Trademark Notice

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

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

Third Party Content, Products, and Services Disclaimer

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.

Hazardous Applications Notice

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

Restricted Rights Notice

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

U.S. GOVERNMENT RIGHTS

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

Copyright © 2006, 2011, Oracle and/or its affiliates. All rights reserved.

License Restrictions Warranty/Consequential Damages Disclaimer

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.

Warranty Disclaimer

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.

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

Third Party Content, Products, and Services Disclaimer

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.
Contents

Preface ...................................................................................................................................................13

1 Oracle Solaris ZFS File System (Introduction) .............................................................................17
  What’s New in ZFS? ...........................................................................................................................17
    ZFS File System Encryption ........................................................................................................19
    ZFS Send Stream Enhancements ..............................................................................................19
    ZFS Snapshot Differences (zfs diff) ........................................................................................19
    ZFS Storage Pool Recovery and Performance Enhancements ................................................20
    Tuning ZFS Synchronous Behavior .........................................................................................20
    Improved ZFS Pool Messages ....................................................................................................21
    ZFS ACL Interoperability Improvements ...............................................................................22
    Splitting a Mirrored ZFS Storage Pool (zpool split) .............................................................23
    Solaris iSCSI Changes ..............................................................................................................23
    New ZFS System Process .........................................................................................................23
    ZFS Deduplication Property ....................................................................................................23
    Changes to the zpool list Command .......................................................................................24
    ZFS Storage Pool Recovery ......................................................................................................24
    ZFS Log Device Enhancements ..............................................................................................25
    Triple Parity RAIDZ (raidz3) ....................................................................................................25
    Holding ZFS Snapshots ............................................................................................................26
    ZFS Device Replacement Enhancements ...............................................................................26
    ZFS User and Group Quotas ....................................................................................................27
    ZFS ACL Pass Through Inheritance for Execute Permission ...............................................28
    Automatic ZFS Snapshots ........................................................................................................29
    ZFS Property Enhancements ...................................................................................................29
    ZFS Log Device Recovery .......................................................................................................32
    Using ZFS ACL Sets ................................................................................................................33
    Using Cache Devices in Your ZFS Storage Pool ...................................................................33
Contents

ZFS Root Pool Management ................................................................. 34
Rolling Back a Dataset Without Unmounting ........................................... 34
Enhancements to the zfs send Command ..................................................... 34
ZFS Quotas and Reservations for File System Data Only .......................... 35
ZFS File System Properties for the Solaris CIFS Service .......................... 35
ZFS Storage Pool Properties ................................................................. 36
ZFS and File System Mirror Mounts ....................................................... 37
ZFS Command History Enhancements (zpool history) ............................ 37
Upgrading ZFS File Systems (zfs upgrade) .............................................. 38
ZFS Delegated Administration ............................................................... 39
Setting Up Separate ZFS Log Devices ..................................................... 39
Creating Intermediate ZFS Datasets ...................................................... 40
ZFS Hot-Plugging Enhancements ........................................................ 41
Recursively Renaming ZFS Snapshots (zfs rename -r) ............................. 41
gzip Compression Is Available for ZFS .................................................. 42
Storing Multiple Copies of ZFS User Data ............................................. 42
Improved zpool status Output ............................................................. 43
ZFS and Solaris iSCSI Improvements ..................................................... 43
Sharing ZFS File System Enhancements ............................................... 44
ZFS Command History (zpool history) ................................................. 45
ZFS Property Improvements ............................................................... 45
Displaying All ZFS File System Information ........................................... 46
New zfs receive -F Option ................................................................. 47
Recursive ZFS Snapshots ................................................................. 47
Double-Parity RAID-Z (raidz2) ............................................................ 47
Hot Spares for ZFS Storage Pool Devices .............................................. 47
Replacing a ZFS File System With a ZFS Clone (zfs promote) ................. 48
Upgrading ZFS Storage Pools (zpool upgrade) ....................................... 48
Using ZFS to Clone Non-Global Zones and Other Enhancements .......... 48
ZFS Backup and Restore Commands Are Renamed ................................. 49
Recovering Destroyed Storage Pools ..................................................... 49
ZFS Is Integrated With Fault Manager ............................................... 49
The zpool clear Command .............................................................. 50
Compact NFSv4 ACL Format ............................................................. 50
File System Monitoring Tool (fsstat) .................................................... 50
What Is ZFS? .................................................................................. 51
Contents

Mounting ZFS File Systems ........................................................................................................ 175
Using Temporary Mount Properties ....................................................................................... 177
Unmounting ZFS File Systems ................................................................................................. 177
Sharing and Unsharing ZFS File Systems ............................................................................. 178
Sharing ZFS Files in an Oracle Solaris SMB Environment .................................................. 179
Setting ZFS Quotas and Reservations .................................................................................. 181
Setting Quotas on ZFS File Systems .................................................................................... 182
Setting Reservations on ZFS File Systems ......................................................................... 185
Encrypting ZFS File Systems ............................................................................................... 187
Changing an Encrypted ZFS File System's Keys ................................................................ 188
Mounting an Encrypted ZFS File System .......................................................................... 190
Interactions Between ZFS Compression, Deduplication, and Encryption Properties ...... 190
Examples of Encrypting ZFS File Systems ......................................................................... 191
Upgrading ZFS File Systems ............................................................................................... 192

7 Working With Oracle Solaris ZFS Snapshots and Clones .................................................... 195
Overview of ZFS Snapshots ................................................................................................. 195
Creating and Destroying ZFS Snapshots ............................................................................ 196
Displaying and Accessing ZFS Snapshots .......................................................................... 199
Rolling Back a ZFS Snapshot ............................................................................................. 200
Identifying ZFS Snapshot Differences (zfs diff) ................................................................. 201
Managing Automatic ZFS Snapshots .................................................................................. 202
Overview of ZFS Clones ...................................................................................................... 205
Creating a ZFS Clone ........................................................................................................... 205
Destroying a ZFS Clone ........................................................................................................ 206
Replacing a ZFS File System With a ZFS Clone ................................................................. 206
Sending and Receiving ZFS Data ........................................................................................ 207
Saving ZFS Data With Other Backup Products ................................................................. 208
Sending a ZFS Snapshot ...................................................................................................... 208
Receiving a ZFS Snapshot ................................................................................................. 209
Applying Different Property Values to a ZFS Snapshot Stream ........................................ 210
Sending and Receiving Complex ZFS Snapshot Streams ................................................. 211
Remote Replication of ZFS Data ........................................................................................ 214
8 Using ACLs and Attributes to Protect Oracle Solaris ZFS Files ................................................................. 215
   New Solaris ACL Model .......................................................................................................................... 215
   Syntax Descriptions for Setting ACLs ................................................................................................. 216
   ACL Inheritance .................................................................................................................................. 220
   ACL Property (aclinherit) .................................................................................................................... 221
   Setting ACLs on ZFS Files ................................................................................................................. 221
   Setting and Displaying ACLs on ZFS Files in Verbose Format ......................................................... 224
   Setting ACL Inheritance on ZFS Files in Verbose Format ................................................................. 228
   Setting and Displaying ACLs on ZFS Files in Compact Format ....................................................... 234
   Applying Special Attributes to ZFS Files ............................................................................................. 238

9 Oracle Solaris ZFS Delegated Administration ......................................................................................... 241
   Overview of ZFS Delegated Administration ....................................................................................... 241
   Disabling ZFS Delegated Permissions ................................................................................................. 242
   Delegating ZFS Permissions ............................................................................................................... 242
   Delegating ZFS Permissions (zfs allow) ............................................................................................. 244
   Removing ZFS Delegated Permissions (zfs unallow) ........................................................................ 245
   Delegating ZFS Permissions (Examples) ............................................................................................ 246
   Displaying ZFS Delegated Permissions (Examples) ........................................................................... 249
   Removing ZFS Delegated Permissions (Examples) ............................................................................ 251

10 Oracle Solaris ZFS Advanced Topics ..................................................................................................... 253
   ZFS Volumes ...................................................................................................................................... 253
   Using a ZFS Volume as a Swap or Dump Device ................................................................................. 254
   Using a ZFS Volume as a Solaris iSCSI LUN .................................................................................... 254
   Using ZFS on a Solaris System With Zones Installed ........................................................................ 256
   Adding ZFS File Systems to a Non-Global Zone ............................................................................... 257
   Delegating Datasets to a Non-Global Zone ......................................................................................... 257
   Adding ZFS Volumes to a Non-Global Zone ....................................................................................... 258
   Using ZFS Storage Pools Within a Zone ............................................................................................. 258
   Managing ZFS Properties Within a Zone ............................................................................................ 259
   Understanding the zoned Property ..................................................................................................... 260
   Using ZFS Alternate Root Pools .......................................................................................................... 261
   Creating ZFS Alternate Root Pools ..................................................................................................... 261
   Importing Alternate Root Pools .......................................................................................................... 261
Index ................................................................................................................................................... 293
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 Oracle Solaris Hardware Compatibility List at http://www.sun.com/bigadmin/hcl. This document cites any implementation differences between the platform types.

For supported systems, see the Oracle Solaris 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.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1, &quot;Oracle Solaris ZFS File System (Introduction)&quot;</td>
<td>Provides an overview of ZFS and its features and benefits. It also covers some basic concepts and terminology.</td>
</tr>
<tr>
<td>Chapter 2, &quot;Getting Started With Oracle Solaris ZFS&quot;</td>
<td>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.</td>
</tr>
</tbody>
</table>
Chapter Description

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, "Managing ZFS Root Pool Components" Describes how to manage ZFS root pool components, such as configuring a mirrored root pool, upgrading your ZFS boot environments, and resizing swap and dump devices.

Chapter 6, "Managing Oracle Solaris ZFS File Systems" Provides detailed information about managing ZFS file systems. Included are such concepts as the hierarchical file system layout, property inheritance, and automatic mount point management and share interactions.

Chapter 7, "Working With Oracle Solaris ZFS Snapshots and Clones" Describes how to create and administer ZFS snapshots and clones.

Chapter 8, "Using ACLs and Attributes to Protect Oracle Solaris ZFS Files" Describes how to use access control lists (ACLs) to protect your ZFS files by providing more granular permissions than the standard UNIX permissions.

Chapter 9, "Oracle Solaris ZFS Delegated Administration" Describes how to use ZFS delegated administration to allow nonprivileged users to perform ZFS administration tasks.

Chapter 10, "Oracle Solaris ZFS Advanced Topics" Provides information about using ZFS volumes, using ZFS on an Oracle Solaris system with zones installed, and using alternate root pools.

Chapter 11, "Oracle Solaris ZFS Troubleshooting and Pool Recovery" Describes how to identify ZFS failures and how to recover from them. Steps for preventing failures are covered as well.

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: Common System Management Tasks
- 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://www.oracle.com/technetwork/indexes/documentation/index.html)
- Training (http://education.oracle.com) – Click the Sun link in the left navigation bar.

Oracle Software Resources

Oracle Technology Network (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).

Typographic Conventions

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

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AaBbCc123</td>
<td>The names of commands, files, and directories, and onscreen computer output</td>
<td>Edit your .login file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use ls -a to list all files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>machine_name% you have mail.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>What you type, contrasted with onscreen computer output</td>
<td>machine_name% su</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Password:</td>
</tr>
<tr>
<td>aabbcc123</td>
<td>Placeholder: replace with a real name or value</td>
<td>The command to remove a file is rm filename.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Shell</th>
<th>Prompt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bash shell, Korn shell, and Bourne shell</td>
<td>$</td>
</tr>
<tr>
<td>Bash shell, Korn shell, and Bourne shell for superuser</td>
<td>#</td>
</tr>
<tr>
<td>C shell</td>
<td>machine_name%</td>
</tr>
<tr>
<td>C shell for superuser</td>
<td>machine_name#</td>
</tr>
</tbody>
</table>

Note: Some emphasized items appear bold online.

Table of Contents

Preface

Oracle Solaris ZFS Administration Guide • January 2011

16
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 17
- “What Is ZFS?” on page 51
- “ZFS Terminology” on page 53
- “ZFS Component Naming Requirements” on page 55

What's New in ZFS?

This section summarizes new features in the ZFS file system.

- “ZFS File System Encryption” on page 19
- “ZFS Send Stream Enhancements” on page 19
- “ZFS Snapshot Differences (zfs diff)” on page 19
- “ZFS Storage Pool Recovery and Performance Enhancements” on page 20
- “Tuning ZFS Synchronous Behavior” on page 20
- “Improved ZFS Pool Messages” on page 21
- “ZFS ACL Interoperability Improvements” on page 22
- “Splitting a Mirrored ZFS Storage Pool (zpool split)” on page 23
- “Solaris iSCSI Changes” on page 23
- “New ZFS System Process” on page 23
- “ZFS Deduplication Property” on page 23
- “Changes to the zpool list Command” on page 24
- “ZFS Storage Pool Recovery” on page 24
- “ZFS Log Device Enhancements” on page 25
- “Triple Parity RAIDZ (raidz3)” on page 25
- “Holding ZFS Snapshots” on page 26
- “ZFS Device Replacement Enhancements” on page 26
What's New in ZFS?

- “ZFS User and Group Quotas” on page 27
- “ZFS ACL Pass Through Inheritance for Execute Permission” on page 28
- “Automatic ZFS Snapshots” on page 29
- “ZFS Property Enhancements” on page 29
- “ZFS Log Device Recovery” on page 32
- “Using ZFS ACL Sets” on page 33
- “Using Cache Devices in Your ZFS Storage Pool” on page 33
- “ZFS Root Pool Management” on page 34
- “Rolling Back a Dataset Without Unmounting” on page 34
- “Enhancements to the zfs send Command” on page 34
- “ZFS Quotas and Reservations for File System Data Only” on page 35
- “ZFS File System Properties for the Solaris CIFS Service” on page 35
- “ZFS Storage Pool Properties” on page 36
- “ZFS and File System Mirror Mounts” on page 37
- “ZFS Command History Enhancements (zpool history)” on page 37
- “Upgrading ZFS File Systems (zfs upgrade)” on page 38
- “ZFS Delegated Administration” on page 39
- “Setting Up Separate ZFS Log Devices” on page 39
- “Creating Intermediate ZFS Datasets” on page 40
- “ZFS Hot-Plugging Enhancements” on page 41
- “Recursively Renaming ZFS Snapshots (zfs rename -r)” on page 41
- “gzip Compression Is Available for ZFS” on page 42
- “Storing Multiple Copies of ZFS User Data” on page 42
- “Improved zpool status Output” on page 43
- “ZFS and Solaris iSCSI Improvements” on page 43
- “Sharing ZFS File System Enhancements” on page 44
- “ZFS Command History (zpool history)” on page 45
- “ZFS Property Improvements” on page 45
- “Displaying All ZFS File System Information” on page 46
- “New zfs receive -F Option” on page 47
- “Recursive ZFS Snapshots” on page 47
- “Double-Parity RAID-Z (raidz2)” on page 47
- “Hot Spares for ZFS Storage Pool Devices” on page 47
- “Replacing a ZFS File System With a ZFS Clone (zfs promote)” on page 48
- “Upgrading ZFS Storage Pools (zpool upgrade)” on page 48
- “Using ZFS to Clone Non-Global Zones and Other Enhancements” on page 48
- “ZFS Backup and Restore Commands Are Renamed” on page 49
- “Recovering Destroyed Storage Pools” on page 49
- “ZFS Is Integrated With Fault Manager” on page 49
- “The zpool clear Command” on page 50
- “Compact NFSv4 ACL Format” on page 50
- “File System Monitoring Tool (fsstat)” on page 50
**ZFS File System Encryption**

**Oracle Solaris 11 Express, build 149:** In this Solaris release, you can encrypt a ZFS file system.

For example, the tank/cindys file system is created with the encryption property enabled. The default encryption policy is to prompt for a passphrase, which must be a minimum of 8 characters in length.

```
# zfs create -o encryption=on tank/cindys
Enter passphrase for 'tank/cindys': xxx
Enter again: xxx
```

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

For more information, see “Encrypting ZFS File Systems” on page 187.

**ZFS Send Stream Enhancements**

**Oracle Solaris 11 Express, build 148:** In this Solaris release, you can set file system properties that are sent and received in a snapshot stream. These enhancements provide flexibility in applying file system properties in a send stream to the receiving file system or in determining whether the local file system properties should be ignored when received, such as a mountpoint property value.

For more information, see “Applying Different Property Values to a ZFS Snapshot Stream” on page 210.

**ZFS Snapshot Differences (zfs diff)**

**Oracle Solaris 11 Express, build 147:** In this Solaris release, you can determine ZFS snapshot differences by using the zfs diff command.

For example, the following two snapshots are created:

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

Identify the snapshot differences. For example:

```
$ zfs diff tank/cindys@0913 tank/cindys@0914
M /tank/cindys/
+ /tank/cindys/fileB
```
In the above output, the M indicates that the directory has been modified. The + indicates that fileB exists in the later snapshot.

For more information, see “Identifying ZFS Snapshot Differences (zfs diff)” on page 201.

ZFS Storage Pool Recovery and Performance Enhancements

Oracle Solaris 11 Express, build 146 and build 147: In this Solaris release, the following new ZFS storage pool features are provided:

- You can import a pool with a missing log by using the zpool import -m command.
  For more information, see “Importing a Pool With a Missing Log Device” on page 119.
- You can import a pool in read-only mode. This feature is primarily for pool recovery. If a damaged pool cannot be accessed because the underlying devices are damaged, you can import the pool read-only to recover the data.
  For more information, see “Importing a Pool in Read-Only Mode” on page 121.
- A RAIDZ (raidz1, raidz2, or raidz3) storage pool, created in this release and upgraded to at least pool version 29, will have some latency-sensitive metadata automatically mirrored to improve read I/O throughput performance. For existing RAIDZ pools that are upgraded to at least pool version 29, some metadata will be mirrored for all newly written data.
  Mirrored metadata in a RAIDZ pool does not provide additional protection against hardware failures, similar to what a mirrored storage pool provides. Additional space is consumed by mirrored metadata, but the RAIDZ protection remains the same as previous releases. This is a performance enhancement only.

Tuning ZFS Synchronous Behavior

Oracle Solaris 11 Express, build 140: In this Solaris release, you can determine a ZFS file system’s synchronous behavior by using the sync property.

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

The property can be set before or after the dataset is created, and it takes effect immediately. For example:

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

The zil_disable parameter is no longer available in Solaris releases that include the sync property.
For more information, see Table 6–1.

**Improved ZFS Pool Messages**

**Oracle Solaris 11 Express, build 140**: In this Solaris release, you can use the `-T` option to provide an interval and count value for the `zpool list` and `zpool status` commands to display additional information.

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

- **Resilver in-progress report.** For example:
  ```
  scan: resilver in progress since Wed Mar 31 11:12:48 2010
  1.26G scanned out of 2.40G at 6.15M/s, 0h3m to go
  1.26G resilvered, 56.3% done
  ```

- **Scrub in-progress report.** For example:
  ```
  scan: scrub in progress since Wed May  5 10:34:33 2010
  18.0M scanned out of 2.35G at 8.99M/s, 0h4m to go
  0 repaired, 0.75% done
  ```

- **Resilver completion message.** For example:
  ```
  scan: resilvered 2.34G in 1h2m with 0 errors on Wed Mar 31 11:19:49 2010
  ```

- **Scrub completion message.** For example:
  ```
  scan: scrub repaired 512B in 1h2m with 0 errors on Wed Mar 31 11:19:49 2010
  ```

- **Ongoing scrub cancellation message.** For example:
  ```
  scan: scrub canceled on Wed May  5 10:37:34 2010
  ```

- **Scrub and resilver completion messages persist across system reboots**

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

```bash
# zpool status -T d tank 3 2
Thu Apr  8 06:46:39 PDT 2010
pool: tank
state: ONLINE
status: One or more devices is currently being resilvered. The pool will
continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
scan: resilver in progress since Thu Apr  8 06:46:18 2010
  81.3M scanned out of 202M at 3.69M/s, 0h0m to go
  37.0M resilvered, 40.21% done
config:
  NAME   STATE
  tank   ONLINE
  mirror-0 ONLINE
  c2t3d0 ONLINE
```
ZFS ACL Interoperability Improvements

Oracle Solaris 11 Express, build 139: In this Solaris release, the following ACL improvements are provided:

- Trivial ACLs do not require deny ACEs except for unusual permissions. For example, a mode of 0644, 0755, 0664 do not need deny ACEs, but modes, such as 0705, 0060 and so on, require deny ACEs.

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

```
# ls -v file.1
-rw-r--r-- 1 root root 206663 Jun 14 11:52 file.1
0:owner@:execute:deny
1:owner@:read_data/write_data/append_data/write_xattr/write_attributes
   /write_acl/write_owner:allow
2:group@:write_data/append_data/execute:deny
3:group@:read data:allow
4:everyone@:write_data/append_data/write_xattr/execute/write_attributes
   /write_acl/write_owner:deny
5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
   :allow
```

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

```
# ls -v file.1
-rw-r--r-- 1 root root 206674 Jun 14 10:54 file.1
0:owner@:read data/write_data/append_data/read_xattr/write_xattr
   /write_attributes/write_acl/write_acl/write_owner
   /synchronize:allow
1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
   :allow
```

- ACLs are no longer split into multiple ACEs during inheritance to try to preserve the original unmodified permission. Instead, the permissions are modified as necessary to enforce the file creation mode.

- The aclinherit property behavior includes a reduction of permissions when the property is set to restricted, which means ACLs are no longer split into multiple ACEs during inheritance.

- An existing ACL is discarded during chmod(2) operations by default. This change means that the ZFS aclmode property is no longer available.

- A new permission mode calculation rule means that if an ACL has a user ACE that is also the file owner, then those permissions are included in the permission mode computation. The same rule applies if a group ACE is the group owner of the file.
For more information, see Chapter 8, "Using ACLs and Attributes to Protect Oracle Solaris ZFS Files."

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

**Oracle Solaris 11 Express, build 140**: 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 89.

**Solaris iSCSI Changes**

**Oracle Solaris 11 Express, build 136**: In this Solaris release, the Solaris iSCSI target daemon is replaced by using the COMSTAR target daemon. This also means that the `share iscsi` property that was used to share a ZFS volume as an iSCSI LUN is no longer available. Use the `sbdadm` command to configure and share a ZFS volume as an iSCSI LUN.

For more information, see “Using a ZFS Volume as a Solaris iSCSI LUN” on page 254.

**New ZFS System Process**

**Oracle Solaris 11 Express, build 129**: 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).

**ZFS Deduplication Property**

**Oracle Solaris 11 Express, build 128**: In this Solaris release, you can use the deduplication property to remove redundant data from your ZFS file systems. If a file system has the `dedup` property enabled, duplicate data blocks are removed synchronously. The result is that only unique data is stored and common components are shared between files.

You can enable this property as follows:

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

For more information, see Chapter 8, "Using ACLs and Attributes to Protect Oracle Solaris ZFS Files."
Although deduplication is set as a file system property, the scope is pool-wide. For example, you can identify the deduplication ratio as follows:

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

The `zpool list` output has been updated to support the deduplication property. For more information, see “Changes to the `zpool list` Command” on page 24.

For more information setting the deduplication property, see “The dedup Property” on page 163.

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

- Determine if your data would benefit from deduplication space savings
- Determine whether your system has enough physical memory to support deduplication
- Known issues exist when attempting to remove a large dataset or destroying a large snapshot with duplicated data

For more information about these considerations, see this FAQ:

http://hub.opensolaris.org/bin/view/Community+Group+zfs/dedup

## Changes to the zpool list Command

### Oracle Solaris 11 Express, build 128:

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 105.

## ZFS Storage Pool Recovery

### Oracle Solaris 11 Express, build 128:

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 286.

### ZFS Log Device Enhancements

**Oracle Solaris 11 Express, builds 122–125:** 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 11 Express, build 120:** 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 75.
Holding ZFS Snapshots

**Oracle Solaris 11 Express, build 120:** 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/snap@1
```

For more information, see “Holding ZFS Snapshots” on page 197.

ZFS Device Replacement Enhancements

**Oracle Solaris 11 Express, build 120:** In this Solaris release, a system event or `sys/event` is provided when an underlying device is expanded. ZFS has been enhanced to recognize these events and adjusts the pool based on the new size of the expanded LUN, depending on the setting of the `autoexpand` property. You can use the `autoexpand` pool property to enable or disable automatic pool expansion when a dynamic LUN expansion event is received.

These features enable you to expand a LUN and the resulting pool can access the expanded space without having to export and import pool or reboot the system.

For example, automatic LUN expansion 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 LUN expanded or not.

A LUN can 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 LUN 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.
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 a LUN. As described above, you can enable the autoexpand property or use the zpool online -e command to expand the full size of a LUN.

For more information about replacing devices, see "Replacing Devices in a Storage Pool" on page 94.

**ZFS User and Group Quotas**

**Oracle Solaris 11 Express, build 120:** 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 181.

## ZFS ACL Pass Through Inheritance for Execute Permission

**Oracle Solaris 11 Express, build 120:** 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 8–13.

**Automatic ZFS Snapshots**

**Oracle Solaris 11 Express, build 120**: This release includes the Time Slider snapshot tool. This tool automatically snapshots ZFS file systems and allows you to browse and recover snapshots of file systems. For more information, see “Managing Automatic ZFS Snapshots” on page 202.

**ZFS Property Enhancements**

**Oracle Solaris 11 Express, build 120**: The following ZFS file system enhancements are included in these releases.

- **Setting ZFS Security Labels** – The mlslabel property is a sensitivity label that determines if a dataset can be mounted in a Trusted Extensions labeled-zone. The default is none. The mlslabel property can be modified only when Trusted Extensions is enabled and only with the appropriate privilege.

- **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 refreservation. The property abbreviation is usedds.

usedbyrefreservation – Identifies the amount of disk space that is used by a refreservation set on this dataset, which would be freed if the refreservation 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:

\[
\text{used property} = \text{usedbychildren} + \text{usedbydataset} + \text{usedbyrefreservation} + \text{usedbysnapshots}
\]

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

```
$ zfs list -o space
NAME   AVAIL  USED  USEDSNAP  USEDSDS  USEDREFRESERV  USEDCHILD
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   18K  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.

**Listingsnapshots** – The listsnapshots pool property controls whether snapshot information is displayed by the `zfs list` command. The default value is off, which means snapshot information is not displayed by default.

You can use the `zfs list -t snapshots` command to display 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    -
```

To display snapshot information by default, set the listsnapshots property. For example:

```
# zpool get listsnapshots pool
NAME  PROPERTY  VALUE  SOURCE
pool  listsnapshots off  default
# zpool set listsnap=on pool
```
Keep in mind that changing the default listsnapshots setting might cause the zfs list output to run slowly in a pool with many snapshots.

ZFS Log Device Recovery

Oracle Solaris 11 Express, build 120: 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 ZFS ACL Sets**

*Oracle Solaris 11 Express, build 120:* This release provides the ability to apply NFSv4–style ACLs in sets, rather than apply different ACL permissions individually. The following ACL sets are provided:

- `full_set` = all permissions
- `modify_set` = all permissions except `write_acl` and `write_owner`
- `read_set` = `read_data`, `read_attributes`, `read_xattr`, and `read_acl`
- `write_set` = `write_data`, `append_data`, `write_attributes`, and `write_xattr`

These ACL sets are prefined and cannot be modified.

For more information about using ACL sets, see Example 8–5.

**Using Cache Devices in Your ZFS Storage Pool**

*Oracle Solaris 11 Express, build 120:* 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:

```bash
# 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     ONLINE 0      0       0
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 77 and Example 4–4.

### ZFS Root Pool Management

**Oracle Solaris 11 Express, build 120:** This Oracle Solaris release uses a ZFS root file system by default. For more information about managing root pool components, see Chapter 5, “Managing ZFS Root Pool Components.”

### Rolling Back a Dataset Without Unmounting

**Oracle Solaris 11 Express, build 120:** 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

**Oracle Solaris 11 Express, build 120:** 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:
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 211.

### ZFS Quotas and Reservations for File System Data Only

**Oracle Solaris 11 Express, build 120:** 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 181.

### ZFS File System Properties for the Solaris CIFS Service

**Oracle Solaris 11 Express, build 120:** This release provides support for the Solaris Common Internet File System (CIFS) service. This product provides the ability to share files between Solaris and Windows or MacOS systems.
To facilitate sharing files between these systems by using the Solaris CIFS service, the following new ZFS properties are provided:

- Case sensitivity support (casesensitivity)
- Non-blocking mandatory locks (nbmand)
- SMB share support (sharesmb)
- Unicode normalization support (normalization)
- UTF-8 character set support (utf8only)

Currently, the sharesmb property is available to share ZFS files in the Solaris CIFS environment. More ZFS CIFS-related properties will be available in an upcoming release. For information about using the sharesmb property, see “Sharing and Unsharing ZFS File Systems” on page 178.

In addition to the ZFS properties added for supporting the Solaris CIFS software product, the vscan property is available for scanning ZFS files if you have a 3rd-party virus scanning engine.

**ZFS Storage Pool Properties**

**Oracle Solaris 11 Express, build 120**: 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 – **Oracle Solaris 11 Express, build 120**. 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:

  ```sh
  # 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 and File System Mirror Mounts**

**Oracle Solaris 11 Express, build 120:** In this Solaris release, NFSv4 mount enhancements are provided to make ZFS file systems more accessible to NFS clients.

When file systems are created on the NFS server, the NFS client can automatically discover these newly created file systems within their existing mount of a parent file system.

For example, if the server neo already shares the tank file system and client zee has it mounted, /tank/baz is automatically visible on the client after it is created on the server.

```
zee# mount neo:/tank /mnt
zee# ls /mnt
baa  bar

neo# zfs create tank/baz

zee% ls /mnt
baa  bar  baz
zee% ls /mnt/baz
file1  file2
```

**ZFS Command History Enhancements (zpool history)**

**Oracle Solaris 11 Express, build 120:** 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 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 mountpoint=/ rpool/ROOT/zfsBE
  2010-06-23.09:30:16 zfs set mountpoint=/ rpool/ROOT/zfsBE
  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:
What's New in ZFS?

The `zpool history` command 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 [user root on neo:global]
2010-06-24.13:08:23 zpool scrub tank
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 267.

Upgrading ZFS File Systems (zfs upgrade)

Oracle Solaris 11 Express, build 120: 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

**Oracle Solaris 11 Express, build 120:** 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:

```bash
# 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

**Oracle Solaris 11 Express, build 120:** 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 76 and “Adding Devices to a Storage Pool” on page 82.

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:
Log devices can be unreplicated or mirrored, but RAID-Z is not supported for log devices.

- If a separate log device is not mirrored and the device that contains the log fails, storing log blocks reverts to the storage pool.
- Log devices can be added, replaced, removed, attached, detached, imported, and exported as part of the larger storage pool.
- 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

Oracle Solaris 11 Express, build 120: 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
```

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

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
```

For more information, see zfs(1M).
ZFS Hot-Plugging Enhancements

**Oracle Solaris 11 Express, build 120:** 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)

**Oracle Solaris 11 Express, build 120:** 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

<table>
<thead>
<tr>
<th>NAME</th>
<th>USED</th>
<th>AVAIL</th>
<th>REFER</th>
<th>MOUNTPOINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>users</td>
<td>216K</td>
<td>16.5G</td>
<td>20K</td>
<td>/users</td>
</tr>
<tr>
<td>users/home</td>
<td>76K</td>
<td>16.5G</td>
<td>22K</td>
<td>/users/home</td>
</tr>
<tr>
<td>users/home@today</td>
<td>0</td>
<td>-</td>
<td>22K</td>
<td>-</td>
</tr>
<tr>
<td>users/home/markm</td>
<td>18K</td>
<td>16.5G</td>
<td>18K</td>
<td>/users/home/markm</td>
</tr>
<tr>
<td>users/home/markm@today</td>
<td>0</td>
<td>-</td>
<td>18K</td>
<td>-</td>
</tr>
<tr>
<td>users/home/marks</td>
<td>18K</td>
<td>16.5G</td>
<td>18K</td>
<td>/users/home/marks</td>
</tr>
<tr>
<td>users/home/marks@today</td>
<td>0</td>
<td>-</td>
<td>18K</td>
<td>-</td>
</tr>
<tr>
<td>users/home/neil</td>
<td>18K</td>
<td>16.5G</td>
<td>18K</td>
<td>/users/home/neil</td>
</tr>
<tr>
<td>users/home/neil@today</td>
<td>0</td>
<td>-</td>
<td>18K</td>
<td>-</td>
</tr>
</tbody>
</table>
```
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 195 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

**Oracle Solaris 11 Express, build 120:** 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 169.

### Storing Multiple Copies of ZFS User Data

**Oracle Solaris 11 Express, build 120:** 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

<table>
<thead>
<tr>
<th>NAME</th>
<th>PROPERTY</th>
<th>VALUE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>users/home</td>
<td>copies</td>
<td>2</td>
<td>local</td>
</tr>
</tbody>
</table>

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-filesystem 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 169.

**Improved zpool status Output**

**Oracle Solaris 11 Express, build 120:** 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 285.

**ZFS and Solaris iSCSI Improvements**

**Oracle Solaris 11 Express, build 120:** 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 iSCSI Storage Devices With COMSTAR," 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 LUN” on page 254.

Sharing ZFS File System Enhancements

Oracle Solaris 11 Express, build 120: In this Solaris release, the process of sharing file systems has been improved. Although modifying system configuration files, such as /etc/dfs/dfstab, is unnecessary for sharing ZFS file systems, you can use the sharemgr command to manage ZFS share properties. The sharemgr command enables you to set and manage share properties on share groups. ZFS shares are automatically designated in the zfs share group.

As in previous releases, you can set the ZFS shares property on a ZFS file system to share a ZFS file system. For example:

# zfs set shares=on tank/home

Or, you can use the new sharemgr add-share subcommand to share a ZFS file system in the zfs share group. For example:

# sharemgr add-share -s tank/data zfs
# sharemgr show -vp zfs
zfs/nfs=()
  zfs/tank/data
  /tank/data
  /tank/data/1
  /tank/data/2
  /tank/data/3

Then, you can use the sharemgr command to manage ZFS shares. The following example shows how to use sharemgr to set the nosuid property on the shared ZFS file systems. You must preface ZFS share paths with a /zfs designation.

# sharemgr set -P nfs -p nosuid=true zfs/tank/data
# sharemgr show -vp zfs
zfs/nfs=()
  zfs/tank/data nfs=(nosuid="true")
  /tank/data
  /tank/data/1
  /tank/data/2
  /tank/data/3
For more information, see `sharemgr(1M)`.

### ZFS Command History (zpool history)

**Oracle Solaris 11 Express, build 120:** In this Solaris release, ZFS automatically logs successful `zfs` and `zpool` commands that modify pool state information. For example:

```bash
# 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:

```bash
# 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`. For more information, see “ZFS Command History Enhancements (zpool history)” on page 37.

For more information about troubleshooting ZFS problems, see “Resolving Problems With ZFS” on page 267.

### ZFS Property Improvements

#### ZFS xattr Property

**Oracle Solaris 11 Express, build 120:** 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 149.
**ZFS canmount Property**

*Oracle Solaris 11 Express, build 120:* The new canmount property enables you to specify whether a dataset can be mounted by using the `zfs mount` command. For more information, see “The canmount Property” on page 162.

**ZFS User Properties**

*Oracle Solaris 11 Express, build 120:* 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 165.

**Setting Properties When Creating ZFS File Systems**

*Oracle Solaris 11 Express, build 120:* 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**

*Oracle Solaris 11 Express, build 120:* 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

Oracle Solaris 11 Express, build 120: 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 209.

Recursive ZFS Snapshots

Oracle Solaris 11 Express, build 120: 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 196.

Double-Parity RAID-Z (raidz2)

Oracle Solaris 11 Express, build 120: 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 75 or zpool(1M).

Hot Spares for ZFS Storage Pool Devices

Oracle Solaris 11 Express, build 120: 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 96 and zpool(1M).
Replacing a ZFS File System With a ZFS Clone (zfs promote)

Oracle Solaris 11 Express, build 120: 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 206 and zfs(1M).

Upgrading ZFS Storage Pools (zpool upgrade)

Oracle Solaris 11 Express, build 120: 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 123 and zpool(1M).

Using ZFS to Clone Non-Global Zones and Other Enhancements

Oracle Solaris 11 Express, build 120: When the source zonepath and the target zonepath both reside on ZFS and are in the same pool, zoneadm clone now automatically uses the ZFS clone feature to clone a zone. This enhancement means that zoneadm clone will take a ZFS snapshot of the source zonepath and set up the target zonepath. The snapshot is named SUNWzoneX, where X is a unique ID used to distinguish between multiple snapshots. The destination zone’s zonepath is used to name the ZFS clone. A software inventory is performed so that a snapshot used at a future time can be validated by the system. Note that you can still specify that the ZFS zonepath be copied instead of the ZFS clone, if desired.

To clone a source zone multiple times, a new parameter added to zoneadm allows you to specify that an existing snapshot should be used. The system validates that the existing snapshot is usable on the target. Additionally, the zone install process now has the capability to detect when a ZFS file system can be created for a zone, and the uninstall process can detect when a ZFS file system in a zone can be destroyed. These steps are then performed automatically by the zoneadm command.

Keep the following points in mind when using ZFS on a system with Solaris containers installed:
- Do not use the ZFS snapshot features to clone a zone
You can delegate or add a ZFS file system to a non-global zone. For more information, see "Adding ZFS File Systems to a Non-Global Zone" on page 257 or "Delegating Datasets to a Non-Global Zone" on page 257.

For more information, see System Administration Guide: Oracle Solaris Zones, Oracle Solaris 10 Containers, and Resource Management.

**ZFS Backup and Restore Commands Are Renamed**

**Oracle Solaris 11 Express, build 120:** 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 207.

**Recovering Destroyed Storage Pools**

**Oracle Solaris 11 Express, build 120:** 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 121.

**ZFS Is Integrated With Fault Manager**

**Oracle Solaris 11 Express, build 120:** 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
PLATFORM: SUNW,Sun-Fire-880, CSN: -, HOSTNAME: neo
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: 504a1188-b270-4ab0-af4e-8a77680576b8
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 272.

**The zpool clear Command**

**Oracle Solaris 11 Express, build 120:** 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 94 and `zpool(1M)`.

**Compact NFSv4 ACL Format**

**Oracle Solaris 11 Express, build 120:** 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 234, `chmod(1)`, and `ls(1)`.

**File System Monitoring Tool (fsstat)**

**Oracle Solaris 11 Express, build 120:** 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  rmdir  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)`.
What Is ZFS?

The Solaris 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 `lucreate` command and possibly updated by the `lupgrade` command, but it is not the active or primary boot environment. The alternate boot environment can become the primary boot environment by running the `livate` 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**: A file system whose initial contents are identical to the contents of a snapshot.

  For information about clones, see “Overview of ZFS Clones” on page 205.

- **dataset**: A generic name for the following ZFS components: clones, file systems, snapshots, and volumes.

  Each dataset is identified by a unique name in the ZFS namespace. Datasets are identified using the following format:

  \[pool\text{/path}\[@\text{snapshot}\]\]

  \[pool\] Identifies the name of the storage pool that contains the dataset
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Is a slash-delimited path name for the dataset component</td>
</tr>
<tr>
<td>snapshot</td>
<td>Is an optional component that identifies a snapshot of a dataset</td>
</tr>
</tbody>
</table>

For more information about datasets, see Chapter 6, “Managing Oracle Solaris ZFS File Systems.”

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>file system</td>
<td>A ZFS dataset of type filesystem that is mounted within the standard system namespace and behaves like other file systems.</td>
</tr>
</tbody>
</table>

For more information about file systems, see Chapter 6, “Managing Oracle Solaris ZFS File Systems.”

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>mirror</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool</td>
<td>A logical group of devices describing the layout and physical characteristics of the available storage. Disk space for datasets is allocated from a pool.</td>
</tr>
</tbody>
</table>

For more information about storage pools, see Chapter 4, “Managing Oracle Solaris ZFS Storage Pools.”

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary boot environment</td>
<td>A boot environment that is used by the lucreate 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 lucreate -s option.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAID-Z</td>
<td>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 71.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>resilvering</td>
<td>The process of copying data from one device to another device is known as resilvering. 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 mirror resynchronization in traditional volume management products.</td>
</tr>
</tbody>
</table>

For more information about ZFS resilvering, see “Viewing Resilvering Status” on page 281.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>snapshot</td>
<td>A read-only copy of a file system or volume at a given point in time.</td>
</tr>
</tbody>
</table>
For more information about snapshots, see “Overview of ZFS Snapshots” on page 195.

**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 78.

**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 253.

### 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 57
- “Creating a Basic ZFS File System” on page 58
- “Creating a ZFS Storage Pool” on page 59
- “Creating a ZFS File System Hierarchy” on page 60

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 Oracle Solaris 11 Express 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 70.

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 60.
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 59, 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 67.

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 59, 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 70.

▼ 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 262.
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 55.

3 Create the pool.
For example, the following command creates a mirrored pool that is named tank:

```bash
# 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 73. For more information about how device usage is determined, see “Detecting In-Use Devices” on page 79.

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

```bash
# 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 105.

---

## 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 61.

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 61, 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 61, 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 149.

▼ 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 262.

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 149.

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.

```bash
# 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 146.

For more information about mounting and sharing file systems, see “Mounting and Sharing ZFS File Systems” on page 173.

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.

```bash
# zfs set quota=10G tank/home/bonwick
```

6 View the results.

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

```bash
# 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 166.

For more information about how disk space is used and calculated, see “ZFS Disk Space Accounting” on page 64.
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 63
- “ZFS Disk Space Accounting” on page 64
- “Out of Space Behavior” on page 64
- “Mounting ZFS File Systems” on page 65
- “Traditional Volume Management” on page 65
- “New Solaris ACL Model” on page 65

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 descendant 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 60.
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 182. 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 185. 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 173.

**Traditional Volume Management**

As described in “ZFS Pooled Storage” on page 51, ZFS eliminates the need for a separate volume manager. ZFS operates on raw devices, so it is possible to create a storage pool comprised of logical volumes, either software or hardware. This configuration is not recommended, as ZFS works best when it uses raw physical devices. Using logical volumes might sacrifice performance, reliability, or both, and should be avoided.

**New Solaris ACL Model**

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

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

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

For more information about using ACLs with ZFS files, see Chapter 8, "Using ACLs and Attributes to Protect Oracle Solaris ZFS Files."
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 67
- “Replication Features of a ZFS Storage Pool” on page 70
- “Creating and Destroying ZFS Storage Pools” on page 73
- “Managing Devices in ZFS Storage Pools” on page 82
- “Managing ZFS Storage Pool Properties” on page 102
- “Querying ZFS Storage Pool Status” on page 105
- “Migrating ZFS Storage Pools” on page 115
- “Upgrading ZFS Storage Pools” on page 123

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 67
- “Using Slices in a ZFS Storage Pool” on page 69
- “Using Files in a ZFS Storage Pool” on page 70

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
requires 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)

<table>
<thead>
<tr>
<th>Part</th>
<th>Tag</th>
<th>Flag</th>
<th>First Sector</th>
<th>Size</th>
<th>Last Sector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>usr</td>
<td>wm</td>
<td>34</td>
<td>136.72GB</td>
<td>286722911</td>
</tr>
<tr>
<td>1</td>
<td>unassigned</td>
<td>wm</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>unassigned</td>
<td>wm</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>unassigned</td>
<td>wm</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>unassigned</td>
<td>wm</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>unassigned</td>
<td>wm</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>unassigned</td>
<td>wm</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>reserved</td>
<td>wm</td>
<td>286722912</td>
<td>8.00MB</td>
<td>286739295</td>
</tr>
</tbody>
</table>

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:


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 c1t1d0
partition> p
Current partition table (original):
Total disk cylinders available: 14087 + 2 (reserved cylinders)
   Part  Tag  Flag Cylinders  Size     Blocks
        0  root  wm  0 - 14086  68.35GB (14087/0/0) 143349312
        1 unassigned  wm  0  0 (0/0/0) 0
        2 backup  wm  0 - 14086  68.35GB (14087/0/0) 143349312
        3 unassigned  wm  0  0 (0/0/0) 0
        4 unassigned  wm  0  0 (0/0/0) 0
        5 unassigned  wm  0  0 (0/0/0) 0
        6 unassigned  wm  0  0 (0/0/0) 0
        7 unassigned  wm  0  0 (0/0/0) 0
```

On an x86 based system, a 72-GB disk has 68 GB of usable disk space located in slice 0, as shown in the following format output. A small amount of boot information is contained in slice 8. Slice 8 requires no administration and cannot be changed.
An fdisk partition also exists on Solaris x86 systems. An fdisk partition is represented by a /dev/dsk/cN[tN]dNpN device name and acts as a container for the disk's available slices. Do not use a cN[tN]dNpN device for a ZFS storage pool component because this configuration is neither tested nor supported.

Using Files in a ZFS Storage Pool

ZFS also allows you to use 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 71
- “RAID-Z Storage Pool Configuration” on page 71
- “Self-Healing Data in a Redundant Configuration” on page 72
- “Dynamic Striping in a Storage Pool” on page 72
- “ZFS Hybrid Storage Pool” on page 72
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 74.

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 \((N-P) \times X\) 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:

\[
\text{raidz c1t0d0 c2t0d0 c3t0d0 c4t0d0 c5t0d0 c6t0d0 c7t0d0 raidz c8t0d0 c9t0d0 c10t0d0 c11t0d0 c12t0d0 c13t0d0 c14t0d0}
\]

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 75.

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:


**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 73
- “Displaying Storage Pool Virtual Device Information” on page 78
- “Handling ZFS Storage Pool Creation Errors” on page 79
- “Destroying ZFS Storage Pools” on page 81

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 55.

Creating a Basic Storage Pool

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

```bash
# 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:


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 82.
- 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 87.
- 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 94.
- 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 87.
- 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 89.

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, "Managing ZFS Root Pool Components."

## Creating a RAID-Z Storage Pool

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

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

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

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

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

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

errors: No known data errors
```
# zpool create tank raidz3 c0t0d0 c1t0d0 c2t0d0 c3t0d0 c4t0d0 c5t0d0 c6t0d0 c7t0d0
# zpool status -v tank
pool: tank
state: ONLINE
scrub: none requested
config:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>raidz3-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c0t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c3t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c4t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c5t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c6t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c7t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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 82.
- 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 94.

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

- Attaching an additional disk to an existing RAID-Z configuration.
- Detaching a disk from a RAID-Z configuration, except when you are detaching a disk that is replaced by a spare disk. Or, when you need to detach a spare disk.
- You cannot outright remove a device that is not a log 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 71.

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 39.

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:
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 provides data redundancy as mirror or RAID-Z virtual devices. These virtual devices consist of disks, disk slices, or files. A spare is a special virtual dev that tracks available hot spares for a pool.

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

```shell
# 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:

```shell
# 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:

```shell
# 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:

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

  NAME   STATE  READ  WRITE  CKSUM
   tank   ONLINE  0      0      0
```
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 `umount` command.

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

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

- **Part of a ZFS pool**
  - The disk or file is part of an active ZFS storage pool. To correct this error, use the `zpool destroy` command to destroy the other pool, if it is no longer needed. Or, use the `zpool detach` command to detach the disk from the other pool. You can only detach a disk from a mirrored storage pool.

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

The disk contains a known file system, though it is not mounted and doesn't appear to be in use.

Part of volume

The disk is part of a Solaris Volume Manager volume.

Part of exported ZFS pool

The disk is part of a storage pool that has been exported or manually removed from a system. In the latter case, the pool is reported as potentially active, as the disk might or might not be a network-attached drive in use by another system. Be cautious when overriding a potentially active pool.

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

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

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

Mismatched Replication Levels

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

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

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

Doing a Dry Run of Storage Pool Creation

Attempts to create a pool can fail unexpectedly in different ways, and formatting disks is a potentially harmful action. For these reasons, the zpool create command has an additional option, -n, which simulates creating the pool without actually writing to the device. This dry run option performs the device in-use checking and replication-level validation, and reports any errors in the process. If no errors are found, you see output similar to the following:
# zpool create -n tank mirror c1t0d0 c1t1d0
would create 'tank' with the following layout:

```
tank
    mirror
    c1t0d0
    c1t1d0
```

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

## Default Mount Point for Storage Pools

When a pool is created, the default mount point for the top-level 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 174.

## 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 121.
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 111.

For more information about importing pools, see "Importing ZFS Storage Pools" on page 118.

Managing Devices in ZFS Storage Pools

Most of the basic information regarding devices is covered in “Components of a ZFS Storage Pool” on page 67. 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 82
- “Attaching and Detaching Devices in a Storage Pool” on page 87
- “Creating a New Pool By Splitting a Mirrored ZFS Storage Pool” on page 89
- “Onlining and Offlining Devices in a Storage Pool” on page 92
- “Clearing Storage Pool Device Errors” on page 94
- “Replacing Devices in a Storage Pool” on page 94
- “Designating Hot Spares in Your Storage Pool” on page 96

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 79.

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

Managing Devices in ZFS Storage Pools
Managing Devices in ZFS Storage Pools

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c0t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-1</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c0t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-2</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c0t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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
cfg:
<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>rzpool</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>raidz1-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t4d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

errors: No known data errors
```

```
# zpool add rzpool raidz c2t2d0 c2t3d0 c2t4d0
# zpool status rzpool
pool: rzpool
state: ONLINE
scrub: none requested
cfg:
<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>rzpool</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>raidz1-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t4d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>raidz1-1</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t4d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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 39.

```
# 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
c@t4d0 ONLINE 0  0   0
c@t5d0 ONLINE 0  0   0
errors: No known data errors
# zpool add newpool log mirror c@t6d0 c@t7d0
# 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
c@t4d0 ONLINE 0  0   0
c@t5d0 ONLINE 0  0   0
  logs
    mirror-1 ONLINE 0  0   0
c@t6d0 ONLINE 0  0   0
c@t7d0 ONLINE 0  0   0
errors: No known data errors
```

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

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:

```
# 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
c@t4d0 ONLINE 0  0   0
```
EXAMPLE 4-3  Adding and Removing a Mirrored Log Device  (Continued)

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       ONLINE 0 0 0
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
c2t3d0    ONLINE 0 0 0
cache     ONLINE 0 0 0
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:
EXAMPLE 4–4  Adding and Removing Cache Devices

(Continued)

# zpool remove tank c2t5d0 c2t8d0
# zpool status tank

pool: tank
state: ONLINE
scrub: none requested

config:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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 77.

Attaching and Detaching Devices in a Storage Pool

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

If you are attaching a disk to create a mirrored root pool, see “How to Configure a Mirrored Root Pool” on page 130.

If you are replacing a disk in a ZFS root pool, see “How to Replace a Disk in the ZFS Root Pool” on page 138.

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

In this example, zeepool 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 zeepool

pool: zeepool
state: ONLINE
scrub: none requested

config:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeepool</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
EXAMPLE 4–5  Converting a Two-Way Mirrored Storage Pool to a Three-way Mirrored Storage Pool
(Continued)

```
mirror-0 ONLINE 0 0 0
c0t1d0 ONLINE 0 0 0
c1t1d0 ONLINE 0 0 0
```

errors: No known data errors

```
# zpool attach zeepool c1t1d0 c2t1d0
# zpool status zeepool
  pool: zeepool
  state: ONLINE
  scrub: resilver completed after 0h0m with 0 errors on Fri Jan 8 12:59:20 2010
  config:

  NAME     STATE    READ  WRITE CKSUM
  zeepool  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
  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
```

Managing Devices in ZFS Storage Pools
EXAMPLE 4–6  Converting a Nonredundant ZFS Storage Pool to a Mirrored ZFS Storage Pool

(Continued)

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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

errors: No known data errors
```

```
# zpool split tank tank2
# zpool import tank2
# zpool status tank tank2
pool: tank
```
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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>trinity</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

events: No known data errors
```

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>neo</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

events: No known data errors
```

```
pool: trinity
state: ONLINE
scrub: none requested
config:
```
Onlining and Offlining Devices in a Storage Pool

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

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

Taking a Device Offline

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

```
# zpool offline tank c1t0d0
```

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
```

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 94.

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 105.

For more information on device health, see "Determining the Health Status of ZFS Storage Pools" on page 111.

**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
PLATFORM: SUNW,Sun-Fire-880, CSN: -, HOSTNAME: neo
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: 504a1188-b270-4ab0-af4e-8a77680576b8
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 272.

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

```
# zpool online -e tank c1t1d0
```

### 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 276.

### 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 138.

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 replacement LUN is expanded to its full size when the disk is added to the pool. By default, the `autoexpand` property is disabled. You can enable this property to expand LUN size before or after the larger LUN is added to the pool.

In the following example, two 16-GB disks in a mirrored pool are replaced with two 72-GB disks. The `autoexpand` property is enabled after the disk replacements to expand the full LUN 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
```
Replacing many disks in a large pool is time-consuming due to resilvering the data onto the new disks. In addition, you might consider running the `zpool scrub` command between disk replacements to ensure that the replacement devices are operational and that the data is written correctly.

If a failed disk has been replaced automatically with a hot spare, then you might need to detach the spare after the failed disk is replaced. You can use the `zpool detach` command to detach a spare in a mirrored or RAIDZ pool. For information about detaching a hot spare, see "Activating and Deactivating Hot Spares in Your Storage Pool" on page 98.

For more information about replacing devices, see "Resolving a Missing Device" on page 272 and "Replacing or Repairing a Damaged Device" on page 274.

**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:

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>trinity</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
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
c1t3d0 ONLINE   0      0       0
c2t3d0 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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeepool</td>
<td>DEGRADED</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>DEGRADED</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spare-1</td>
<td>DEGRADED</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t1d0</td>
<td>UNAVAIL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Managing Devices in ZFS Storage Pools

---

Managing Devices in ZFS Storage Pools

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeepool</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spare-1</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

errors: No known data errors
```

Then, detach the disk c2t1d0.

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeepool</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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 zeepool c2t1d0
# zpool status zeepool
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeepool</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spare-1</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0 90K resilvered</td>
</tr>
<tr>
<td>c2t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spares</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c2t3d0</td>
<td>AVAIL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeepool</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spares</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c2t3d0</td>
<td>AVAIL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeepool</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t3d0</td>
<td>AVAIL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

errors: No known data errors
EXAMPLE 4–10  Detaching a Failed Disk and Using the Hot Spare  (Continued)

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-8000-20

scrub: resilver in progress for 0h0m, 70.47% done, 0h0m to go

config:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeepool</td>
<td>DEGRADE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>DEGRADE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spare-1</td>
<td>DEGRADE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t1d0</td>
<td>UNAVAIL</td>
<td>0</td>
<td>0</td>
<td>cannot open</td>
</tr>
<tr>
<td>c2t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spares</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c2t3d0</td>
<td>INUSE</td>
<td></td>
<td></td>
<td>currently in use</td>
</tr>
</tbody>
</table>

errors: No known data errors

# zpool detach zeepool c2t1d0

# zpool status zeepool

pool: zeepool
state: ONLINE
scrub: resilver completed after 0h0m with 0 errors on Wed Jan 20 13:46:46 2010

config:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeepool</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeepool</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spares</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c2t1d0</td>
<td>AVAIL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

errors: No known data errors
Managing ZFS Storage Pool Properties

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

```
# zpool get all zeepool
NAME PROPERTY VALUE SOURCE
zeepool size 33.8G -
zeepool capacity 0% -
zeepool altroot - default
zeepool health ONLINE -
zeepool guid 858873752016230819 default
zeepool version 31 default
zeepool bootfs - default
zeepool delegation on default
zeepool autoreplace off default
zeepool cachefile - default
zeepool failmode wait default
zeepool listsnapshots off default
zeepool autoexpand off default
zeepool dedupditto 0 default
zeepool dedupratio 1.00x -
zeepool free 33.7G -
zeepool allocated 104K -
zeepool readonly off -
```

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

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

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocated</td>
<td>String</td>
<td>N/A</td>
<td>Read-only value that identifies the amount of storage space within the pool that has been physically allocated.</td>
</tr>
<tr>
<td>altroot</td>
<td>String</td>
<td>off</td>
<td>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.</td>
</tr>
<tr>
<td>autoreplace</td>
<td>Boolean</td>
<td>off</td>
<td>Controls automatic device replacement. If set to off, device replacement must be initiated by using the <code>zpool replace</code> command. If set to on, any new device found in the same physical location as a device that previously belonged to the pool is automatically formatted and replaced. The property abbreviation is <code>replace</code>.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Type</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>bootfs</td>
<td>Boolean</td>
<td>N/A</td>
<td>Identifies the default bootable dataset for the root pool. This property is typically set by the installation and upgrade programs.</td>
</tr>
<tr>
<td>cachefile</td>
<td>String</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>capacity</td>
<td>Number</td>
<td>N/A</td>
<td>Read-only value that identifies the percentage of pool space used. The property abbreviation is <code>cap</code>.</td>
</tr>
<tr>
<td>dedupditto</td>
<td>String</td>
<td>N/A</td>
<td>Sets a threshold, and if the reference count for a deduped block goes above the threshold, another ditto copy of the block is stored automatically.</td>
</tr>
<tr>
<td>dedupratio</td>
<td>String</td>
<td>N/A</td>
<td>Read-only deduplication ratio achieved for a pool, expressed as a multiplier.</td>
</tr>
<tr>
<td>delegation</td>
<td>Boolean</td>
<td>on</td>
<td>Controls whether a nonprivileged user can be granted access permissions that are defined for a dataset. For more information, see Chapter 9, &quot;Oracle Solaris ZFS Delegated Administration.&quot;</td>
</tr>
</tbody>
</table>
### TABLE 4-1 ZFS Pool Property Descriptions (Continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| failmode      | String  | wait          | Controls the system behavior if a catastrophic pool failure occurs. This condition is typically a result of a loss of connectivity to the underlying storage device or devices or a failure of all devices within the pool. The behavior of such an event is determined by one of the following values:  
  ■ **wait** – Blocks all I/O requests to the pool until device connectivity is restored, and the errors are cleared by using the `zpool clear` command. In this state, I/O operations to the pool are blocked, but read operations might succeed. A pool remains in the wait state until the device issue is resolved.  
  ■ **continue** – 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 `zpool clear` command.  
  ■ **panic** – Prints a message to the console and generates a system crash dump. |
| free          | String  | N/A           | Read-only value that identifies the number of blocks within the pool that are not allocated. |
| guid          | String  | N/A           | Read-only property that identifies the unique identifier for the pool. |
| health        | String  | N/A           | Read-only property that identifies the current health of the pool, as either ONLINE, DEGRADED, FAULTED, OFFLINE, REMOVED, or UNAVAIL. |
| listsnapshots | String  | off           | Controls whether snapshot information that is associated with this pool is displayed with the `zfs list` command. If this property is disabled, snapshot information can be displayed with the `zfs list -t snapshot` command. |
| readonly      | Boolean | off           | Identifies whether a pool can be modified. This property is only enabled when a pool is has been imported in read-only mode. If enabled, any synchronous data that exists only in the intent log will not be accessible until the pool is re-imported in read-write mode. |
| size          | Number  | N/A           | Read-only property that identifies the total size of the storage pool. |
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 105
- “Viewing I/O Statistics for ZFS Storage Pools ” on page 109
- “Determining the Health Status of ZFS Storage Pools” on page 111

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 64.
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 111.

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 261.

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 ALTROOT
tank     80.0G  22.3G  47.7G  28%  ONLINE -
```

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

```
# zpool list -T d 3 2
Tue Nov  2 10:36:11 MDT 2010
NAME     SIZE  ALLOC  FREE  CAP  HEALTH ALTROOT
pool     33.8G  83.5K  33.7G  0%  1.00x  ONLINE -
rpool   33.8G  12.2G  21.5G  36%  1.00x  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 105.
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 cammount=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 cammount=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 cammount=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 11:10 zpool set bootfs=rpool rpool
2010-05-11 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
2010-05-14.08:10:29 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]
2010-05-14.08:10:29 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
History for 'tank':
2010-05-13.14:13:15 zpool create -f tank mirror c1t2d0 c1t3d0
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:

- **alloc capacity**  The amount of data currently stored in the pool or device. This amount differs from the amount of disk space available to actual file systems by a small margin due to internal implementation details.

  For more information about the differences between pool space and dataset space, see “ZFS Disk Space Accounting” on page 64.

- **free capacity**  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.

- **read operations**  The number of read I/O operations sent to the pool or device, including metadata requests.

- **write operations**  The number of write I/O operations sent to the pool or device.

- **read bandwidth**  The bandwidth of all read operations (including metadata), expressed as units per second.

- **write bandwidth**  The bandwidth of all write operations, expressed as units per second.

Listing Pool-Wide I/O Statistics

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

```
# zpool iostat

pool | capacity  | operations | bandwidth
-----|----------|------------|-----------
     | 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

pool | capacity  | operations | bandwidth
-----|----------|------------|-----------
     | 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  
```
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:

```bash
# zpool iostat -v

<table>
<thead>
<tr>
<th>pool</th>
<th>alloc</th>
<th>free</th>
<th>read</th>
<th>write</th>
<th>read</th>
<th>write</th>
</tr>
</thead>
<tbody>
<tr>
<td>rpool</td>
<td>6.05G</td>
<td>61.9G</td>
<td>0</td>
<td>0</td>
<td>785</td>
<td>107</td>
</tr>
<tr>
<td>mirror</td>
<td>6.05G</td>
<td>61.9G</td>
<td>0</td>
<td>0</td>
<td>785</td>
<td>107</td>
</tr>
<tr>
<td>c1t0d0s0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>578</td>
<td>109</td>
</tr>
<tr>
<td>c1t1d0s0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>595</td>
<td>109</td>
</tr>
<tr>
<td>tank</td>
<td>36.5G</td>
<td>31.5G</td>
<td>4</td>
<td>1</td>
<td>295K</td>
<td>146K</td>
</tr>
<tr>
<td>mirror</td>
<td>36.5G</td>
<td>31.5G</td>
<td>126</td>
<td>45</td>
<td>8.13M</td>
<td>4.01M</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>3</td>
<td>100K</td>
<td>386K</td>
</tr>
<tr>
<td>c1t3d0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>3</td>
<td>104K</td>
<td>386K</td>
</tr>
</tbody>
</table>
```

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>DEGRADE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>DEGRADE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t1d0</td>
<td>OFFLINE</td>
<td>0</td>
<td>0</td>
<td>40K</td>
</tr>
</tbody>
</table>

**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.

The **zpool status** command displays the following scrub and resilver information:

- **Resilver in-progress report.** For example:
  
  ```
  scan: resilver in progress since Thu Oct 28 14:08:04 2010  
  24.3M scanned out of 12.2G at 3.03M/s, 1h5m to go  
  24.0M resilvered, 0.19% done
  ```

- **Resilver completion message.** For example:
  
  ```
  scan: resilvered 12.2G in 0h14m with 0 errors on Thu Oct 28 14:55:57 2010
  ```

- **Scrub in-progress report.** For example:
  
  ```
  scan: scrub in progress since Thu Oct 28 13:50:22 2010  
  98.6M scanned out of 13.3G at 16.4M/s, 0h1m to go  
  0 repaired, 0.72% done
  ```

- **Scrub completion message.** For example:
  
  ```
  scan: scrub repaired 512B in 1h2m with 0 errors on Thu Oct 28 14:52:22 2010
  ```

For more information about diagnosing and repairing faulted pools and data, see Chapter 11, “Oracle Solaris ZFS Troubleshooting and Pool Recovery.”
Gathering ZFS Storage Pool Status Information

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

```
# zpool status -Td 3 2
zpool status -Td 3 2
Tue Nov  2 10:38:18 MDT 2010
pool: pool
  state: ONLINE
  scan: none requested
config:

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

errors: No known data errors

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

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

errors: No known data errors
```

Querying ZFS Storage Pool Status
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 207.

- “Preparing for ZFS Storage Pool Migration” on page 115
- “Exporting a ZFS Storage Pool” on page 115
- “Determining Available Storage Pools to Import” on page 116
- “Importing ZFS Storage Pools From Alternate Directories” on page 118
- “Importing ZFS Storage Pools” on page 118
- “Recovering Destroyed ZFS Storage Pools” on page 121

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:
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: 11009215114195894163
  state: ONLINE
  action: The pool can be imported using its name or numeric identifier.
  config:
    tank ONLINE
    mirror-0 ONLINE
    clt0d0 ONLINE
    clt1d0 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:

```bash
# 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-8000-20
config:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>DEGRADE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>DEGRADE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t0d0</td>
<td>UNAVAIL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

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:

```bash
# 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:

<table>
<thead>
<tr>
<th>RAID-Z</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>raidz1-0</td>
<td>FAULTED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c1t0d0</td>
<td>ONLINE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c1t1d0</td>
<td>FAULTED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c1t3d0</td>
<td>FAULTED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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:

```bash
# 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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>dozer</td>
<td>FAULTED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>raidz1-0</td>
<td>ONLINE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
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:

```bash
# 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
```

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:

```bash
# 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:

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

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

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

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

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

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

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

Pools can also be imported under an alternate root by using the `-R` option. For more information on alternate root pools, see “Using ZFS Alternate Root Pools” on page 261.

### Importing a Pool With a Missing Log Device

You can import a pool with a missing log device. If a pool has a missing log device, then it will not import. For example:

```
zpool import dozer
```
The devices below are missing, use '-m' to import the pool anyway:
c3t3d0 [log]
cannot import 'dozer': one or more devices is currently unavailable
Import the pool with the missing log device. For example:

```
# zpool import -m dozer
# zpool status dozer

pool: dozer
state: DEGRADED
status: One or more devices could not be opened. Sufficient replicas exist for the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
see: http://www.sun.com/msg/ZFS-8000-2Q
scan: scrub repaired 0 in 0h0m with 0 errors on Fri Oct 15 16:43:03 2010
config:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>dozer</td>
<td>DEGRADE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c3t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c3t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>logs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14685044587769991702</td>
<td>UNAVAIL</td>
<td>0</td>
<td>0</td>
<td>0 was c3t3d0</td>
</tr>
</tbody>
</table>
```

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

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

```
# zpool import dozer
The devices below are missing, use '-m' to import the pool anyway:
mirror-1 [log]
c3t3d0
c3t4d0

cannot import 'dozer': one or more devices is currently unavailable
# zpool import -m dozer
# zpool status dozer

pool: dozer
state: DEGRADED
status: One or more devices could not be opened. Sufficient replicas exist for the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
see: http://www.sun.com/msg/ZFS-8000-2Q
scan: scrub repaired 0 in 0h0m with 0 errors on Fri Oct 15 16:51:39 2010
config:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>dozer</td>
<td>DEGRADE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c3t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c3t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>logs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mirror-1</td>
<td>UNAVAIL</td>
<td>0</td>
<td>0</td>
<td>0 insufficient replicas</td>
</tr>
<tr>
<td>1351406142645294202</td>
<td>UNAVAIL</td>
<td>0</td>
<td>0</td>
<td>0 was c3t3d0</td>
</tr>
<tr>
<td>16839344638582008929</td>
<td>UNAVAIL</td>
<td>0</td>
<td>0</td>
<td>0 was c3t4d0</td>
</tr>
</tbody>
</table>
```

After attaching the missing log devices, run the `zpool clear` command to clear the pool errors.
Importing a Pool in Read-Only Mode

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

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

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

- All file systems and volumes are mounted in read-only mode.
- Pool transaction processing is disabled. This also means that any pending synchronous writes in the intent log are not played until the pool is imported read-write.
- Attempts to set a pool property during the read-only import are ignored.

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

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

Importing a Pool By a Specific Device Path

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

```
# zpool import -d /dev/dsk/c2t3d0s0 dpool
# zpool status dpool
  pool: dpool  state: ONLINE
  scan: resilvered 952K in 0h0m with 0 errors on Thu Mar 10 10:28:46 2011
  config:
      NAME   STATE    READ  WRITE  CKSUM
      dpool  ONLINE  0   0       0
      mirror-0 ONLINE  0   0       0
      c2t3d0 ONLINE  0   0       0
      c2t1d0 ONLINE  0   0       0
```

An existing limitation is that even though this pool is comprised of whole disks, the command must include the specific device’s slice identifier.

Recovering Destroyed ZFS Storage Pools

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

pool: tank
id: 5154272182900538157
state: ONLINE (DESTROYED)
action: The pool can be imported using its name or numeric identifier.

config:

<table>
<thead>
<tr>
<th>tank</th>
<th>ONLINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
</tr>
<tr>
<td>c1t0d0</td>
<td>ONLINE</td>
</tr>
<tr>
<td>c1t1d0</td>
<td>ONLINE</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>ONLINE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c1t0d0</td>
<td>ONLINE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c1t1d0</td>
<td>ONLINE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>dozer</td>
<td>DEGRADED</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>raidz2-0</td>
<td>DEGRADED</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t8d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Upgrading ZFS Storage Pools

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

```
# zpool status
```

This displays information about the pool, including its state and any issues such as degraded state due to missing devices. The `zpool upgrade` command can be used to update the pool:

```
# zpool upgrade
```

This will upgrade the pool to the current version. If the pool is still usable, the upgrade will proceed without losing data.

For example:

```
UpgradingZFSStoragePools
Chapter4 • ManagingOracleSolarisZFSStoragePools 123
```
You can use the following syntax to identify additional information about a particular version and supported releases:

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

```
# zpool upgrade -a
```

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

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

The following sections are provided in this chapter:

- “Managing ZFS Root Pool Components (Overview)” on page 125
- “Oracle Solaris 11 Express Installation Requirements for ZFS Support” on page 126
- “Managing Your ZFS Root Pool” on page 128
- “Managing Your ZFS Swap and Dump Devices” on page 131
- “Booting From a ZFS Root File System” on page 134
- “Recovering the ZFS Root Pool or Root Pool Snapshots” on page 138

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


Managing ZFS Root Pool Components (Overview)

ZFS is the default root file system in the Oracle Solaris 11 Express releases. In the Oracle Solaris 11 Express release, you can install and boot from a ZFS root file system in the following ways:

- Oracle Solaris 11 Express Live CD – Installs a ZFS root pool on an x86 based system on a single disk. You can use the fdisk partition menu during the installation to partition the disk for your environment.
- Oracle Solaris 11 Express Automated Installer (AI) – Automatically installs a ZFS root pool on a SPARC based or x86 based system. You can use an AI manifest to determine the disk and the disk partitions to be used for the ZFS root pool.

Swap and dump devices are automatically created on ZFS volumes in the ZFS root pool by both of the above installation methods. For more information about managing ZFS swap and dump devices, see “Managing Your ZFS Swap and Dump Devices” on page 131.
You cannot configure a mirrored root pool during an Oracle Solaris 11 Express installation. For more information about configuring a mirrored root pool, see “How to Configure a Mirrored Root Pool” on page 130.

Oracle Solaris 11 Express Installation Requirements for ZFS Support

Review the following Oracle Solaris 11 Express installation requirement sections.

Oracle Solaris 11 Express Release Requirements

ZFS is the default root file system for all Oracle Solaris 11 Express releases. In addition, the following features are available:

- Oracle Solaris 11 Express Live CD is available for x86 systems only
- Oracle Solaris 11 Express provides the Automated Installer features for SPARC based and x86 based systems

General ZFS Storage Pool Requirements

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

ZFS Storage Pool Space Requirements

When a system is installed, 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 ZFS storage pool space requirements:

- 1 GB of memory is recommended to install a ZFS root file system and for overall better ZFS performance
- At least 13 GBs of disk space is recommended. The space is consumed as follows:
  - **Swap area and dump device** – The default sizes of the swap and dump volumes that are created by the Solaris installation programs are as follows:
    - The default swap volume size is calculated as half the size of physical memory
    - The default dump volume size is calculated by the kernel based on *dumpadm* information and the size of physical memory

After installation, you can adjust the sizes of your swap and dump volumes to sizes of your choosing as long as the new sizes support system operation. For more information, see "Adjusting the Sizes of Your ZFS Swap and Dump Devices" on page 132.
■ **Boot environment (BE)** – A ZFS BE is approximately 4-6 GB. Swap and dump volumes are determined by the amount of physical memory on the system. For example, 8 GB swap and dump devices are created on a system with 16 GBs of memory. Each ZFS BE that is cloned from another ZFS BE doesn’t need additional disk space unless it is patched or upgraded. Consider that BE size will increase when patches are applied or the BE is upgraded. 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.

### ZFS Storage Pool Configuration Requirements

Review the following ZFS storage pool configuration requirements:

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

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

  For a detailed description of supported ZFS root pool configurations, see “Creating a ZFS Root Pool” on page 74.
- On an x86 based system, the disk must contain a Solaris `fdisk` partition. A Solaris `fdisk` partition is created automatically when the x86 based system is installed. For more information about Solaris `fdisk` partitions, see “Guidelines for Creating an fdisk Partition” in *System Administration Guide: Devices and File 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. Renaming the root pool might cause an unbootable system.
Managing Your ZFS Root Pool

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

Installing a ZFS Root Pool

The Oracle Solaris 11 Express Live CD installation method installs a default ZFS root pool on a single disk. With the Oracle Solaris 11 Express automated installation (AI) method, you can create an AI manifest with the `<ai_target_device>` tag to identify the disk that is used to install the ZFS root pool. If you do not identify a target disk for the root pool, the default target disk is selected as follows:

- The installer searches for a disk based on a recommended size of approximately 13 GB
- The disks are searched based on an order determined by the `libdiskmgt` library
- The installer selects the first disk that matches the recommended size
- If no disk matches the recommended size, the automated installation fails

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

Also keep in mind that the disk intended for the root pool must have an SMI label. Otherwise, the installation will fail.

Similar to the Oracle Solaris 11 Express Live CD installation method, you can only install a root pool onto one disk with the automated installer. See the next section for configuring a mirrored root pool.

After the installation, review your ZFS storage pool and file system information. For example:

```
# zpool status
pool: rpool
  state: ONLINE
  requested
config:
    NAME    STATE  READ  WRITE  CKSUM
    rpool   ONLINE 0 0 0
    c1t3d0s0 ONLINE 0 0 0
errors: No known data errors
# zfs list
# rpool 22.9G 111G 76.5K /rpool
rpool/ROOT 6.80G 111G 31K legacy
rpool/ROOT/solaris 6.80G 111G 5.20G /
```
Review your ZFS BE information. For example:

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

In the above output, NR means *now running*.

### How to Update Your ZFS Boot Environment

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

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

In the above output, NR means *now running*.

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

1. **Update your ZFS BE.**

   ```bash
   # pkg update
   Download PKGS FILES XFER (MB)
   Completed 707/707 10529/10529 194.9/194.9
   ```

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

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

   ```bash
   # init 6
   ```

   ```bash
   # beadm list
   BE    Active Mountpoint Space Policy Created
   --    ------ ---------- ------ -------
   ```
How to Configure a Mirrored Root Pool

You cannot configure a mirrored root pool with any of the Oracle Solaris 11 Express installation methods, but you can easily configure a mirrored root pool 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 138.

1 Display your current root pool status.

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

   NAME      STATE READ WRITE CKSUM
rpool      ONLINE 0  0  0
  c1t3d0s0  ONLINE 0  0  0
errors: No known data errors
```

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

```bash
# zpool attach rpool c1t3d0s0 c1t2d0s0
```

Make sure to wait until resilver is done before rebooting.

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

```bash
# zpool status rpool
pool: rpool
  state: ONLINE
  status: One or more devices is currently being resilvered. The pool will continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
scan: resilver in progress since Thu Jan 13 15:54:54 2011
  2.16G scanned out of 14.8G at 71.3M/s, 0h3m to go
  2.16G resilvered, 14.59% done
config:

   NAME      STATE READ WRITE CKSUM
rpool      ONLINE 0  0  0
  mirror-0  ONLINE 0  0  0
  c1t3d0s0  ONLINE 0  0  0
  c1t2d0s0  ONLINE 0  0  0 (resilvering)
errors: No known data errors
```

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

```bash
scrub: resilver completed after 0h10m with 0 errors on Thu Mar 11 11:27:22 2010
```
Verify that you can boot successfully from the second disk.

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.

### Managing Your ZFS Swap and Dump Devices

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

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

During the installation process, a dump device is created on a ZFS volume in the ZFS root pool. In general, a dump device requires no administration because it is setup automatically at installation time. For example:

```bash
# dumpadm
Dump content: kernel pages
Dump device: /dev/zvol/dsk/rpool/dump (dedicated)
Savecore directory: /var/crash/t5120
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 11 Express Installation Requirements for ZFS Support” on page 126.

Both the swap volume size and the dump volume size can be adjusted after installation. For more information, see “Adjusting the Sizes of Your ZFS Swap and Dump Devices” on page 132.

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

- Separate ZFS volumes must be used for the swap area and dump devices.
- Currently, using a swap file on a ZFS file system is not supported.
- If you need to change your swap area or dump device after the system is installed or upgraded, use the `swap` and `dumpadm` commands as in previous Solaris releases. For more information, see Chapter 21, “Configuring Additional Swap Space (Tasks),” in System Administration Guide: Devices and File Systems and Chapter 13, “Managing System Crash Information (Tasks),” in System Administration Guide: Common System Management Tasks.
Adjusting the Sizes of Your ZFS Swap and Dump Devices

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

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

  ```
  # zfs set vsize=2G rpool/dump
  # zfs get vsize rpool/dump
  NAME     PROPERTY   VALUE   SOURCE
  rpool/dump vsize    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 vsize=2G rpool/swap
  # swap -a /dev/zvol/dsk/rpool/swap
  ```

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


- 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
```

Add an entry for the second swap volume to the `/etc/vfstab` file.

- Select one of the following if you need to recreate your swap area:
  - On a SPARC based system, create your swap area. Set the block size to 8 KB.
    ```
    # zfs create -V 2G -b 8k rpool/swap
    ```
  - On an x86 based system, create your swap area. Set the block size to 4 KB.
    ```
    # zfs create -V 2G -b 4k rpool/swap
    ```
  - You must enable the swap area when a new swap device is added or changed.
  - Add an entry for the swap volume to the `/etc/vfstab` file.
Troubleshooting ZFS Dump Device Issues

Review the following items if you have problems either capturing a system crash dump or resizing the dump device.

- If a crash dump was not created automatically, you can use the `savecore` command to save the crash dump.

- A dump device is created automatically when you initially install a ZFS root file system or migrate to a ZFS root file system. In most cases, you will only need to adjust the size of the dump device if the default dump device size is too small. For example, on a large-memory system, the dump device size is increased to 40 GB as follows:

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

  Resizing a large dump device can be a time-consuming process.

  If, for any reason, you need to enable a dump device after you create a dump device manually, use syntax similar to the following:

  ```bash
  # dumpadm -d /dev/zvol/dsk/rpool/dump
  ```

  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:

  ```bash
  # dumpadm -d /dev/zvol/dsk/rpool/dump
  Error: dumpadm: dump device /dev/zvol/dsk/rpool/dump is too small to hold a system dump
  ```

  For information on 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:

  ```bash
  # dumpadm -d /dev/zvol/dsk/datapool/dump
  Error: dumpadm: dump is not supported on device '/dev/zvol/dsk/datapool/dump': `datapool’ has multiple top level vdevs
  ```

  Add the dump device to the root pool, which cannot have multiple top-level devices.
Booting From a ZFS Root File System

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

Booting from a ZFS file system differs from booting from a UFS file system because with ZFS, a device specifier identifies a storage pool, not a single root file system. A storage pool can contain multiple bootable ZFS root file systems. When booting from ZFS, you must specify a boot device and a root file system within the pool that was identified by the boot device.

By default, the dataset selected for booting is the one identified by the pool's bootfs property. This default selection can be overridden by specifying an alternate bootable dataset that is included in the boot -Z command on a SPARC system or by selecting an alternate boot device from the BIOS on an x86 based system.

Booting From an Alternate Disk in a Mirrored ZFS Root Pool

You can attach a disk to create a mirrored ZFS root pool after installation. For more information about creating a mirrored root pool, see “How to Configure a Mirrored Root Pool” on page 130.

Review the following known issues regarding mirrored ZFS root pools:

- If you replace a root pool disk by using the zpool replace command, you must install the boot information on the newly replaced disk by using the installboot or installgrub commands. If you create a mirrored ZFS root pool with the initial installation method or if you use the zpool attach command to attach a disk to the root pool, then this step is unnecessary. The installboot or installgrub command is as follows:

  ```
  sparc# installboot -F zfs /usr/platform/’uname -i’/lib/fs/zfs/bootblk /dev/rdsk/c0t1d0s0
  x86# installgrub /boot/grub/stage1 /boot/grub/stage2 /dev/rdsk/c0t1d0s0
  ```

- You can boot from different devices in a mirrored ZFS root pool. Depending on the hardware configuration, you might need to update the PROM or the BIOS to specify a different boot device.

  For example, you can boot from either disk (c1t0d0s0 or c1t1d0s0) in this pool.

  ```
  # zpool status
  pool: rpool
  state: ONLINE
  scrub: none requested
  config:
  ```
On a SPARC based system, enter the alternate disk at the `ok` prompt.

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

After the system is rebooted, confirm the active boot device. For example:

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

On an x86 based system, use syntax similar to the following:

```
x86# prtconf -v|sed -n '/bootpath/,/value/p'
  name='bootpath' type=string items=1
  value='/pci@0,0/pci8086,25f8@4/pci108e,286@0/disk@0,0:a'
```

- On an x86 based system, select an alternate disk in the mirrored ZFS root pool from the appropriate BIOS menu.

## Boooting From a ZFS Root File System on a SPARC Based System

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

During an installation and `beadm` activation process, the ZFS root file system is automatically designated with the `bootfs` property.

Multiple bootable 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, an entry similar to the following is added to the `menu.lst` file:

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

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

On a SPARC based system, two boot options are available:
After a ZFS BE is activated, you can use the boot -L command to display a list of bootable 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.

Use the boot -Z dataset command to boot a specific ZFS dataset.

**EXAMPLE 5-1  Booting From a Specific ZFS Boot Environment**

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

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

```
# beadm list
BE   Active Mountpoint Space Policy Created
------ ---------- ----- ----- -------
solaris - - 19.18M static 2011-01-13 15:31
solaris-1 NR / 8.48G static 2011-01-13 15:44
```

If you have multiple ZFS BEs on your SPARC based system, you can use the `boot -L` command. For example:

```
ok boot -L
Boot device: /pci@0/pci@0/pci@2/scsi@0/disk@3,0:a File and args: -L
1 solaris
2 solaris-1
Select environment to boot: [1 - 2 ]: 2
To boot the selected entry, invoke:
boot [<root-device>] -Z rpool/ROOT/solaris-1
Program terminated
ok boot -Z rpool/ROOT/solaris-1
```

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

The following entries are added to the `/pool-name/boot/grub/menu.lst` file during the installation process or `beadm activate` operation to boot ZFS automatically:

```plaintext
title Oracle Solaris 11 Express snv_152
findroot (pool_rpool,0,a)
bootfs rpool/ROOT/solaris
textkernel$ /platform/86pc/kernel/$ISADIR/unix -B $ZFS-BOOTFS
module$ /platform/86pc/$ISADIR/boot_archive
```

If the device identified by GRUB as the boot device contains a ZFS storage pool, the `menu.lst` file is used to create the GRUB menu.
On an x86 based system with multiple ZFS BEs, you can select a BE from the GRUB menu. If the root file system corresponding to this menu entry is a ZFS dataset, the following option is added.

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

**EXAMPLE 5–2 x86: Booting a ZFS File System**

When booting from a ZFS file system, the root device is specified by the `boot -B $ZFS-BOOTFS` parameter. For example:

```
title Oracle Solaris 11 Express snv_152
findroot (pool_rpool,0,a)
bootfs rpool/ROOT/solaris
kernel$ /platform/i86pc/kernel/$ISADIR/unix -B $ZFS-BOOTFS
module$ /platform/i86pc/$ISADIR/boot_archive
```

**EXAMPLE 5–3 x86: Fast Rebooting a ZFS Root File System**

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

You must still use the `init 6` command when transitioning between BEs with the `beadm activate` command. For other system operations where the `reboot` command is appropriate, you can use the `reboot -f` command. For example:

```
# reboot -f
```

**Booting For Recovery Purposes in a ZFS Root Environment**

Use the following procedure if you need to boot the system so that you can recover from a lost root password or similar problem.

If you need to recover a root pool or root pool snapshot, see “Recovering the ZFS Root Pool or Root Pool Snapshots” on page 138.

**How to Boot ZFS for Recovery Purposes**

Keep multiple boot environments to prevent a system boot failure. Follow these steps if your system won’t boot because of a problem with the BE contents.

1. **Boot from the Oracle Solaris 11 Express Live CD or from an Oracle Solaris 11 Express AI server.**
Recovering the ZFS Root Pool or Root Pool Snapshots

2 At the terminal prompt, import the root pool.
   # zpool import rpool

3 Mount the ZFS BE on /a
   # beadm mount solaris-1 /a

4 Modify the BE contents to resolve the boot failure.
   For example, review the menu.lst file to determine if the boot entries are correct.
   
   # cat /a/rpool/boot/menu.lst
   title solaris
   bootfs rpool/ROOT/solaris
   title solaris-1
   bootfs rpool/ROOT/solaris-1

   If a password or shadow entry is preventing a console login, then correct that problem.

   # cd /a/etc
   # vi passwd

5 If necessary, set the TERM type.
   # TERM=vt100
   # export TERM

6 Update the boot archive
   # bootadm update-archive -R /a

7 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 138
- “How to Create Root Pool Snapshots” on page 140
- “How to Recreate a ZFS Root Pool and Restore Root Pool Snapshots” on page 142

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 it with a larger disk
- The root pool disk is failing. In a non-redundant pool, if the disk is failing so that the system won’t boot, you’ll need to boot from an alternate media, such as a CD or the network, before you replace the root pool disk.
In a mirrored root pool configuration, you might be able to attempt a disk replacement without having to boot from alternate media. You can replace a failed disk by using the `zpool replace` command or if you have an additional disk, you can use the `zpool attach` command. See the steps below for an example of attaching an additional disk and detaching a root pool disk.

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

```bash
# zpool offline rpool c1t0d0s0
# cfgadm -c unconfigure cl::dsk/c1t0d0
# Physically remove failed disk c1t0d0
# Physically insert replacement disk c1t0d0
# cfgadm -c configure cl::dsk/c1t0d0
# Confirm that the new disk has an SMI label and a slice 0
# zpool replace rpool c1t0d0s0
# zpool online rpool c1t0d0s0
# zpool status rpool
# Let disk resilver before installing the boot blocks
```

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

Identify the boot device pathnames of the current and new disk so that you can test booting from the replacement disk and also manually boot from the existing disk, if necessary, if the replacement disk fails. In the example below, the current root pool disk (`c1t10d0s0`) is:

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

In the example below, the replacement boot disk is (`c1t9d0s0`):

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

1. **Physically connect the replacement disk.**

2. **Confirm that the replacement (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:
   

3. **Attach the new disk to the root pool.**
   
   For example:
   ```bash
   # zpool attach rpool c1t10d0s0 c1t9d0s0
   ```
   
   Make sure to wait until resilver is done before rebooting.
4 **Confirm the root pool status.**

For example:

```
# zpool status rpool
state: ONLINE
status: One or more devices is currently being resilvered. The pool will continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
scan: resilver in progress since Fri Jan 14 13:35:45 2011
814M scanned out of 15.3G at 16.3M/s, 0h15m to go
813M resilvered, 5.18% done
config:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>rpool</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t3d0s0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0s0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

errors: No known data errors
```

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

For example, on a SPARC based system:

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

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

For example:

```
# zpool detach rpool c1t1d0s0
```

7 **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

Create root pool snapshots for recovery purposes. The best way to create root pool snapshots is to do a recursive snapshot of the root pool.

The procedure below creates a recursive root pool snapshot and stores the snapshot as a file in a pool on a remote system. In the case of a root pool failure, the remote dataset can be mounted by using NFS and the snapshot file received into the recreated pool. You can also 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 and the most up-to-date information about root pool recovery, go to this site:

Validating remotely stored snapshots as files or snapshots is an important step in root pool recovery and in 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 example, the system is booted from the zfs\texttt{BE} boot environment.

1. **Create space on a remote system to store the snapshots.**
   
   For example:
   ```
   remote\# zfs create rpool/snaps
   ```

2. **Share the space to the local system.**
   
   For example:
   ```
   remote\# zfs set sharenfs='\texttt{rw}=local-system,\texttt{root}=local-system' rpool/snaps
   # share
   @rpool/snaps /rpool/snaps sec=sys,\texttt{rw}=local-system,\texttt{root}=local-system
   ```

3. **Create a recursive snapshot of the root pool.**
   
   In this example, the system has two \texttt{BE}s, \texttt{osolBE} and \texttt{osol2BE}. The active \texttt{BE} is \texttt{osolBE}.
   ```
   local\# zpool set listsnapshots=on rpool
   local\# zfs snapshot -r rpool@0311
   local\# zfs list -r rpool
   ```

4. **Send the root pool snapshots to the remote system.**
   
   For example:
   ```
   local\# zfs send -Rv rpool@0311 > /net/remote-system/rpool/snaps/rpool.0311
   ```
How to Recreate a ZFS Root Pool and Restore Root Pool Snapshots

In this scenario, assume the following conditions:

- ZFS root pool cannot be recovered
- ZFS root pool snapshots are stored on a remote system and are shared over NFS
- The system is booted from an equivalent Solaris release to the root pool version so that the Solaris release and the pool version match. Otherwise, you will need to add the \texttt{-o version=version-number} property option and value when you recreate the root pool in step 4 below.

All steps below are performed on the local system.

1. **Boot from CD/DVD or the network.**
   On a SPARC based system, select one of the following boot methods:
   
   \begin{verbatim}
   ok boot net -s
   ok boot cdrom -s
   \end{verbatim}
   
   If you don’t use \texttt{-s} option, you’ll need to exit the installation program.

   On an x86 based system, select the option for booting from the DVD or the network. Then, exit the installation program.

2. **Mount the remote snapshot dataset.**
   For example:
   
   \begin{verbatim}
   # mount -F nfs remote-system:/rpool/snaps /mnt
   \end{verbatim}
   
   If your network services are not configured, you might need to specify the \texttt{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 will have to relabel the disk.**
   For more information about relabeling the disk, go to the following site:
   
4  Recreate the root pool.
For example:

```
# zpool create -f -o failmode=continue -R /a -m legacy -o cachefile=/etc/zfs/zpool.cache rpool c1t0d0s0
```

5  Restore the root pool snapshots.
This step might take some time. For example:

```
# cat /mnt/rpool.0311 | zfs receive -Fdu rpool
```

Using the `-u` option means that the restored archive is not mounted when the `zfs receive` operation completes.

6  (Optional) If you want to modify something in the BE, you will need to explicitly mount them like this:

a.  Mount the BE components. For example:

```
# zfs mount rpool/ROOT/osolBE
```

b.  Mount everything in the pool that is not part of a BE. For example:

```
# zfs mount -a rpool
```

Other BEs are not mounted since they have `canmount=noauto`, which suppresses mounting when the `zfs mount -a` operation is done.

7  Verify that the root pool datasets are restored.
For example:

```
# zfs list
NAME USED AVAIL REFER MOUNTPOINT
rpool 20.1G 114G 67K /rpool
rpool@0311 0 - 67K -
rpool/ROOT 4.00G 114G 21K legacy
rpool/ROOT@0311 0 - 21K -
rpool/ROOT/opensolaris 5.11M 114G 3.96G /
rpool/ROOT/opensolaris@0311 0 - 3.96G -
rpool/ROOT/osolBE 4.00G 114G 3.96G /
rpool/ROOT/osolBE@0311 0 - 3.96G -
rpool/ROOT/osolBE@install 30.9M - 3.89G -
rpool/ROOT/osolBE@osolBE 2.97M - 3.96G -
rpool/ROOT/osolBE@0311 0 - 3.96G -
rpool/dump 7.94G 114G 7.94G -
rpool/dump@0311 0 - 7.94G -
rpool/export 69.5K 114G 23K /export
rpool/export@0311 0 - 23K -
rpool/export/home 46.5K 114G 23K /export/home
rpool/export/home@0311 0 - 23K -
rpool/export/home/admin 23.5K 114G 23.5K /export/home/admin
rpool/export/home/admin@0311 0 - 23.5K -
rpool/swap 8.20G 122G 14.7M -
rpool/swap@0311 0 - 14.7M -
```
8 Set the `bootfs` property on the root pool BE.
   For example:
   
   ```
   # zpool set bootfs=rpool/ROOT/osolBE rpool
   ```

9 Install the boot blocks on the new disk.
   On a SPARC based system:
   
   ```
   # installboot -F zfs /usr/platform/'uname -i'/lib/fs/zfs/bootblk /dev/rdsk/c1t0d0s0
   ```
   On an x86 based system:
   
   ```
   # installgrub /boot/grub/stage1 /boot/grub/stage2 /dev/rdsk/c1t0d0s0
   ```

10 Reboot the system.
   ```
   # init 6
   ```
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 145
- “Creating, Destroying, and Renaming ZFS File Systems” on page 146
- “Introducing ZFS Properties” on page 149
- “Querying ZFS File System Information” on page 166
- “Managing ZFS Properties” on page 169
- “Mounting and Sharing ZFS File Systems” on page 173
- “Setting ZFS Quotas and Reservations” on page 181
- “Encrypting ZFS File Systems” on page 187
- “Upgrading ZFS File Systems” on page 192

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 253.
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 146
- “Destroying a ZFS File System” on page 147
- “Renaming a ZFS File System” on page 148

**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 55.

Encrypting a ZFS file system must be enabled when the file system is created. For information about encrypting a ZFS file system, see “Encrypting ZFS File Systems” on page 187.

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 174.

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 149.

## 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 174.

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:
```

```
# zfs destroy -R tank/home/schrock
```
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 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 181.

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 198.
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 159
- “Settable ZFS Native Properties” on page 160
- “ZFS User Properties” on page 165

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 165.

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 descendents.

All inheritable properties have an associated source that indicates how a property was obtained. The source of a property can have the following values:

- **local** Indicates that the property was explicitly set on the dataset by using the `zfs set` command as described in “Setting ZFS Properties” on page 169.
- **inherited from dataset-name** Indicates that the property was inherited from the named ancestor.
- **default** 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 `local`.

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 165.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclinherit</td>
<td>String</td>
<td>secure</td>
<td>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 Property (aclinherit)” on page 221.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Type</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>atime</td>
<td>Boolean</td>
<td>on</td>
<td>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.</td>
</tr>
<tr>
<td>available</td>
<td>Number</td>
<td>N/A</td>
<td>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. The property abbreviation is <code>avail</code>. For more information about disk space accounting, see “ZFSDiskSpaceAccounting” on page 64.</td>
</tr>
<tr>
<td>canmount</td>
<td>Boolean</td>
<td>on</td>
<td>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. 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. For more information, see “The canmount Property” on page 162.</td>
</tr>
</tbody>
</table>
### TABLE 6–1  ZFS Native Property Descriptions (Continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>casesensitivity</td>
<td>String</td>
<td>mixed</td>
<td>This property indicates whether the file name matching algorithm used by the file system should be casesensitive, caseinsensitive, or allow a combination of both styles of matching (mixed). Traditionally, UNIX and POSIX file systems have case-sensitive file names. The mixed value for this property indicates the file system can support requests for both case-sensitive and case-insensitive matching behavior. Currently, case-insensitive matching behavior on a file system that supports mixed behavior is limited to the Oracle Solaris SMB server product. For more information about using the mixed value, see &quot;The casesensitivity Property&quot; on page 162. Regardless of the casesensitivity property setting, the file system preserves the case of the name specified to create a file. This property cannot be changed after the file system is created.</td>
</tr>
<tr>
<td>checksum</td>
<td>String</td>
<td>on</td>
<td>Controls the checksum used to verify data integrity. The default value is on, which automatically selects an appropriate algorithm, currently fletcher4. The values are on, off, fletcher2, fletcher4, sha256, and sha256+mac. A value of off disables integrity checking on user data. A value of off is not recommended.</td>
</tr>
<tr>
<td>compression</td>
<td>String</td>
<td>off</td>
<td>Enables or disables compression for a dataset. The values are on, off, lzo, gzip, and gzip -N. Currently, setting this property to lzo, gzip, or gzip -N has the same effect as setting this property to on. Enabling compression on a file system with existing data only compresses new data. Existing data remains uncompressed. The property abbreviation is compress.</td>
</tr>
<tr>
<td>compressratio</td>
<td>Number</td>
<td>N/A</td>
<td>Read-only property that identifies the compression ratio achieved for a dataset, expressed as a multiplier. Compression can be enabled by the zfs set compression=on dataset command. The value is calculated from the logical size of all files and the amount of referenced physical data. It includes explicit savings through the use of the compression property.</td>
</tr>
</tbody>
</table>
### TABLE 6–1  ZFS Native Property Descriptions  (Continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copies</td>
<td>Number</td>
<td>1</td>
<td>Sets the number of copies of user data per file system. Available values are 1, 2, or 3. These copies are in addition to any pool-level redundancy. Disk space used by multiple copies of user data is charged to the corresponding file and dataset, and counts against quotas and reservations. In addition, the used property is updated when multiple copies are enabled. Consider setting this property when the filesystem is created because changing this property on an existing file system only affects newly written data.</td>
</tr>
<tr>
<td>creation</td>
<td>String</td>
<td>N/A</td>
<td>Read-only property that identifies the date and time that a dataset was created.</td>
</tr>
<tr>
<td>dedup</td>
<td>String</td>
<td>off</td>
<td>Controls the ability to remove duplicate data in a ZFS filesystem. Possible values are on, off, verify, and sha256[, verify]. The default checksum for deduplication is sha256. For more information, see &quot;The dedup Property&quot; on page 163.</td>
</tr>
<tr>
<td>devices</td>
<td>Boolean</td>
<td>on</td>
<td>Controls whether device files in a file system can be opened.</td>
</tr>
<tr>
<td>encryption</td>
<td>Boolean</td>
<td>off</td>
<td>Controls whether a dataset is encrypted. An encrypted dataset means that data is encoded and a key is needed by the dataset owner to access the data.</td>
</tr>
<tr>
<td>exec</td>
<td>Boolean</td>
<td>on</td>
<td>Controls whether programs in a file system are allowed to be executed. Also, when set to off, mmap(2) calls with PROT_EXEC are disallowed.</td>
</tr>
<tr>
<td>keysource</td>
<td>String</td>
<td>none</td>
<td>Identifies the format and location of the key that wraps the dataset keys. The valid property values are raw, hex, passphrase, prompt, or file. The key must be present when the dataset is created, mounted, or loaded by using the zfs key -l command. If encryption is enabled for a new dataset, the default keysource is passphrase, prompt.</td>
</tr>
<tr>
<td>keystatus</td>
<td>String</td>
<td>none</td>
<td>Read-only property that identifies the dataset's encryption key status. The availability of a dataset's key is indicated by available or unavailable. For datasets that do not have encryption enabled, none is displayed.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Type</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>logbias</td>
<td>String</td>
<td>latency</td>
<td>Controls how ZFS optimizes synchronous requests for this dataset. If <code>logbias</code> is set to <code>latency</code>, ZFS uses the pool's separate log devices, if any, to handle the requests at low latency. If <code>logbias</code> is set to <code>throughput</code>, ZFS does not use the pool's separate log devices. Instead, ZFS optimizes synchronous operations for global pool throughput and efficient use of resources. The default value is <code>latency</code>.</td>
</tr>
<tr>
<td>mlslabel</td>
<td>String</td>
<td>None</td>
<td>Provides a sensitivity label that determines if a dataset can be mounted in a Trusted Extensions zone. If the labeled dataset matches the labeled zone, the dataset can be mounted and accessed from the labeled zone. The default value is <code>none</code>. This property can only be modified when Trusted Extensions is enabled and only with the appropriate privilege.</td>
</tr>
<tr>
<td>mounted</td>
<td>Boolean</td>
<td>N/A</td>
<td>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>.</td>
</tr>
<tr>
<td>mountpoint</td>
<td>String</td>
<td>N/A</td>
<td>Controls the mount point used for this file system. When the <code>mountpoint</code> property is changed for a file system, the file system and any descendents that inherit the mount point are unmounted. If the new value is <code>legacy</code>, then they remain unmounted. Otherwise, they are automatically remounted in the new location if the property was previously <code>legacy</code> or <code>none</code>, or if they were mounted before the property was changed. In addition, any shared file systems are unshared and shared in the new location. For more information about using this property, see &quot;Managing ZFS Mount Points&quot; on page 174.</td>
</tr>
<tr>
<td>primarycache</td>
<td>String</td>
<td>all</td>
<td>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.</td>
</tr>
<tr>
<td>nbmand</td>
<td>Boolean</td>
<td>off</td>
<td>Controls whether the file system should be mounted with <code>nbmand</code> (Non-blocking mandatory) locks. This property is for SMB clients only. Changes to this property only take effect when the file system is unmounted and remounted.</td>
</tr>
</tbody>
</table>
### TABLE 6–1  ZFS Native Property Descriptions  (Continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normalization</td>
<td>String</td>
<td>None</td>
<td>This property indicates whether a file system should perform a unicode normalization of file names whenever two file names are compared, and which normalization algorithm should be used. File names are always stored unmodified, names are normalized as part of any comparison process. If this property is set to a legal value other than none, and the utf8only property was left unspecified, the utf8only property is automatically set to on. The default value of the normalization property is none. This property cannot be changed after the file system is created.</td>
</tr>
<tr>
<td>origin</td>
<td>String</td>
<td>N/A</td>
<td>Read-only property for cloned file systems or volumes that identifies the snapshot from which the clone was created. The origin cannot be destroyed (even with the -r or -f option) as long as a clone exists. Non-cloned file systems have an origin of none.</td>
</tr>
<tr>
<td>quota</td>
<td>Number (or none)</td>
<td>none</td>
<td>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 volsize property acts as an implicit quota. For information about setting quotas, see &quot;Setting Quotas on ZFS File Systems&quot; on page 182.</td>
</tr>
<tr>
<td>rekeydate</td>
<td>String</td>
<td>N/A</td>
<td>Read-only property that indicates the date of the last data encryption key change from a zfs key -K or zfs clone -K operation on this dataset. If no rekey operation has been performed, the value of this property is the same as the creation date.</td>
</tr>
<tr>
<td>readonly</td>
<td>Boolean</td>
<td>off</td>
<td>Controls whether a dataset can be modified. When set to on, no modifications can be made. The property abbreviation is ronly.</td>
</tr>
<tr>
<td>recordsize</td>
<td>Number</td>
<td>128K</td>
<td>Specifies a suggested block size for files in a file system. The property abbreviation is recsize. For a detailed description, see &quot;The recordsize Property&quot; on page 164.</td>
</tr>
</tbody>
</table>
### TABLE 6-1  ZFS Native Property Descriptions (Continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>referenced</td>
<td>Number</td>
<td>N/A</td>
<td>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. 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. The property abbreviation is refer.</td>
</tr>
<tr>
<td>refquota</td>
<td>Number (or none)</td>
<td>none</td>
<td>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.</td>
</tr>
<tr>
<td>refreservation</td>
<td>Number (or none)</td>
<td>none</td>
<td>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 refreservation. The refreservation reservation is accounted for in the parent dataset’s disk space used, and counts against the parent dataset’s quotas and reservations. If refreservation is set, a snapshot is only allowed if enough free pool space is available outside of this reservation to accommodate the current number of referenced bytes in the dataset. The property abbreviation is refreserv.</td>
</tr>
<tr>
<td>reservation</td>
<td>Number (or none)</td>
<td>none</td>
<td>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. The property abbreviation is reserv. For more information, see “Setting Reservations on ZFS File Systems” on page 185.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Type</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>rstchown</td>
<td>Boolean</td>
<td>on</td>
<td>Indicates whether the file system owner can grant file ownership changes. The default is to restrict chown operations. When <code>rstchown</code> is set to off, the user has the <code>PRIV_FILE_CHOWN_SELF</code> privilege for chown operations.</td>
</tr>
<tr>
<td>secondarycache</td>
<td>String</td>
<td>all</td>
<td>Controls what is cached in the secondary cache (L2ARC). Possible values are all, none, and metadata. If set to all, both user data and metadata are cached. If set to none, neither user data nor metadata is cached. If set to metadata, only metadata is cached.</td>
</tr>
<tr>
<td>setuid</td>
<td>Boolean</td>
<td>on</td>
<td>Controls whether the setuid bit is honored in a file system.</td>
</tr>
<tr>
<td>share nfs</td>
<td>String</td>
<td>off</td>
<td>Controls whether a file system is available over NFS and what options are used. If set to on, the <code>zfs</code> share command is invoked with no options. Otherwise, the <code>zfs</code> share command is invoked with options equivalent to the contents of this property. If set to off, 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 178.</td>
</tr>
<tr>
<td>sharesmb</td>
<td>String</td>
<td>off</td>
<td>Controls whether the file system is shared by using the Oracle Solaris SMB service, and what options are to be used. A file system with the <code>sharesmb</code> property set to off is managed through traditional tools, such as the <code>sharemgr</code> command. Otherwise, the file system is automatically shared and unshared by using the <code>zfs share</code> and <code>zfs unshare</code> commands. If the property is set to on, the <code>sharemgr</code> command is invoked with no options. Otherwise, the <code>sharemgr</code> command is invoked with options that are equivalent to the contents of this property.</td>
</tr>
<tr>
<td>snapdir</td>
<td>String</td>
<td>hidden</td>
<td>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 195.</td>
</tr>
</tbody>
</table>
### TABLE 6–1  ZFS Native Property Descriptions  (Continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sync</td>
<td>String</td>
<td>standard</td>
<td>Determines the synchronous behavior of a file system's transactions. Possible values are: standard, the default value, which means synchronous file system transactions, such as fsync, O_DSYNC, O_SYNC, and so on, are written to the intent log; always, ensures that every file system transaction is written and flushed to stable storage by a returning system call. This value has a significant performance penalty; disabled, means that synchronous requests are disabled. File system transactions are only committed to stable storage on the next transaction group commit, which might be after many seconds. This value gives the best performance, with no risk of corrupting the pool. However, this value is very dangerous because ZFS is ignoring the synchronous transaction demands of applications, such as databases or NFS operations. Setting this value on the currently active root or /var file system might result in out-of-spec behavior, application data loss, or increased vulnerability to replay attacks. You should only use this value if you fully understand all the associated risks.</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>N/A</td>
<td>Read-only property that identifies the dataset type as filesystem (file system or clone), volume, or snapshot.</td>
</tr>
<tr>
<td>used</td>
<td>Number</td>
<td>N/A</td>
<td>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 160.</td>
</tr>
<tr>
<td>usedbychildren</td>
<td>Number</td>
<td>off</td>
<td>Read-only property that identifies the amount of disk space that is used by children of this dataset, which would be freed if all the dataset's children were destroyed. The property abbreviation is usedchild.</td>
</tr>
<tr>
<td>usedbydataset</td>
<td>Number</td>
<td>off</td>
<td>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 useddds.</td>
</tr>
<tr>
<td>usedbyrefreservation</td>
<td>Number</td>
<td>off</td>
<td>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.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Type</td>
<td>Default Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>usedbysnapshots</td>
<td>Number</td>
<td>off</td>
<td>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.</td>
</tr>
<tr>
<td>version</td>
<td>Number</td>
<td>N/A</td>
<td>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 zfs upgrade command.</td>
</tr>
<tr>
<td>utf8only</td>
<td>Boolean</td>
<td>Off</td>
<td>This property indicates whether a file system should reject file names that include characters that are not present in the UTF-8 character code set. If this property is explicitly set to off, the normalization property must either not be explicitly set or be set to none. The default value for the utf8only property is off. This property cannot be changed after the file system is created.</td>
</tr>
<tr>
<td>volsize</td>
<td>Number</td>
<td>N/A</td>
<td>For volumes, specifies the logical size of the volume. For a detailed description, see “The volsize Property” on page 165.</td>
</tr>
<tr>
<td>volblocksize</td>
<td>Number</td>
<td>8 KB</td>
<td>For volumes, specifies the block size of the volume. The block size cannot be changed after the volume has been written, so set the block size at volume creation time. The default block size for volumes is 8 KB. Any power of 2 from 512 bytes to 128 KB is valid. The property abbreviation is volblock.</td>
</tr>
<tr>
<td>vscan</td>
<td>Boolean</td>
<td>Off</td>
<td>Controls whether regular files should be scanned for viruses when a file is opened and closed. In addition to enabling this property, a virus scanning service must also be enabled for virus scanning to occur if you have third-party virus scanning software. The default value is off.</td>
</tr>
</tbody>
</table>
### TABLE 6–1 ZFS Native Property Descriptions (Continued)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zoned</td>
<td>Boolean</td>
<td>N/A</td>
<td>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 256.</td>
</tr>
<tr>
<td>xattr</td>
<td>Boolean</td>
<td>on</td>
<td>Indicates whether extended attributes are enabled (on) or disabled (off) for this file system.</td>
</tr>
</tbody>
</table>

### ZFS Read-Only Native Properties

Read-only native properties can be retrieved but not set. Read-only native properties are not inherited. Some native properties are specific to a particular type of dataset. In such cases, the dataset type is mentioned in the description in Table 6–1.

The read-only native properties are listed here and described in Table 6–1.

- available
- compressratio
- creation
- keystatus
- mounted
- origin
- referenced
- rekeydate
- type
- used
  - For detailed information, see “The used Property” on page 160.
- usedbychildren
- usedbydataset
- usedbyrefreservation
- usedbysnapshots

For more information about disk space accounting, including the used, referenced, and available properties, see “ZFS Disk Space Accounting” on page 64.
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 64.

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 169 or by using the zfs create command as described in “Creating a ZFS File System” on page 146. 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 181.

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 Property (aclinherit)” on page 221.
- atime
- canmount
Introducing ZFS Properties

- casesensitivity
- checksum
- compression
- copies
- devices
- dedup
- encryption
- exec
- keysource
- logbias
- mlslabel
- mountpoint
- nbmand
- normalization
- primarycache
- quota
- readonly
- recordsize
  For a detailed description, see “The recordsize Property” on page 164.
- refquota
- refreservation
- reservation
- rstchown
- secondarycache
- sharesmb
- sharenfs
- setuid
- snapdir
- version
- vscan
- utf8only
- volsize
  For a detailed description, see “The volsize Property” on page 165.
The `canmount` Property

If the `canmount` property is set to `off`, the file system cannot be mounted by using the `zfs mount` or `zfs mount -a` commands. Setting this property to `off` is similar to setting the `mountpoint` property to `none`, except that the 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.

The `casesensitivity` Property

This property indicates whether the file name matching algorithm used by the file system should be `casesensitive`, `caseinsensitive`, or allow a combination of both styles of matching (mixed).

When a case-insensitive matching request is made of a mixed sensitivity file system, the behavior is generally the same as would be expected of a purely case-insensitive file system. The difference is that a mixed sensitivity file system might contain directories with multiple names that are unique from a case-sensitive perspective, but not unique from the case-insensitive perspective.

For example, a directory might contain files `foo`, `Foo`, and `F00`. If a request is made to case-insensitively match any of the possible forms of `foo`, (for example `foo`, `F00`, `Fo0`, `f0o`, and so on) one of the three existing files is chosen as the match by the matching algorithm. Exactly
which file the algorithm chooses as a match is not guaranteed, but what is guaranteed is that the same file is chosen as a match for any of the forms of foo. The file chosen as a case-insensitive match for foo, FOO, foO, Foo, and so on, is always the same, so long as the directory remains unchanged.

The utf8only, normalization, and casesensitivity properties also provide new permissions that can be assigned to non-privileged users by using ZFS delegated administration. For more information, see “Delegating ZFS Permissions” on page 242.

The dedup Property

This property controls whether duplicate data is removed from the file system. If a file system has the dedup property enabled, duplicate data blocks are removed synchronously. The result is that only unique data is stored and common components are shared between files.

Do not enable the dedup property on your file systems until you have reviewed the information in this FAQ:

http://hub.opensolaris.org/bin/view/Community+Group+zfs/dedup

When dedup is enabled, the dedup checksum algorithm overrides the checksum property. Setting the value to verify is equivalent to specifying sha256, verify. If the property is set to verify and two blocks have the same signature, ZFS does a byte-for-byte comparison with the existing block to ensure that the contents are identical.

This property can be enabled per file system as follows:

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

You can use the zfs get command to determine if the dedup property is set.

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

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

The DEDUP column indicates how much deduplication has occurred. If the dedup property is not enabled on any dataset or if the dedup property was just enabled on the dataset, the DEDUP ratio is 1.00x.

You can use the zpool get command to determine the value of the dedup_ratio property.

```
# zpool get all export
NAME PROPERTY VALUE SOURCE
export size 33.8G -
export capacity 0% -
export altroot - default
```
This pool property illustrates how much deduplication we have been able to achieve.

**The encryption Property**

You can use the encryption property to encrypt ZFS file systems. For more information, see "Encrypting ZFS File Systems" on page 187.

**The recordsize Property**

The recordsize property specifies a suggested block size for files in the file system.

This property is designed solely for use with database workloads that access files in fixed-size records. ZFS automatically adjust block sizes according to internal algorithms optimized for typical access patterns. For databases that create very large files but access the files in small random chunks, these algorithms might be suboptimal. Specifying a recordsize value greater than or equal to the record size of the database can result in significant performance gains. Use of this property for general purpose file systems is strongly discouraged and might adversely affect performance. The size specified must be a power of 2 greater than or equal to 512 bytes and less than or equal to 128 KB. Changing the file system’s recordsize value only affects files created afterward. Existing files are unaffected.

The property abbreviation is recsize.

**The sharesmb Property**

This property enables sharing of ZFS file systems with the Oracle Solaris SMB service, and identifies options to be used.

Because SMB shares requires a resource name, a unique resource name is constructed from the dataset name. The constructed name is a copy of the dataset name except that the characters in the dataset name, which would be illegal in the resource name, are replaced with underscore (_) characters. A pseudo property name is also supported that allows you to replace the dataset name with a specific name. The specific name is then used to replace the prefix dataset in the case of inheritance.

```
export health ONLINE -
export guid 2064230982813446135 default
export version 22 default
export bootfs - default
export delegation on default
export autoreplace off default
export cachefile - default
export failmode wait default
export listsnapshots off default
export autoexpand off default
export dedupditto 0 default
export dedupratio 3.00x -
export free 33.6G -
export allocated 105M -
```
For example, if the dataset, `data/home/john`, is set to `name=john`, then `data/home/john` has a resource name of `john`. If a child dataset of `data/home/john/backups` exists, it has a resource name of `john_backups`. When the `sharesmb` property is changed for a dataset, the dataset and any children inheriting the property are re-shared with the new options, only if the property was previously set to `off`, or if they were shared before the property was changed. If the new property is set to `off`, the file systems are unshared.

For examples of using the `sharesmb` property, see “Sharing and Unsharing ZFS File Systems” on page 178.

**The `volsize` Property**

The `volsize` property specifies the logical size of the volume. By default, creating a volume establishes a reservation for the same amount. Any changes to `volsize` are reflected in an equivalent change to the reservation. These checks are used to prevent unexpected behavior for users. A volume that contains less space than it claims is available can result in undefined behavior or data corruption, depending on how the volume is used. These effects can also occur when the volume size is changed while the volume is in use, particularly when you shrink the size. Use extreme care when adjusting the volume size.

Though not recommended, you can create a sparse volume by specifying the `-s` flag to `zfs create -V` or by changing the reservation after the volume has been created. A `sparse volume` is a volume whose reservation is not equal to the volume size. For a sparse volume, changes to `volsize` are not reflected in the reservation.

For more information about using volumes, see “ZFS Volumes” on page 253.

**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
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>PROPERTY</th>
<th>VALUE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>userpool</td>
<td>dept:users</td>
<td>all</td>
<td>local</td>
</tr>
<tr>
<td>userpool/user1</td>
<td>dept:users</td>
<td>finance</td>
<td>local</td>
</tr>
<tr>
<td>userpool/user2</td>
<td>dept:users</td>
<td>general</td>
<td>local</td>
</tr>
<tr>
<td>userpool/user3</td>
<td>dept:users</td>
<td>itops</td>
<td>local</td>
</tr>
</tbody>
</table>

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 149.

For example:
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
```

You can use the `zfs list` command with the mount point of a file system. For example:

```
# zfs list /pool/home/marks
```

The following example shows how to display basic information about tank/home/chua and all of its descendents:

```
# zfs list -r tank/home/chua
```

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 149. In addition to the properties defined, the `-o` option list can also contain the literal `name` to indicate that the output should include the name of the dataset.

The following example uses `zfs list` to display the dataset name, along with the `share nfs` and `mount point` property values.

```
# zfs list -o name,share nfs,mount point
```

---

**Chapter 6 • Managing Oracle Solaris ZFS File Systems**

167
You can use the `-t` option to specify the types of datasets to display. The valid types are described in the following table.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filesystem</td>
<td>File systems and clones</td>
</tr>
<tr>
<td>volume</td>
<td>Volumes</td>
</tr>
<tr>
<td>snapshot</td>
<td>Snapshots</td>
</tr>
</tbody>
</table>

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 169
- "Inheriting ZFS Properties" on page 170
- "Querying ZFS Properties" on page 170

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 160.

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 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` filesystem. 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
# zfs inherit compression tank/home/bonwick
# zfs get -r compression tank
NAME PROPERTY VALUE SOURCE
tank/home compression off default
# zfs inherit -r compression tank/home
```

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 166. 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

<table>
<thead>
<tr>
<th>NAME</th>
<th>PROPERTY</th>
<th>VALUE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank/ws</td>
<td>checksum</td>
<td>on</td>
<td>default</td>
</tr>
</tbody>
</table>

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)**

<table>
<thead>
<tr>
<th>Source Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>This property value was never explicitly set for this dataset or any of its ancestors. The default value for this property is being used.</td>
</tr>
<tr>
<td>inherited from dataset-name</td>
<td>This property value is inherited from the parent dataset specified in dataset-name.</td>
</tr>
<tr>
<td>local</td>
<td>This property value was explicitly set for this dataset by using zfs set.</td>
</tr>
<tr>
<td>temporary</td>
<td>This property value was set by using the zfs mount -o option and is only valid for the duration of the mount. For more information about temporary mount point properties, see &quot;Using Temporary Mount Properties&quot; on page 177.</td>
</tr>
<tr>
<td>(none)</td>
<td>This property is read-only. Its value is generated by ZFS.</td>
</tr>
</tbody>
</table>

You can use the special keyword all to retrieve all dataset property values. The following examples use the all keyword:

# zfs get all tank

<table>
<thead>
<tr>
<th>NAME</th>
<th>PROPERTY</th>
<th>VALUE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>type</td>
<td>filesystem</td>
<td>-</td>
</tr>
<tr>
<td>tank</td>
<td>creation</td>
<td>Fri Oct 8 16:47 2010</td>
<td>-</td>
</tr>
<tr>
<td>tank</td>
<td>used</td>
<td>1.31M</td>
<td>-</td>
</tr>
<tr>
<td>tank</td>
<td>available</td>
<td>66.9G</td>
<td>-</td>
</tr>
<tr>
<td>tank</td>
<td>referenced</td>
<td>35K</td>
<td>-</td>
</tr>
<tr>
<td>tank</td>
<td>compressratio</td>
<td>1.00x</td>
<td>-</td>
</tr>
<tr>
<td>tank</td>
<td>mounted</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>tank</td>
<td>quota</td>
<td>none</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>reservation</td>
<td>none</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>recordsize</td>
<td>128K</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>mountpoint</td>
<td>/tank</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>sharerfs</td>
<td>off</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>checksum</td>
<td>on</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>compression</td>
<td>off</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>atime</td>
<td>on</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>devices</td>
<td>on</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>exec</td>
<td>on</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>setuid</td>
<td>on</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>readonly</td>
<td>off</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>zoned</td>
<td>off</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>snapdir</td>
<td>hidden</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>aclinherit</td>
<td>restricted</td>
<td>default</td>
</tr>
<tr>
<td>tank</td>
<td>canmount</td>
<td>on</td>
<td>default</td>
</tr>
</tbody>
</table>
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 149 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`:

```bash
# zfs get -H -o value compression tank/home
```

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:

```bash
# 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:

```bash
# zfs get -H -o name,value -r used export/home
export/home 5.57G
export/home/marks 1.43G
export/home/maybe 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 174
- "Mounting ZFS File Systems” on page 175
- "Using Temporary Mount Properties” on page 177
- “Unmounting ZFS File Systems” on page 177
- “Sharing and Unsharing ZFS File Systems” on page 178
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 73.

All ZFS file systems are mounted by ZFS at boot time by using the Service Management Facility’s (SMF) `svc://system/filesystem/local` service. File systems are mounted under /path, where path is the name of the file system.

You can override the default mount point by using the `zfs set` command to set the mountpoint property to a specific path. ZFS automatically creates the specified mount point, if needed, and automatically mounts the associated file system.

ZFS file systems are automatically mounted at boot time without requiring you to edit the `/etc/vfstab` file.

The mountpoint property is inherited. For example, if pool/home has the mountpoint property set to /export/stuff, then pool/home/user inherits /export/stuff/user for its mountpoint property value.

To prevent a file system from being mounted, set the mountpoint property to none. In addition, the canmount property can be used to control whether a file system can be mounted. For more information about the canmount property, see “The canmount Property” on page 162.

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 175.

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
```

---

Mounting and Sharing ZFS File Systems
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 51.

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 umount 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>
<thead>
<tr>
<th>Property</th>
<th>Mount Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>atime</td>
<td>atime/noatime</td>
</tr>
<tr>
<td>devices</td>
<td>devices/nodevices</td>
</tr>
<tr>
<td>exec</td>
<td>exec/noexec</td>
</tr>
<tr>
<td>nbmand</td>
<td>nbmand/nonbmand</td>
</tr>
<tr>
<td>readonly</td>
<td>ro/rw</td>
</tr>
<tr>
<td>setuid</td>
<td>setuid/nosetuid</td>
</tr>
<tr>
<td>xattr</td>
<td>xattr/noxattr</td>
</tr>
</tbody>
</table>

The mount option `nosuid` is an alias for `nodevices, nosetuid`.
You can use the NFSv4 mirror mount features to help you better manage NFS-mounted ZFS home directories. For a description of mirror mounts, see “ZFS and File System Mirror Mounts” on page 37.

**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 umount` 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 share nfs 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 share nfs 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.

Sharing ZFS Files in an Oracle Solaris SMB Environment

The sharesmb property is provided to share ZFS files by using the Oracle Solaris SMB software product. When this property is set on a ZFS file system, these shares are visible to SMB client systems. For more information about using the Oracle Solaris SMB software product, see the System Administration Guide: Windows Interoperability.

For a detailed description of the sharesmb property, see “The sharesmb Property” on page 164.

```
# svcadm enable -r smb/server
svcadm: svc:/milestone/network depends on svc:/network/physical, which has multiple instances.
# svcs | grep smb
online 10:47:15 svc:/network/smb/server:default
```

EXAMPLE 6-1  Example—Sharing ZFS File Systems (sharesmb)

In this example, a ZFS file system `sandbox/fs1` is created and shared with the sharesmb property. If necessary, enable the SMB services.

```
# svcadm enable -r smb/server
svcadm: svc:/milestone/network depends on svc:/network/physical, which has multiple instances.
# svcs | grep smb
online 10:47:15 svc:/network/smb/server:default
```
# zpool create sandbox mirror c0t2d0 c0t4d0
# zfs create sandbox/fs1
# zfs set sharesmb=on sandbox/fs1

The `sharesmb` property is set for `sandbox/fs1` and its descendents.

Verify that the file system was shared. For example:

```
# sharemgr show -vp
default nfs=()
zfs nfs=()
  zfs/sandbox/fs1 smb=()
    sandbox_fs1=/sandbox/fs1

A default SMB resource name, `sandbox_fs1`, is assigned automatically.

In this example, another file system is created, `sandbox/fs2`, and shared with a resource name, `myshare`.

```
# zfs create sandbox/fs2
# zfs set sharesmb=name=myshare sandbox/fs2
# sharemgr show -vp
default nfs=()
zfs nfs=()
  zfs/sandbox/fs1 smb=()
    sandbox_fs1=/sandbox/fs1
  zfs/sandbox/fs2 smb=()
    myshare=/sandbox/fs2

The `sandbox/fs2/fs2_sub1` file system is created and is automatically shared. The inherited resource name is `myshare_fs2_sub1`.

```
# zfs create sandbox/fs2/fs2_sub1
# sharemgr show -vp
default nfs=()
zfs nfs=()
  zfs/sandbox/fs1 smb=()
    sandbox_fs1=/sandbox/fs1
  zfs/sandbox/fs2 smb=()
    myshare=/sandbox/fs2
    myshare_fs2_sub1=/sandbox/fs2/fs2_sub1

Disable SMB sharing for `sandbox/fs2` and its descendents.

```
# zfs set sharesmb=off sandbox/fs2
# sharemgr show -vp
default nfs=()
zfs nfs=()
  zfs/sandbox/fs1 smb=()
    sandbox_fs1=/sandbox/fs1

In this example, the `sharesmb` property is set on the pool's top-level file system. The descendent file systems are automatically shared.
EXAMPLE 6–1 Example—Sharing ZFS File Systems (sharesmb) (Continued)

```
# zpool create sandbox mirror c0t2d0 c0t4d0
# zfs set sharesmb=on sandbox
# zfs create sandbox/fs1
# zfs create sandbox/fs2
```

The top-level file system has a resource name of sandbox, but the descendents have their dataset name appended to the resource name.

```
# sharemgr show -vp
default nfs=()
zfs nfs=()
  zfs/sandbox smb=()
  sandbox=/sandbox
    sandbox_fs1=/sandbox/fs1 smb=()
    sandbox_fs2=/sandbox/fs2 smb=()
```

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 182 and “Setting Reservations on ZFS File Systems” on page 185.

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
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>PROPERTY</th>
<th>VALUE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank/home/bonwick</td>
<td>quota</td>
<td>10.0G</td>
<td>local</td>
</tr>
</tbody>
</table>

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
```
For additional convenience, you can set another quota on a dataset to help manage the disk space that is consumed by snapshots. For example:

```bash
# 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 -
```

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:

```bash
# 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:

```bash
# 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:

```bash
# zfs get userquota@student1 students/compsci
NAME PROPERTY VALUE SOURCE
students/compsci userquota@student1 10G local
```

You can display general user or group disk space usage by querying the following properties:
To identify individual user or group disk space usage, query the following properties:

```bash
# 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:

```bash
# 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:

```bash
# zfs set userquota@student1=10m students/compsci
# zfs userspace students/compsci
```
Setting ZFS Quotas and Reservations

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:

```bash
# 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:

```bash
# 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:

```bash
# 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:

```bash
# zfs get reservation,refreserv profs/prof1
NAME PROPERTY VALUE SOURCE
profs/prof1 reservation 20G local
profs/prof1 refreserv 10G local
```

If `refreservation` is set, a snapshot is only allowed if sufficient unreserved pool space exists outside of this reservation to accommodate the current number of referenced bytes in the dataset.
Encrypting ZFS File Systems

Encryption is the process where data is encoded for privacy and a key is needed by the data owner to access the encoded data. The benefits of using ZFS encryption are as follows:

- ZFS encryption is integrated with the ZFS command set. Like other ZFS operations, encryption operations such as key changes and rekey are performed online.
- You can use your existing storage pools as long as they are upgraded. You have the flexibility of encrypting specific file systems.
- ZFS encryption is inheritable to descendent file systems. Key management can be delegated through ZFS delegated administration.
- Data is encrypted using AES (Advanced Encryption Standard) with key lengths of 128, 192, and 256 in the CCM and GCM operation modes.
- ZFS encryption uses the Oracle Solaris Cryptographic Framework, which gives it access to any available hardware acceleration or optimized software implementations of the encryption algorithms automatically.

You can set an encryption policy when a ZFS dataset is created, but the policy cannot be changed. For example, the tank/home/darren file system is created with the encryption property enabled. The default encryption policy is to prompt for a passphrase, which must be a minimum of 8 characters in length.

```
# zfs create -o encryption=on tank/home/darren
Enter passphrase for 'tank/home/darren': xxxxxxxx
Enter again: xxxxxxxx
```

Confirm that the file system has encryption enabled. For example:

```
# zfs get encryption tank/home/darren
NAME PROPERTY VALUE SOURCE
tank/home/darren encryption on local
```

The default encryption algorithm is aes - 128 - ccm when a file system's encryption value is on.

A wrapping key is used to encrypt the actual data encryption keys. The wrapping key is passed from the zfs command, as in the above example when the encrypted file system is created, to the kernel. A wrapping key is either in a file (in raw or hex format) or it is derived from a passphrase.

The format and location of the wrapping key are specified in the keysource property as follows:

```
keysouce=format,location
```

- Format is one of the following:
  - raw — The raw key bytes
  - hex — A hexadecimal key string
  - passphrase — A character string that generates a key
Location is one of the following:
- **prompt** — You are prompted for a key when the file system is created or mounted
- **file://filename** — The key file location in a file system

If the keysource format is *passphrase*, then the wrapping key is derived from the passphrase. Otherwise, the keysource property value points to the actual wrapping key, as raw bytes or in hexadecimal format. You can specify that the passphrase is stored in a file or stored in a raw stream of bytes that are prompted for, which is likely only suitable for scripting.

When a file system’s keysource property values identifies *passphrase*, then the wrapping key is derived from the passphrase using PKCS#5 PBKDF2 and a per file system randomly generated salt. This means that the same passphrase generates a different wrapping key if used on descendent datasets.

A file system’s encryption policy is inherited by descendent file systems and cannot be removed. For example:

```
# zfs snapshot tank/home/darren@now
# zfs clone tank/home/darren@now tank/home/darren-new
Enter passphrase for 'tank/home/darren-new': xxxxxxxx
Enter again: xxxxxxxx
# zfs set encryption=off tank/home/darren-new
```

If you need to copy or migrate encrypted or unencrypted ZFS file systems, then consider the following points:

- Currently, you cannot send an unencrypted dataset stream and receive it as an encrypted stream even if the receiving pool’s dataset has encryption enabled.
- You can use the following commands to migrate unencrypted data to a pool/dataset with encryption enabled:
  - `cp -r`
  - `find | cpio`
  - `tar`
  - `rsync`
- A replicated encrypted dataset stream can be received into an encrypted dataset and the data remains encrypted. For more information, see *Example 6–5*.

### Changing an Encrypted ZFS File System's Keys

You can change an encrypted file system’s wrapping key by using the `zfs key -c` command. The existing wrapping key must have been loaded first, either at boot time or by explicitly loading the file system key (`zfs key -l`) or by mounting the file system (`zfs mount filesystem`). For example:
In the following example, the wrapping key is changed and the keysource property value is changed to specify that the wrapping key comes from a file.

```
# zfs key -c -o keysource=raw,file:///media/stick/key tank/home/darren
```

The data encryption key for an encrypted file system can be changed by using the `zfs key -K` command, but the new encryption key is only used for newly written data. This feature can be used to provide compliance with NIST 800-57 guidelines on a data encryption key's time limit. For example:

```
# zfs key -K tank/home/darren
```

In the above example, the data encryption key is not visible nor is it directly managed by you. In addition, you need the keychange delegation to perform a key change operation.

The following encryption algorithms are available:

- `aes-128-ccm`, `aes-192-ccm`, `aes-256-ccm`
- `aes-128-gcm`, `aes-192-gcm`, `aes-256-gcm`

The ZFS keysource property identifies the format and location of the key that wraps the file system's data encryption keys. For example:

```
# zfs get keysource tank/home/darren
NAME PROPERTY  VALUE SOURCE
tank/home/darren keysource passphrase,prompt local
```

The ZFS rekeydate property identifies the date of the last `zfs key -K` operation. For example:

```
# zfs get rekeydate tank/home/darren
NAME PROPERTY  VALUE SOURCE
tank/home/darren rekeydate Tue Oct 12 15:36 2010 local
```

If an encrypted file system's creation and rekeydate properties have the same value, the file system has never been rekeyed by an `zfs key -K` operation.

### Delegating ZFS Key Operation Permissions

Review the following permission descriptions for delegating key operations:

- Loading or unloading a dataset key by using the `zfs key -l` and `zfs key -u` commands require the key permission. In most cases, you will need the mount permission as well.
- Changing a dataset key by using the `zfs key -c` and `zfs key -K` commands require the keychange permission.
Consider delegating separate permissions for key use (load or unload) and key change, which allows you to have a two-person key operation model. For example, determine which users can use the keys versus which users can change them. Or, both users need to be present for a key change. This model also allows you to build a key escrow system.

**Mounting an Encrypted ZFS File System**

Review the following considerations when attempting to mount an encrypted ZFS file system:

- If an encrypted file system key is not available during boot time, the file system is not mounted automatically. For example, a file system with an encryption policy set to `passphrase,prompt` will not mount during boot time because the boot process is not interrupted to prompt for a passphrase.

- If you want to mount a file system with an encryption policy set to `passphrase,prompt` at boot time, you will need to either explicitly mount it with the `zfs mount` command and specify the passphrase or use the `zfs key -l` command to be prompted for the key after the system is booted.

  For example:

  ```
  # zfs mount -a
  Enter passphrase for 'tank/home/darren': xxxxxxxx
  Enter passphrase for 'tank/home/ws': xxxxxxxx
  Enter passphrase for 'users/home/marks': xxxxxxxx
  ```

- If an encrypted file system's `keysource` property points to a file in another file system, the mount order of the file systems can impact whether the encrypted file system is mounted at boot, particularly if the file is on removable media.

**Interactions Between ZFS Compression, Deduplication, and Encryption Properties**

Review the following considerations when using the ZFS compression, deduplication, and encryption properties:

- When a file is written, the data is compressed, encrypted, and the checksum is verified. Then, the data is deduplicated, if possible.

- When a file is read, the checksum is verified and the data is decrypted. Then, the data is decompressed, if required.

- If the `dedup` property is enabled on an encrypted dataset that is also cloned and the `zfs key -K` or `zfs clone -K` commands have not been used on the clones, data from all the clones will be deduplicated, if possible.
Examples of Encrypting ZFS File Systems

EXAMPLE 6-2 Encrypting a ZFS File System by Using a Raw Key

In the following example, an aes-256-ccm encryption key is generated by using the pktool command and is written to a file, /cindykey.file.

```
# pktool genkey keystore=file outkey=/cindykey.file keytype=aes keylen=256
```

Then, the /cindykey.file is specified when the tank/home/cindy file system is created.

```
# zfs create -o encryption=aes-256-ccm -o keysource=raw,file:///cindykey.file tank/home/cindy
```

EXAMPLE 6-3 Encrypting a ZFS File System With a Different Encryption Algorithm

You can create a ZFS storage pool and have all the file systems in the storage pool inherit an encryption algorithm. In this example, the users pool is created and the users/home file system is created and encrypted by using a passphrase. The default encryption algorithm is aes-128-ccm.

Then, the users/home/marks file system is created and encrypted by using the aes-256-ccm encryption algorithm.

```
# zpool create -O encryption=on users mirror c0t1d0 c1t1d0 mirror c2t1d0 c3t1d0
Enter passphrase for 'users': xxxxxxxx
Enter again: xxxxxxxx
# zfs create users/home
# zfs get encryption users/home
NAME PROPERTY VALUE SOURCE
users/home encryption on inherited from users
# zfs create -o encryption=aes-256-ccm users/home/marks
# zfs get encryption users/home/marks
NAME PROPERTY VALUE SOURCE
users/home/marks encryption aes-256-ccm local
```

EXAMPLE 6-4 Cloning an Encrypted ZFS File System

If the clone dataset inherits the keysource property from the same dataset as its origin snapshot, then a new keysource is not necessary, and you are not prompted for a new passphrase if keysource=passphrase,prompt. The same keysource is used for the clone. For example:

By default, you are not prompted for a key when cloning a descendent of an encrypted file system.

```
# zfs create -o encryption=on tank/ws
Enter passphrase for 'tank/ws': xxxxxxxx
Enter again: xxxxxxxx
# zfs create tank/ws/fs1
# zfs snapshot tank/ws/fs1@snap1
# zfs clone tank/ws/fs1@snap1 tank/ws/fs1clone
```
EXAMPLE 6–4  Cloning an Encrypted ZFS File System  (Continued)

If you want to create a new key for the clone file system, use the `zfs clone -K` command.

If you clone an encrypted file system rather than a descendent encrypted file system, you are prompted to provide a new key. For example:

```
# zfs create -o encryption=on tank/ws
Enter passphrase for ‘tank/ws’: xxxxxxxx
Enter again: xxxxxxxx
# zfs snapshot tank/ws@1
# zfs clone tank/ws@1 tank/ws1clone
Enter passphrase for ‘tank/ws1clone’: xxxxxxxx
Enter again: xxxxxxxx
```

EXAMPLE 6–5  Sending and Receiving an Encrypted ZFS File System

In the following example, the `tank/home/darren@snap1` snapshot is created from the encrypted `/tank/home/darren` file system. Then, the snapshot is sent to `bpool/snaps`, with the encryption property enabled so the resulting received data is encrypted. However, the `tank/home/darren@snap1` stream is not encrypted during the send process.

```
# zfs get encryption tank/home/darren
NAME      PROPERTY  VALUE  SOURCE
tank/home/darren encryption on local
# zfs snapshot tank/home/darren@snap1
# zfs get encryption bpool/snaps
NAME      PROPERTY  VALUE  SOURCE
bpool/snaps encryption on inherited from bpool
# zfs send tank/home/darren@snap1 | zfs receive bpool/snaps/darren1012
# zfs get encryption bpool/snaps/darren1012
NAME      PROPERTY  VALUE  SOURCE
bpool/snaps/darren1012 encryption on inherited from bpool
```

In this case, a new key is automatically generated for the received encrypted dataset.

**Upgrading ZFS File Systems**

If you have ZFS file systems from a previous Solaris release, you can upgrade your file systems with the `zfs upgrade` command to take advantage of the file system features in the current release. In addition, this command notifies you when your file systems are running older versions.

For example, this file system is at the current version 5.

```
# zfs upgrade
This system is currently running ZFS filesystem version 5.
```

All file systems are formatted with the current version.

Use this command to identify the features that are available with each file system version.
The following filesystem versions are supported:

<table>
<thead>
<tr>
<th>VER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initial ZFS filesystem version</td>
</tr>
<tr>
<td>2</td>
<td>Enhanced directory entries</td>
</tr>
<tr>
<td>3</td>
<td>Case insensitive and File system unique identifier (FUID)</td>
</tr>
<tr>
<td>4</td>
<td>userquota, groupquota properties</td>
</tr>
<tr>
<td>5</td>
<td>System attributes</td>
</tr>
</tbody>
</table>

For more information on a particular version, including supported releases, see the ZFS Administration Guide.
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 207.

“Creating and Destroying ZFS Snapshots” on page 196
“Displaying and Accessing ZFS Snapshots” on page 199
“Rolling Back a ZFS Snapshot” on page 200

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 55.

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/ahrens@now
# zfs list -t snapshot
```

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 147.

**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 descendental 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
```

Overview of ZFS Snapshots
If you want to destroy a held snapshot, use the \texttt{-d} option. For example:

\texttt{# zfs destroy -d tank/home/cindys@snap1}

Use the \texttt{zfs holds} command to display a list of held snapshots. For example:

\texttt{# zfs holds tank/home@now}

\begin{verbatim}
NAME    TAG  TIMESTAMP
tank/home@now keep Thu Jul 15 11:25:39 2010
\end{verbatim}

\texttt{# zfs holds -r tank/home@now}

\begin{verbatim}
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
\end{verbatim}

You can use the \texttt{zfs release} command to release a hold on a snapshot or set of snapshots. For example:

\texttt{# zfs release -r keep tank/home@now}

If the snapshot is released, the snapshot can be destroyed by using the \texttt{zfs destroy} command. For example:

\texttt{# zfs destroy -r tank/home@now}

Two new properties identify snapshot hold information:

\begin{itemize}
\item The \texttt{defer_destroy} property is on if the snapshot has been marked for deferred destruction by using the \texttt{zfs destroy -d} command. Otherwise, the property is off.
\item The \texttt{userrefs} property is set to the number of holds on this snapshot, also referred to as the user-reference count.
\end{itemize}

\textbf{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:

\texttt{# zfs rename tank/home/cindys@083006 tank/home/cindys@today}

In addition, the following shortcut syntax is equivalent to the preceding syntax:

\texttt{# 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:

\texttt{# zfs rename tank/home/cindys@today pool/home/cindys@saturday}
\texttt{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

By default, snapshots are no longer displayed in the `zfs list` output. You must use the `zfs list -t snapshot` command to display snapshot information. Or, enable the `listsnapshots` pool property. For example:

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

Snapshots of file systems are accessible in the `.zfs/snapshot` directory within the root of the file system. For example, if `tank/home/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 -
```
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
NAME     AVAIL  USED  USEDSNAP  USEDDS  USEDREFRESERV  USEDCHILD
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  28K      0        18K
rpool/export/home 25.4G   18K  0  18K      0        0
rpool/swap  25.8G  512M  0   111M     401M    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
```

### Identifying ZFS Snapshot Differences (zfs diff)

You can determine ZFS snapshot differences by using the `zfs diff` command.

For example, the following two snapshots are created:

```
$ ls /tank/home/timh
fileA
$ zfs snapshot tank/home/timh@1014
$ ls /tank/home/timh
fileA  fileB
$ zfs snapshot tank/home/timh@01015
```

Identify the snapshot differences. For example:

```
$ zfs diff tank/home/timh@1014 tank/home/timh@01015
M /tank/home/timh/
+ /tank/home/timh/fileB
```

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

```
$ mv /tank/cindys/fileB /tank/cindys/fileC
$ zfs snapshot tank/cindys@0915
$ zfs diff tank/cindys@0914 tank/cindys@0915
M /tank/cindys/
R /tank/cindys/fileB -> /tank/cindys/fileC
```

The following table summarizes the file or directory changes that are identified by the `zfs diff` command.

<table>
<thead>
<tr>
<th>File or Directory Change</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>File or directory is modified or file or directory link changed</td>
<td>M</td>
</tr>
<tr>
<td>File or directory is present in the older snapshot but not in the newer snapshot</td>
<td>—</td>
</tr>
<tr>
<td>File or directory is present in the newer snapshot but not in the older snapshot</td>
<td>+</td>
</tr>
<tr>
<td>File or directory is renamed</td>
<td>R</td>
</tr>
</tbody>
</table>

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

**Managing Automatic ZFS Snapshots**

The Time Slider snapshot tool automatically snapshots ZFS file systems and allows you to browse and recover snapshots of file systems. This tool provides the following features:

- SMF snapshot service instances that schedule reoccurring snapshots
- Automatic snapshots are scheduled from the `zfssnap crontab`
- Older snapshots are removed based on a predefined percentage of file system space used
- Browse and recover files from snapshots by using the GNOME file manager

You will need to add yourself to the `zfssnap` role to use the GNOME file manager to modify Time Slider behavior.

When the Time Slider tool is enabled, ZFS file system snapshots are created based on the following criteria:

- **frequent** snapshots every 15 mins, keeping 4 snapshots
- **hourly** snapshots every hour, keeping 24 snapshots
- **daily** snapshots every day, keeping 31 snapshots
- **weekly** snapshots every week, keeping 7 snapshots
- **monthly** snapshots every month, keeping 12 snapshots
How to Manage Automatic ZFS Snapshots

1 Become an administrator.

2 Enable the Time Slider service, which is disabled by default.
   You can enable or disable these services from the command line or from the
   System->Preferences->Time Slider Setup menu. From this menu, you can also customize which
   ZFS file systems to snapshot and adjust the file system capacity setting for when snapshots are
   removed.

3 Review the default automatic snapshot service instances that are started when the Time Slider
   service is enabled.

   # svc | grep auto-snapshot
   online Oct_22 svc:/system/filesystem/zfs/auto-snapshot:frequent
   online Oct_22 svc:/system/filesystem/zfs/auto-snapshot:hourly
   online Oct_22 svc:/system/filesystem/zfs/auto-snapshot:weekly
   online Oct_22 svc:/system/filesystem/zfs/auto-snapshot:monthly
   online Oct_22 svc:/system/filesystem/zfs/auto-snapshot:daily

4 Confirm that automatic snapshots are created.
   For example:

   # zfs list -t snapshot
   NAME USED AVAIL REFER MOUNTPOINT
   rpool@zfs-auto-snap:weekly-2008-11-13-15:39 0 - 46.5K -
   rpool@zfs-auto-snap:daily-2008-11-13-15:39 0 - 46.5K -
   rpool@zfs-auto-snap:hourly-2008-11-13-15:39 0 - 46.5K -
   rpool@zfs-auto-snap:frequent-2008-11-13-15:39 0 - 46.5K -
   rpool/ROOT@zfs-auto-snap:weekly-2008-11-13-15:39 0 - 18K -
   rpool/ROOT@zfs-auto-snap:daily-2008-11-13-15:39 0 - 18K -
   rpool/ROOT@zfs-auto-snap:hourly-2008-11-13-15:39 0 - 18K -
   rpool/ROOT@zfs-auto-snap:frequent-2008-11-13-15:39 0 - 18K -

5 Disable or enable specific automatic snapshot services for the top-level dataset and all
   descendent datasets from the command line.
   For example:

   # zfs set com.sun:auto-snapshot=false rpool
   # zfs set com.sun:auto-snapshot=true rpool/ROOT/opensolaris

6 Choose to only take snapshots under a given schedule for a dataset and all direct descendent
   datasets from the command line.
   For example:

   # zfs set com.sun:auto-snapshot=false rpool/export
   # zfs set com.sun:auto-snapshot:weekly=true rpool/export
7 Change the frequency of a given snapshot schedule from the command line.
   For example:
   
   # svccfg -s svc:/system/filesystem/zfs/auto-snapshot:frequent setprop zfs/period = 30
   # svccfg -s svc:/system/filesystem/zfs/auto-snapshot:frequent refresh
   # svcadm restart svc:/system/filesystem/zfs/auto-snapshot:frequent

8 Disable automatic snapshot services for the swap and dump volumes, from the command line, if necessary.
   For example:
   
   # zfs set com.sun:auto-snapshot=false rpool/dump
   # zfs set com.sun:auto-snapshot=false rpool/swap

9 Remove a range of unwanted snapshots, from the command line, if necessary.
   For example, remove all automatic snapshots in the bash shell, as follows:
   
   for s in 'zfs list -H -o name -t snapshot | grep @zfs-auto-snap'; do zfs destroy $s; done

\[\textbf{How to Recover Automatic ZFS Snapshots (GNOME File Manager)}\]

You can browse and recover snapshots from the GNOME desktop file manager, which is accessed as follows:

- Click the Desktop icon under the Places tab
- Click the clock icon with the slider from the Desktop - File Browser

1 Browse your snapshots by opening any folder in the GNOME file manager.
   If the Restore icon (clock with slider) is enabled, snapshots of this directory are available.

2 Click on the clock icon to access snapshot navigation features.
   Directly below the location: URL section is a one line description of the following information:
   
   - The timeline or date and time that the snapshot was taken
   - The location in the backup timeline of the current snapshot
   - The number of snapshots available for this directory and the space consumed by ZFS snapshots

3 Drag the slider into the past to retrieve previous versions of your files.
   You can open all your files in read-only mode. Or, you can browse in list view mode (as oppose to icon view) so that a restore information column automatically appears. This column gives you contextual information about either of the following:
   
   - The file version number, if you browse in the current or latest version of a directory
   - The difference, if any, between the file snapshot and the latest version of the file

4 Recover files from an automatic snapshot from either of the following methods:
Drag and drop a snapshot into another file manager window
Right click the mouse and select the restore to Desktop icon. Then, copy and paste the selected snapshot to the present time.

**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 169.

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 205
- “Destroying a ZFS Clone” on page 206
- “Replacing a ZFS File System With a ZFS Clone” on page 206

**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.
Destroying a ZFS Clone

ZFS clones are destroyed by using the `zfs destroy` command. For example:

```bash
# 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 206.

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.

```bash
# 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:

```bash
# zfs rename tank/test/productA tank/test/productAlegacy
# zfs rename tank/test/productAbeta tank/test/productA
# zfs list -r tank/test
```
Optionally, you can remove the legacy file system. For example:

```
# zfs destroy tank/test/productAlegacy
```

## Sending and Receiving ZFS Data

The `zfs send` command creates a stream representation of a snapshot that is written to standard output. By default, a full stream is generated. You can redirect the output to a file or to a different system. The `zfs receive` command creates a snapshot whose contents are specified in the stream that is provided on standard input. If a full stream is received, a new file system is created as well. You can send ZFS snapshot data and receive ZFS snapshot data and file systems with these commands. See the examples in the next section.

- “Saving ZFS Data With Other Backup Products” on page 208
- “Sending a ZFS Snapshot” on page 208
- “Receiving a ZFS Snapshot” on page 209
- “Applying Different Property Values to a ZFS Snapshot Stream” on page 210
- “Sending and Receiving Complex ZFS Snapshot Streams” on page 211
- “Remote Replication of ZFS Data” on page 214

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 195.

- **Saving snapshots** – Use the `zfs send` and `zfs receive` commands to send and receive a ZFS snapshot. You can save incremental changes between snapshots, but you cannot restore files individually. You must restore the entire file system snapshot. These commands do not provide a complete backup solution for saving your ZFS data.

- **Remote replication** – Use the `zfs send` and `zfs receive` commands to copy a file system from one system to another system. This process is different from a traditional volume management product that might mirror devices across a WAN. No special configuration or hardware is required. The advantage of replicating a ZFS file system is that you can re-create a file system on a storage pool on another system, and specify different levels of configuration for the newly created pool, such as RAID-Z, but with identical file system data.
- **Archive utilities** – Save ZFS data with archive utilities such as `tar`, `cpio`, and `pax` or third-party backup products. Currently, both `tar` and `cpio` translate NFSv4-style ACLs correctly, but `pax` does not.

### Saving ZFS Data With Other Backup Products

In addition to the `zfs send` and `zfs receive` commands, you can also use archive utilities, such as the `tar` and `cpio` commands, to save ZFS files. These utilities save and restore ZFS file attributes and ACLs. Check the appropriate options for both the `tar` and `cpio` commands.

For up-to-date information about issues with ZFS and third-party backup products, see the [Oracle Solaris 11 Express Release Notes](http://hub.opensolaris.org/bin/view/Community+Group+zfs/faq/#backupsoftware) 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/dana@snap1 | zfs recv spool/ds01
```

You can use `zfs recv` as an alias for the `zfs receive` command.

If you are sending the snapshot stream to a different system, pipe the `zfs send` output through the `ssh` command. For example:

```
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 snapshot:

```
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

### Applying Different Property Values to a ZFS Snapshot Stream

You can send a ZFS snapshot stream with a certain file system property, but you can specify a different local property value when the snapshot stream is received. Or, you can specify that the original property value is used when the snapshot stream is received to recreate the original file system. In addition, you can also disable a file system property when the snapshot stream is received.

In some cases, file system properties in a send stream might not apply to the receiving file system or the local file system properties might interfere with a restore, such as a mountpoint property value.

For example, the tank/data file system has the compression property is disabled. A snapshot of the tank/data file system is sent with properties (-p) to a backup pool and is received with the compression property enabled.

```
# zfs get compression tank/data
NAME PROPERTY VALUE SOURCE
tank/data compression off default
# zfs snapshot tank/data@snap1
# zfs send -p tank/data@snap1 | zfs recv -o compression=on -d bpool
# zfs get -o all compression bpool/data
NAME PROPERTY VALUE RECEIVED SOURCE
bpool/data compression on off local
```

In the above example, the compression property is enabled when the snapshot was received into the bpool, so for bpool/data, the compression value is on.

If this snapshot stream is sent to a new pool, restorepool, for recovery purposes, you might want to keep all the original snapshot properties. In this case, you would use the zfs send -b option to restore the original snapshot properties. For example:

```
# zfs send -b bpool/data@snap1 | zfs recv -d restorepool
# zfs get -o all compression restorepool/data
```
In the above example, the compression value is off, which represents the snapshot compression value from the original tank/data file system.

If you have a local file system property value in a snapshot stream and you want to disable the property when it is received, use the zfs receive -x command. For example, send a recursive snapshot stream of home directory file systems with all file system properties reserved to a backup pool, but without the quota property values.

```
# zfs send -R tank/home@1020 | zfs recv -x quota bpool/home
```

```
NAME  PROPERTY  VALUE  RECIPE  SOURCE
bpool/home quota none  default
bpool/home@1020 quota - -
bpool/home/cindys quota none  local
bpool/home/cindys@1020 quota - -
bpool/home/tom quota none  local
bpool/home/tom@1020 quota - -
```

If the recursive snapshot was not received with the -x option, the quota property would be set in the received file systems.

```
# zfs send -R tank/home@1020 | zfs recv bpool/home
# zfs get -r quota bpool/home
```

```
NAME  PROPERTY  VALUE  SOURCE
bpool/home quota none  default
bpool/home@1020 quota 2G  received
bpool/home/cindys quota 4G  received
bpool/home/tom@1020 quota - -
```

---

**Sending and Receiving Complex ZFS Snapshot Streams**

This section describes how to use the zfs send -I and -R options to send and receive more complex snapshot streams.

Keep the following points in mind when sending and receiving complex ZFS snapshot streams:

- Use the zfs send -I option to send all incremental streams from one snapshot to a cumulative snapshot. Or, use this option to send an incremental stream from the original snapshot to create a clone. The original snapshot must already exist on the receiving side to accept the incremental stream.
- Use the zfs send -R option to send a replication stream of all descendent file systems. When the replication stream is received, all properties, snapshots, descendent file systems, and clones are preserved.
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
```

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
```
EXAMPLE 7–1  Sending and Receiving Complex ZFS Snapshot Streams  (Continued)

# 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 descendant file systems, up to the named snapshot. When this stream is received, all properties, snapshots, descendant 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 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

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
# 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
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.
This chapter provides information about using access control lists (ACLs) to protect your ZFS files by providing more granular permissions than the standard UNIX permissions.

The following sections are provided in this chapter:
- “New Solaris ACL Model” on page 215
- “Setting ACLs on ZFS Files” on page 221
- “Setting and Displaying ACLs on ZFS Files in Verbose Format” on page 224
- “Setting and Displaying ACLs on ZFS Files in Compact Format” on page 234
- “Applying Special Attributes to ZFS Files” on page 238

**New Solaris ACL Model**

Previous versions of Solaris supported an ACL implementation that was primarily based on the POSIX-draft ACL specification. The POSIX-draft based ACLs are used to protect UFS files and are translated by versions of NFS prior to NFSv4.

With the introduction of NFSv4, a new ACL model fully supports the interoperability that NFSv4 offers between UNIX and non-UNIX clients. The new ACL implementation, as defined in the NFSv4 specification, provides much richer semantics that are based on NT-style ACLs.

The main differences of the new ACL model are as follows:
- Based on the NFSv4 specification and similar to NT-style ACLs.
- Provide much more granular set of access privileges. For more information, see Table 8–2.
- Set and displayed with the chmod and ls commands rather than the setfacl and getfacl commands.
- Provide richer inheritance semantics for designating how access privileges are applied from directory to subdirectories, and so on. For more information, see “ACL Inheritance” on page 220.
Both ACL models provide more fine-grained access control than is available with the standard file permissions. Much like POSIX-draft ACLs, the new ACLs are composed of multiple Access Control Entries (ACEs).

POSIX-draft style ACLs use a single entry to define what permissions are allowed and what permissions are denied. The new ACL model has two types of ACEs that affect access checking: ALLOW and DENY. As such, you cannot infer from any single ACE that defines a set of permissions whether or not the permissions that weren’t defined in that ACE are allowed or denied.

Translation between NFSv4-style ACLs and POSIX-draft ACLs is as follows:

- If you use any ACL-aware utility, such as the cp, mv, tar, cpio, or rcp commands, to transfer UFS files with ACLs to a ZFS file system, the POSIX-draft ACLs are translated into the equivalent NFSv4-style ACLs.
- Some NFSv4-style ACLs are translated to POSIX-draft ACLs. You see a message similar to the following if an NFSv4-style ACL isn’t translated to a POSIX-draft ACL:
  
  ```
  # cp -p filea /var/tmp
  cp: failed to set acl entries on /var/tmp/filea
  ```

- If you create a UFS tar or cpio archive with the preserve ACL option (tar -p or cpio -P) on a system that runs a current Solaris release, you will lose the ACLs when the archive is extracted on a system that runs a previous Solaris release.

  All of the files are extracted with the correct file modes, but the ACL entries are ignored.

- You can use the ufsrestore command to restore data into a ZFS file system. If the original data includes POSIX-style ACLs, they are converted to NFSv4-style ACLs.

- If you attempt to set an NFSv4-style ACL on a UFS file, you see a message similar to the following:

  ```
  chmod: ERROR: ACL type's are different
  ```

- If you attempt to set a POSIX-style ACL on a ZFS file, you will see messages similar to the following:

  ```
  # getfacl filea
  File system doesn't support aclent_t style ACL's.
  See acl(5) for more information on Solaris ACL support.
  ```

For information about other limitations with ACLs and backup products, see “Saving ZFS Data With Other Backup Products” on page 208.

### Syntax Descriptions for Setting ACLs

Two basic ACL formats are provided as follows:

#### Syntax for Setting Trivial ACLs

```bash
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 ...


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 ...


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

The 2 or the index-ID designation in this example identifies the ACL entry in the larger ACL, which might have multiple entries for owner, specific UIDs, group, and everyone. You can
specify the index-ID with the chmod command to identify which part of the ACL you want to modify. For example, you can identify index ID 3 as A3 to the chmod command, similar to the following:

```chmod A3=user:venkman:read_acl:allow filename```

ACL entry types, which are the ACL representations of owner, group, and other, are described in the following table.

<table>
<thead>
<tr>
<th>ACL Entry Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner@</td>
<td>Specifies the access granted to the owner of the object.</td>
</tr>
<tr>
<td>group@</td>
<td>Specifies the access granted to the owning group of the object.</td>
</tr>
<tr>
<td>everyone@</td>
<td>Specifies the access granted to any user or group that does not match any other ACL entry.</td>
</tr>
<tr>
<td>user</td>
<td>With a username, specifies the access granted to an additional user of the object. Must include the ACL-entry-ID, which contains a username or user ID. If the value is not a valid numeric UID or username, the ACL entry type is invalid.</td>
</tr>
<tr>
<td>group</td>
<td>With a group name, specifies the access granted to an additional group of the object. Must include the ACL-entry-ID, which contains a groupname or group ID. If the value is not a valid numeric GID or group name, the ACL entry type is invalid.</td>
</tr>
</tbody>
</table>

ACL access privileges are described in the following table.

<table>
<thead>
<tr>
<th>Access Privilege</th>
<th>Compact Access Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_file</td>
<td>w</td>
<td>Permission to add a new file to a directory.</td>
</tr>
<tr>
<td>add_subdirectory</td>
<td>p</td>
<td>On a directory, permission to create a subdirectory.</td>
</tr>
<tr>
<td>append_data</td>
<td>p</td>
<td>Placeholder. Not currently implemented.</td>
</tr>
<tr>
<td>delete</td>
<td>d</td>
<td>Permission to delete a file.</td>
</tr>
<tr>
<td>delete_child</td>
<td>D</td>
<td>Permission to delete a file or directory within a directory.</td>
</tr>
<tr>
<td>execute</td>
<td>x</td>
<td>Permission to execute a file or search the contents of a directory.</td>
</tr>
<tr>
<td>list_directory</td>
<td>r</td>
<td>Permission to list the contents of a directory.</td>
</tr>
<tr>
<td>read_acl</td>
<td>c</td>
<td>Permission to read the ACL (ls).</td>
</tr>
</tbody>
</table>
TABLE 8–2  ACL Access Privileges (Continued)

<table>
<thead>
<tr>
<th>Access Privilege</th>
<th>Compact Access Privilege</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_attributes</td>
<td>a</td>
<td>Permission to read basic attributes (non-ACLs) of a file. Think of basic attributes as the stat level attributes. Allowing this access mask bit means the entity can execute <code>ls(1)</code> and <code>stat(2)</code>.</td>
</tr>
<tr>
<td>read_data</td>
<td>r</td>
<td>Permission to read the contents of the file.</td>
</tr>
<tr>
<td>read_xattr</td>
<td>R</td>
<td>Permission to read the extended attributes of a file or perform a lookup in the file’s extended attributes directory.</td>
</tr>
<tr>
<td>synchronize</td>
<td>s</td>
<td>Placeholder. Not currently implemented.</td>
</tr>
<tr>
<td>write_xattr</td>
<td>W</td>
<td>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.</td>
</tr>
<tr>
<td>write_data</td>
<td>w</td>
<td>Permission to modify or replace the contents of a file.</td>
</tr>
<tr>
<td>write_attributes</td>
<td>A</td>
<td>Permission to change the times associated with a file or directory to an arbitrary value.</td>
</tr>
<tr>
<td>write_acl</td>
<td>C</td>
<td>Permission to write the ACL or the ability to modify the ACL by using the <code>chmod</code> command.</td>
</tr>
<tr>
<td>write_owner</td>
<td>o</td>
<td>Permission to change the file’s owner or group. Or, the ability to execute the <code>chown</code> or <code>chgrp</code> commands on the file. Permission to take ownership of a file or permission to change the group ownership of the file to a group of which the user is a member. If you want to change the file or group ownership to an arbitrary user or group, then the PRIV_FILE_CHOWN privilege is required.</td>
</tr>
</tbody>
</table>

New Solaris ACL Model

ZFS ACL Sets

The following ACL combinations can be applied in an ACL set rather than setting individual permissions separately. The following ACL sets are available.

<table>
<thead>
<tr>
<th>ACL Set Name</th>
<th>Included ACL Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>full_set</td>
<td>All permissions</td>
</tr>
<tr>
<td>modify_set</td>
<td>all permissions except write_acl and write_owner</td>
</tr>
<tr>
<td>read_set</td>
<td>read_data, read_attributes, read_xattr, and read_acl</td>
</tr>
</tbody>
</table>
### ACL Set Name

<table>
<thead>
<tr>
<th>ACL Set Name</th>
<th>Included ACL Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>write_set</td>
<td>write_data, append_data, write_attributes, and write_xattr</td>
</tr>
</tbody>
</table>

These ACL sets are predefined and cannot be modified.

## ACL Inheritance

The purpose of using ACL inheritance is so that a newly created file or directory can inherit the ACLs they are intended to inherit, but without disregarding the existing permission bits on the parent directory.

By default, ACLs are not propagated. If you set a non-trivial ACL on a directory, it is not inherited to any subsequent directory. You must specify the inheritance of an ACL on a file or directory.

The optional inheritance flags are described in the following table.

### TABLE 8–3  ACL Inheritance Flags

<table>
<thead>
<tr>
<th>Inheritance Flag</th>
<th>Compact Inheritance Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_inherit</td>
<td>f</td>
<td>Only inherit the ACL from the parent directory to the directory’s files.</td>
</tr>
<tr>
<td>dir_inherit</td>
<td>d</td>
<td>Only inherit the ACL from the parent directory to the directory’s subdirectories.</td>
</tr>
<tr>
<td>inherit_only</td>
<td>i</td>
<td>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 file_inherit flag, the dir_inherit flag, or both, to indicate what to inherit.</td>
</tr>
<tr>
<td>no_propagate</td>
<td>n</td>
<td>Only inherit the ACL from the parent directory to the first-level contents of the directory, not the second-level or subsequent contents. This flag requires the file_inherit flag, the dir_inherit flag, or both, to indicate what to inherit.</td>
</tr>
<tr>
<td>-</td>
<td>N/A</td>
<td>No permission granted.</td>
</tr>
</tbody>
</table>

Currently, the following flags are only applicable to an SMB client or server.

| successful_access | S | Indicates whether an alarm or audit record should be initiated upon a successful access. This flag is used with audit or alarm ACE types. |
### Table 8–3  ACL Inheritance Flags  (Continued)

<table>
<thead>
<tr>
<th>Inheritance Flag</th>
<th>Compact Inheritance Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failed_access</td>
<td>F</td>
<td>Indicates whether an alarm or audit record should be initiated when an access fails. This flag is used with audit or alarm ACE types.</td>
</tr>
<tr>
<td>inherited</td>
<td>I</td>
<td>Indicates that an ACE was inherited.</td>
</tr>
</tbody>
</table>

In addition, you can set a default ACL inheritance policy on the file system that is more strict or less strict by using the aclinherit file system property. For more information, see the next section.

## ACL Property (aclinherit)

The ZFS file system includes the aclinherit property to determine the behavior of ACL inheritance. Values include the following:

- **discard** – For new objects, no ACL entries are inherited when a file or directory is created. The ACL on the file or directory is equal to the permission mode of the file or directory.
- **noallow** – For new objects, only inheritable ACL entries that have an access type of deny are inherited.
- **restricted** – For new objects, the write_owner and write_acl permissions are removed when an ACL entry is inherited.
- **passthrough** – When property value is set to passthrough, files are created with a mode determined by the inheritable ACEs. If no inheritable ACEs exist that affect the mode, then the mode is set in accordance to the requested mode from the application.
- **passthrough-x** – Has the same semantics as passthrough, except that when passthrough-x is enabled, files are created with the execute (x) permission, but only if execute permission is set in the file creation mode and in an inheritable ACE that affects the mode.

The default mode for the aclinherit is restricted.

### Setting ACLs on ZFS Files

As implemented with ZFS, ACLs are composed of an array of ACL entries. ZFS provides a *pure* ACL model, where all files have an ACL. Typically, the ACL is *trivial* in that it only represents the traditional UNIX owner/group/other entries.

ZFS files still have permission bits and a mode, but these values are more of a cache of what the ACL represents. As such, if you change the permissions of the file, the file’s ACL is updated accordingly. In addition, if you remove a non-trivial ACL that granted a user access to a file or
directory, that user could still have access to the file or directory because of the file or directory's permission bits that grant access to group or everyone. All access control decisions are governed by the permissions represented in a file or directory's ACL.

The primary rules of ACL access on a ZFS file are as follows:

- ZFS processes ACL entries in the order they are listed in the ACL, from the top down.
- Only ACL entries that have a “who” that matches the requester of the access are processed.
- Once an allow permission has been granted, it cannot be denied by a subsequent ACL deny entry in the same ACL permission set.
- The owner of the file is granted the write_acl permission unconditionally, even if the permission is explicitly denied. Otherwise, any permission left unspecified is denied.

In the cases of deny permissions or when an access permission is missing, the privilege subsystem determines what access request is granted for the owner of the file or for superuser. This mechanism prevents owners of files from getting locked out of their files and enables superuser to modify files for recovery purposes.

If you set a non-trivial ACL on a directory, the ACL is not automatically inherited by the directory's children. If you set a non-trivial ACL and you want it inherited to the directory's children, you have to use the ACL inheritance flags. For more information, see Table 8–3 and “Setting ACL Inheritance on ZFS Files in Verbose Format” on page 228.

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 206674 Jun 14 14:48 /tank/file.1
0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow
```

Each user category (owner@, group@, everyone@) has an ACL entry in this example.

A description of this file ACL is as follows:

- **0:owner@**: The owner can read and modify the contents of the file (read_data/write_data/append_data/read_xattr). The owner can also modify the file’s attributes such as timestamps, extended attributes, and ACLs (write_xattr/read_attributes/write_attributes/read_acl/write_acl). In addition, the owner can modify the ownership of the file (write_owner:allow).

  The synchronize access permission is not currently implemented.
1:group@ The group is granted read permissions to the file and the file’s attributes
(read_data/read_xattr/read_attributes/read_acl:allow).

2:everyone@ Everyone who is not user or group is granted read permissions to the file and
the file’s attributes (read_data/read_xattr/read_attributes/read_acl::allow).

When a new directory is created and depending on the umask value, a default directory ACL is
similar to the following:

$ ls -dv dir.1
drwxr-xr-x 2 root root 2 Jun 14 14:58 dir.1
0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/write_xattr/execute/read_attributes
/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
1:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
2:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow

A description of this directory ACL is as follows:

0:owner@ The owner can read and modify the directory contents
(list_directory/read_data/add_file/write_data/add_subdirectory
/append_data), search the contents (execute), and read and modify the file’s
attributes such as timestamps, extended attributes, and ACLs
(/read_xattr/write_xattr/read_attributes/write_attributes/read_acl/write_acl). In addition, the owner can modify the ownership of the directory
(write_owner::allow).

The synchronize access permission is not currently implemented.

1:group@ The group can list and read the directory contents and the directory’s
attributes. In addition, the group has execute permission to search the
directory contents (list_directory/read_data/read_xattr/execute/read_attributes
/read_acl).

2:everyone@ Everyone who is not user or group is granted read and execute permissions to
the directory contents and the directory’s attributes
(list_directory/read_data/read_xattr/execute/read_attributes/read_acl::allow).
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 216.

- Adding ACL entries
  - Adding an ACL entry for a user
    ```
    % chmod A+ acl-specification filename
    ```
  - Adding an ACL entry by `index-ID`
    ```
    % chmod A index-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 A index-ID= acl-specification filename
  ```

- Removing ACL entries
  - Removing an ACL entry by `index-ID`
    ```
    % chmod A index-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 206674 Jun 15 09:56 file.1
 0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
   /read_attributes/write_attributes/read_acl/write_acl/write_owner
    /synchronize:allow
 1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
 2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
   :allow
```

For information about using the compact ACL format, see "Setting and Displaying ACLs on ZFS Files in Compact Format" on page 234.

**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`:
Modifying Trivial ACLs on ZFS Files

# ls -v file.1
-rw-r--r-- 1 root root 206674 Jun 14 10:54 file.1
0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
   /read_attributes/write_attributes/read_acl/write_acl/write_owner
   /synchronize:allow
1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
   :allow

In the following example, write_data permissions are granted for group@.

# chmod A1=group@:read_data/write_data:allow file.1
# ls -v file.1
-rw-rw-r-- 1 root root 206674 Jun 14 14:45 file.1
0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
   /read_attributes/write_attributes/read_acl/write_acl/write_owner
   /synchronize:allow
1:group@:read_data/write_data:allow
2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
   :allow

In the following example, permissions on file.1 are set back to 644.

# chmod 644 file.1
# ls -v file.1
0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
   /read_attributes/write_attributes/read_acl/write_acl/write_owner
   /synchronize:allow
1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
   :allow

Setting Non-Trivial ACLs on ZFS Files

This section provides examples of setting and displaying non-trivial ACLs.

In the following example, read_data/execute permissions are added for the user gozer on the test.dir directory.

# chmod A+user:gozer:read_data/execute:allow test.dir
# ls -dv test.dir
drwxr-xr-x+ 2 root root 2 Jun 15 10:01 test.dir
0:user:gozer:list_directory/read_data/execute:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
   /append_data/read_xattr/write_xattr/execute/read_attributes
   /read_acl/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
   /read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
   /read_acl/synchronize:allow

In the following example, read_data/execute permissions are removed for user gozer.
EXAMPLE 8–2  Setting Non-Trivial ACLs on ZFS Files  (Continued)

# chmod A0- test.dir
# ls -dv test.dir
drwxr-xr-x 2 root root 2 Jun 15 10:01 test.dir
  0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
    /append_data/read_xattr/write_xattr/execute/read_attributes
    /write_attributes/read_acl/write_acl/write_owner/synchronize:allow
  1:group@:list_directory/read_data/execute/read_attributes
    /read_acl/synchronize:allow
  2:everyone@:list_directory/read_data/execute/read_attributes
    /read_acl/synchronize:allow

EXAMPLE 8–3  ACL Interaction With Permissions on ZFS Files

These ACL examples illustrate the interaction between setting ACLs and then changing the file or directory's permission bits.

In the following example, a trivial ACL exists on file.2:

# ls -v file.2
-rw-r--r-- 1 root root 28092 Jun 15 10:08 file.2
  0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
    /read_attributes/write_attributes/read_acl/write_acl/write_owner
    /synchronize:allow
  1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
  2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow

In the following example, ACL allow permissions are removed from everyone@.

# chmod A2- file.2
# ls -v file.2
-rw-r----- 1 root root 28092 Jun 15 10:08 file.2
  0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
    /read_attributes/write_attributes/read_acl/write_acl/write_owner
    /synchronize:allow
  1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
  2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow

In this output, the file's permission bits are reset from 644 to 640. Read permissions for everyone@ have been effectively removed from the file's permissions bits when the ACL allow permissions are removed for everyone@.

In the following example, the existing ACL is replaced with read_data/write_data permissions for everyone@.

# chmod A=everyone@:read_data/write_data:allow file.3
# ls -v file.3
-rw-rw-rw- 1 root root 2380 Jun 15 10:17 file.3
  0:everyone@:read_data/write_data:allow
EXAMPLE 8–3  ACL Interaction With Permissions on ZFS Files  (Continued)

In this output, the chmod syntax effectively replaces the existing ACL with permissions to read/write permissions for owner, group, and everyone@. In this model, everyone@ specifies access to any user or group. Since no owner@ or group@ ACL entry exists to override the permissions for owner and group, the permission bits are set to 666.

In the following example, the existing ACL is replaced with read permissions for user gozer.

```bash
# chmod A=user:gozer:read_data:allow file.3
# ls -v file.3
----------+ 1 root root 2380 Jun 15 10:17 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:

```bash
# chmod 655 file.3
# ls -v file.3
-rw-r-xr-x 1 root root 2380 Jun 15 10:17 file.3
0:owner@:execute:deny
1:owner@:read_data/append_data/read_xattr/write_xattr
/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:read_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow
```

EXAMPLE 8–4  Restoring Trivial ACLs on ZFS Files

You can use the chmod command to remove all non-trivial ACLs on a file or directory.

In the following example, two non-trivial ACEs exist on test5.dir.

```bash
# ls -dv test5.dir
drwxr-xr-x+ 2 root root 2 Jun 15 10:24 test5.dir
0:user:lp:read_data/file_inherit:deny
1:user:gozer:read_data/file_inherit:deny
2:owner@:list_directory/add_file/write_data/append_data/read_xattr/write_xattr/execute/read_attributes/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
3:group@:list_directory/test5.dir/execute/read_attributes/write_attributes/read_acl/synchronize:allow
4:everyone@:list_directory/test5.dir/execute/read_attributes/write_attributes/read_acl/synchronize:allow
```
EXAMPLE 8–4  Restoring Trivial ACLs on ZFS Files  

In the following example, the non-trivial ACLs for users gozer and lp are removed. The remaining ACL contains the default values for owner@, group@, and everyone@.

```
# chmod A- test5.dir
# ls -dv test5.dir
drwxr-xr-x  2 root root  2 Jun 15 10:24 test5.dir
  0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
      /append_data/read_xattr/write_xattr/execute/read_attributes
      /write_attributes/read_acl/write_acl/write_owner/synchronize:allow
  1:group@:list_directory/read_data/read_xattr/write_xattr/execute/read_attributes
      /read_acl/synchronize:allow
  2:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
      /read_acl/synchronize:allow
```

EXAMPLE 8–5  Applying an ACL Set to ZFS Files

ACL sets are available so that you do not have to apply ACL permissions separately. For a description of ACL sets, see “ZFS ACL Sets” on page 219.

For example, you can apply the write_set as follows:

```
# chmod A+user:otto:read_set:allow file.1
# ls -v file.1
-r--r--r--+ 1 root root 206674 Jun 15 10:34 file.1
  0:user:otto:read_data/read_xattr/read_attributes/read_acl:allow
  1:owner@:read_data/write_data/append_data/read_xattr/write_xattr/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
  2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
  3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

You can apply the write_set and read_set as follows:

```
# chmod A+user:otto:read_set/write_set:allow file.2
# ls -v file.2
-rw-r----- 1 root root 28092 Jun 15 10:08 file.2
  0:user:otto:write_data/append_data/read_xattr/write_xattr
      /read_attributes/write_acl/write_acl/write_owner/synchronize:allow
  1:owner@:write_data/append_data/read_xattr/write_xattr/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
  2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

Setting ACL Inheritance on ZFS Files inVerbose Format

You can determine how ACLs are inherited or not inherited on files and directories. By default, ACLs are not propagated. If you set a non-trivial ACL on a directory, the ACL is not inherited by any subsequent directory. You must specify the inheritance of an ACL on a file or directory.
The aclinherit property can be set globally on a file system. By default, aclinherit is set to restricted.

For more information, see “ACL Inheritance” on page 220.

EXAMPLE 8–6  Granting Default ACL Inheritance

By default, ACLs are not propagated through a directory structure.

In the following example, a non-trivial ACE of read_data/write_data/execute is applied for user gozer on test.dir.

```
# chmod A+user:gozer:read_data/write_data/execute:allow test.dir
# ls -dv test.dir
drwxr-xr-x+ 2 root  root 2 Jun 15 10:40 test.dir
0:user:gozer:list_directory/read_data/add_file/write_data/execute:allow
1:owner@:list_directory/read_data/add_file/write_data/execute:allow
 /append_data/read_xattr/write_xattr/execute/read_attributes
 /write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
 /read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
 /read_acl/synchronize:allow
```

If a test.dir subdirectory is created, the ACE for user gozer is not propagated. User gozer would only have access to sub.dir if the permissions on sub.dir granted him access as the file owner, group member, or everyone@.

```
# mkdir test.dir/sub.dir
# ls -dv test.dir/sub.dir
drwxr-xr-x 2 root  root 2 Jun 15 10:41 test.dir/sub.dir
0:owner@:list_directory/read_data/add_file/write_data/execute:allow
 /append_data/read_xattr/write_xattr/execute/read_attributes
 /write_attributes/read_acl/write_acl/write_owner/synchronize:allow
1:group@:list_directory/read_data/read_xattr/execute/read_attributes
 /read_acl/synchronize:allow
2:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
 /read_acl/synchronize:allow
```

EXAMPLE 8–7  Granting ACL Inheritance on Files and Directories

This series of examples identify the file and directory ACEs that are applied when the file_inherit flag is set.

In the following example, read_data/write_data permissions are added for files in the test.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 Jun 15 10:42 test2.dir
0:user:gozer:read_data/write_data:file_inherit:allow
1:owner@:list_directory/read_data/add_file/write_data/execute:allow
 /append_data/read_xattr/write_xattr/execute/read_attributes
 /write_attributes/read_acl/write_acl/write_owner/synchronize:allow
```
EXAMPLE 8–7  Granting ACL Inheritance on Files and Directories  (Continued)

/append_data/read_xattr/write_xattr/execute/read_attributes
/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow

In the following example, user gozer’s permissions are applied on the newly created test2.dir/file.2 file. The ACL inheritance granted, read_data:file_inherit:allow, means user gozer can read the contents of any newly created file.

```
# touch test2.dir/file.2
# ls -v test2.dir/file.2
-rw-r--r--+ 1 root root 0 Jun 15 10:43 test2.dir/file.2
 0:user:gozer:read_data:inherited:allow
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_xattr/read_attributes/read_acl/synchronize:allow
```

Because the aclinherit property for this file system is set to the default mode, restricted, user gozer does not have write_data permission on file.2 because the group permission of the file does not allow it.

Note the inherit_only permission, which is applied when the file_inherit or dir_inherit flags are set, is used to propagate the ACL through the directory structure. As such, user gozer is only granted or denied permission from everyone@ permissions unless he is the file owner or is a member of the file’s group owner. For example:

```
# mkdir test2.dir/subdir.2
# ls -dv test2.dir/subdir.2
drwxr-xr-x+ 2 root root 2 Jun 15 10:48 test2.dir/subdir.2
 0:user:gozer:list_directory/read_data/add_file/write_data:file_inherit
/inherit_only/inherited:allow
1:owner@:list_directory/read_data/add_file/write_data:add_subdirectory
/append_data/read_xattr/write_xattr/execute/read_attributes
/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
```

The following series of examples identify the file and directory ACLs that are applied when both the file_inherit and dir_inherit flags are set.

In the following example, user gozer is granted read, write, and execute permissions that are inherited for newly created files and directories.
EXAMPLE 8-7  Granting ACL Inheritance on Files and Directories (Continued)

```
# chmod A+user:gozer:read_data/write_data/execute:file_inherit/dir_inherit:allow
# ls -dv test3.dir
# touch test3.dir/file.3
# ls -v test3.dir/file.3
# mkdir test3.dir/subdir.1
# ls -dv test3.dir/subdir.1
```

The inherited text in the output below is an informational message that indicates that the ACE is inherited.

```
The inherited text in the output below is an informational message that indicates that the ACE is inherited.
```

```
# chmod A+user:gozer:read_data/write_data/execute:file_inherit/no_propagate:allow
# test4.dir
```

In these examples, because the permission bits of the parent directory for group@ and everyone@ deny write and execute permissions, user gozer is denied write and execute permissions. The default acl_inherit property is restricted, which means that write_data and execute permissions are not inherited.

In the following example, user gozer is granted read, write, and execute permissions that are inherited for newly created files, but are not propagated to subsequent contents of the directory.

```
# chmod A+user:gozer:read_data/write_data/execute:file_inherit/no_propagate:allow
```

Setting and Displaying ACLs on ZFS Files in Verbose Format
Granting ACL Inheritance on Files and Directories (Continued)

```bash
# ls -dv test4.dir
-rwxr-xr-x+ 2 root root 2 Jun 15 11:54 test4.dir
0:user:gozer:list_directory/read_data/add_file/write_data/execute
   :file inherit/no propagate:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
   /append_data/read_xattr/write_xattr/execute/read_attributes
   /write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
   /read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
   /read_acl/synchronize:allow
```

Asthe following example illustrates, when a new subdirectory is created, user gozer’s read_data/write_data/execute permission for files are not propagated to the new sub4.dir directory.

```bash
# mkdir test4.dir/sub4.dir
# ls -dv test4.dir/sub4.dir
-rwxr-xr-x 2 root root 2 Jun 15 11:55 test4.dir/sub4.dir
0:owner@:list_directory/read_data/add_file/write_data/append_data/read_xattr/write_xattr/execute/read_attributes
   /write_attributes/read_acl/write_acl/write_owner/synchronize:allow
1:group@:list_directory/read_data/read_xattr/execute/read_attributes
   /read_acl/synchronize:allow
2:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
   /read_acl/synchronize:allow
```

As the following example illustrates, gozer’s read_data/write_data/execute permission for files is propagated to the newly created file.

```bash
# touch test4.dir/file.4
# ls -v test4.dir/file.4
-rw-r--r--+ 1 root root 0 Jun 15 11:56 test4.dir/file.4
0:user:gozer:read_data:inherited:allow
1:owner@:read_data/write_data/append_data/read_xattr/write_xattr/execute/inherit:
   attributes/write_attributes/write_acl/write_acl/write_owner/synchronize:allow
2:group@:read_data/read_xattr/execute/inherit:attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/execute/inherit:attributes/read_acl/synchronize:allow
```

**EXAMPLE 8-8 ACL Inheritance With ACL Mode Set to Pass Through**

If the aclinherit property on the tank/cindys filesystem is set to passthrough, then user gozer would inherit the ACL applied on test4.dir for the newly created file.4 as follows:

```bash
# zfs set aclinherit=passthrough tank/cindys
# touch test4.dir/file.4
# ls -v test4.dir/file.4
-rw-r--r--+ 1 root root 0 Jun 15 12:11 test4.dir/file.4
0:user:gozer:read_data:inherited:allow
1:owner@:read_data/write_data/execute:inherited:allow
```

Oracle Solaris ZFS Administration Guide • January 2011
EXAMPLE 8–8  ACL Inheritance With ACL Mode Set to Pass Through  (Continued)

/read_attributes/write_attributes/read_acl/write_acl/write_owner
/synchronize:allow
2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
:allow

This output illustrates that the read_data/write_data/execute:allow:file_inherit ACL that was set on the parent directory, test4.dir, is passed through to user gozer.

EXAMPLE 8–9  ACL Inheritance With ACL Mode Set to Discard

If the aclinherit property on a file system is set to discard, then ACLs can potentially be discarded when the permission bits on a directory change. For example:

# zfs set aclinherit=discard 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 Jun 15 12:15 test5.dir
0:user:gozer:list_directory/read_data/add_file/write_data/execute
:dir_inherit:allow
1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/read_attributes
/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
2:group@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow
3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow

If, at a later time, you decide to tighten the permission bits on a directory, the non-trivial ACL is discarded. For example:

# chmod 744 test5.dir
# ls -dv test5.dir
drwxr--r-- 2 root root 2 Jun 15 12:15 test5.dir
0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
/append_data/read_xattr/write_xattr/execute/read_attributes
/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
1:group@:list_directory/read_data/read_xattr/execute/read_attributes
/write_acl/synchronize:allow
2:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
/read_acl/synchronize:allow

EXAMPLE 8–10  ACL Inheritance With ACL Inherit Mode Set to Noallow

In the following example, two non-trivial ACLs with file inheritance are set. One ACL allows read_data permission, and one ACL denies read_data permission. This example also illustrates how you can specify two ACEs in the same chmod command.

# zfs set aclinherit=noallow tank/cindys
# chmod A+user:gozer:read_data:file_inherit:deny,user:lp:read_data:file_inherit:allow
test6.dir
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 Jun 15 12:19 test6.dir/file.6
 0:user:gozer:read_data:inherited:deny
 1:owner@:read_data/write_data/append_data/read_xattr/write_xattr/write_attributes/read_acl/write_acl/write_owner/synchronize:allow
 2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
 3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow

The compact ACL output is described as follows:

owner@ The owner can read and modify the contents of the file (rw=read_data/write_data, p=append_data). The owner can also modify the file's attributes such as timestamps, extended attributes, and ACLs.
(a=read_attributes, A=write_xattr, R=read_xattr, W=write_attributes, c=read_acl, C=write_acl). In addition, the owner can modify the ownership of the file (o=write_owner).

The synchronize access permission is not currently implemented.

group@ The group is granted read permissions to the file (r=read_data) and the file's attributes (a=read_attributes, R=read_xattr, c=read_acl).

The synchronize access permission is not currently implemented.

everyone@ Everyone who is not user or group is granted read permissions to the file and the file's attributes (r=read_data, a=append_data, R=read_xattr, c=read_acl, and s=synchronize).

The synchronize access permission is not currently implemented.

Compact ACL format provides the following advantages over verbose ACL format:

- Permissions can be specified as positional arguments to the chmod command.
- The hyphen (-) characters, which identify no permissions, can be removed and only the required letters need to be specified.
- Both permissions and inheritance flags are set in the same fashion.

For information about using the verbose ACL format, see "Setting and Displaying ACLs on ZFS Files in Verbose Format" on page 224.

**EXAMPLE 8-11 Setting and Displaying ACLs in Compact Format**

In the following example, a trivial ACL exists on file.1:

```bash
# ls -V file.1
-rw-r--r-- 1 root root 206674 Jun 15 10:34 file.1
  owner@:rw-p--aARWcCos:-------:allow
  group@:r-----a-R-c--s:-------:allow
  everyone@:r-----a-R-c--s:-------:allow
```

In this example, read_data/execute permissions are added for the user gozer on file.1.

```bash
# chmod A+user:gozer:rx:allow file.1
# ls -V file.1
-rw-r--r--+ 1 root root 206674 Jun 15 10:34 file.1
  user:gozer:r-x------------:allow
  owner@:rw-p--aARWcCos:-------:allow
  group@:r-----a-R-c--s:-------:allow
  everyone@:r-----a-R-c--s:-------:allow
```

In the following example, user gozer is granted read, write, and execute permissions that are inherited for newly created files and directories by using the compact ACL format.
EXAMPLE 8–11  Setting and Displaying ACLs in Compact Format  (Continued)

```bash
# chmod A+user:gozer:rwx:fd:allow dir.2
# ls -dV dir.2
drwxr-xr-x+ 2 root root 2 Jun 15 13:14 dir.2
user:gozer:rwx----------:fd-----:allow
    owner@:rwxp--aARWcCos:-------:allow
    group@:r-x---a-R-c--s:-------:allow
everyone@:r-x---a-R-c--s:-------:allow
```

You can also cut and paste permissions and inheritance flags from the `ls -V` output into the compact `chmod` format. For example, to duplicate the permissions and inheritance flags on `dir.2` for user `gozer` to user `cindys` on `dir.2`, copy and paste the permission and inheritance flags (`rwx----------:fd-----:allow`) into your `chmod` command. For example:

```bash
# chmod A+user:cindys:rwx----------:fd-----:allow dir.2
# ls -dV dir.2
drwxr-xr-x+ 2 root root 2 Jun 16 12:47 dir.2
user:cindys:rwx----------:fd-----:allow
    user:gozer:rwx----------:fd-----:allow
    owner@:rwxp--aARWcCos:-------:allow
    group@:r-x---a-R-c--s:-------:allow
everyone@:r-x---a-R-c--s:-------:allow
```

EXAMPLE 8–12  ACL Inheritance With ACL Inherit Mode Set to Pass Through

A file system that has the aclinherit property set to passthrough inherits all inheritable ACL entries without any modifications made to the ACL entries when they are inherited. When this property is set to passthrough, files are created with a permission mode that is determined by the inheritable ACEs. If no inheritable ACEs exist that affect the permission mode, then the permission mode is set in accordance to the requested mode from the application.

The following examples use compact ACL syntax to show how to inherit permission bits by setting aclinherit mode to passthrough.

In this example, an ACL is set on `test1.dir` to force inheritance. The syntax creates an owner@, group@, and everyone@ ACL entry for newly created files. Newly created directories inherit an @owner, group@, and everyone@ ACL entry.

```bash
# 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 -dV test1.dir
drwxrwx---+ 2 root root 2 Jun 15 13:17 test1.dir
    owner@:rwxpdDaARWcCos:fd-----:allow
    group@:rwxp----------:fd-----:allow
everyone@:--------------:fd-----:allow
```
In this example, a newly created file inherits the ACL that was specified to be inherited to newly created files.

```bash
# cd test1.dir
# touch file.1
# ls -V file.1
-rwxrwx---+ 1 root root 0 Jun 15 13:19 file.1
       owner@:rwxpdDaARWcCos:------I:allow
       group@:rwxp------------:------I:allow
       everyone@:--------------:------I:allow
```

In this example, a newly created directory inherits both ACEs that control access to this directory as well as ACEs for future propagation to children of the newly created directory.

```bash
# mkdir subdir.1
# ls -dV subdir.1
drwxrwx---+ 2 root root 2 Jun 15 13:20 subdir.1
       owner@:rwxpdDaARWcCos:fd----I:allow
       group@:rwxp------------:fd----I:allow
       everyone@:--------------:fd----I:allow
```

The `fd----` 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.

```bash
# cd /tank/cindys
# mkdir test2.dir
# cd test2.dir
# touch file.2
# ls -V file.2
-rw-r--r-- 1 root root 0 Jun 15 13:21 file.2
       owner@:rw-p--aARWcCos:-------:allow
       group@:r-----a-R-c--s:-------:allow
       everyone@:r-----a-R-c--s:-------:allow
```

**Example 8-13  ACL Inheritance With ACL Inherit Mode Set to Pass Through-X**

When `aclinherit=passthrough-x` is enabled, files are created with the execute (x) permission for `owner@`, `group@`, or `everyone@`, but only if execute permission is set in the file creation mode and in an inheritable ACE that affects the mode.

The following example shows how to inherit the execute permission by setting `aclinherit` mode to `passthrough-x`.

```bash
# 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@`. 

---

**Note:**

- `owner@`: The owner of the file or directory.
- `group@`: The group to which the owner belongs.
- `everyone@`: Everyone, which includes any user that does not have a specific permission set.
- `r`: Read permission.
- `w`: Write permission.
- `x`: Execute permission.
- `p`: Propagate permission.
- `a`: All groups.
- `R`: Reset permissions.
- `c`: Change group.
- `s`: Set group.
- `-`: No permission.
- `+`: Add permission.

---

### Setting and Displaying ACLs on ZFS Files in Compact Format

**Chapter 8 • Using ACLs and Attributes to Protect Oracle Solaris ZFS Files 237**
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.

Next, an executable called t is generated by using the cc compiler in the testdir directory.

The resulting permissions are 0770 because cc requested permissions 0777, which caused the execute permission to be inherited from the owner@, group@, and everyone@ entries.

### Applying Special Attributes to ZFS Files

The following examples show how to apply and display special attributes, such as immutability or read-only access, to ZFS files.

For more information about displaying and applying special attributes, see `ls(1)` and `chmod(1)`.

**EXAMPLE 8–14**  Apply Immutability to a ZFS File

Use the following syntax to make a file immutable:

```
# chmod S+ci file.1
# echo this >>file.1
  bash: file.1: Not owner
# rm file.1
rm: cannot remove 'file.1': Not owner
```

You can display special attributes on ZFS files by using the following syntax:
EXAMPLE 8–14  Apply Immutability to a ZFS File  (Continued)

Use the following syntax to remove file immutability:

```
# chmod S-ci file.1
# ls -l/c file.1
   -rw-r--r-- 1 root root 206674 Jun 15 13:31 file.1
   {A-----im--}
```

EXAMPLE 8–15  Apply Read-Only Access to a ZFS File

The following example shows how to apply read-only access to a ZFS file.

```
# chmod S+cR file.2
# echo this >>file.2
   -bash: file.2: Not owner
```

EXAMPLE 8–16  Displaying and Removing ZFS File Attributes

You can display all special attributes with the following syntax:

```
# ls -l/v file.3
   -rw-r-xr-x 1 root root 2380 Jun 15 10:17 file.3
   {archive,nohidden,noreadonly,nosystem,noappendonly,nonodump,noimmutable,av_modified,av_quarantined,nonounlink}
# chmod S+cR file.3
# ls -l/v file.3
   -rw-r-xr-x 1 root root 2380 Jun 15 10:17 file.3
   {archive,nohidden,readonly,nosystem,noappendonly,nonodump,noimmutable,av_modified,av_quarantined,nonounlink}
```

Some of these attributes only apply in an Oracle Solaris SMB environment.

You can clear all attributes on a file. For example:

```
# chmod S-a file.3
# ls -l/v file.3
   -rw-r-xr-x 1 root root 2380 Jun 15 10:17 file.3
   {noarchive,nohidden,noreadonly,nosystem,noappendonly,nonodump,noimmutable,av_modified,av_quarantined,nonounlink}
```
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 241
- “Delegating ZFS Permissions” on page 242
- “Displaying ZFS Delegated Permissions (Examples)” on page 249
- “Delegating ZFS Permissions (Examples)” on page 246
- “Removing ZFS Delegated Permissions (Examples)” on page 251

### 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.


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.

<table>
<thead>
<tr>
<th>Permission (Subcommand)</th>
<th>Description</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>The permission to grant permissions that you have to another user.</td>
<td>Must also have the permission that is being allowed.</td>
</tr>
</tbody>
</table>
Delegating ZFS Permissions

<table>
<thead>
<tr>
<th>Permission (Subcommand)</th>
<th>Description</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>clone</td>
<td>The permission to clone any of the dataset’s snapshots.</td>
<td>Must also have the create permission and the mount permission in the original file system.</td>
</tr>
<tr>
<td>create</td>
<td>The permission to create descendent datasets.</td>
<td>Must also have the mount permission.</td>
</tr>
<tr>
<td>destroy</td>
<td>The permission to destroy a dataset.</td>
<td>Must also have the mount permission.</td>
</tr>
<tr>
<td>hold</td>
<td>The permission to hold a snapshot.</td>
<td></td>
</tr>
<tr>
<td>mount</td>
<td>The permission to mount and unmount a dataset, and create and destroy volume device links.</td>
<td></td>
</tr>
<tr>
<td>promote</td>
<td>The permission to promote a clone to a dataset.</td>
<td>Must also have the mount permission and the promote permission in the original file system.</td>
</tr>
<tr>
<td>receive</td>
<td>The permission to create descendent file systems with the zfs receive command.</td>
<td>Must also have the mount permission and the create permission.</td>
</tr>
<tr>
<td>rename</td>
<td>The permission to rename a dataset.</td>
<td>Must also have the create permission and the mount permission in the new parent.</td>
</tr>
<tr>
<td>rollback</td>
<td>The permission to roll back a snapshot.</td>
<td></td>
</tr>
<tr>
<td>send</td>
<td>The permission to send a snapshot stream.</td>
<td></td>
</tr>
<tr>
<td>share</td>
<td>The permission to share and unshare a dataset.</td>
<td></td>
</tr>
<tr>
<td>snapshot</td>
<td>The permission to create a snapshot of a dataset.</td>
<td></td>
</tr>
</tbody>
</table>

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
- atime
- canmount
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 descendents 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:

```bash
# chmod A+user:marks:add_subdirectory:fd:allow /tank
```

Then, use the `zfs allow` command to delegate create, destroy, and mount permissions. For example:

```bash
# zfs allow marks create,destroy,mount tank
```

Now, user marks can create his own file systems in the tank file system. For example:

```bash
# su marks
marks$ zfs create tank/marks
marks$ ^D
# su lp
lp$ zfs create tank/lp
```

```bash
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.

```bash
# zfs allow staff create,mount tank
# zfs allow -c create,destroy tank
# zfs allow tank
```

Create time permissions on (tank)
create,destroy
Local+Descendant permissions on (tank)
group staff create,mount

```bash
# su cindys
```

cindys$ zfs create tank/cindys

cindys$ exit

```bash
# su marks
```

marks$ zfs create tank/marks/data

marks$ exit

```bash
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
$s 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@snapp
cannot create snapshot ‘tank/marks@snapp’: 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
```
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`.

```bash
# 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:

```bash
# 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 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.
Removing ZFS Delegated Permissions (Examples)

```
# zfs allow tank
permission sets on (tank)
@myset clone,create,destroy,mount,promote,readonly,snapshot
create time permissions on (tank)
  create,destroy,mount
local+descendant permissions on (tank)
  group staff create,mount

# zfs unallow -s @myset tank
$ zfs allow tank
```

Create time permissions on (tank)
  create,destroy,mount
Local+Descendant permissions on (tank)
  group staff create,mount
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 253
- “Using ZFS on a Solaris System With Zones Installed” on page 256
- “Using ZFS Alternate Root Pools” on page 261
- “ZFS Rights Profiles” on page 262

ZFS Volumes

A ZFS volume is a dataset that represents a block device. ZFS volumes are identified as devices in the /dev/zvol/{dsk, rdsk}/pool directory.

In the following example, a 5-GB ZFS volume, 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 258.
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 and Dump Devices” on page 132.

Using a ZFS Volume as a Solaris iSCSI LUN

The Common Multiprotocol SCSI Target (COMSTAR) software framework enables you to convert any Solaris host into an iSCSI target device that can be accessed over a storage network by initiator hosts. You can create and configure a ZFS volume to be shared as an iSCSI logical unit (LUN).

First, install the COMSTAR package.

```
# pkg install solaris/storage-server
```

Create a ZFS volume to be used as an iSCSI target and then create the SCSI-block-device-based LUN. For example:
ZFS Volumes

```bash
# zfs create -V 2g tank/volumes/v2
# sbdadm create-lu /dev/zvol/rdsk/tank/volumes/v2
Created the following LU:

GUID DATA SIZE SOURCE
-------------------------------- ------------------- ----------------
600144f000144f1dafaa4c0faff20001 2147483648 /dev/zvol/rdsk/tank/volumes/v2

# sbdadm list-lu
Found 1 LU(s)

GUID DATA SIZE SOURCE
-------------------------------- ------------------- ----------------
600144f000144f1dafaa4c0faff20001 2147483648 /dev/zvol/rdsk/tank/volumes/v2
```

You can expose the LUN views to all clients or selected clients. Identify the LUN GUID and then share the LUN view. In the following example, the LUN view is shared to all clients.

```bash
# stmfdadm list-lu
LU Name: 600144F000144F1DAFAA4C0FAFF20001
# stmfdadm add-view 600144F000144F1DAFAA4C0FAFF20001
# stmfdadm list-view -l 600144F000144F1DAFAA4C0FAFF20001
View Entry: 0
  Host group : All
  Target group : All
  LUN : 0
```

The next step is to create the iSCSI targets. For information about creating the iSCSI targets, go to the following sites:

- [http://wikis.sun.com/display/OpenSolarisInfo/COMSTAR+Administration](http://wikis.sun.com/display/OpenSolarisInfo/COMSTAR+Administration)
- [http://blogs.sun.com/observatory/entry/iscsi_san_part_2_the](http://blogs.sun.com/observatory/entry/iscsi_san_part_2_the)

A ZFS volume as an iSCSI target is managed just like any other ZFS dataset except that you cannot rename the dataset, rollback a volume snapshot, or export the pool while the ZFS volumes are shared as iSCSI LUNs. You will see messages similar to the following:

```bash
# zfs rename tank/volumes/v2 tank/volumes/v1
cannot rename 'tank/volumes/v2': dataset is busy
# zpool export tank
cannot export 'tank': pool is busy
```

All iSCSI target configuration information is stored within the dataset. Like an NFS shared file system, an iSCSI target that is imported on a different system is shared appropriately.
Using ZFS on a Solaris System With Zones Installed

The following sections describe how to use ZFS on a system with Oracle Solaris zones:

- “Adding ZFS File Systems to a Non-Global Zone” on page 257
- “Delegating Datasets to a Non-Global Zone” on page 257
- “Adding ZFS Volumes to a Non-Global Zone” on page 258
- “Using ZFS Storage Pools Within a Zone” on page 258
- “Managing ZFS Properties Within a Zone” on page 259
- “Understanding the zoned Property” on page 260

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, “Oracle Solaris Zones,” in System Administration Guide: Oracle Solaris Zones, Oracle Solaris 10 Containers, and Resource Management.
Adding ZFS File Systems to a Non-Global Zone

You can add a ZFS file system as a generic file system when the goal is solely to share space with the global zone. A ZFS file system that is added to a non-global zone must have its mountpoint property set to legacy. For example, if the tank/zone/zion file system will be added to a non-global zone, set the mountpoint property in the global zone as follows:

```
# zfs set mountpoint=legacy tank/zone/zion
```

You can add a ZFS file system to a non-global zone by using the zonecfg command’s add fs subcommand.

In the following example, a ZFS file system is added to a non-global zone by a global zone administrator from the global zone:

```
# zonecfg -z zion
zonecfg:zion> add fs
zonecfg:zion:fs> set type=zfs
zonecfg:zion:fs> set special=tank/zone/zion
zonecfg:zion:fs> set dir=/export/shared
zonecfg:zion:fs> end
```

This syntax adds the ZFS file system, tank/zone/zion, to the already configured zion zone, which is mounted at /export/shared. The mountpoint property of the file system must be set to legacy, and the file system cannot already be mounted in another location. The zone administrator can create and destroy files within the file system. The file system cannot be remounted in a different location, nor can the zone administrator change properties on the file system such as atime, readonly, compression, and so on. The global zone administrator is responsible for setting and controlling properties of the file system.


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.

**Adding ZFS Volumes to a Non-Global Zone**

You can add or create a ZFS volume in a non-global zone or you can add access to a volume’s data in a non-global zone in the following ways:

- In a non-global zone, a privileged zone administrator can create a ZFS volume as descendent of a previously delegated file system. For example:

  ```
  # zfs create -V 2g tank/zone/zion/vol1
  ```

  The above syntax means that the zone administrator can manage the volume’s properties and data in the non-global zone.

- In a global zone, use the `zonecfg add dataset` subcommand and specify a ZFS volume to be added to a non-global zone. For example:

  ```
  # zonecfg -z zion
  zonecfg:zion> add dataset
  zonecfg:zion:dataset> set name=tank/volumes/vol1
  zonecfg:zion:dataset> end
  ```

  The above syntax means that the zone administrator can manage the volume’s properties and data in the non-global zone.

- In a global zone, use the `zonecfg add device` subcommand and specify a ZFS volume whose data can be accessed in a non-global zone. For example:

  ```
  # zonecfg -z zion
  zonecfg:zion> add device
  zonecfg:zion:device> set match=/dev/zvol/dsk/tank/volumes/vol2
  zonecfg:zion:device> end
  ```

  The above syntax means that only the volume data can be accessed in the non-global zone.

**Using ZFS Storage Pools Within a Zone**

ZFS storage pools cannot be created or modified within a zone. The delegated administration model centralizes control of physical storage devices within the global zone and control of virtual storage to non-global zones. Although a pool-level dataset can be added to a zone, any command that modifies the physical characteristics of the pool, such as creating, adding, or removing devices, is not allowed from within a zone. Even if physical devices are added to a zone by using the `zonecfg` command’s `add device` subcommand, or if files are used, the `zpool` command does not allow the creation of any new pools within the zone.
Managing ZFS Properties Within a Zone

After a dataset is delegated to a zone, the zone administrator can control specific dataset properties. After a dataset is delegated to a zone, all its ancestors are visible as read-only datasets, while the dataset itself is writable, as are all of its descendents. For example, consider the following configuration:

global# zfs list -Ho name
tank
tank/home
tank/data
tank/data/matrix
tank/data/zion
tank/data/zion/home

If tank/data/zion were added to a zone, each dataset would have the following properties.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Visible</th>
<th>Writable</th>
<th>Immutable Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>tank/home</td>
<td>No</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>tank/data</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>tank/data/matrix</td>
<td>No</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>tank/data/zion</td>
<td>Yes</td>
<td>Yes</td>
<td>sharefs, zoned, quota, reservation</td>
</tr>
<tr>
<td>tank/data/zion/home</td>
<td>Yes</td>
<td>Yes</td>
<td>sharefs, zoned</td>
</tr>
</tbody>
</table>

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 sharefs 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 sharefs 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
```

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:

```bash
# 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`.

```bash
# 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.
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."
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 263
- “Checking ZFS File System Integrity” on page 265
- “Resolving Problems With ZFS” on page 267
- “Repairing a Damaged ZFS Configuration” on page 272
- “Resolving a Missing Device” on page 272
- “Replacing or Repairing a Damaged Device” on page 274
- “Repairing Damaged Data” on page 283
- “Repairing an Unbootable System” on page 287

**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 264
- “Damaged Devices in a ZFS Storage Pool” on page 264
- “Corrupted ZFS Data” on page 264

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 286.

If your pool is not redundant, the risk that filesystem 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
scan: scrub in progress since Mon Jun 7 12:07:52 2010
  201M scanned out of 222M at 9.55M/s, 0h0m to go
  0 repaired, 90.44% done
config:
  NAME STATE READ WRITE CKSUM
  tank ONLINE 0 0 0
  mirror-0 ONLINE 0 0 0
  c1t0d0 ONLINE 0 0 0
  c1t1d0 ONLINE 0 0 0
errors: No known data errors
```

Only one active scrubbing operation per pool can occur at one time.

You can stop a scrubbing operation that is in progress by using the `-s` option. For example:

```
# zpool scrub -s tank
```

In most cases, a scrubbing operation to ensure data integrity should continue to completion. Stop a scrubbing operation at your own discretion if system performance is impacted by the operation.

Performing routine scrubbing guarantees continuous I/O to all disks on the system. Routine scrubbing has the side effect of preventing power management from placing idle disks in low-power mode. If the system is generally performing I/O all the time, or if power consumption is not a concern, then this issue can safely be ignored.

For more information about interpreting `zpool status` output, see “Querying ZFS Storage Pool Status” on page 105.
**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 281.

---

**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 268
- “Reviewing *zpool status* Output” on page 268
- “System Reporting of ZFS Error Messages” on page 272

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 105.

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-8000-20

**scrub:** none requested

**config:**

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>DEGRADED</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>DEGRADED</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t0d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t1d0</td>
<td>UNAVAIL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

- **pool** Identifies the name of the pool.
- **state** Indicates the current health of the pool. This information refers only to the ability of the pool to provide the necessary replication level.
- **status** Describes what is wrong with the pool. This field is omitted if no errors are found.
- **action** A recommended action for repairing the errors. This field is omitted if no errors are found.
- **see** Refers to a knowledge article containing detailed repair information. Online articles are updated more often than this guide can be updated. So, always reference them for the most up-to-date repair procedures. This field is omitted if no errors are found.
- **scrub** Identifies the current status of a scrub operation, which might include the date and time that the last scrub was completed, a scrub is in progress, or if no scrub was requested.
- **errors** Identifies known data errors or the absence of known data errors.

**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 274.

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 281.

**Scrubbing Status**

The scrub section of the `zpool status` output describes the current status of any explicit scrubbing operations. This information is distinct from whether any errors are detected on the system, though this information can be used to determine the accuracy of the data corruption error reporting. If the last scrub ended recently, most likely, any known data corruption has been discovered.

The following `zpool status scrub status` messages are provided:

- **Scrub in-progress report.** For example:
  ```
  scan: scrub in progress since Mon Jun 7 08:56:04 2010
  1.90G scanned out of 16.2G at 9.33M/s, 0h26m to go
  0 repaired, 11.69% done
  ```

- **Scrub completion message.** For example:
  ```
  scrub repaired 0 in 0h12m with 0 errors on Mon Jun 7 09:08:48 2010
  ```

- **Ongoing scrub cancellation message.** For example:
  ```
  scan: scrub canceled on Thu Jun 3 09:39:39 2010
  ```
Scrub completion messages persist across system reboots.

For more information about the data scrubbing and how to interpret this information, see "Checking ZFS File System Integrity" on page 265.

**Data Corruption Errors**

The `zpool status` command also shows whether any known errors are associated with the pool. These errors might have been found during data scrubbing or during normal operation. ZFS maintains a persistent log of all data errors associated with a pool. This log is rotated whenever a complete scrub of the system finishes.

Data corruption errors are always fatal. Their presence indicates that at least one application experienced an I/O error due to corrupt data within the pool. Device errors within a redundant pool do not result in data corruption and are not recorded as part of this log. By default, only the number of errors found is displayed. A complete list of errors and their specifics can be found by using the `zpool status -v` option. For example:

```
# zpool status -v
pool: tank
  state: UNAVAIL
status: One or more devices are faulted in response to IO failures.
  action: Make sure the affected devices are connected, then run 'zpool clear'.
  see: http://www.sun.com/msg/ZFS-8000-HC
  scrub: scrub completed after 0h0m with 0 errors on Tue Feb  2 13:08:42 2010
config:

    NAME STATE   READ WRITE CKSUM
    tank  UNAVAIL 0   0    0 insufficient replicas
   c1t0d0 ONLINE 0   0    0
   c1t1d0 UNAVAIL 4   1    0 cannot open

errors: Permanent errors have been detected in the following files:
/tank/data/aaa
/tank/data/bbb
/tank/data/ccc
```

A similar message is also displayed by `fmd` on the system console and the `/var/adm/messages` file. These messages can also be tracked by using the `fmdump` command.

For more information about interpreting data corruption errors, see "Identifying the Type of Data Corruption" on page 284.
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 fmdump 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 115.

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 fnd 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: a1f66d0-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 274.

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 93.

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:

```bash
# 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.

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 283.

### 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 94.

### 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 277
- “Devices That Cannot be Replaced” on page 277
- “Replacing a Device in a ZFS Storage Pool” on page 278
- “Viewing Resilvering Status” on page 281
Determining If a Device Can Be Replaced

If the device to be replaced is part of a redundant configuration, sufficient replicas from which to retrieve good data must exist. For example, if two disks in a four-way mirror are faulted, then either disk can be replaced because healthy replicas are available. However, if two disks in a four-way RAID-Z (raidz1) virtual device are faulted, then neither disk can be replaced because insufficient replicas from which to retrieve data exist. If the device is damaged but otherwise online, it can be replaced as long as the pool is not in the FAULTED state. However, any corrupted data on the device is copied to the new device, unless sufficient replicas with good data exist.

In the following configuration, the c1t1d0 disk can be replaced, and any data in the pool is copied from the healthy replica, c1t0d0:

<table>
<thead>
<tr>
<th>mirror</th>
<th>c1t0d0</th>
<th>c1t1d0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ONLINE</td>
<td>FAULTED</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>raidz</th>
<th>c1t0d0</th>
<th>c2t0d0</th>
<th>c3t0d0</th>
<th>c4t0d0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ONLINE</td>
<td>FAULTED</td>
<td>FAULTED</td>
<td>ONLINE</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>c1t0d0</th>
<th>c1t1d0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONLINE</td>
<td>ONLINE</td>
</tr>
</tbody>
</table>

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 286.
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 94.

**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).
ReReplacing a Device in a ZFS Storage Pool  (Continued)

**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.

  ```bash
  # zpool detach tank c2t4d0
  ```

The following example walks through the steps to replace a disk in a ZFS storage pool.

```bash
# 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
# cfgadm -c configure sata1/3
# cfgadm | grep sata1/3
sata1/3::dsk/c1t3d0 disk connected configured ok
# zpool online tank c1t3d0
# zpool replace tank c1t3d0
# zpool status tank
pool: tank
state: ONLINE
scrub: resilver completed after 0h0m with 0 errors on Tue Feb  2 13:17:32 2010
cfgadm | grep tank
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
```
Replacing or Repairing a Damaged Device

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:
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>DEGRADED</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c0t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-1</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c0t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-2</td>
<td>DEGRADED</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c0t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>replacing</td>
<td>DEGRADED</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t3d0</td>
<td>OFFLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c4t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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:
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>tank</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c0t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t1d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mirror-1</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c0t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t2d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Replacing or Repairing a Damaged Device

Example 11-1  Replacing a Device in a ZFS Storage Pool  (Continued)

<table>
<thead>
<tr>
<th>Device</th>
<th>State</th>
<th>Read</th>
<th>Write</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>mirror-2</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c0t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c4t3d0</td>
<td>ONLINE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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:
  

- Physically replace the failed log device.
- Bring the new log device online.
- Clear the pool’s error condition.

```bash
# zpool status -x
pool: pool
  state: FAULTED
  status: One or more of the intent logs could not be read.
  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     | bad intent log |
    mirror-0 | ONLINE | 0    | 0     | 0     |
    c0t1d0  | ONLINE | 0    | 0     | 0     |
    c0t4d0  | ONLINE | 0    | 0     | 0     |
    logs    | FAULTED| 0    | 0     | bad intent log |
    c0t5d0  | UNAVAIL| 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.

The following zpool status resilver status messages are provided:

- Resilver in-progress report. For example:
Resilver completion message. For example:

resilvered 16.2G in 0h16m with 0 errors on Mon Jun 7 09:34:21 2010

Resilver completion messages persist across system reboots.

Traditional file systems resilver data at the block level. Because ZFS eliminates the artificial layering of the volume manager, it can perform resilvering in a much more powerful and controlled manner. The two main advantages of this feature are as follows:

- ZFS only resilver 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:

```bash
# zpool status tank
pool: tank
  state: ONLINE
status: One or more devices is currently being resilvered. The pool will continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
scan: resilver in progress since Mon Jun 7 10:49:20 2010
    54.6M scanned out of 222M at 5.46M/s, 0h0m to go
    54.5M resilvered, 24.64% done
config:

   NAME   STATE     READ  WRITE  CKSUM
  tank   ONLINE    0      0      0
 mirror-0 ONLINE    0      0      0
 replacing-0 ONLINE  0      0      0
 c1t0d0 ONLINE    0      0      0 (resilvering)
 c2t0d0 ONLINE    0      0      0
 c1t1d0 ONLINE    0      0      0
```

In this example, the disk `c1t0d0` is being replaced by `c2t0d0`. This event is observed in the status output by the presence of the `replacing` virtual device in the configuration. This device is not real, nor is it possible for you to create a pool by using it. The purpose of this device is solely to display the resilvering progress and to identify which device is being replaced.

Note that any pool currently undergoing resilvering is placed in the `ONLINE` or `DEGRADED` state because the pool cannot provide the desired level of redundancy until the resilvering process is completed. Resilvering proceeds as fast as possible, though the I/O is always scheduled with a
lower priority than user-requested I/O, to minimize impact on the system. After the resilvering is completed, the configuration reverts to the new, complete, configuration. For example:

```
# zpool status tank
pool: tank
  state: ONLINE
  scrub: resilver completed after 0h1m with 0 errors on Tue Feb 2 13:54:30 2010
config:

  NAME    STATE    READ  WRITE   CKSUM
  tank    ONLINE   0      0        0
  mirror-0 ONLINE   0      0        0
  c2t0d0  ONLINE   0      0            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 284
- “Repairing a Corrupted File or Directory” on page 285
- “Repairing ZFS Storage Pool-Wide Damage” on page 286

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 265.
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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>monkey</td>
<td>ONLINE</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t1d0</td>
<td>ONLINE</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t5d0</td>
<td>ONLINE</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>morpheus</td>
<td>FAULTED</td>
<td>corrupted data</td>
<td></td>
</tr>
<tr>
<td>c1t1d0</td>
<td>ONLINE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

```bash
# 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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
<th>READ</th>
<th>WRITE</th>
<th>CKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>monkey</td>
<td>ONLINE</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c1t1d0</td>
<td>ONLINE</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c2t5d0</td>
<td>ONLINE</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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: FAULTED
  status: The pool metadata is corrupted and the pool cannot be opened.
  action: Recovery is possible, but will result in some data loss. Returning the pool to its state as of Wed Jul 14 11:44:10 2010 should correct the problem. Approximately 5 seconds of data must be discarded, irreversibly. Recovery can be attempted by executing 'zpool clear -F tpool'. A scrub of the pool is strongly recommended after recovery.
  see: http://www.sun.com/msg/ZFS-0000-72
  scrub: none requested
  config:

  NAME  STATE     READ WRITE CKSUM
  tpool FAULTED  0  0    1  corrupted data
  c1t1d0 ONLINE  0  0    2
  c1t3d0 ONLINE  0  0    4
  ```

  The recovery process as described 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 boot, 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.

- You can import a damaged pool in read-only mode. This method allows you to import the pool so that you can access the data. For example:

```
# zpool import -o readonly=on tpool
```

For more information about importing a pool read-only, see “Importing a Pool in Read-Only Mode” on page 121.

- You can import a pool with a missing log device by using the zpool import -m command. For more information, see “Importing a Pool With a Missing Log Device” on page 119.

- 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 `fmdump -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 `fmdump` output.
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 289
- “ZFS Pool Versions” on page 289
- “ZFS File System Versions” on page 291

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` command 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 38 and “Upgrading ZFS Storage Pools” on page 123.

ZFS Pool Versions

The following table provides a list of ZFS pool versions that are available in the Solaris releases.

<table>
<thead>
<tr>
<th>Version</th>
<th>Oracle Solaris 11 Express</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>snv_36</td>
<td>Initial ZFS version</td>
</tr>
<tr>
<td>2</td>
<td>snv_38</td>
<td>Ditto blocks (replicated metadata)</td>
</tr>
<tr>
<td>Version</td>
<td>Oracle Solaris 11 Express</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>3</td>
<td>snv_42</td>
<td>Hot spares and double parity RAID-Z</td>
</tr>
<tr>
<td>4</td>
<td>snv_62</td>
<td>zpool history</td>
</tr>
<tr>
<td>5</td>
<td>snv_62</td>
<td>gzip compression algorithm</td>
</tr>
<tr>
<td>6</td>
<td>snv_62</td>
<td>bootfs pool property</td>
</tr>
<tr>
<td>7</td>
<td>snv_68</td>
<td>Separate intent log devices</td>
</tr>
<tr>
<td>8</td>
<td>snv_69</td>
<td>Delegated administration</td>
</tr>
<tr>
<td>9</td>
<td>snv_77</td>
<td>refquota and refreservation properties</td>
</tr>
<tr>
<td>10</td>
<td>snv_78</td>
<td>Cache devices</td>
</tr>
<tr>
<td>11</td>
<td>snv_94</td>
<td>Improved scrub performance</td>
</tr>
<tr>
<td>12</td>
<td>snv_96</td>
<td>Snapshot properties</td>
</tr>
<tr>
<td>13</td>
<td>snv_98</td>
<td>snapused property</td>
</tr>
<tr>
<td>14</td>
<td>snv_103</td>
<td>aclinherit passthrough-x property</td>
</tr>
<tr>
<td>15</td>
<td>snv_114</td>
<td>user and group space accounting</td>
</tr>
<tr>
<td>16</td>
<td>snv_116</td>
<td>stmf property</td>
</tr>
<tr>
<td>17</td>
<td>snv_120</td>
<td>Triple-parity RAID-Z</td>
</tr>
<tr>
<td>18</td>
<td>snv_121</td>
<td>Snapshot user holds</td>
</tr>
<tr>
<td>19</td>
<td>snv_125</td>
<td>Log device removal</td>
</tr>
<tr>
<td>20</td>
<td>snv_128</td>
<td>zle (zero-length encoding) compression algorithm</td>
</tr>
<tr>
<td>21</td>
<td>snv_128</td>
<td>Deduplication</td>
</tr>
<tr>
<td>22</td>
<td>snv_128</td>
<td>Received properties</td>
</tr>
<tr>
<td>23</td>
<td>snv_135</td>
<td>Slim ZIL</td>
</tr>
<tr>
<td>24</td>
<td>snv_137</td>
<td>System attributes</td>
</tr>
<tr>
<td>25</td>
<td>snv_140</td>
<td>Improved scrub stats</td>
</tr>
<tr>
<td>26</td>
<td>snv_141</td>
<td>Improved snapshot deletion performance</td>
</tr>
<tr>
<td>27</td>
<td>snv_145</td>
<td>Improved snapshot creation performance</td>
</tr>
<tr>
<td>28</td>
<td>snv_147</td>
<td>Multiple vdev replacements</td>
</tr>
<tr>
<td>29</td>
<td>snv_148</td>
<td>RAID-Z/mirror hybrid allocator</td>
</tr>
<tr>
<td>30</td>
<td>snv_149</td>
<td>Encryption</td>
</tr>
</tbody>
</table>
### ZFS File System Versions

The following table lists the ZFS file system versions that are available in the Solaris releases.

<table>
<thead>
<tr>
<th>Version</th>
<th>Oracle Solaris 11 Express</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>snv_36</td>
<td>Initial ZFS file system version</td>
</tr>
<tr>
<td>2</td>
<td>snv_69</td>
<td>Enhanced directory entries</td>
</tr>
<tr>
<td>3</td>
<td>snv_77</td>
<td>Case insensitivity and file system unique identifier (FUID)</td>
</tr>
<tr>
<td>4</td>
<td>snv_114</td>
<td>user quota and group quota properties</td>
</tr>
<tr>
<td>5</td>
<td>snv_137</td>
<td>System attributes</td>
</tr>
</tbody>
</table>
Index

ACLs (Continued)
setting ACLs on ZFS file (verbose mode)
description, 224
setting on ZFS files
description, 221
adding
cache devices (example of), 86
devices to a ZFS storage pool (zpool add)
(example of), 82
disks to a RAID-Z configuration (example of), 84
mirrored log device (example of), 85
ZFS file system to a non-global zone
(example of), 257
ZFS volume to a non-global zone
(example of), 258
adjusting, sizes of swap and dump devices, 132
allocated property, description, 102
alternate root pools
creating
(example of), 261
description, 261
importing
(example of), 261
altroot property, description, 102
atime property, description, 150
attaching
devices to ZFS storage pool (zpool attach)
(example of), 87
autoreplace property, description, 102
available property, description, 150

A
accessing
ZFS snapshot
(example of), 199
ACL model, Solaris, differences between ZFS and traditional file systems, 65
ACL property mode, aclinherit, 149
aclinherit property, 221
ACLS
access privileges, 218
ACL inheritance, 220
ACL inheritance flags, 220
ACL on ZFS directory
detailed description, 223
ACL on ZFS file
detailed description, 222
ACL property, 221
aclinherit property, 221
description, 215
differences from POSIX-draft ACLs, 216
disk entry types, 218
format description, 216
modifying trivial ACL on ZFS file (verbose mode)
(example of), 224
restoring trivial ACL on ZFS file (verbose mode)
(example of), 227
setting ACL inheritance on ZFS file (verbose mode)
(example of), 228
setting ACLs on ZFS file (compact mode)
(example of), 235
description, 234

adding
accessing
ZFS snapshot
(example of), 199
ACL model, Solaris, differences between ZFS and traditional file systems, 65
ACL property mode, aclinherit, 149
aclinherit property, 221
ACLS
access privileges, 218
ACL inheritance, 220
ACL inheritance flags, 220
ACL on ZFS directory
detailed description, 223
ACL on ZFS file
detailed description, 222
ACL property, 221
aclinherit property, 221
description, 215
differences from POSIX-draft ACLs, 216
disk entry types, 218
format description, 216
modifying trivial ACL on ZFS file (verbose mode)
(example of), 224
restoring trivial ACL on ZFS file (verbose mode)
(example of), 227
setting ACL inheritance on ZFS file (verbose mode)
(example of), 228
setting ACLs on ZFS file (compact mode)
(example of), 235
description, 234

ACLs (Continued)
setting ACLs on ZFS file (verbose mode)
description, 224
setting on ZFS files
description, 221
adding
cache devices (example of), 86
devices to a ZFS storage pool (zpool add)
(example of), 82
disks to a RAID-Z configuration (example of), 84
mirrored log device (example of), 85
ZFS file system to a non-global zone
(example of), 257
ZFS volume to a non-global zone
(example of), 258
adjusting, sizes of swap and dump devices, 132
allocated property, description, 102
alternate root pools
creating
(example of), 261
description, 261
importing
(example of), 261
altroot property, description, 102
atime property, description, 150
attaching
devices to ZFS storage pool (zpool attach)
(example of), 87
autoreplace property, description, 102
available property, description, 150

A
accessing
ZFS snapshot
(example of), 199
ACL model, Solaris, differences between ZFS and traditional file systems, 65
ACL property mode, aclinherit, 149
aclinherit property, 221
ACLS
access privileges, 218
ACL inheritance, 220
ACL inheritance flags, 220
ACL on ZFS directory
detailed description, 223
ACL on ZFS file
detailed description, 222
ACL property, 221
aclinherit property, 221
description, 215
differences from POSIX-draft ACLs, 216
disk entry types, 218
format description, 216
modifying trivial ACL on ZFS file (verbose mode)
(example of), 224
restoring trivial ACL on ZFS file (verbose mode)
(example of), 227
setting ACL inheritance on ZFS file (verbose mode)
(example of), 228
setting ACLs on ZFS file (compact mode)
(example of), 235
description, 234

adding
accessing
ZFS snapshot
(example of), 199
ACL model, Solaris, differences between ZFS and traditional file systems, 65
ACL property mode, aclinherit, 149
aclinherit property, 221
ACLS
access privileges, 218
ACL inheritance, 220
ACL inheritance flags, 220
ACL on ZFS directory
detailed description, 223
ACL on ZFS file
detailed description, 222
ACL property, 221
aclinherit property, 221
description, 215
differences from POSIX-draft ACLs, 216
disk entry types, 218
format description, 216
modifying trivial ACL on ZFS file (verbose mode)
(example of), 224
restoring trivial ACL on ZFS file (verbose mode)
(example of), 227
setting ACL inheritance on ZFS file (verbose mode)
(example of), 228
setting ACLs on ZFS file (compact mode)
(example of), 235
description, 234

ACLs (Continued)
setting ACLs on ZFS file (verbose mode)
description, 224
setting on ZFS files
description, 221
adding
cache devices (example of), 86
devices to a ZFS storage pool (zpool add)
(example of), 82
disks to a RAID-Z configuration (example of), 84
mirrored log device (example of), 85
ZFS file system to a non-global zone
(example of), 257
ZFS volume to a non-global zone
(example of), 258
adjusting, sizes of swap and dump devices, 132
allocated property, description, 102
alternate root pools
creating
(example of), 261
description, 261
importing
(example of), 261
altroot property, description, 102
atime property, description, 150
attaching
devices to ZFS storage pool (zpool attach)
(example of), 87
autoreplace property, description, 102
available property, description, 150
bootblocks, installing with installboot and installgrub, 134
bootfs property, description, 103
booting
  a ZFS BE with boot -L and boot -Z on SPARC systems, 136
root file system, 134
cache devices
  considerations for using, 77
  creating a ZFS storage pool with (example of), 77
cache devices, adding, (example of), 86
cache devices, removing, (example of), 86
cachefile property, description, 103
canmount property
  description, 150
  detailed description, 162
capacity property, description, 103
casesensitivity property, description, 151
checking, ZFS data integrity, 265
checksum, definition, 53
checksum property, description, 151
checksummed data, description, 52
clearing
  a device in a ZFS storage pool (zpool clear)
    description, 94
  device errors (zpool clear)
    (example of), 276
clearing a device
  ZFS storage pool
    (example of), 94
clone, definition, 53
clones
  creating (example of), 205
  destroying (example of), 206
features, 205
command history, zpool history, 45
components of ZFS storage pool, 67
components of ZFS, naming requirements, 55
compression property, description, 151
compressratio property, description, 151
controlling, data validation (scrubbing), 265
copies property, description, 152
crash dump, saving, 133
creating
  a basic ZFS file system (zpool create)
    (example of), 58
  a new pool by splitting a mirrored storage pool
    (zpool split)
    (example of), 89
  a ZFS storage pool (zpool create)
    (example of), 58
  alternate root pools
    (example of), 261
double-parity RAID-Z storage pool (zpool create)
  (example of), 75
mirrored ZFS storage pool (zpool create)
  (example of), 74
single-parity RAID-Z storage pool (zpool create)
  (example of), 75
triple-parity RAID-Z storage pool (zpool create)
  (example of), 75
ZFS clone (example of), 205
ZFS file system, 61
  (example of), 146
  description, 146
ZFS file system hierarchy, 60
ZFS snapshot
  (example of), 196
ZFS storage pool
  description, 73
ZFS storage pool (zpool create)
  (example of), 73
ZFS storage pool with cache devices (example of), 77
ZFS storage pool with log devices (example of), 76
ZFS volume
  (example of), 253
creation property, description, 152
data
corrupted, 264
data (Continued)
corruption identified (zpool status -v)
   (example of), 271
repair, 265
resilvering
   description, 267
scrubbing
   (example of), 266
validation (scrubbing), 265
dataset
definition, 54
description, 146
dataset types, description, 168
dedup property, description, 152
dedupratio property, description, 103
dedupratio property, description, 103
delegated administration, overview, 241
deleagting
dataset to a non-global zone
   (example of), 257
permissions (example of), 246
delegating permissions, zfs allow, 244
delegating permissions to a group, (example of), 246
delegating permissions to an individual user, (example of), 246
delegation property, description, 103
delegation property, disabling, 242
destroying
ZFS clone (example of), 206
ZFS file system
   (example of), 147
ZFS file system with dependents
   (example of), 147
ZFS snapshot
   (example of), 197
ZFS storage pool
description, 73
ZFS storage pool (zpool destroy)
   (example of), 81
detaching
devices to ZFS storage pool (zpool detach)
   (example of), 89
detecting
   in-use devices
      (example of), 79
mismatched replication levels
   (example of), 80
determining
   if a device can be replaced
description, 277
type of device failure
description, 274
devices property, description, 152
differences between ZFS and traditional file systems
   file system granularity, 63
   mounting ZFS file systems, 65
   new Solaris ACL model, 65
   out of space behavior, 64
   traditional volume management, 65
   ZFS space accounting, 64
disks, as components of ZFS storage pools, 68
displaying
   command history, 45
   delegated permissions (example of), 249
detailed ZFS storage pool health status
   (example of), 112
   health status of storage pools
description of, 111
   syslog reporting of ZFS error messages
description, 272
ZFS storage pool health status
   (example of), 112
ZFS storage pool I/O statistics
description, 109
ZFS storage pool vdev I/O statistics
   (example of), 110
ZFS storage pool-wide I/O statistics
   (example of), 109
dry run
ZFS storage pool creation (zpool create -n)
   (example of), 80
dumpadm, enabling a dump device, 133
dynamic striping
description, 72
storage pool feature, 72
M

migrating ZFS storage pools, description, 115
mirror, definition, 54
mirrored configuration
   conceptual view, 71
description, 71
   redundancy feature, 71
mirrored log device, adding, (example of), 85
mirrored log devices, creating a ZFS storage pool with
   (example of), 76
mirrored storage pool (zpool create), (example of), 74
mismatched replication levels
detecting
   (example of), 80
mlslabel property, description, 153
modifying
   trivial ACL on ZFS file (verbose mode)
   (example of), 224
mount point, default for ZFS storage pools, 81
mount points
   automatic, 174
   legacy, 174
   managing ZFS
description, 174
mounted property, description, 153
mounting
   ZFS file systems
   (example of), 176
mounting ZFS file systems
   differences between ZFS and traditional file
   systems, 65

mounting ZFS file systems (Continued)
   with NFSv4 mirror mounts (example of), 37
mountpoint, default for ZFS file system, 146
mountpoint property, description, 153

N

naming requirements, ZFS components, 55
NFSv4 ACLs
   ACL inheritance, 220
   ACL inheritance flags, 220
   ACL property, 221
differences from POSIX-draft ACLs, 216
   format description, 216
   model
description, 215
NFSv4 mirror mounts, 37
notifying
   ZFS of reattached device (zpool online)
   (example of), 274

O

offlining a device (zpool offline)
   ZFS storage pool
   (example of), 92
onlining a device
   ZFS storage pool (zpool online)
   (example of), 93
onlining and offlineing devices
   ZFS storage pool
description, 92
origin property, description, 154
out of space behavior, differences between ZFS and
   traditional file systems, 64

P

permission sets, defined, 241
pool, definition, 54
pooled storage, description, 51
POSIX-draft ACLs, description, 216
Index

primarycache property, description, 153
properties of ZFS
  description, 149
    description of heritable properties, 149

Q
quota property, description, 154
quotas and reservations, description, 181

R
RAID-Z, definition, 54
RAID-Z configuration
  (example of), 75
    conceptual view, 71
    double-parity, description, 71
    redundancy feature, 71
    single-parity, description, 71
RAID-Z configuration, adding disks to, (example of), 84
read-only properties of ZFS
  available, 150
  compression, 151
  creation, 152
  description, 159
  mounted, 153
  origin, 154
  referenced, 155
  type, 157
  used, 157
  usedbychildren, 157
  usedbydataset, 157
  usedbyrefreservation, 157
  usedbysnapshots, 158
read-only property, description, 154
receiving
  ZFS file system data (zfs receive)
    (example of), 209
recordsize property
  description, 154
  detailed description, 164
recovering
  destroyed ZFS storage pool
    (example of), 122
  referenced property, description, 155
  refquota property, description, 155
  refreservation property, description, 155
  removing, cache devices (example of), 86
  removing permissions, zfs unallow, 245
renaming
  ZFS file system
    (example of), 148
  ZFS snapshot
    (example of), 198
repairing
  a damaged ZFS configuration
    description, 272
  an unbootable system
    description, 287
  pool-wide damage
    description, 287
  repairing a corrupted file or directory
    description, 285
replacing
  a device (zpool replace)
    (example of), 94, 278, 282
  a missing device
    (example of), 273
replication features of ZFS, mirrored or RAID-Z, 70
requirements, installation, 126
reservation property, description, 155
resilvering, definition, 54
resilvering and data scrubbing, description, 267
restoring
  trivial ACL on ZFS file (verbose mode)
    (example of), 227
rights profiles, for management of ZFS file systems and storage pools, 262
rolling back
  ZFS snapshot
    (example of), 201
S
savecore, saving crash dumps, 133
saving
  crash dumps
    savecore, 133
    ZFS file system data (zfs send)
      (example of), 208
scripting
  ZFS storage pool output
    (example of), 107
scrubbing
  (example of), 266
  data validation, 265
secondarycache property, description, 156
self-healing data, description, 72
sending and receiving
  ZFS file system data
    description, 207
separate log devices, considerations for using, 39
settable properties of ZFS
  aclinherit, 149
  atime, 150
  canmount, 150
    detailed description, 162
  casesensitivity, 151
  checksum, 151
  compression, 151
  copies, 152
  dedup, 152
  description, 160
  devices, 152
  exec, 152
  mountpoint, 153
  primarycache, 153
  quota, 154
  read-only, 154
  recordsize, 154
    detailed description, 164
  refquota, 155
  refreservation, 155
  reservation, 155
  secondarycache, 156
  setuid, 156
  sharesmb, 156
  snapdir, 156
settable properties of ZFS (Continued)
  sync, 157
  used
    detailed description, 160
  version, 158
  volblocksize, 158
  volsize, 158
    detailed description, 165
  xattr, 159
  zoned, 159
setting
  ACL inheritance on ZFS file (verbose mode)
    (example of), 228
  ACLs on ZFS file (compact mode)
    (example of), 235
    description, 234
  ACLs on ZFS file (verbose mode)
    (description, 224
  ACLs on ZFS files
    description, 221
  compression property
    (example of), 62
  legacy mount property
    (example of), 175
  mountpoint property, 62
  quota property (example of), 62
  sharesmb property
    (example of), 62
  ZFS atime property
    (example of), 169
  ZFS file system quota (zfs set quota)
    example of, 182
  ZFS file system reservation
    (example of), 185
  ZFS mount points (zfs set mountpoint)
    (example of), 175
  ZFS quota
    (example of), 169
  setuid property, description, 156
  sharesmb property
    description, 156
  sharesmb property
    (example of), 179
    description, 156
sharesmb property, description, detailed, 164
sharing
  ZFS file systems
    description, 178
    example of, 178
sharing ZFS file systems
  sharesmb property, 164
  with sharesmb property (example of), 179
simplified administration, description, 53
size property, description, 104
snapdir property, description, 156
snapshot
  accessing
    (example of), 199
  creating
    (example of), 196
  definition, 55
  destroying
    (example of), 197
  features, 195
  renaming
    (example of), 198
  rolling back
    (example of), 201
  space accounting, 200
Solaris ACLs
  ACL inheritance, 220
  ACL inheritance flags, 220
  ACL property, 221
  differences from POSIX-draft ACLs, 216
  format description, 216
  new model
    description, 215
splitting a mirrored storage pool
  (zpool split)
    (example of), 89
storage requirements, identifying, 59
swap and dump devices
  adjusting sizes of, 132
  description, 131
  issues, 131
sync property, description, 157

t terminology
  checksum, 53
  clone, 53
  dataset, 54
  file system, 54
  mirror, 54
  pool, 54
  RAID-Z, 54
  resilvering, 54
  snapshot, 55
  virtual device, 55
  volume, 55
traditional volume management, differences between
  ZFS and traditional file systems, 65
transactional semantics, description, 51
troubleshooting
  clear device errors (zpool clear)
    (example of), 276
  damaged devices, 264
  data corruption identified (zpool status -v)
    (example of), 271
  determining if a device can be replaced
    description, 277
  determining if problems exist (zpool status -x), 268
  determining type of data corruption (zpool status
    -v)
    (example of), 284
  determining type of device failure
    description, 274
  identifying problems, 267
  missing (faulted) devices, 264
  notifying ZFS of reattached device (zpool online)
    (example of), 274
  overall pool status information
    description, 269
  repairing a corrupted file or directory
    description, 285
  repairing a damaged ZFS configuration, 272
  repairing an unbootable system
    description, 287
  repairing pool-wide damage
    description, 287
troubleshooting (Continued)
  replacing a device (zpool replace)
    (example of), 278, 282
  replacing a missing device
    (example of), 273
  syslog reporting of ZFS error messages, 272
  ZFS failures, 263
  type property, description, 157

U
  unmounting
    ZFS file systems
      (example of), 177
  unsharing
    ZFS file systems
      example of, 178
  upgrading
    ZFS file systems
      description, 192
    ZFS storage pool
      description, 123
  used property
    description, 157
    detailed description, 160
  usedbychildren property, description, 157
  usedbydataset property, description, 157
  usedbyrefreservation property, description, 157
  usedbysnapshots property, description, 158
  user properties of ZFS
    (example of), 165
    detailed description, 165

V
  version property, description, 158
  version property, description, 105
  virtual device, definition, 55
  virtual devices, as components of ZFS storage pools, 78
  volblocksize property, description, 158
  volsize property
    description, 158
    detailed description, 165
  volume, definition, 55

W
  whole disks, as components of ZFS storage pools, 68

X
  xattr property, description, 159

Z
  zfs allow
    description, 244
    displaying delegated permissions, 249
  zfs create
    (example of), 61, 146
    description, 146
  ZFS delegated administration, overview, 241
  zfs destroy, (example of), 147
  zfs destroy -r, (example of), 147
  ZFS file system
    description, 145
    versions
      description, 289
  ZFS file systems
    ACL on ZFS directory
      detailed description, 223
    ACL on ZFS file
      detailed description, 222
    adding ZFS file system to a non-global zone
      (example of), 257
    adding ZFS volume to a non-global zone
      (example of), 258
    and NFSv4 mirror mounts, 37
    booting a root file system
      description, 134
    booting a ZFS BE with boot -Land boot -Z
      (SPARC example of), 136
    checksum
      definition, 53
ZFS file systems (Continued)
setting mount point (zfs set mountpoint)
(example of), 175
setting quota property
(example of), 169
sharing
description, 178
telephone of, 178
simplified administration
description, 53
snapshot
accessing, 199
creating, 196
definition, 55
description, 195
destroying, 197
renaming, 198
rolling back, 201
snapshot space accounting, 200
swap and dump devices
adjusting sizes of, 132
description, 131
issues, 131
transactional semantics
description, 51
unmounting
(example of), 177
unsharing
example of, 178
upgrading
description, 192
using on a Solaris system with zones installed
description, 256
volume
definition, 55
ZFS file systems (zfs set quota)
setting a quota
example of, 182
zfs get, (example of), 170
zfs get -H -o, (example of), 173
zfs get -s, (example of), 172
zfs inherit, (example of), 170
ZFS intent log (ZIL), description, 39
zfs list
(example of), 62, 166
zfs list -H, (example of), 168
zfs list -r, (example of), 167
zfs list -t, (example of), 168
zfs mount, (example of), 176
ZFS pool properties
allocated, 102
alroot, 102
autoreplace, 102
bootfs, 103
cache, 103
capacity, 103
dedupfile, 103
dedupratio, 103
dedupratio, 103
dedupratio, 103
defailmode, 104
free, 104
guid, 104
health, 104
listsnapshots, 104
size, 104
version, 105
zfs promote, clone promotion (example of), 206
ZFS properties
acl inherit, 149
atime, 150
available, 150
canmount, 150
detailed description, 162
casesensitivity, 151
checksum, 151
compression, 151
compressratio, 151
copies, 152
creation, 152
dedup, 152
description, 149
devices, 152
description, 149
exec, 152
inheritable, description of, 149
logbias, 153
management within a zone
description, 259
### ZFS properties (Continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlslabel</td>
<td>153</td>
</tr>
<tr>
<td>mounted</td>
<td>153</td>
</tr>
<tr>
<td>mountpoint</td>
<td>153</td>
</tr>
<tr>
<td>origin</td>
<td>154</td>
</tr>
<tr>
<td>quota</td>
<td>154</td>
</tr>
<tr>
<td>read-only</td>
<td>154</td>
</tr>
<tr>
<td>recordsize</td>
<td>154</td>
</tr>
<tr>
<td>detailed description</td>
<td>164</td>
</tr>
<tr>
<td>referenced</td>
<td>155</td>
</tr>
<tr>
<td>refquota</td>
<td>155</td>
</tr>
<tr>
<td>refreservation</td>
<td>155</td>
</tr>
<tr>
<td>reservation</td>
<td>155</td>
</tr>
<tr>
<td>secondarycache</td>
<td>153, 156</td>
</tr>
<tr>
<td>settable</td>
<td>160</td>
</tr>
<tr>
<td>setuid</td>
<td>156</td>
</tr>
<tr>
<td>sharenfs</td>
<td>156</td>
</tr>
<tr>
<td>sharesmb</td>
<td>156</td>
</tr>
<tr>
<td>sharesmb property</td>
<td>179</td>
</tr>
<tr>
<td>snapdir</td>
<td>156</td>
</tr>
<tr>
<td>sync</td>
<td>157</td>
</tr>
<tr>
<td>type</td>
<td>157</td>
</tr>
<tr>
<td>used</td>
<td>157</td>
</tr>
<tr>
<td>detailed description</td>
<td>160</td>
</tr>
<tr>
<td>usedbychildren</td>
<td>157</td>
</tr>
<tr>
<td>usedbydataset</td>
<td>157</td>
</tr>
<tr>
<td>usedbyrefreservation</td>
<td>157</td>
</tr>
<tr>
<td>usedbysnapshots</td>
<td>158</td>
</tr>
<tr>
<td>user properties</td>
<td></td>
</tr>
<tr>
<td>detailed description</td>
<td>165</td>
</tr>
<tr>
<td>version</td>
<td>158</td>
</tr>
<tr>
<td>volblocksize</td>
<td>158</td>
</tr>
<tr>
<td>volsize</td>
<td>158</td>
</tr>
<tr>
<td>detailed description</td>
<td>165</td>
</tr>
<tr>
<td>xattr</td>
<td>159</td>
</tr>
<tr>
<td>zoned</td>
<td>159</td>
</tr>
<tr>
<td>zoned property</td>
<td></td>
</tr>
<tr>
<td>detailed description</td>
<td>260</td>
</tr>
<tr>
<td>zfs receive</td>
<td>209</td>
</tr>
<tr>
<td>zfs rename</td>
<td>148</td>
</tr>
<tr>
<td>zfs send</td>
<td>208</td>
</tr>
<tr>
<td>zfs set atime</td>
<td>169</td>
</tr>
<tr>
<td>zfs set compression</td>
<td>62</td>
</tr>
<tr>
<td>zfs set mountpoint</td>
<td>62, 175</td>
</tr>
<tr>
<td>zfs set mountpoint=legacy</td>
<td>175</td>
</tr>
<tr>
<td>zfs set quota</td>
<td>62</td>
</tr>
<tr>
<td>zfs set quota</td>
<td>169</td>
</tr>
<tr>
<td>example of</td>
<td>182</td>
</tr>
<tr>
<td>zfs set reservation</td>
<td>185</td>
</tr>
<tr>
<td>zfs set sharenfs</td>
<td>62</td>
</tr>
<tr>
<td>zfs set sharenfs=on</td>
<td>178</td>
</tr>
<tr>
<td>ZFS space accounting</td>
<td></td>
</tr>
<tr>
<td>differences between ZFS</td>
<td>64</td>
</tr>
<tr>
<td>and traditional file</td>
<td></td>
</tr>
<tr>
<td>systems</td>
<td></td>
</tr>
<tr>
<td>ZFS storage pool</td>
<td></td>
</tr>
<tr>
<td>versions</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>289</td>
</tr>
<tr>
<td>ZFS storage pools</td>
<td></td>
</tr>
<tr>
<td>adding devices to</td>
<td>82</td>
</tr>
<tr>
<td>(zpool add)</td>
<td></td>
</tr>
<tr>
<td>alternate root pools</td>
<td>261</td>
</tr>
<tr>
<td>attaching devices to</td>
<td>87</td>
</tr>
<tr>
<td>(zpool attach)</td>
<td></td>
</tr>
<tr>
<td>clearing a device</td>
<td>94</td>
</tr>
<tr>
<td>clearing device errors</td>
<td>276</td>
</tr>
<tr>
<td>(zpool clear)</td>
<td></td>
</tr>
<tr>
<td>components</td>
<td>67</td>
</tr>
<tr>
<td>corrupted data</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>264</td>
</tr>
<tr>
<td>creating (zpool create)</td>
<td>73</td>
</tr>
<tr>
<td>(example of)</td>
<td></td>
</tr>
<tr>
<td>creating a RAID-Z</td>
<td>75</td>
</tr>
<tr>
<td>configuration (zpool</td>
<td></td>
</tr>
<tr>
<td>create)</td>
<td></td>
</tr>
<tr>
<td>creating mirrored</td>
<td>74</td>
</tr>
<tr>
<td>configuration (zpool</td>
<td></td>
</tr>
<tr>
<td>create)</td>
<td></td>
</tr>
<tr>
<td>damaged devices</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>264</td>
</tr>
<tr>
<td>data corruption identified</td>
<td>271</td>
</tr>
<tr>
<td>(zpool status -v)</td>
<td></td>
</tr>
<tr>
<td>data repair</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>265</td>
</tr>
<tr>
<td>data scrubbing</td>
<td></td>
</tr>
<tr>
<td>(example of)</td>
<td>266</td>
</tr>
<tr>
<td>description</td>
<td>265</td>
</tr>
</tbody>
</table>

---

**Index**

**Oracle Solaris ZFS Administration Guide** • January 2011

304
ZFS storage pools (Continued)
data scrubbing and resilvering
description, 267
data validation
description, 265
default mount point, 81
destroying (zpool destroy)
(example of), 81
detaching devices from (zpool detach)
(example of), 89
determining if a device can be replaced
description, 277
determining if problems exist (zpool status -x)
description, 268
determining type of device failure
description, 274
displaying detailed health status
(example of), 112
displaying health status, 111
(example of), 112
doing a dry run (zpool create -n)
(example of), 80
dynamic striping, 72
exporting
(example of), 116
failures, 263
identifying for import (zpool import -a)
(example of), 116
identifying problems
description, 267
identifying type of data corruption (zpool status
-v)
(example of), 284
importing
(example of), 119
importing from alternate directories (zpool import
-d)
(example of), 118
listing
(example of), 106
migrating
description, 115
mirror
definition, 54
ZFS storage pools (Continued)
mirrored configuration, description, 71
missing (faulted) devices
description, 264
notifying ZFS of reattached device (zpool online)
(example of), 274
offlining a device (zpool offline)
(example of), 92
onlining and offlining devices
description, 92
overall pool status information for troubleshooting
description, 269
pool
definition, 54
pool-wide I/O statistics
(example of), 109
RAID-Z
definition, 54
RAID-Z configuration, description, 71
recovering a destroyed pool
(example of), 122
repairing a corrupted file or directory
description, 285
repairing a damaged ZFS configuration, 272
repairing an unbootable system
description, 287
repairing pool-wide damage
description, 287
replacing a device (zpool replace)
(example of), 94, 278
replacing a missing device
(example of), 273
resilvering
definition, 54
rights profiles, 262
scripting storage pool output
(example of), 107
splitting a mirrored storage pool (zpool split)
(example of), 89
system error messages
description, 272
upgrading
description, 123
using files, 70
Index
ZFS storage pools (Continued) using whole disks, 68
vdev I/O statistics (example of), 110
viewing resilvering process (example of), 282
virtual device definition, 55
virtual devices, 78
ZFS storage pools (zpool online) onlining a device (example of), 93
zfs unallow, description, 245
zfs unmount, (example of), 177
zfs upgrade, 192
ZFS version ZFS feature and Solaris OS description, 289
ZFS volume, description, 253
zoned property description, 159
detailed description, 260
zones adding ZFS file system to a non-global zone (example of), 257
adding ZFS volume to a non-global zone (example of), 258
deleagating dataset to a non-global zone (example of), 257
using with ZFS file systems description, 256
ZFS property management within a zone description, 259
zoned property detailed description, 260
zpool add, (example of), 82
zpool attach, (example of), 87
zpool clear (example of), 94
description, 94
zpool create (example of), 58, 60
basic pool (example of), 73
zpool create (Continued) mirrored storage pool (example of), 74
RAID-Z storage pool (example of), 75
zpool create -n, dry run (example of), 80
zpool destroy, (example of), 81
zpool detach, (example of), 89
zpool export, (example of), 116
zpool history, (example of), 45
zpool import -a, (example of), 116
zpool import -d, (example of), 122
zpool import -d, (example of), 118
zpool import name, (example of), 119
zpool iostat, pool-wide (example of), 109
zpool iostat -v, vdev (example of), 110
zpool list (example of), 60, 106
description, 105
zpool list -Ho name, (example of), 107
zpool offline, (example of), 92
zpool online, (example of), 93
zpool replace, (example of), 94
zpool split, (example of), 89
zpool status -v, (example of), 112
zpool status -x, (example of), 112
zpool upgrade, 123