181
 - refquota, 175
 - refreservation, 175
 - reservation, 175
 - secondarycache, 176
 - setuid, 176
 - shareiscsi, 176
 - sharenfs, 176
 - snaddir, 176
 - used
 - detailed description, 179
 - version, 177
 - volblocksize, 177
 - volsize, 177
 - detailed description, 181
 - xattr, 178
 - zoned, 178
 - setting
 - ACL inheritance on ZFS file (verbose format)
 - (example of), 233
 - ACLs on ZFS file (compact format)
 - (example of), 240
 - description, 239
 - ACLs on ZFS file (verbose format)
 - (description), 228

setting (*Continued*)

- ACLs on ZFS files
 - description, 225
 - compression property
 - (example of), 52
 - legacy mount points
 - (example of), 192
 - mountpoint property, 52
 - quota property (example of), 52
 - sharenfs property
 - (example of), 52
 - ZFS atime property
 - (example of), 185
 - ZFS file system quota (zfs set quota)
 - example of, 196
 - ZFS file system reservation
 - (example of), 200
 - ZFS mount points (zfs set mountpoint)
 - (example of), 191
 - ZFS quota
 - (example of), 186
- setuid property, description, 176
- shareiscsi property, description, 176
- sharenfs property
 - description, 176, 194
- sharing
 - ZFS file systems
 - description, 194
 - example of, 194
- simplified administration, description, 43
- size property, description, 94
- snappdir property, description, 176
- snapshot
 - accessing
 - (example of), 207
 - creating
 - (example of), 204
 - definition, 45
 - destroying
 - (example of), 205
 - features, 203
 - renaming
 - (example of), 206

snapshot (*Continued*)

- rolling back
 - (example of), 209
 - space accounting, 208
- Solaris ACLs
- ACL inheritance, 223
 - ACL inheritance flags, 223
 - ACL property, 224
 - differences from POSIX-draft ACLs, 220
 - format description, 220
 - new model
 - description, 219
- splitting a mirrored storage pool (zpool split)
 - (example of), 79
- storage requirements, identifying, 49
- swap and dump devices
 - adjusting sizes of, 149
 - description, 148
 - issues, 149

T

terminology

- checksum, 44
 - clone, 44
 - dataset, 44
 - file system, 44
 - mirror, 44
 - pool, 44
 - RAID-Z, 45
 - resilvering, 45
 - snapshot, 45
 - virtual device, 45
 - volume, 45
- traditional volume management, differences between ZFS and traditional file systems, 55
- transactional semantics, description, 42
- troubleshooting
 - clear device errors (zpool clear)
 - (example of), 279
 - damaged devices, 268
 - data corruption identified (zpool status -v)
 - (example of), 274

- troubleshooting (*Continued*)
- determining if a device can be replaced
 - description, 280
 - determining if problems exist (zpool status -x), 272
 - determining type of data corruption (zpool status -v)
 - (example of), 287
 - determining type of device failure
 - description, 278
 - identifying problems, 271
 - missing (faulted) devices, 268
 - notifying ZFS of reattached device (zpool online)
 - (example of), 277
 - overall pool status information
 - description, 273
 - repairing a corrupted file or directory
 - description, 288
 - repairing a damaged ZFS configuration, 276
 - repairing an unbootable system
 - description, 291
 - repairing pool-wide damage
 - description, 290
 - replacing a device (zpool replace)
 - (example of), 281, 285
 - replacing a missing device
 - (example of), 276
 - syslog reporting of ZFS error messages, 275
 - ZFS failures, 267
- type property, description, 176
- U**
- unmounting
 - ZFS file systems
 - (example of), 194
 - unsharing
 - ZFS file systems
 - example of, 195
 - upgrading
 - ZFS storage pool
 - description, 109
- used property
- description, 176
 - used property (*Continued*)
 - detailed description, 179
 - usedbychildren property, description, 176
 - usedbydataset property, description, 177
 - usedbyreservation property, description, 177
 - usedbysnapshots property, description, 177
 - user properties of ZFS
 - (example of), 182
 - detailed description, 182
- V**
- version property, description, 177
 - version property, description, 94
 - virtual device, definition, 45
 - virtual devices, as components of ZFS storage pools, 68
 - volblocksize property, description, 177
 - volsize property
 - description, 177
 - detailed description, 181
 - volume, definition, 45
- W**
- whole disks, as components of ZFS storage pools, 58
- X**
- xattr property, description, 178
- Z**
- zfs allow
 - description, 248
 - displaying delegated permissions, 253
 - zfs create
 - (example of), 51, 168
 - description, 168
 - ZFS delegated administration, overview, 245
 - zfs destroy, (example of), 169
 - zfs destroy -r, (example of), 169

- ZFS file system
 - description, 167
 - versions
 - description, 293
- ZFS file systems
 - ACL on ZFS directory
 - detailed description, 227
 - ACL on ZFS file
 - detailed description, 226
 - adding ZFS file system to a non-global zone
 - (example of), 260
 - adding ZFS volume to a non-global zone
 - (example of), 262
 - booting a root file system
 - description, 152
 - booting a ZFS BE with boot -Land boot -Z (SPARC example of), 154
 - checksum
 - definition, 44
 - checksummed data
 - description, 42
 - clone
 - replacing a file system with (example of), 210
 - clones
 - definition, 44
 - description, 209
 - component naming requirements, 45
 - creating
 - (example of), 168
 - creating a clone, 210
 - creating a ZFS volume
 - (example of), 257
 - dataset
 - definition, 44
 - dataset types
 - description, 184
 - default mountpoint
 - (example of), 168
 - delegating dataset to a non-global zone
 - (example of), 261
 - description, 41
 - destroying
 - (example of), 169
 - destroying a clone, 210
- ZFS file systems (*Continued*)
 - destroying with dependents
 - (example of), 169
 - file system
 - definition, 44
 - inheriting property of (`zfs inherit`)
 - (example of), 186
 - initial installation of ZFS root file system, 115
 - installation and Oracle Solaris Live Upgrade requirements, 113
 - installing a root file system, 112
 - JumpStart installation of root file system, 124
 - listing
 - (example of), 183
 - listing descendents
 - (example of), 184
 - listing properties by source value
 - (example of), 189
 - listing properties for scripting
 - (example of), 189
 - listing properties of (`zfs list`)
 - (example of), 187
 - listing types of
 - (example of), 185
 - listing without header information
 - (example of), 185
 - managing automatic mount points, 190
 - managing legacy mount points
 - description, 190
 - managing mount points
 - description, 190
 - modifying trivial ACL on ZFS file (verbose format)
 - (example of), 229
 - mounting
 - (example of), 192
 - pooled storage
 - description, 41
 - property management within a zone
 - description, 263
 - receiving data streams (`zfs receive`)
 - (example of), 213
 - renaming
 - (example of), 170

ZFS file systems (*Continued*)

- restoring trivial ACL on ZFS file (verbose format)
 - (example of), 232
- rights profiles, 266
- root file system migration issues, 129
- root file system migration with Oracle Solaris Live Upgrade, 128
 - (example of), 130
- saving data streams (`zfs send`)
 - (example of), 212
- sending and receiving
 - description, 211
- setting a reservation
 - (example of), 200
- setting ACL inheritance on ZFS file (verbose format)
 - (example of), 233
- setting ACLs on ZFS file (compact format)
 - (example of), 240
 - description, 239
- setting ACLs on ZFS file (verbose format)
 - description, 228
- setting ACLs on ZFS files
 - description, 225
- setting atime property
 - (example of), 185
- setting legacy mount point
 - (example of), 192
- setting mount point (`zfs set mountpoint`)
 - (example of), 191
- setting quota property
 - (example of), 186
- sharing
 - description, 194
 - example of, 194
- simplified administration
 - description, 43
- snapshot
 - accessing, 207
 - creating, 204
 - definition, 45
 - description, 203
 - destroying, 205
 - renaming, 206
 - rolling back, 209

ZFS file systems (*Continued*)

- snapshot space accounting, 208
- swap and dump devices
 - adjusting sizes of, 149
 - description, 148
 - issues, 149
- transactional semantics
 - description, 42
- unmounting
 - (example of), 194
- unsharing
 - example of, 195
- using on a Solaris system with zones installed
 - description, 260
- volume
 - definition, 45
- ZFS file systems (`zfs set quota`)
 - setting a quota
 - example of, 196
- `zfs get`, (example of), 187
- `zfs get -H -o`, (example of), 189
- `zfs get -s`, (example of), 189
- `zfs inherit`, (example of), 186
- ZFS intent log (ZIL), description, 31
- `zfs list`
 - (example of), 52, 183
- `zfs list -H`, (example of), 185
- `zfs list -r`, (example of), 184
- `zfs list -t`, (example of), 185
- `zfs mount`, (example of), 192
- ZFS pool properties
 - allocated, 93
 - alroot, 93
 - autoreplace, 93
 - bootfs, 93
 - cachefile, 93
 - capacity, 93
 - delegation, 93
 - failmode, 94
 - free, 94
 - guid, 94
 - health, 94
 - listsnapshots, 94
 - size, 94

ZFS pool properties (*Continued*)

- version, 94
- zfs promote, clone promotion (example of), 210
- ZFS properties
 - aclinherit, 171
 - aclmode, 172
 - atime, 172
 - available, 172
 - canmount, 172
 - detailed description, 181
 - checksum, 172
 - compression, 173
 - compressratio, 173
 - copies, 173
 - creation, 173
 - description, 171
 - devices, 173
 - exec, 173
 - inheritable, description of, 171
 - management within a zone
 - description, 263
 - mounted, 173
 - mountpoint, 174
 - origin, 174
 - quota, 174
 - read-only, 174
 - read-only, 178
 - recordsize, 174
 - detailed description, 181
 - referenced, 175
 - refquota, 175
 - refreservation, 175
 - reservation, 175
 - secondarycache, 174, 176
 - settable, 179
 - setuid, 176
 - shareiscsi, 176
 - sharenfs, 176
 - snapdir, 176
 - type, 176
 - used, 176
 - detailed description, 179
 - usedbychildren, 176
 - usedbydataset, 177

ZFS properties (*Continued*)

- usedbyrefreservation, 177
- usedbysnapshots, 177
- user properties
 - detailed description, 182
- version, 177
- volblocksize, 177
- volsize, 177
 - detailed description, 181
- xattr, 178
- zoned, 178
- zoned property
 - detailed description, 264
- zfs receive, (example of), 213
- zfs rename, (example of), 170
- zfs send, (example of), 212
- zfs set atime, (example of), 185
- zfs set compression, (example of), 52
- zfs set mountpoint
 - (example of), 52, 191
- zfs set mountpoint=legacy, (example of), 192
- zfs set quota
 - (example of), 52
- zfs set quota, (example of), 186
- zfs set quota
 - example of, 196
- zfs set reservation, (example of), 200
- zfs set sharenfs, (example of), 52
- zfs set sharenfs=on, example of, 194
- ZFS space accounting, differences between ZFS and traditional file systems, 54
- ZFS storage pool
 - versions
 - description, 293
- ZFS storage pools
 - adding devices to (zpool add)
 - (example of), 73
 - alternate root pools, 265
 - attaching devices to (zpool attach)
 - (example of), 77
 - clearing a device
 - (example of), 85
 - clearing device errors (zpool clear)
 - (example of), 279

ZFS storage pools (*Continued*)

- components, 57
- corrupted data
 - description, 268
- creating (zpool create)
 - (example of), 63
- creating a RAID-Z configuration (zpool create)
 - (example of), 65
- creating mirrored configuration (zpool create)
 - (example of), 64
- damaged devices
 - description, 268
- data corruption identified (zpool status -v)
 - (example of), 274
- data repair
 - description, 269
- data scrubbing
 - (example of), 270
 - description, 269
- data scrubbing and resilvering
 - description, 270
- data validation
 - description, 269
- default mount point, 71
- destroying (zpool destroy)
 - (example of), 71
- detaching devices from (zpool detach)
 - (example of), 79
- determining if a device can be replaced
 - description, 280
- determining if problems exist (zpool status -x)
 - description, 272
- determining type of device failure
 - description, 278
- displaying detailed health status
 - (example of), 102
- displaying health status, 100
 - (example of), 101
- doing a dry run (zpool create -n)
 - (example of), 71
- dynamic striping, 62
- exporting
 - (example of), 104
- failures, 267

ZFS storage pools (*Continued*)

- identifying for import (zpool import -a)
 - (example of), 104
- identifying problems
 - description, 271
- identifying type of data corruption (zpool status -v)
 - (example of), 287
- importing
 - (example of), 107
- importing from alternate directories (zpool import -d)
 - (example of), 106
- listing
 - (example of), 96
- migrating
 - description, 103
- mirror
 - definition, 44
- mirrored configuration, description, 61
- missing (faulted) devices
 - description, 268
- notifying ZFS of reattached device (zpool onLine)
 - (example of), 277
- offlining a device (zpool offline)
 - (example of), 83
- onlining and offlining devices
 - description, 82
- overall pool status information for troubleshooting
 - description, 273
- pool
 - definition, 44
- pool-wide I/O statistics
 - (example of), 98
- RAID-Z
 - definition, 45
- RAID-Z configuration, description, 61
- recovering a destroyed pool
 - (example of), 108
- repairing a corrupted file or directory
 - description, 288
- repairing a damaged ZFS configuration, 276
- repairing an unbootable system
 - description, 291

ZFS storage pools (*Continued*)

- repairing pool-wide damage
 - description, 290
 - replacing a device (zpool replace)
 - (example of), 85, 281
 - replacing a missing device
 - (example of), 276
 - resilvering
 - definition, 45
 - rights profiles, 266
 - scripting storage pool output
 - (example of), 96
 - splitting a mirrored storage pool (zpool split)
 - (example of), 79
 - system error messages
 - description, 275
 - upgrading
 - description, 109
 - using files, 60
 - using whole disks, 58
 - vdev I/O statistics
 - (example of), 99
 - viewing resilvering process
 - (example of), 285
 - virtual device
 - definition, 45
 - virtual devices, 68
- ZFS storage pools (zpool online)
- onlining a device
 - (example of), 84
- zfs unallow, description, 249
- zfs unmount, (example of), 194
- ZFS version

- ZFS feature and Solaris OS
 - description, 293

ZFS volume, description, 257

zoned property

- description, 178
- detailed description, 264

zones

- adding ZFS file system to a non-global zone
 - (example of), 260
- adding ZFS volume to a non-global zone
 - (example of), 262

zones (*Continued*)

- delegating dataset to a non-global zone
 - (example of), 261
 - using with ZFS file systems
 - description, 260
 - ZFS property management within a zone
 - description, 263
 - zoned property
 - detailed description, 264
- zpool add, (example of), 73
- zpool attach, (example of), 77
- zpool clear
 - (example of), 85
 - description, 84
- zpool create
 - (example of), 48, 50
 - basic pool
 - (example of), 63
 - mirrored storage pool
 - (example of), 64
 - RAID-Z storage pool
 - (example of), 65
- zpool create -n, dry run (example of), 71
- zpool destroy, (example of), 71
- zpool detach, (example of), 79
- zpool export, (example of), 104
- zpool history, (example of), 35
- zpool import -a, (example of), 104
- zpool import -D, (example of), 108
- zpool import -d, (example of), 106
- zpool import *name*, (example of), 107
- zpool iostat, pool-wide (example of), 98
- zpool iostat -v, vdev (example of), 99
- zpool list
 - (example of), 50, 96
 - description, 95
- zpool list -Ho name, (example of), 96
- zpool offline, (example of), 83
- zpool online, (example of), 84
- zpool replace, (example of), 85
- zpool split, (example of), 79
- zpool status -v, (example of), 102
- zpool status -x, (example of), 101

zpool upgrade, 109

