Oracle® Solaris ZFS Administration Guide

Part No: E37384–02
June 2013

ORACLE

130613@25097

Contents

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. The supported systems appear in the *Oracle Solaris Hardware Compatibility List* at http://www.oracle.com/webfolder/technetwork/hcl/index.html. This document cites any implementation differences between the platform types.

Who Should Use This Book

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

How This Book Is Organized

The following table describes the chapters in this book.

Chapter	Description
Chapter 1, "Oracle Solaris ZFS File System (Introduction)"	Provides an overview of ZFS and its features and benefits. It also covers some basic concepts and terminology.
Chapter 2, "Getting Started With Oracle Solaris ZFS"	Provides step-by-step instructions on setting up basic ZFS configurations with basic pools and file systems. This chapter also provides the hardware and software required to create ZFS file systems.
Chapter 3, "Managing Oracle Solaris ZFS Storage Pools"	Provides a detailed description of how to create and administer ZFS storage pools.
Chapter 4, "Installing and Booting an Oracle Solaris ZFS Root File System"	Describes how to install and boot a ZFS file system. Migrating a UFS root file system to a ZFS root file system by using Oracle Solaris Live Upgrade is also covered.

Chapter	Description
Chapter 5, "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 6, "Working With Oracle Solaris ZFS Snapshots and Clones"	Describes how to create and administer ZFS snapshots and clones.
Chapter 7, "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 8, "Oracle Solaris ZFS Delegated Administration"	Describes how to use ZFS delegated administration to allow nonprivileged users to perform ZFS administration tasks.
Chapter 9, "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 10, "Oracle Solaris ZFS Troubleshooting and Pool Recovery"	Describes how to identify ZFS failures and how to recover from them. Steps for preventing failures are covered as well.
Chapter 11, "Recommended Oracle Solaris ZFS Practices"	Describes recommended practices for creating, monitoring, and maintaining your ZFS storage pools and file systems.
Appendix A, "Oracle Solaris ZFS Version Descriptions"	Describes available ZFS versions, features of each version, and the Solaris OS that provides the ZFS version and feature.

Related Books

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

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

Access to Oracle Support

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

Typographic Conventions

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

TABLE P–1 Typographic Conventions

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

Shell Prompts in Command Examples

The following table shows UNIX system prompts and superuser prompts for shells that are included in the Oracle Solaris OS. In command examples, the shell prompt indicates whether the command should be executed by a regular user or a user with privileges.

TABLE P–2 Shell Prompts

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

◆ ◆ ◆ CHAPTER 1

Oracle Solaris ZFS File System (Introduction)

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

The following sections are provided in this chapter:

What's New in ZFS?

This section summarizes new features in the ZFS file system.

ZFS Command Usage Enhancements

Oracle Solaris 10 1/13: The `zfs` and `zpool` command have a `help` subcommand that you can use to provide more information about the `zfs` and `zpool` subcommands and their supported options. For example:

```
# zfs help
The following commands are supported:
allow       clone       create      destroy     diff        get
groupspace  help        hold        holds       inherit     list
mount       promote     receive     release     rename      rollback
send        set         share       snapshot    unallow     unmount
unshare     upgrade     userspace
For more info, run: zfs help <command>
# zfs help create
usage:
        create [-p] [-o property=value] ... <filesystem>
        create [-ps] [-b blocksize] [-o property=value] ... -V <size> <volume>

# zpool help
The following commands are supported:
add      attach   clear    create   destroy  detach   export   get
help     history  import   iostat   list     offline  online   remove
replace  scrub    set      split    status   upgrade
For more info, run: zpool help <command>
# zpool help attach
usage:
        attach [-f] <pool> <device> <new-device>
```

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

ZFS Snapshot Enhancements

Oracle Solaris 10 1/13: This release includes the following ZFS snapshot enhancements:

- The `zfs snapshot` command has a `snap` alias that provides abbreviated syntax for this command. For example:

  ```
  # zfs snap -r users/home@snap1
  ```

- The `zfs diff` command provides an enumeration option, `-e`, to identify all the files that were added or modified between the two snapshots. The generated output identifies all files added, but does not provide possible deletions. For example:

  ```
  # zfs diff -e tank/cindy@yesterday tank/cindy@now
  +       /tank/cindy/
  +       /tank/cindy/file.1
  ```

  You can also use the `-o` option to identify selected fields to be displayed. For example:

  ```
  # zfs diff -e -o size -o name tank/cindy@yesterday tank/cindy@now
  +       7       /tank/cindy/
  +       206695  /tank/cindy/file.1
  ```

For more information about creating ZFS snapshots, see Chapter 6, "Working With Oracle Solaris ZFS Snapshots and Clones."

Improved `aclmode` Property

Oracle Solaris 10 1/13: The `aclmode` property modifies Access Control List (ACL) behavior whenever ACL permissions on a file are modified during a `chmod` operation. The `aclmode` property has been reintroduced with the following property values:

- `discard` – A file system with an `aclmode` property of `discard` deletes all ACL entries that do not represent the mode of the file. This is the default value.
- `mask` – A file system with an `aclmode` property of `mask` reduces user or group permissions. The permissions are reduced, such that they are no greater than the group permission bits, unless it is a user entry that has the same UID as the owner of the file or directory. In this case, the ACL permissions are reduced so that they are no greater than owner permission bits. The mask value also preserves the ACL across mode changes, provided an explicit ACL set operation has not been performed.
- `passthrough` – A file system with an `aclmode` property of `passthrough` indicates that no changes are made to the ACL other than generating the necessary ACL entries to represent the new mode of the file or directory.

For more information, see Example 7–13.

Oracle Solaris ZFS Installation Features

Oracle Solaris 10 8/11: In this release, the following new installation features are available:

- You can use the text mode installation method to install a system with a ZFS flash archive. For more information, see Example 4–3.
- You can use the Oracle Solaris Live Upgrade `luupgrade` command to install a ZFS root flash archive. For more information, see Example 4–8.
- You can use the Oracle Solaris Live Upgrade `lucreate` command to specify a separate /var file system. For more information, see Example 4–5.

ZFS Send Stream Enhancements

Oracle Solaris 10 8/11: In this release, you can set file system properties that are sent and received in a snapshot stream. These enhancements provide flexibility in applying file system properties in a send stream to the receiving file system or in determining whether the local file system properties, such as the `mountpoint` property value, should be ignored when received.

For more information, see "Applying Different Property Values to a ZFS Snapshot Stream" on page 216.

ZFS Snapshot Differences (`zfs diff`)

Oracle Solaris 10 8/11: In this release, you can determine ZFS snapshot differences by using the `zfs diff` command.

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

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

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

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

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

For more information, see "Identifying ZFS Snapshot Differences (`zfs diff`)" on page 207.

ZFS Storage Pool Recovery and Performance Enhancements

Oracle Solaris 10 8/11: In this release, the following new ZFS storage pool features are provided:

- You can import a pool with a missing log by using the `zpool import -m` command. For more information, see "Importing a Pool With a Missing Log Device" on page 96.
- You can import a pool in read-only mode. This feature is primarily for pool recovery. If a damaged pool cannot be accessed because the underlying devices are damaged, you can import the pool read-only to recover the data. For more information, see "Importing a Pool in Read-Only Mode" on page 97.
- A RAID-Z (`raidz1`, `raidz2`, or `raidz3`) storage pool that is created in this release will have some latency-sensitive metadata automatically mirrored to improve read I/O throughput performance. For existing RAID-Z pools that are upgraded to at least pool version 29, some metadata will be mirrored for all newly written data.

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

Tuning ZFS Synchronous Behavior

Solaris 10 8/11: In this release, you can determine a ZFS file system's synchronous behavior by using the `sync` property.

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

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

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

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

For more information, see Table 5–1.

Improved ZFS Pool Messages

Oracle Solaris 10 8/11: In this release, you can use the `-T` option to provide an interval and count value for the `zpool list` and `zpool status` commands to display additional information.

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

- Resilver in-progress report. For example:

```
scan: resilver in progress since Thu Jun  7 14:41:11 2012
    3.83G scanned out of 73.3G at 106M/s, 0h11m to go
    3.80G resilvered, 5.22% done
```

- Scrub in-progress report. For example:

```
scan: scrub in progress since Thu Jun  7 14:59:25 2012
    1.95G scanned out of 73.3G at 118M/s, 0h10m to go
    0 repaired, 2.66% done
```

- Resilver completion message. For example:

```
resilvered 73.3G in 0h13m with 0 errors on Thu Jun  7 14:54:16 2012
```

- Scrub completion message. For example:

```
scan: scrub repaired 512B in 1h2m with 0 errors on Thu Jun  7 15:10:32 2012
```

- Ongoing scrub cancellation message. For example:

```
scan: scrub canceled on Thu Jun  7 15:19:20 MDT 2012
```

- Scrub and resilver completion messages persist across system reboots

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

```
# zpool status -T d tank 3 2
Wed Nov 14 15:44:34 MST 2012
  pool: tank
 state: DEGRADED
status: One or more devices is currently being resilvered.  The pool will
        continue to function in a degraded state.
action: Wait for the resilver to complete.
  scan: resilver in progress since Wed Nov 14 15:44:34 2012
    2.96G scanned out of 4.19G at 189M/s, 0h0m to go
    1.48G resilvered, 70.60% done
config:

        NAME                       STATE     READ WRITE CKSUM
        tank                       DEGRADED     0     0     0
          mirror-0                 ONLINE       0     0     0
            c0t5000C500335F95E3d0  ONLINE       0     0     0
            c0t5000C500335F907Fd0  ONLINE       0     0     0
          mirror-1                 DEGRADED     0     0     0
            c0t5000C500335BD117d0  ONLINE       0     0     0
            c0t5000C500335DC60Fd0  DEGRADED     0     0     0  (resilvering)

errors: No known data errors
```

ZFS ACL Interoperability Enhancements

Oracle Solaris 10 8/11: In this release, the following ACL enhancements are provided:

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

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

  ```
  # ls -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      206663 Jun 22 14:30 file.1
       0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
  ```

```
        /read_attributes/write_attributes/read_acl/write_acl/write_owner
        /synchronize:allow
    1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
    2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
        :allow
```

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

For more information, see Chapter 7, "Using ACLs and Attributes to Protect Oracle Solaris ZFS Files."

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

Oracle Solaris 10 9/10: In this release, you can use the `zpool split` command to split a mirrored storage pool, which detaches a disk or disks in the original mirrored pool to create another identical pool.

For more information, see "Creating a New Pool By Splitting a Mirrored ZFS Storage Pool" on page 65.

New ZFS System Process

Oracle Solaris 10 9/10: In this release, each ZFS storage pool has an associated process, `zpool-`*poolname*. The threads in this process are the pool's I/O processing threads to handle I/O tasks, such as compression and checksum validation, that are associated with the pool. The purpose of this process is to provide visibility into each storage pool's CPU utilization.

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

ZFS Device Replacement Enhancements

Oracle Solaris 10 9/10: In this release, a system event or *sysevent* is provided when the disks in a pool are replaced with larger disks. ZFS has been enhanced to recognize these events and adjusts

the pool based on the new size of the disk, depending on the setting of the `autoexpand` property. You can use the `autoexpand` pool property to enable or disable automatic pool expansion when a larger disk replaces a smaller disk.

These enhancements enable you to increase the pool size 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 pool size expanded when a larger disk replaces a smaller disk.

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

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

Another way to expand the disk 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:

- In previous releases, ZFS was unable 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 almost the same size provided that the pool is not already full.
- In this release, you do not need to reboot the system or export and import a pool to expand the pool size. As described previously, you can enable the `autoexpand` property or use the `zpool online -e` command to expand the pool size.

For more information about replacing devices, see "Replacing Devices in a Storage Pool" on page 70.

ZFS and Flash Installation Support

Solaris 10 10/09: In this release, you can set up a JumpStart profile to identify a flash archive of a ZFS root pool. For more information, see "Installing a ZFS Root File System (Oracle Solaris Flash Archive Installation)" on page 114.

Zone Migration in a ZFS Environment

Solaris 10 5/09: This release extends support for migrating zones in a ZFS environment with Oracle Solaris Live Upgrade. For more information, see "Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (at Least Solaris 10 5/09)" on page 136.

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

ZFS Installation and Boot Support

Solaris 10 10/08: This release enables you to install and boot a ZFS root file system. You can use the initial installation option or the JumpStart feature to install a ZFS root file system. Or, you can use Oracle Solaris Live Upgrade to migrate a UFS root file system to a ZFS root file system. ZFS support for swap and dump devices is also provided. For more information, see Chapter 4, "Installing and Booting an Oracle Solaris ZFS Root File System."

For a list of known issues with this release, see the Solaris 10 10/08 release notes.

ZFS Web-Based Management

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

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

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

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

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

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

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

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

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

What Is Oracle Solaris ZFS?

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

ZFS Pooled Storage

ZFS uses the concept of *storage pools* to manage physical storage. Historically, file systems were constructed on top of a single physical device. To address multiple devices and provide for data redundancy, the concept of a *volume manager* was introduced to provide a representation of a

single device so that file systems would not need to be modified to take advantage of multiple devices. This design added another layer of complexity and ultimately prevented certain file system advances because the file system had no control over the physical placement of data on the virtualized volumes.

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

Transactional Semantics

ZFS is a transactional file system, which means that the file system state is always consistent on disk. Traditional file systems overwrite data in place, which means that if the system loses power, for example, between the time a data block is allocated and when it is linked into a directory, the file system will be left in an inconsistent state. Historically, this problem was solved through the use of the `fsck` command. This command was responsible for reviewing and verifying the file system state, and attempting to repair any inconsistencies during the process. This problem of inconsistent file systems caused great pain to administrators, and the `fsck` command was never guaranteed to fix all possible problems. More recently, file systems have introduced the concept of *journaling*. The journaling process records actions in a separate journal, which can then be *replayed* safely if a system crash occurs. This process introduces unnecessary overhead because the data needs to be written twice, often resulting in a new set of problems, such as when the journal cannot be replayed properly.

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

Checksums and Self-Healing Data

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

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

Unparalleled Scalability

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

ZFS Snapshots

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

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

Simplified Administration

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

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

ZFS Terminology

This section describes the basic terminology used throughout this book:

alternate boot environment
: A boot environment that is created by the `lucreate` command and possibly updated by the `luupgrade` command, but it is not the active or primary boot environment. The alternate boot environment can become the primary boot environment by running the `luactivate` 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 208.

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

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

  pool/*path*[*@snapshot*]

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

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

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

  For more information about datasets, see Chapter 5, "Managing Oracle Solaris ZFS File Systems."

file system
: A ZFS dataset of type `filesystem` that is mounted within the standard system namespace and behaves like other file systems.

	For more information about file systems, see Chapter 5, "Managing Oracle Solaris ZFS File Systems."
mirror	A virtual device that stores identical copies of data on two or more disks. If any disk in a mirror fails, any other disk in that mirror can provide the same data.
pool	A logical group of devices describing the layout and physical characteristics of the available storage. Disk space for datasets is allocated from a pool. For more information about storage pools, see Chapter 3, "Managing Oracle Solaris ZFS Storage Pools."
primary boot environment	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.
RAID-Z	A virtual device that stores data and parity on multiple disks. For more information about RAID-Z, see "RAID-Z Storage Pool Configuration" on page 45.
resilvering	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. For more information about ZFS resilvering, see "Viewing Resilvering Status" on page 288.
snapshot	A read-only copy of a file system or volume at a given point in time. For more information about snapshots, see "Overview of ZFS Snapshots" on page 201.
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 53.
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 259.

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

In addition, empty components are not allowed.

Oracle Solaris ZFS and Traditional File System Differences

ZFS File System Granularity

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

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

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

For an example that shows how to create a file system hierarchy, see "Creating a ZFS File System Hierarchy" on page 36.

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 195. 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 198. This model is very similar to the NFS model, where multiple directories are mounted from the same file system (consider `/home`).

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

ZFS is a transactional file system. Most file system modifications are bundled into transaction groups and committed to disk asynchronously. Until these modifications are committed to disk, they are called *pending changes*. The amount of disk space used, available, and referenced by a file or file system does not consider pending changes. Pending changes are generally

accounted for within a few seconds. Even committing a change to disk by using `fsync(3c)` or `O_SYNC` does not necessarily guarantee that the disk space usage information is updated immediately.

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

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

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

- For files that are larger than `recordsize`, the last block of the file is generally about 1/2 full. With the default `recordsize` set to 128 KB, approximately 64 KB is wasted per file, which might be a large impact. The integration of RFE 6812608 would resolve this scenario. You can work around this by enabling compression. Even if your data is already compressed, the unused portion of the last block will be zero-filled, and compresses very well.
- On a RAIDZ-2 pool, every block consumes at least 2 sectors (512-byte chunks) of parity information. The space consumed by the parity information is not reported, but because it can vary, and be a much larger percentage for small blocks, an impact to space reporting might be seen. The impact is more extreme for a `recordsize` set to 512 bytes, where each 512-byte logical block consumes 1.5 KB (3 times the space). Regardless of the data being stored, if space efficiency is your primary concern, you should leave the `recordsize` at the default (128 KB), and enable compression (to the default of `lzjb`).
- The `df` command is not aware of deduplicated file data.

Out of Space Behavior

File system snapshots are inexpensive and easy to create in ZFS. Snapshots are common in most ZFS environments. For information about ZFS snapshots, see Chapter 6, “Working With Oracle Solaris ZFS Snapshots and Clones.”

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

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

Mounting ZFS File Systems

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

For more information about mounting and sharing ZFS file systems, see "Mounting ZFS File Systems" on page 188.

Traditional Volume Management

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

Solaris ACL Model Based on NFSv4

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 7, "Using ACLs and Attributes to Protect Oracle Solaris ZFS Files."

◆ ◆ ◆ CHAPTER 2

Getting Started With Oracle Solaris ZFS

This chapter provides step-by-step instructions on setting up a basic Oracle Solaris ZFS configuration. By the end of this chapter, you will have a basic understanding of how the ZFS commands work, and should be able to create a basic pool and file systems. This chapter does not provide a comprehensive overview and refers to later chapters for more detailed information.

The following sections are provided in this chapter:

ZFS Rights Profiles

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 8, "Oracle Solaris ZFS Delegated Administration."

ZFS Hardware and Software Requirements and Recommendations

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

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

Creating a Basic ZFS File System

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

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

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

For more information about redundant ZFS pool configurations, see "Replication Features of a ZFS Storage Pool" on page 45.

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

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

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

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

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

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

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

Typically, you want to create and organize a hierarchy of file systems that matches your organizational needs. For information about creating a hierarchy of ZFS file systems, see "Creating a ZFS File System Hierarchy" on page 36.

Creating a Basic ZFS Storage Pool

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

▼ How to Identify Storage Requirements for Your ZFS Storage Pool

1 Determine available devices for your storage pool.

Before creating a storage pool, you must determine which devices will store your data. These devices must be disks of at least 128 MB in size, and they must not be in use by other parts of the operating system. The devices can be individual slices on a preformatted disk, or they can be entire disks that ZFS formats as a single large slice.

In the storage example in "How to Create a ZFS Storage Pool" on page 36, assume that the whole disks `/dev/dsk/c1t0d0` and `/dev/dsk/c2t0d0` are available for use.

For more information about disks and how they are used and labeled, see "Using Disks in a ZFS Storage Pool" on page 41.

2 Choose data replication.

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

In the storage example in "How to Create a ZFS Storage Pool" on page 36, basic mirroring of two available disks is used.

For more information about ZFS replication features, see "Replication Features of a ZFS Storage Pool" on page 45.

▼ How to Create a ZFS Storage Pool

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

For more information about the ZFS rights profiles, see "ZFS Rights Profiles" on page 33.

2 **Pick a name for your storage pool.**

This name is used to identify the storage pool when you are using the `zpool` and `zfs` commands. Pick any pool name that you prefer, but it must satisfy the naming requirements in "ZFS Component Naming Requirements" on page 29.

3 **Create the pool.**

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

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

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

For more information about creating storage pools, see "Creating ZFS Storage Pools" on page 48. For more information about how device usage is determined, see "Detecting In-Use Devices" on page 55.

4 **View the results.**

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

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

For more information about viewing pool status, see "Querying ZFS Storage Pool Status" on page 81.

Creating a ZFS File System Hierarchy

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

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

How to Determine Your ZFS File System Hierarchy

1 **Pick the file system granularity.**

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

Two ZFS file systems, `jeff` and `bill`, are created in "How to Create ZFS File Systems" on page 37.

For more information about managing file systems, see Chapter 5, "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 37, 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 37, all home directories are mounted at `/export/zfs/`*user*, are shared by using NFS, and have compression enabled. In addition, a quota of 10 GB on user `jeff` is enforced.

For more information about properties, see "Introducing ZFS Properties" on page 169.

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 33.

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 169.

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

4 Create the individual file systems.

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

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

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

For more information about creating file systems, see "Creating a ZFS File System" on page 166.

For more information about mounting and sharing file systems, see "Mounting ZFS File Systems" on page 188.

5 Set the file system-specific properties.

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

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

6 View the results.

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

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

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

For more information about viewing file system status, see "Querying ZFS File System Information" on page 181.

For more information about how disk space is used and calculated, see "ZFS Disk Space Accounting" on page 30.

◆ ◆ ◆ CHAPTER 3

Managing Oracle Solaris ZFS Storage Pools

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

The following sections are provided in this chapter:

Components of a ZFS Storage Pool

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

Using Disks in a ZFS Storage Pool

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

A storage device can be a whole disk (`c1t0d0`) or an individual slice (`c0t0d0s7`). The recommended mode of operation is to use an entire disk, in which case the disk does not

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

```
Current partition table (original):
Total disk sectors available: 143358287 + 16384 (reserved sectors)

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

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

- When using a whole disk, the disk is generally named by using the `/dev/dsk/cNtNdN` naming convention. Some third-party drivers use a different naming convention or place disks in a location other than the `/dev/dsk` directory. To use these disks, you must manually label the disk and provide a slice to ZFS.
- On an x86 based system, the disk must have a valid Solaris `fdisk` partition. For more information about creating or changing a Solaris `fdisk` partition, see "Setting Up Disks for ZFS File Systems (Task Map)" in *System Administration Guide: Devices and File Systems*.
- ZFS applies an EFI label when you create a storage pool with whole disks. For more information about EFI labels, see "EFI (GPT) Disk Label" in *System Administration Guide: Devices and File Systems*.

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

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

Using Slices in a ZFS Storage Pool

Disks can be labeled with a Solaris VTOC (SMI) label when you create a storage pool with a disk slice, but using disk slices for a pool is not recommended because management of disk slices is more difficult.

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

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

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

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

```
# format
.
.
.
selecting c1t0d0
partition> p
Current partition table (original):
Total disk cylinders available: 49779 + 2 (reserved cylinders)

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

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

Using Files in a ZFS Storage Pool

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

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

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

Considerations for ZFS Storage Pools

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

- Using whole physical disks is the easiest way to create ZFS storage pools. ZFS configurations become progressively more complex, from management, reliability, and performance perspectives, when you build pools from disk slices, LUNs in hardware RAID arrays, or volumes presented by software-based volume managers. The following considerations might help you determine how to configure ZFS with other hardware or software storage solutions:
  - If you construct a ZFS configuration on top of LUNs from hardware RAID arrays, you need to understand the relationship between ZFS redundancy features and the redundancy features offered by the array. Certain configurations might provide adequate redundancy and performance, but other configurations might not.
  - You can construct logical devices for ZFS using volumes presented by software-based volume managers, 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 Chapter 11, "Recommended Oracle Solaris ZFS Practices."
- Disks are identified both by their path and by their device ID, if available. On systems where device ID information is available, this identification method allows devices to be reconfigured without updating ZFS. Because device ID generation and management can vary by system, export the pool first before moving devices, such as moving a disk from one controller to another controller. A system event, such as a firmware update or other hardware change, might change the device IDs in your ZFS storage pool, which can cause the devices to become unavailable.

Replication Features of a ZFS Storage Pool

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

Mirrored Storage Pool Configuration

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

```
mirror c1t0d0 c2t0d0
```

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

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

For information about creating a mirrored storage pool, see "Creating a Mirrored Storage Pool" on page 48.

RAID-Z Storage Pool Configuration

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

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

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

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

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

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

```
raidz c1t0d0 c2t0d0 c3t0d0
```

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

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

If you are creating a RAID-Z configuration with many disks, consider splitting the disks into multiple groupings. For example, a RAID-Z configuration with 14 disks is better split into two 7-disk groupings. RAID-Z configurations with single-digit groupings of disks should perform better.

For information about creating a RAID-Z storage pool, see "Creating a RAID-Z Storage Pool" on page 49.

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

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

For additional information about RAID-Z storage pool recommendations, see Chapter 11, "Recommended Oracle Solaris ZFS Practices."

ZFS Hybrid Storage Pool

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

For more information about this product, see the *Sun Storage Unified Storage System Administration Guide*.

Self-Healing Data in a Redundant Configuration

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

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

Dynamic Striping in a Storage Pool

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

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

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

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

Creating and Destroying ZFS Storage Pools

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

- "Creating ZFS Storage Pools" on page 48
- "Displaying Storage Pool Virtual Device Information" on page 53
- "Handling ZFS Storage Pool Creation Errors" on page 54
- "Destroying ZFS Storage Pools" on page 57

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

Creating ZFS Storage Pools

To create a storage pool, use the `zpool create` command. This command takes a pool name and any number of virtual devices as arguments. The pool name must satisfy the naming requirements in "ZFS Component Naming Requirements" on page 29.

Creating a Basic Storage Pool

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

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

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

Creating a Mirrored Storage Pool

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

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

The second `mirror` keyword indicates that a new top-level virtual device is being specified. Data is dynamically striped across both mirrors, with data being redundant between each disk appropriately.

For more information about recommended mirrored configurations, see Chapter 11, "Recommended Oracle Solaris ZFS Practices."

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

- Adding another set of disks for an additional top-level virtual device (vdev) to an existing mirrored configuration. For more information, see "Adding Devices to a Storage Pool" on page 58.
- Attaching additional disks to an existing mirrored configuration. Or, attaching additional disks to a non-replicated configuration to create a mirrored configuration. For more information, see "Attaching and Detaching Devices in a Storage Pool" on page 63.
- Replacing a disk or disks in an existing mirrored configuration as long as the replacement disks are greater than or equal to the size of the device to be replaced. For more information, see "Replacing Devices in a Storage Pool" on page 70.
- Detaching a disk in a mirrored configuration as long as the remaining devices provide adequate redundancy for the configuration. For more information, see "Attaching and Detaching Devices in a Storage Pool" on page 63.

- Splitting a mirrored configuration by detaching one of the disks to create a new, identical pool. For more information, see "Creating a New Pool By Splitting a Mirrored ZFS Storage Pool" on page 65.

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

Creating a ZFS Root Pool

Consider the following root pool configuration requirements:

- 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 4, "Installing and Booting an Oracle Solaris ZFS Root File System."

Creating a RAID-Z Storage Pool

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

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

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

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

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

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

errors: No known data errors

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

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          raidz3-0  ONLINE       0     0     0
            c0t0d0  ONLINE       0     0     0
            c1t0d0  ONLINE       0     0     0
            c2t0d0  ONLINE       0     0     0
            c3t0d0  ONLINE       0     0     0
            c4t0d0  ONLINE       0     0     0
            c5t0d0  ONLINE       0     0     0
            c6t0d0  ONLINE       0     0     0
            c7t0d0  ONLINE       0     0     0
            c8t0d0  ONLINE       0     0     0
errors: No known data errors
```

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

- Adding another set of disks for an additional top-level virtual device to an existing RAID-Z configuration. For more information, see "Adding Devices to a Storage Pool" on page 58.
- Replacing a disk or disks in an existing RAID-Z configuration as long as the replacement disks are greater than or equal to the size of the device to be replaced. For more information, see "Replacing Devices in a Storage Pool" on page 70.

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

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

For more information about a RAID-Z configuration, see "RAID-Z Storage Pool Configuration" on page 45.

Creating a ZFS Storage Pool With Log Devices

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

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

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

- Log devices for the ZFS intent log are not related to database log files.
- Any performance improvement seen by implementing a separate log device depends on the device type, the hardware configuration of the pool, and the application workload. For preliminary performance information, see this blog:

  `http://blogs.oracle.com/perrin/entry/slog_blog_or_blogging_on`
- Log devices can be unreplicated or mirrored, but RAID-Z is not supported for log devices.
- If a separate log device is not mirrored and the device that contains the log fails, storing log blocks reverts to the storage pool.
- Log devices can be added, replaced, removed, attached, detached, imported, and exported as part of the larger storage pool.
- You can attach a log device to an existing log device to create a mirrored log device. This operation is identical to attaching a device in a unmirrored storage pool.
- The minimum size of a log device is the same as the minimum size of each device in a pool, which is 64 MB. The amount of in-play data that might be stored on a log device is relatively small. Log blocks are freed when the log transaction (system call) is committed.
- The maximum size of a log device should be approximately 1/2 the size of physical memory because that is the maximum amount of potential in-play data that can be stored. For example, if a system has 16 GB of physical memory, consider a maximum log device size of 8 GB.

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

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

```
# zpool create datap mirror c0t5000C500335F95E3d0 c0t5000C500335F907Fd0 mirror
c0t5000C500335BD117d0 c0t5000C500335DC60Fd0 log mirror c0t5000C500335E106Bd0 c0t5000C500335FC3E7d0
# zpool status datap
  pool: datap
 state: ONLINE
 scrub: none requested
config:

        NAME                       STATE     READ WRITE CKSUM
```

```
        datap                     ONLINE       0     0     0
          mirror-0                ONLINE       0     0     0
            c0t5000C500335F95E3d0 ONLINE       0     0     0
            c0t5000C500335F907Fd0 ONLINE       0     0     0
          mirror-1                ONLINE       0     0     0
            c0t5000C500335BD117d0 ONLINE       0     0     0
            c0t5000C500335DC60Fd0 ONLINE       0     0     0
        logs
          mirror-2                ONLINE       0     0     0
            c0t5000C500335E106Bd0 ONLINE       0     0     0
            c0t5000C500335FC3E7d0 ONLINE       0     0     0

errors: No known data errors
```

For information about recovering from a log device failure, see Example 10–2.

Creating a ZFS Storage Pool With Cache Devices

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

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

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

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

errors: No known data errors
```

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

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

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

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

- Using cache devices provides the greatest performance improvement for random-read workloads of mostly static content.
- Capacity and reads can be monitored by using the `zpool iostat` command.
- Single or multiple cache devices can be added when the pool is created. They can also be added and removed after the pool is created. For more information, see Example 3–4.
- Cache devices cannot be mirrored or be part of a RAID-Z configuration.
- If a read error is encountered on a cache device, that read I/O is reissued to the original storage pool device, which might be part of a mirrored or a RAID-Z configuration. The content of the cache devices is considered volatile, similar to other system caches.

Cautions For Creating Storage Pools

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

- Do not repartition or relabel disks that are part of an existing storage pool. If you attempt to repartition or relabel a root pool disk, you might have to reinstall the OS.
- Do not create a storage pool that contains components from another storage pool, such files or volumes. Deadlocks can occur in this unsupported configuration.
- A pool created with a single slice or single disk has no redundancy and is at risk for data loss. A pool created with multiple slices but no redundancy is also at risk for data loss. A pool created with multiple slices across disks is harder to manage than a pool created with whole disks.
- A pool that is not created with ZFS redundancy (RAIDZ or mirror) can only report data inconsistencies. It cannot repair data inconsistencies.
- Although a pool that is created with ZFS redundancy can help reduce down time due to hardware failures, it is not immune to hardware failures, power failures, or disconnected cables. Make sure you backup your data on a regular basis. Performing routine backups of pool data on non-enterprise grade hardware is important.
- A pool cannot be shared across systems. ZFS is not a cluster file system.

Displaying Storage Pool Virtual Device Information

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

If the top-level virtual device contains two or more physical devices, the configuration provides data redundancy as mirror or RAID-Z virtual devices. These virtual devices consist of disks, disk slices, or files. A spare is a special virtual dev that tracks available hot spares for a pool.

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

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

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

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

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

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

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

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

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

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            c0t1d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            c0t2d0  ONLINE       0     0     0
            c1t2d0  ONLINE       0     0     0
          mirror-2  ONLINE       0     0     0
            c0t3d0  ONLINE       0     0     0
            c1t3d0  ONLINE       0     0     0

errors: No known data errors
```

Handling ZFS Storage Pool Creation Errors

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

Detecting In-Use Devices

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

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

Live upgrade — The disk is in use as an alternate boot environment for Oracle Solaris Live Upgrade.

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

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

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

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

Mismatched Replication Levels

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

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

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

Doing a Dry Run of Storage Pool Creation

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

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

        tank
          mirror
            c1t0d0
            c1t1d0
```

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

Default Mount Point for Storage Pools

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

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

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

For more information about mount points, see "Managing ZFS Mount Points" on page 188.

Destroying ZFS Storage Pools

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

```
# zpool destroy tank
```

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

If you destroy a pool with the `zpool destroy` command, the pool is still available for import as described in "Recovering Destroyed ZFS Storage Pools" on page 98. This means that confidential data might still be available on the disks that were part of the pool. If you want to destroy data on the destroyed pool's disks, you must use a feature like the `format` utility's `analyze->purge` option on every disk in the destroyed pool.

Destroying a Pool With Unavailable 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 `UNAVAIL` devices such that the pool itself is `UNAVAIL` (meaning that a top-level virtual device is `UNAVAIL`), 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 86.

For more information about importing pools, see "Importing ZFS Storage Pools" on page 95.

Managing Devices in ZFS Storage Pools

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

Adding Devices to a Storage Pool

You can dynamically add disk space to a pool by adding a new top-level virtual device. This disk space is immediately available to all datasets in the pool. To add a new virtual device to a pool, use the `zpool add` command. For example:

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

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

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

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

For more information about how virtual device validation is done, see "Detecting In-Use Devices" on page 55.

EXAMPLE 3–1 Adding Disks to a Mirrored ZFS Configuration

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

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

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            c0t1d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            c0t2d0  ONLINE       0     0     0
            c1t2d0  ONLINE       0     0     0

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

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            c0t1d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            c0t2d0  ONLINE       0     0     0
            c1t2d0  ONLINE       0     0     0
          mirror-2  ONLINE       0     0     0
            c0t3d0  ONLINE       0     0     0
```

EXAMPLE 3–1 Adding Disks to a Mirrored ZFS Configuration *(Continued)*

```
            c1t3d0  ONLINE       0     0     0

errors: No known data errors
```

EXAMPLE 3–2 Adding Disks to a RAID-Z Configuration

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

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

        NAME        STATE     READ WRITE CKSUM
        rzpool      ONLINE       0     0     0
          raidz1-0  ONLINE       0     0     0
            c1t2d0  ONLINE       0     0     0
            c1t3d0  ONLINE       0     0     0
            c1t4d0  ONLINE       0     0     0

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

        NAME        STATE     READ WRITE CKSUM
        rzpool      ONLINE       0     0     0
          raidz1-0  ONLINE       0     0     0
            c1t0d0  ONLINE       0     0     0
            c1t2d0  ONLINE       0     0     0
            c1t3d0  ONLINE       0     0     0
          raidz1-1  ONLINE       0     0     0
            c2t2d0  ONLINE       0     0     0
            c2t3d0  ONLINE       0     0     0
            c2t4d0  ONLINE       0     0     0

errors: No known data errors
```

EXAMPLE 3–3 Adding and Removing a Mirrored Log Device

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

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

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

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

errors: No known data errors
# zpool add newpool log mirror c0t6d0 c0t7d0
# zpool status newpool
  pool: newpool
 state: ONLINE
 scrub: none requested
config:

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

errors: No known data errors
```

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

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

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

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

errors: No known data errors
```

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

```
# zpool status pool
  pool: pool
```

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

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

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

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

EXAMPLE 3–4 Adding and Removing Cache Devices

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

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

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

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

errors: No known data errors
```

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

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

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

        NAME        STATE     READ WRITE CKSUM
```

EXAMPLE 3–4 Adding and Removing Cache Devices *(Continued)*

```
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            c2t0d0  ONLINE       0     0     0
            c2t1d0  ONLINE       0     0     0
            c2t3d0  ONLINE       0     0     0

errors: No known data errors
```

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

For more information about using cache devices in a ZFS storage pool, see "Creating a ZFS Storage Pool With Cache Devices" on page 52.

Attaching and Detaching Devices in a Storage Pool

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

If you are attaching a disk to create a mirrored root pool, see "How to Create a Mirrored ZFS Root Pool (Postinstallation)" on page 113.

If you are replacing a disk in a ZFS root pool, see "Recovering the ZFS Root Pool or Root Pool Snapshots" on page 157.

EXAMPLE 3–5 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:

        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

errors: No known data errors
# zpool attach zeepool c1t1d0 c2t1d0
# zpool status zeepool
  pool: zeepool
```

EXAMPLE 3–5 Converting a Two-Way Mirrored Storage Pool to a Three-way Mirrored Storage Pool *(Continued)*

```
 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 3–6 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

errors: No known data errors
```

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

```
# zpool detach zeepool c2t1d0
```

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

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

Creating a New Pool By Splitting a Mirrored ZFS Storage Pool

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

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

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

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

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

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

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

errors: No known data errors

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

```
config:

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

errors: No known data errors
```

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

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

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

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

```
# zpool split -R /tank2 tank tank2
```

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

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

- This feature is not available for a RAID-Z configuration or a non-redundant pool of multiple disks.
- Data and application operations should be quiesced before attempting a `zpool split` operation.
- A pool cannot be split if resilvering is in process.
- Splitting a mirrored pool is optimal when the pool contains two to three disks, where the last disk in the original pool is used for the newly created pool. Then, you can use the `zpool attach` command to re-create your original mirrored storage pool or convert your newly created pool into a mirrored storage pool. No method currently exists to create a *new* mirrored pool from an *existing* mirrored pool in one `zpool split` operation because the new (split) pool is non-redundant
- If the existing pool is a three-way mirror, then the new pool will contain one disk after the split operation. If the existing pool is a two-way mirror of two disks, then the outcome is two non-redundant pools of two disks. You must attach two additional disks to convert the non-redundant pools to mirrored pools.

- A good way to keep your data redundant during a split operation is to split a mirrored storage pool that contains three disks so that the original pool contains two mirrored disks after the split operation.
- Confirm that your hardware is configured correctly before splitting a mirrored pool. For related information about confirming your hardware's cache flush settings, see "General System Practices" on page 301.

EXAMPLE 3–7 Splitting a Mirrored ZFS Pool

In the following example, a mirrored storage pool called `mothership`, with three disks is split. The two resulting pools are the mirrored pool `mothership`, with two disks and the new pool, `luna`, with one disk. Each pool has identical content.

The pool `luna` could be imported on another system for backup purposes. After the backup is complete, the pool `luna` could be destroyed and the disk reattached to the `mothership`. Then, the process could be repeated.

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

        NAME                       STATE     READ WRITE CKSUM
        mothership                 ONLINE       0     0     0
          mirror-0                 ONLINE       0     0     0
            c0t5000C500335F95E3d0  ONLINE       0     0     0
            c0t5000C500335BD117d0  ONLINE       0     0     0
            c0t5000C500335F907Fd0  ONLINE       0     0     0

errors: No known data errors
# zpool split mothership luna
# zpool import luna
# zpool status mothership luna
  pool: luna
 state: ONLINE
  scan: none requested
config:

        NAME                     STATE     READ WRITE CKSUM
        luna                     ONLINE       0     0     0
          c0t5000C500335F907Fd0  ONLINE       0     0     0

errors: No known data errors

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

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

EXAMPLE 3–7 Splitting a Mirrored ZFS Pool *(Continued)*

```
            c0t5000C500335F95E3d0  ONLINE       0     0     0
            c0t5000C500335BD117d0  ONLINE       0     0     0

errors: No known data errors
```

Onlining and Offlining Devices in a Storage Pool

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

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

Taking a Device Offline

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

```
# zpool offline tank c0t5000C500335F95E3d0
```

Consider the following points when taking a device offline:

- You cannot take a pool offline to the point where it becomes `UNAVAIL`. 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 c0t5000C500335F95E3d0
  cannot offline c0t5000C500335F95E3d0: 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 70.

Offline devices are in the `OFFLINE` state when you query pool status. For information about querying pool status, see "Querying ZFS Storage Pool Status" on page 81.

For more information on device health, see "Determining the Health Status of ZFS Storage Pools" on page 86.

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 c0t5000C500335F95E3d0
```

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 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 `UNAVAIL` state.

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

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

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

```
SUNW-MSG-ID: ZFS-8000-D3, TYPE: Fault, VER: 1, SEVERITY: Major
EVENT-TIME: Wed Jun 20 11:35:26 MDT 2012
PLATFORM: SUNW,Sun-Fire-880, CSN: -, HOSTNAME: neo
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: 504a1188-b270-4ab0-af4e-8a77680576b8
DESC: A ZFS device failed.  Refer to http://sun.com/msg/ZFS-8000-D3 for more information.
AUTO-RESPONSE: No automated response will occur.
IMPACT: Fault tolerance of the pool may be compromised.
REC-ACTION: Run 'zpool status -x' and replace the bad device.
```

For more information about replacing a faulted device, see "Resolving a Missing or Removed Device" on page 276.

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

```
# zpool online -e tank c0t5000C500335F95E3d0
```

Clearing Storage Pool Device Errors

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

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

```
# zpool clear tank
```

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

```
# zpool clear tank c0t5000C500335F95E3d0
```

For more information about clearing `zpool` errors, see "Clearing Transient Device Errors" on page 281.

Replacing Devices in a Storage Pool

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

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

```
# zpool replace tank c1t1d0
```

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

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

If you are replacing a disk in the ZFS root pool, see "Recovering the ZFS Root Pool or Root Pool Snapshots" on page 157.

The following are the basic steps for replacing a disk:

1. Offline the disk, if necessary, with the `zpool offline` command.
2. Remove the disk to be replaced.
3. Insert the replacement disk.
4. Review the `format` output to determine if the replacement disk is visible.

   In addition, check to see whether the device ID has changed. If the replacement disk has a WWN, then the device ID for the failed disk is changed.
5. Let ZFS know the disk is replaced. For example:

   ```
   # zpool replace tank c1t1d0
   ```

   If the replacement disk has a different device ID as identified above, include the new device ID.

   ```
   # zpool replace tank c0t5000C500335FC3E7d0 c0t5000C500335BA8C3d0
   ```
6. Bring the disk online with the `zpool online` command, if necessary.
7. Notify FMA that the device is replaced.

   From `fmadm faulty` output, identify the `zfs://pool=name/vdev=guid` string in the `Affects:` section and provide that string as an argument to the `fmadm repaired` command.

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

On some systems with SATA disks, you must unconfigure a disk before you can take it offline. If you are replacing a disk in the same slot position on this system, then you can just run the `zpool replace` command as described in the first example in this section.

For an example of replacing a SATA disk, see Example 10–1.

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

- If you set the `autoreplace` pool property to `on`, then any new device found in the same physical location as a device that previously belonged to the pool is automatically formatted and replaced. You are not required to use the `zpool replace` command when this property is enabled. This feature might not be available on all hardware types.
- The storage pool state `REMOVED` is provided when a device or hot spare has been physically removed while the system was running. A hot spare device is substituted for the removed device, if available.
- If a device is removed and then reinserted, the device is placed online. If a hot spare was activated when the device was reinserted, the hot spare is removed when the online operation completes.

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

  In the following example, two 16-GB disks in a mirrored pool are replaced with two 72-GB disks. Ensure that the first device is completely resilvered before attempting the second device replacement. The `autoexpand` property is enabled after the disk replacements to expand the full disk sizes.

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

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

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

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

- If a failed disk has been replaced automatically with a hot spare, then you might need to detach the spare after the failed disk is replaced. You can use the `zpool detach` command to detach a spare in a mirrored or RAID-Z pool. For information about detaching a hot spare, see "Activating and Deactivating Hot Spares in Your Storage Pool" on page 75.

For more information about replacing devices, see "Resolving a Missing or Removed Device" on page 276 and "Replacing or Repairing a Damaged Device" on page 279.

Designating Hot Spares in Your Storage Pool

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

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

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

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

```
# zpool create zeepool mirror c0t5000C500335F95E3d0 c0t5000C500335F907Fd0
mirror c0t5000C500335BD117d0 c0t5000C500335DC60Fd0 spare c0t5000C500335E106Bd0 c0t5000C500335FC3E7d0
# zpool status zeepool
  pool: zeepool
 state: ONLINE
  scan: none requested
config:

        NAME                       STATE     READ WRITE CKSUM
        zeepool                    ONLINE       0     0     0
          mirror-0                 ONLINE       0     0     0
            c0t5000C500335F95E3d0  ONLINE       0     0     0
            c0t5000C500335F907Fd0  ONLINE       0     0     0
          mirror-1                 ONLINE       0     0     0
            c0t5000C500335BD117d0  ONLINE       0     0     0
            c0t5000C500335DC60Fd0  ONLINE       0     0     0
        spares
          c0t5000C500335E106Bd0    AVAIL
          c0t5000C500335FC3E7d0    AVAIL

errors: No known data errors
```

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

```
# zpool add zeepool spare c0t5000C500335E106Bd0 c0t5000C500335FC3E7d0
# zpool status zeepool
  pool: zeepool
```

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

        NAME                      STATE     READ WRITE CKSUM
        zeepool                   ONLINE       0     0     0
          mirror-0                ONLINE       0     0     0
            c0t5000C500335F95E3d0 ONLINE       0     0     0
            c0t5000C500335F907Fd0 ONLINE       0     0     0
          mirror-1                ONLINE       0     0     0
            c0t5000C500335BD117d0 ONLINE       0     0     0
            c0t5000C500335DC60Fd0 ONLINE       0     0     0
        spares
          c0t5000C500335E106Bd0   AVAIL
          c0t5000C500335FC3E7d0   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 c0t5000C500335FC3E7d0
# zpool status zeepool
  pool: zeepool
 state: ONLINE
  scan: none requested
config:

        NAME                      STATE     READ WRITE CKSUM
        zeepool                   ONLINE       0     0     0
          mirror-0                ONLINE       0     0     0
            c0t5000C500335F95E3d0 ONLINE       0     0     0
            c0t5000C500335F907Fd0 ONLINE       0     0     0
          mirror-1                ONLINE       0     0     0
            c0t5000C500335BD117d0 ONLINE       0     0     0
            c0t5000C500335DC60Fd0 ONLINE       0     0     0
        spares
          c0t5000C500335E106Bd0   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.

An `UNAVAIL` device is automatically replaced if a hot spare is available. For example:

```
# zpool status -x
  pool: zeepool
 state: DEGRADED
status: One or more devices are unavailable in response to persistent errors.
        Sufficient replicas exist for the pool to continue functioning in a
        degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://www.sun.com/msg/ZFS-8000-2Q
  scan: resilvered 3.15G in 0h0m with 0 errors on Mon Nov 12 15:53:42 2012
config:

        NAME                       STATE     READ WRITE CKSUM
        zeepool                    DEGRADED     0     0     0
          mirror-0                 ONLINE       0     0     0
            c0t5000C500335F95E3d0  ONLINE       0     0     0
            c0t5000C500335F907Fd0  ONLINE       0     0     0
          mirror-1                 DEGRADED     0     0     0
            c0t5000C500335BD117d0  ONLINE       0     0     0
            spare-1                DEGRADED   449     0     0
              c0t5000C500335DC60Fd0  UNAVAIL      0     0     0
              c0t5000C500335E106Bd0  ONLINE       0     0     0
        spares
          c0t5000C500335E106Bd0    INUSE

errors: No known data errors
```

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

- By removing the hot spare from the storage pool.
- By detaching a hot spare after a failed disk is physically replaced. See Example 3–8.
- By temporarily or permanently swapping in another hot spare. See Example 3–9.

EXAMPLE 3–8 Detaching a Hot Spare After the Failed Disk Is Replaced

In this example, the failed disk (`c0t5000C500335DC60Fd0`) is physically replaced and ZFS is notified by using the `zpool replace` command.

```
# zpool replace zeepool c0t5000C500335DC60Fd0
# zpool status zeepool
  pool: zeepool
 state: ONLINE
  scan: resilvered 3.15G in 0h0m with 0 errors on Thu Jun 21 16:53:43 2012
config:

        NAME                       STATE     READ WRITE CKSUM
        zeepool                    ONLINE       0     0     0
          mirror-0                 ONLINE       0     0     0
            c0t5000C500335F95E3d0  ONLINE       0     0     0
            c0t5000C500335F907Fd0  ONLINE       0     0     0
          mirror-1                 ONLINE       0     0     0
            c0t5000C500335BD117d0  ONLINE       0     0     0
            c0t5000C500335DC60Fd0  ONLINE       0     0     0
        spares
          c0t5000C500335E106Bd0    AVAIL
```

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

```
# zpool detach zeepool c0t5000C500335E106Bd0
```

EXAMPLE 3–9 Detaching a Failed Disk and Using the Hot Spare

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

```
# zpool status zeepool
  pool: zeepool
 state: DEGRADED
status: One or more devices are unavailable in response to persistent errors.
        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: resilver in progress since Mon Nov 12 16:04:12 2012
     4.80G scanned out of 12.0G at 55.8M/s, 0h2m to go
     4.80G scanned out of 12.0G at 55.8M/s, 0h2m to go
     4.77G resilvered, 39.97% done
config:

        NAME                       STATE     READ WRITE CKSUM
        zeepool                    DEGRADED     0     0     0
          mirror-0                 ONLINE       0     0     0
            c0t5000C500335F95E3d0  ONLINE       0     0     0
            c0t5000C500335F907Fd0  ONLINE       0     0     0
          mirror-1                 DEGRADED     0     0     0
```

EXAMPLE 3–9 Detaching a Failed Disk and Using the Hot Spare *(Continued)*

```
            c0t5000C500335BD117d0  ONLINE       0     0     0
            c0t5000C500335DC60Fd0  UNAVAIL      0     0     0
        spares
          c0t5000C500335E106Bd0    AVAIL

errors: No known data errors
# zpool detach zeepool c0t5000C500335DC60Fd0
# zpool status zeepool
  pool: zeepool
 state: ONLINE
  scan: resilvered 11.3G in 0h3m with 0 errors on Mon Nov 12 16:07:12 2012
config:

        NAME                      STATE     READ WRITE CKSUM
        zeepool                   ONLINE       0     0     0
          mirror-0                ONLINE       0     0     0
            c0t5000C500335F95E3d0  ONLINE       0     0     0
            c0t5000C500335F907Fd0  ONLINE       0     0     0
          mirror-1                ONLINE       0     0     0
            c0t5000C500335BD117d0  ONLINE       0     0     0
            c0t5000C500335E106Bd0  ONLINE       0     0     0

errors: No known data errors
(Original failed disk c0t5000C500335DC60Fd0 is physically replaced)
# zpool add zeepool spare c0t5000C500335DC60Fd0
# zpool status zeepool
  pool: zeepool
 state: ONLINE
  scan: resilvered 11.2G in 0h3m with 0 errors on Mon Nov 12 16:07:12 2012

config:

        NAME                      STATE     READ WRITE CKSUM
        zeepool                   ONLINE       0     0     0
          mirror-0                ONLINE       0     0     0
            c0t5000C500335F95E3d0  ONLINE       0     0     0
            c0t5000C500335F907Fd0  ONLINE       0     0     0
          mirror-1                ONLINE       0     0     0
            c0t5000C500335BD117d0  ONLINE       0     0     0
            c0t5000C500335E106Bd0  ONLINE       0     0     0
        spares
          c0t5000C500335DC60Fd0    AVAIL

errors: No known data errors
```

After a disk is replaced and the spare is detached, let FMA know that the disk is repaired.

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

Managing ZFS Storage Pool Properties

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

```
# zpool get all tank
tank  size           68G                  -
tank  capacity       0%                   -
tank  altroot        -                    default
tank  health         ONLINE               -
tank  guid           15560293364730146756  -
tank  version        32                   default
tank  bootfs         -                    default
tank  delegation     on                   default
tank  autoreplace    off                  default
tank  cachefile      -                    default
tank  failmode       wait                 default
tank  listsnapshots  on                   default
tank  autoexpand     off                  default
tank  free           68.0G                -
tank  allocated      124K                 -
tank  readonly       off                  -
```

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

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

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

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

For information on preventing pool space capacity problems, see Chapter 11, "Recommended Oracle Solaris ZFS Practices."

TABLE 3–1 ZFS Pool Property Descriptions

Property Name	Type	Default Value	Description
`allocated`	String	N/A	Read-only value that identifies the amount of storage space within the pool that has been physically allocated.
`altroot`	String	`off`	Identifies an alternate root directory. If set, this directory is prepended to any mount points within the pool. This property can be used when you are examining an unknown pool, if the mount points cannot be trusted, or in an alternate boot environment, where the typical paths are not valid.

TABLE 3–1 ZFS Pool Property Descriptions *(Continued)*

Property Name	Type	Default Value	Description
`autoreplace`	Boolean	`off`	Controls automatic device replacement. If set to `off`, device replacement must be initiated by using the `zpool replace` 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 `replace`.
`bootfs`	Boolean	N/A	Identifies the default bootable file system for the root pool. This property is typically set by the installation programs.
`cachefile`	String	N/A	Controls where pool configuration information is cached. All pools in the cache are automatically imported when the system boots. However, installation and clustering environments might require this information to be cached in a different location so that pools are not automatically imported. You can set this property to cache pool configuration information in a different location. This information can be imported later by using the `zpool import -c` command. For most ZFS configurations, this property is not used.
`capacity`	Number	N/A	Read-only value that identifies the percentage of pool space used. The property abbreviation is `cap`.
`delegation`	Boolean	`on`	Controls whether a nonprivileged user can be granted access permissions that are defined for a file system. For more information, see Chapter 8, "Oracle Solaris ZFS Delegated Administration."

TABLE 3–1 ZFS Pool Property Descriptions *(Continued)*

Property Name	Type	Default Value	Description
`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, SUSPENDED, REMOVED, or UNAVAIL.
`listshares`	String	off	Controls whether share information in this pool is displayed with the `zfs list` command. The default value is `off`.
`listsnapshots`	String	`on`	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.
`size`	Number	N/A	Read-only property that identifies the total size of the storage pool.

TABLE 3–1 ZFS Pool Property Descriptions *(Continued)*

Property Name	Type	Default Value	Description
`version`	Number	N/A	Identifies the current on-disk version of the pool. The preferred method of updating pools is with the `zpool upgrade` command, although this property can be used when a specific version is needed for backwards compatibility. This property can be set to any number between 1 and the current version reported by the `zpool upgrade -v` command.

Querying ZFS Storage Pool Status

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

- "Displaying Information About ZFS Storage Pools" on page 81
- "Viewing I/O Statistics for ZFS Storage Pools " on page 84
- "Determining the Health Status of ZFS Storage Pools" on page 86

Displaying Information About ZFS Storage Pools

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

Displaying Information About All Storage Pools or a Specific Pool

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

```
# zpool list
NAME                    SIZE    ALLOC   FREE    CAP  HEALTH     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 30.

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 86.
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 267.

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  DEDUP  HEALTH  ALTROOT
pool   33.8G  83.5K  33.7G     0%  1.00x  ONLINE  -
rpool  33.8G  12.2G  21.5G    36%  1.00x  ONLINE  -
Tue Nov  2 10:36:14 MDT 2010
pool   33.8G  83.5K  33.7G     0%  1.00x  ONLINE  -
rpool  33.8G  12.2G  21.5G    36%  1.00x  ONLINE  -
```

Displaying Specific Storage Pool Statistics

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

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

The column names correspond to the properties that are listed in "Displaying Information About All Storage Pools or a Specific Pool" on page 81.

Scripting ZFS Storage Pool Output

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

```
# zpool list -Ho name
tank
dozer
```

Here is another example:

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

Displaying ZFS Storage Pool Command History

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

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

```
# zpool history
History for 'rpool':
2010-05-11.10:18:54 zpool create -f -o failmode=continue -R /a -m legacy -o
cachefile=/tmp/root/etc/zfs/zpool.cache rpool mirror c1t0d0s0 c1t1d0s0
2010-05-11.10:18:55 zfs set canmount=noauto rpool
2010-05-11.10:18:55 zfs set mountpoint=/rpool rpool
2010-05-11.10:18:56 zfs create -o mountpoint=legacy rpool/ROOT
2010-05-11.10:18:57 zfs create -b 8192 -V 2048m rpool/swap
2010-05-11.10:18:58 zfs create -b 131072 -V 1536m rpool/dump
2010-05-11.10:19:01 zfs create -o canmount=noauto rpool/ROOT/zfsBE
2010-05-11.10:19:02 zpool set bootfs=rpool/ROOT/zfsBE rpool
2010-05-11.10:19:02 zfs set mountpoint=/ rpool/ROOT/zfsBE
2010-05-11.10:19:03 zfs set canmount=on rpool
2010-05-11.10:19:04 zfs create -o mountpoint=/export rpool/export
2010-05-11.10:19:05 zfs create rpool/export/home
2010-05-11.11:11:10 zpool set bootfs=rpool rpool
2010-05-11.11:11:10 zpool set bootfs=rpool/ROOT/zfsBE rpool
```

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

The features of the history log are as follows:

- The log cannot be disabled.
- The log is saved persistently on disk, which means that the log is saved across system reboots.

- The log is implemented as a ring buffer. The minimum size is 128 KB. The maximum size is 32 MB.
- For smaller pools, the maximum size is capped at 1 percent of the pool size, where the *size* is determined at pool creation time.
- The log requires no administration, which means that tuning the size of the log or changing the location of the log is unnecessary.

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

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

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

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

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

Viewing I/O Statistics for ZFS Storage Pools

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

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

	For more information about the differences between pool space and dataset space, see "ZFS Disk Space Accounting" on page 30.
`free capacity`	The amount of disk space available in the pool or device. Like the `used` statistic, this amount differs from the amount of disk space available to datasets by a small margin.
`read operations`	The number of read I/O operations sent to the pool or device, including metadata requests.
`write operations`	The number of write I/O operations sent to the pool or device.
`read bandwidth`	The bandwidth of all read operations (including metadata), expressed as units per second.
`write bandwidth`	The bandwidth of all write operations, expressed as units per second.

Listing Pool-Wide I/O Statistics

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

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

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

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

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

For example, `zpool iostat 2 3` would print a summary every two seconds for three iterations, for a total of six seconds. If there is only a single pool, then the statistics are displayed on consecutive lines. If more than one pool exists, then an additional dashed line delineates each iteration to provide visual separation.

Listing Virtual Device I/O Statistics

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

```
# zpool iostat -v
               capacity     operations    bandwidth
pool        alloc   free   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
rpool       6.05G  61.9G      0      0    785    107
  mirror    6.05G  61.9G      0      0    785    107
    c1t0d0s0    -      -      0      0    578    109
    c1t1d0s0    -      -      0      0    595    109
----------  -----  -----  -----  -----  -----  -----
tank        36.5G  31.5G      4      1   295K   146K
  mirror    36.5G  31.5G    126     45  8.13M  4.01M
    c1t2d0      -      -      0      3   100K   386K
    c1t3d0      -      -      0      3   104K   386K
----------  -----  -----  -----  -----  -----  -----
```

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

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

You can use the same set of options (interval and count) when examining virtual device statistics.

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 10, "Oracle Solaris ZFS Troubleshooting and Pool Recovery."

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

`DEGRADED`
: A pool with one or more failed devices, but the data is still available due to a redundant configuration.

`ONLINE`
: A pool that has all devices operating normally.

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

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

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

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

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

The `zpool status` command also provides details about resilver and scrub operations.

- Resilver in-progress report. For example:

```
scan: resilver in progress since Wed Jun 20 14:19:38 2012
    7.43G scanned out of 71.8G at 36.4M/s, 0h30m to go
    7.43G resilvered, 10.35% done
```

- Scrub in-progress report. For example:

```
scan: scrub in progress since Wed Jun 20 14:56:52 2012
    529M scanned out of 71.8G at 48.1M/s, 0h25m to go
    0 repaired, 0.72% done
```

- Resilver completion message. For example:

```
scan: resilvered 71.8G in 0h14m with 0 errors on Wed Jun 20 14:33:42 2012
```

- Scrub completion message. For example:

```
scan: scrub repaired 0 in 0h11m with 0 errors on Wed Jun 20 15:08:23 2012
```

- Ongoing scrub cancellation message. For example:

```
scan: scrub canceled on Wed Jun 20 16:04:40 2012
```

- Scrub and resilver completion messages persist across system reboots

Basic Storage Pool Health Status

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

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

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

Detailed Health Status

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

```
# zpool status -v tank
  pool: tank
 state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist for
        the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://www.sun.com/msg/ZFS-8000-2Q
 scrub: scrub completed after 0h0m with 0 errors on Wed Jan 20 15:13:59 2010
config:

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

```
            c1t0d0  ONLINE       0     0     0
            c1t1d0  UNAVAIL      0     0     0  cannot open

errors: No known data errors
```

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

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

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

# zpool online pond c0t5000C500335F907Fd0
warning: device 'c0t5000C500335DC60Fd0' onlined, but remains in degraded state
# zpool status -x
all pools are healthy
```

The above output identifies that the device remains in a degraded state until any resilvering is complete.

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

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

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

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

errors: No known data errors

# zpool status -x
 pool: pond
 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'.
config:

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

errors: No known data errors
```

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

The `errors:` field identifies any known data errors.

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

For more information about diagnosing and repairing `UNAVAIL` pools and data, see Chapter 10, "Oracle Solaris ZFS Troubleshooting and Pool Recovery."

Gathering ZFS Storage Pool Status Information

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

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

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

errors: No known data errors
```

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

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

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

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

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

errors: No known data errors

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

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

errors: No known data errors
```

Migrating ZFS Storage Pools

Occasionally, you might need to move a storage pool between systems. To do so, the storage devices must be disconnected from the original system and reconnected to the destination system. This task can be accomplished by physically recabling the devices, or by using multiported devices such as the devices on a SAN. ZFS enables you to export the pool from one system and import it on the destination system, even if the systems are of different architectural

endianness. For information about replicating or migrating file systems between different storage pools, which might reside on different systems, see "Sending and Receiving ZFS Data" on page 210.

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 `UNAVAIL` on the original system because the devices are no longer present. By default, the destination system cannot import a pool that has not been explicitly exported. This condition is necessary to prevent you from accidentally importing an active pool that consists of network-attached storage that is still in use on another system.

Exporting a ZFS Storage Pool

To export a pool, use the `zpool export` command. For example:

```
# zpool export tank
```

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

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

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

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

If ZFS volumes are in use in the pool, the pool cannot be exported, even with the `-f` option. To export a pool with a ZFS volume, first ensure that all consumers of the volume are no longer active.

For more information about ZFS volumes, see "ZFS Volumes" on page 259.

Determining Available Storage Pools to Import

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

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

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

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

Similar to the `zpool status` command output, the `zpool import` output includes a link to a knowledge article with the most up-to-date information regarding repair procedures for the problem that is preventing a pool from being imported. In this case, the user can force the pool to be imported. However, importing a pool that is currently in use by another system over a storage network can result in data corruption and panics as both systems attempt to write to the same storage. If some devices in the pool are not available but sufficient redundant data exists to provide a usable pool, the pool appears in the `DEGRADED` state. For example:

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

```
config:

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

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

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

```
# zpool import
pool: dozer
    id: 9784486589352144634
 state: FAULTED
status: One or more devices are missing from the system.
action: The pool cannot be imported. Attach the missing
        devices and try again.
   see: http://www.sun.com/msg/ZFS-8000-6X
config:
        dozer          FAULTED   missing device
          raidz1-0     ONLINE
            c1t0d0     ONLINE
            c1t1d0     ONLINE
            c1t2d0     ONLINE
            c1t3d0     ONLINE
Additional devices are known to be part of this pool, though their
exact configuration cannot be determined.
```

Importing ZFS Storage Pools From Alternate Directories

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

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

```
config:

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

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

Importing ZFS Storage Pools

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

```
# zpool import tank
```

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

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

        dozer       ONLINE
          c1t9d0    ONLINE

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

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

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

```
# zpool import dozer zeepool
```

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

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

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

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

Pools can also be imported under an alternate root by using the `-R` option. For more information on alternate root pools, see "Using ZFS Alternate Root Pools" on page 267.

Importing a Pool With a Missing Log Device

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

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

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

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

```
# zpool import -m dozer
# zpool status dozer
  pool: dozer
 state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist for
        the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://www.sun.com/msg/ZFS-8000-2Q
  scan: none requested
config:

        NAME                    STATE     READ WRITE CKSUM
        dozer                   DEGRADED     0     0     0
          mirror-0              ONLINE       0     0     0
            c8t0d0              ONLINE       0     0     0
            c8t1d0              ONLINE       0     0     0
        logs
          2189413556875979854   UNAVAIL      0     0     0

errors: No known data errors
```

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

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

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

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

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

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

Importing a Pool in Read-Only Mode

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

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

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

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

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

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

Importing a Pool By a Specific Device Path

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

```
# zpool import -d /dev/dsk/c2t3d0s0 dpool
# zpool status dpool
  pool: dpool
 state: ONLINE
 scan: resilvered 952K in 0h0m with 0 errors on Fri Jun 29 16:22:06 2012
config:

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

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

Recovering Destroyed ZFS Storage Pools

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

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

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

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

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

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

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

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

errors: No known data errors
```

If one of the devices in the destroyed pool is 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: 4107023015970708695
 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:
        dozer              DEGRADED
          raidz2-0         DEGRADED
            c8t0d0         ONLINE
            c8t1d0         ONLINE
            c8t2d0         ONLINE
            c8t3d0         UNAVAIL  cannot open
            c8t4d0         ONLINE
errors: No known data errors
# zpool import -Df dozer
# zpool status -x
  pool: dozer
 state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist for
        the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://www.sun.com/msg/ZFS-8000-2Q
   scan: none requested
config:

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

```
errors: No known data errors
# zpool online dozer c8t4d0
# zpool status -x
all pools are healthy
```

Upgrading ZFS Storage Pools

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

```
# zpool status
  pool: tank
 state: ONLINE
status: The pool is formatted using an older on-disk format.  The pool can
        still be used, but some features are unavailable.
action: Upgrade the pool using 'zpool upgrade'.  Once this is done, the
        pool will no longer be accessible on older software versions.
 scrub: none requested
config:
        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            c1t0d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0
errors: No known data errors
```

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

```
# zpool upgrade -v
This system is currently running ZFS pool version 22.

The following versions are supported:

VER  DESCRIPTION
---  --------------------------------------------------------
 1   Initial ZFS version
 2   Ditto blocks (replicated metadata)
 3   Hot spares and double parity RAID-Z
 4   zpool history
 5   Compression using the gzip algorithm
 6   bootfs pool property
 7   Separate intent log devices
 8   Delegated administration
 9   refquota and refreservation properties
 10  Cache devices
 11  Improved scrub performance
 12  Snapshot properties
 13  snapused property
 14  passthrough-x aclinherit
```

```
 15  user/group space accounting
 16  stmf property support
 17  Triple-parity RAID-Z
 18  Snapshot user holds
 19  Log device removal
 20  Compression using zle (zero-length encoding)
 21  Reserved
 22  Received properties

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

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

```
# zpool upgrade -a
```

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

If you want to use the ZFS Administration console on a system with a pool from a previous Solaris release, ensure that you upgrade your pools before using the console. To determine if your pools need to be upgraded, use the `zpool status` command.

◆ ◆ ◆ CHAPTER 4

Installing and Booting an Oracle Solaris ZFS Root File System

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

The following sections are provided in this chapter:

For a list of known issues in this release, see *Oracle Solaris 10 1/13 Release Notes*.

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

You can install and boot from a ZFS root file system in the following ways:

- **Oracle Solaris initial installation (interactive text mode installation method)**
  - Select and install ZFS as the root file system.
  - Install a ZFS flash archive.
- **Oracle Solaris Live Upgrade feature**
  - Migrate a UFS root file system to a ZFS root file system.

- Create a new boot environment in a new ZFS root pool.
- Create or update a boot environment in an existing ZFS root pool.
- Upgrade an alternate boot environment (BE) with a ZFS flash archive.

- **Oracle Solaris JumpStart feature**.
  - Create a profile to automatically install a system with a ZFS root file system.
  - Create a profile to automatically install a system with a ZFS flash archive.

After a SPARC based or an x86 based system is installed with or migrated to a ZFS root file system, the system boots automatically from the ZFS root file system. For more information about boot changes, see "Booting From a ZFS Root File System" on page 150.

ZFS Installation Features

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

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

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

    ```
    ok boot net - text
    ```
  - x86: Select the text-mode installation method.
- A custom JumpStart profile provides the following features:
  - You can set up a profile to create a ZFS storage pool and designate a bootable ZFS file system.
  - You can set up a profile to install a flash archive of a ZFS root pool.
- Using Live Upgrade, you can migrate a UFS root file system to a ZFS root file system.
- You can set up a mirrored ZFS root pool by selecting two disks during installation. Or, you can attach additional disks after installation to create a mirrored ZFS root pool.
- Swap and dump devices are automatically created on ZFS volumes in the ZFS root pool.

The following installation features are not provided in this release:

- The GUI installation feature for installing a ZFS root file system is not currently available. You must select the text mode installation method to install a ZFS root file system.
- You cannot use the standard upgrade program to upgrade your UFS root file system to a ZFS root file system.

Oracle Solaris Installation and Live Upgrade Requirements for ZFS Support

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

Oracle Solaris Release Requirements

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

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

General ZFS Root Pool Requirements

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

Disk Space Requirements for ZFS Root Pools

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

When a system is installed or upgraded with a ZFS root file system, the size of the swap area and the dump device depends upon the amount of physical memory. The minimum amount of available pool space for a bootable ZFS root file system depends on the amount of physical memory, the disk space available, and the number of boot environments (BEs) to be created.

Review the following disk space requirements for ZFS storage pools:

- 1536 MB is the minimum amount of memory required to install a ZFS root file system.
- 1536 MB of memory or greater is recommended for better overall ZFS performance.
- At least 16 GB of disk space is recommended. The disk space is consumed as follows:
  - **Swap area and dump device** – The default sizes of the swap and dump volumes that are created by the Oracle Solaris installation programs are as follows:
    - **Initial installation** – In the new ZFS boot environment, the default swap size is calculated as half the size of physical memory, generally in the 512-MB to 2- GB range. You can adjust the swap size during an initial installation.
    - The default dump size is calculated by the kernel based on `dumpadm` information and the size of physical memory. You can adjust the dump size during an initial installation.

- **Live Upgrade** – When a UFS root file system is migrated to a ZFS root file system, the default swap size for the ZFS BE is calculated as the size of the swap device of the UFS BE. The default swap size calculation adds the sizes of all the swap devices in the UFS BE and creates a ZFS volume of that size in the ZFS BE. If no swap devices are defined in the UFS BE, then the default swap size is set to 512 MB.
- In the ZFS BE, the default dump size is set to half the size of physical memory, between 512 MB and 2 GB.

You can adjust the sizes of your swap and dump volumes to sizes of your choosing as long as the new sizes support system operations. For more information, see "Adjusting the Sizes of Your ZFS Swap Device and Dump Device" on page 147.

- **Boot environment (BE)** – In addition to either new swap and dump space requirements or adjusted swap and dump device sizes, a ZFS BE that is migrated from a UFS BE requires approximately 6 GB. Each ZFS BE that is cloned from another ZFS BE doesn't require additional disk space, but consider that the BE size will increase when patches are applied. All ZFS BEs in the same root pool use the same swap and dump devices.
- **Oracle Solaris OS Components** – All subdirectories of the root file system that are part of the OS image, with the exception of `/var`, must be in the same dataset as the root file system. In addition, all OS components must reside in the root pool, with the exception of the swap and dump devices.

  For information about changing default swap and dump devices with Live Upgrade, see "Customizing ZFS Swap and Dump Volumes" on page 148.

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

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

ZFS Root Pool Configuration Requirements

Review the following ZFS root pool configuration requirements:

- The pool that is intended to be the root pool must have an SMI label. This requirement is usually met if the pool is created with disk slices.
- The pool must exist either on a disk slice or on disk slices that are mirrored. If you attempt to use an unsupported pool configuration during a Live Upgrade migration, you see a message similar to the following:

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

For a detailed description of supported ZFS root pool configurations, see "Creating a ZFS Root Pool" on page 49.

- x86: The disk must contain an Oracle Solaris `fdisk` partition. This `fdisk` partition is created automatically when the x86 based system is installed. For more information about Solaris `fdisk` partitions, see "Guidelines for Creating an fdisk Partition" in *System Administration Guide: Devices and File Systems*.
- Disks that are designated for booting in a ZFS root pool must be less than 2 TBs in size on both SPARC based and x86 based systems.
- Compression can be enabled on the root pool but only after the root pool is installed. No way exists to enable compression on a root pool during installation. The `gzip` compression algorithm is not supported on root pools.
- Do not rename the root pool after it is created by an initial installation or after Solaris Live Upgrade migration to a ZFS root file system. Renaming the root pool might cause an unbootable system.

  In addition, do not change the default mount point of the root pool components, if you want to use Live Upgrade.
- If you want to change your swap and dump devices and use Live Upgrade, see "Customizing ZFS Swap and Dump Volumes" on page 148.

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

In this Oracle Solaris release, you can perform an initial installation by using the following methods:

- Use the interactive text installer to initially install a ZFS storage pool that contains a bootable ZFS root file system. If you have an existing ZFS storage pool that you want to use for your ZFS root file system, then you must use Live Upgrade to migrate your existing UFS root file system to a ZFS root file system in an existing ZFS storage pool. For more information, see "Migrating to a ZFS Root File System or Updating a ZFS Root File System (Live Upgrade)" on page 122.
- Use the interactive text installer to initially install a ZFS storage pool that contains a bootable ZFS root file system from a ZFS flash archive.

Before you begin the initial installation to create a ZFS storage pool, see "Oracle Solaris Installation and Live Upgrade Requirements for ZFS Support" on page 105.

If you will be configuring zones after the initial installation of a ZFS root file system and you plan on patching or upgrading the system, see "Using Live Upgrade to Migrate or Upgrade a System With Zones (Solaris 10 10/08)" on page 131 or "Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (at Least Solaris 10 5/09)" on page 136.

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

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

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

EXAMPLE 4–1 Initial Installation of a Bootable ZFS Root File System

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

1. Insert the Oracle Solaris installation media or boot the system from an installation server. Then, select the interactive text installation method to create a bootable ZFS root file system.
   - SPARC: Use the following syntax for the Oracle Solaris Installation DVD:

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

     ```
     ok boot net - text
     ```
   - x86: Select the text-mode installation method.

   You can also create a ZFS flash archive to be installed by using the following methods:
   - JumpStart installation. For more information, see Example 4–2.
   - Initial installation. For more information, see Example 4–3.

   You can perform a standard upgrade to upgrade an existing bootable ZFS file system, but you cannot use this option to create a new bootable ZFS file system. Starting in the Solaris 10 10/08 release, you can migrate a UFS root file system to a ZFS root file system, as long as at least the Solaris 10 10/08 release is already installed. For more information about migrating to a ZFS root file system, see "Migrating to a ZFS Root File System or Updating a ZFS Root File System (Live Upgrade)" on page 122.
2. To create a ZFS root file system, select the ZFS option. For example:

   ```
   Choose Filesystem Type

     Select the filesystem to use for your Solaris installation


                [ ] UFS
                [X] ZFS
   ```

EXAMPLE 4–1 Initial Installation of a Bootable ZFS Root File System *(Continued)*

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

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

  Disk Device                                                 Available Space
  ============================================================================
  [X] ** c1t0d0                                             139989 MB  (F4 to edit
) [ ]    c1t1d0                                             139989 MB
  [ ]    c1t2d0                                             139989 MB
  [ ]    c1t3d0                                             139989 MB
  [ ]    c2t0d0                                             139989 MB
  [ ]    c2t1d0                                             139989 MB
  [ ]    c2t2d0                                             139989 MB
  [ ]    c2t3d0                                             139989 MB
                                  Maximum Root Size: 139989 MB
                                  Suggested Minimum:  11102 MB
```

You can select one or more disks to be used for your ZFS root pool. If you select two disks, a mirrored two-disk configuration is set up for your root pool. Either a two-disk or a three-disk mirrored pool is optimal. If you have eight disks and you select all of them, those eight disks are used for the root pool as one large mirror. This configuration is not optimal. Another option is to create a mirrored root pool after the initial installation is complete. A RAID-Z pool configuration for the root pool is not supported.

For more information about configuring ZFS storage pools, see "Replication Features of a ZFS Storage Pool" on page 45.

4. To select two disks to create a mirrored root pool, use the cursor control keys to select the second disk.

In the following example, both `c1t0d0` and `c1t1d0` are selected for the root pool disks. Both disks must have an SMI label and a slice 0. If the disks are not labeled with an SMI label or they don't contain slices, then you must exit the installation program, use the `format` utility to relabel and repartition the disks, and then restart the installation program.

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

  Disk Device                                                 Available Space
  ============================================================================
  [X] ** c1t0d0                                             139989 MB  (F4 to edit
```

EXAMPLE 4–1 Initial Installation of a Bootable ZFS Root File System *(Continued)*

```
) [X]    c1t1d0                                          139989 MB
  [ ]    c1t2d0                                          139989 MB
  [ ]    c1t3d0                                          139989 MB
  [ ]    c2t0d0                                          139989 MB
  [ ]    c2t1d0                                          139989 MB
  [ ]    c2t2d0                                          139989 MB
  [ ]    c2t3d0                                          139989 MB

                                    Maximum Root Size: 139989 MB
                                    Suggested Minimum:  11102 MB
```

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

If you do not create a mirrored root pool during installation, you can easily create one after the installation. For information, see "How to Create a Mirrored ZFS Root Pool (Postinstallation)" on page 113.

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

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


              ZFS Pool Name: rpool
      ZFS Root Dataset Name: s10nameBE
      ZFS Pool Size (in MB): 139990
  Size of Swap Area (in MB): 4096
  Size of Dump Area (in MB): 1024
        (Pool size must be between 7006 MB and 139990 MB)

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

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

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

   ```
              ZFS Pool Name: rpool
      ZFS Root Dataset Name: zfsBE
      ZFS Pool Size (in MB): 139990
  Size of Swap Area (in MB): 4096
  Size of Dump Area (in MB): 1024
        (Pool size must be between 7006 MB and 139990 MB)
   ```

EXAMPLE 4–1 Initial Installation of a Bootable ZFS Root File System *(Continued)*

6. At this final installation, screen, you can optionally change the installation profile. For example:

```
Profile

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

  ============================================================================

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

                          Regions: North America
                     System Locale: C ( C )

                          Software: Solaris 10, Entire Distribution
                         Pool Name: rpool
             Boot Environment Name: zfsBE
                         Pool Size: 139990 MB
                   Devices in Pool: c1t0d0
                                    c1t1d0
```

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

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

        NAME          STATE     READ WRITE CKSUM
        rpool         ONLINE       0     0     0
          mirror-0    ONLINE       0     0     0
            c1t0d0s0  ONLINE       0     0     0
            c1t1d0s0  ONLINE       0     0     0

errors: No known data errors
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
rpool              10.1G   124G   106K  /rpool
rpool/ROOT         5.01G   124G    31K  legacy
rpool/ROOT/zfsBE   5.01G   124G  5.01G  /
rpool/dump         1.00G   124G  1.00G  -
rpool/export         63K   124G    32K  /export
rpool/export/home    31K   124G    31K  /export/home
rpool/swap         4.13G   124G  4.00G  -
```

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

8. To create another ZFS boot environment (BE) in the same storage pool, use the `lucreate` command.

EXAMPLE 4–1 Initial Installation of a Bootable ZFS Root File System *(Continued)*

In the following example, a new BE named `zfs2BE` is created. The current BE is named `zfsBE`, as shown in the `zfs list` output. However, the current BE is not acknowledged in the `lustatus` output until the new BE is created.

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

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

```
# lucreate -n zfs2BE
INFORMATION: The current boot environment is not named - assigning name <zfsBE>.
Current boot environment is named <zfsBE>.
Creating initial configuration for primary boot environment <zfsBE>.
The device </dev/dsk/c1t0d0s0> is not a root device for any boot environment; cannot get BE ID.
PBE configuration successful: PBE name <zfsBE> PBE Boot Device </dev/dsk/c1t0d0s0>.
Comparing source boot environment <zfsBE> file systems with the file
system(s) you specified for the new boot environment. Determining which
file systems should be in the new boot environment.
Updating boot environment description database on all BEs.
Updating system configuration files.
Creating configuration for boot environment <zfs2BE>.
Source boot environment is <zfsBE>.
Creating boot environment <zfs2BE>.
Cloning file systems from boot environment <zfsBE> to create boot environment <zfs2BE>.
Creating snapshot for <rpool/ROOT/zfsBE> on <rpool/ROOT/zfsBE@zfs2BE>.
Creating clone for <rpool/ROOT/zfsBE@zfs2BE> on <rpool/ROOT/zfs2BE>.
Setting canmount=noauto for </> in zone <global> on <rpool/ROOT/zfs2BE>.
Population of boot environment <zfs2BE> successful.
Creation of boot environment <zfs2BE> successful.
```

Creating a ZFS BE within the same pool uses ZFS clone and snapshot features to instantly create the BE. For more details about using Live Upgrade for a ZFS root migration, see "Migrating to a ZFS Root File System or Updating a ZFS Root File System (Live Upgrade)" on page 122.

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

```
# lustatus
Boot Environment           Is       Active Active    Can    Copy
Name                       Complete Now    On Reboot Delete Status
-------------------------- -------- ------ --------- ------ ----------
zfsBE                      yes      yes    yes       no     -
zfs2BE                     yes      no     no        yes    -
# zfs list
NAME                      USED  AVAIL  REFER  MOUNTPOINT
rpool                    10.1G   124G   106K  /rpool
rpool/ROOT               5.00G   124G    31K  legacy
rpool/ROOT/zfs2BE         218K   124G  5.00G  /
rpool/ROOT/zfsBE         5.00G   124G  5.00G  /
rpool/ROOT/zfsBE@zfs2BE   104K      -  5.00G  -
rpool/dump               1.00G   124G  1.00G  -
rpool/export               63K   124G    32K  /export
rpool/export/home          31K   124G    31K  /export/home
rpool/swap               4.13G   124G  4.00G  -
```

EXAMPLE 4–1 Initial Installation of a Bootable ZFS Root File System *(Continued)*

10. To boot from an alternate BE, use the `luactivate` command.
    - SPARC - Use the `boot -L` command to identify the available BEs when the boot device contains a ZFS storage pool.

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

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

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

    - x86 – Identify the BE to be booted from the GRUB menu.

For more information about booting a ZFS file system, see "Booting From a ZFS Root File System" on page 150.

How to Create a Mirrored ZFS Root Pool (Postinstallation)

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

For information about replacing a disk in a root pool, see "How to Replace a Disk in the ZFS Root Pool" on page 157.

1 Display the current root pool status.

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

        NAME        STATE     READ WRITE CKSUM
        rpool       ONLINE       0     0     0
          c1t0d0s0  ONLINE       0     0     0

errors: No known data errors
```

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

```
# zpool attach rpool c1t0d0s0 c1t1d0s0
Make sure to wait until resilver is done before rebooting.
```

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

```
# zpool status rpool
  pool: rpool
 state: ONLINE
status: One or more devices is currently being resilvered.  The pool will
        continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
 scrub: resilver in progress for 0h1m, 24.26% done, 0h3m to go
config:

        NAME          STATE     READ WRITE CKSUM
        rpool         ONLINE       0     0     0
          mirror-0    ONLINE       0     0     0
            c1t0d0s0  ONLINE       0     0     0
            c1t1d0s0  ONLINE       0     0     0  3.18G resilvered

errors: No known data errors
```

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

```
resilvered 10.0G in 0h10m with 0 errors on Thu Nov 15 12:48:33 2012
```

4 Verify that you can boot successfully from the second disk.

5 If necessary, set up the system to boot automatically from the new disk.

- SPARC - Use the `eeprom` command or the `setenv` command from the SPARC boot PROM to reset the default boot device.
- x86 - reconfigure the system BIOS.

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

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

- Create a flash archive that can be used to install and boot a system with a ZFS root file system.

- Perform a JumpStart installation or initial installation of a clone system by using a ZFS flash archive. Creating a ZFS flash archive clones an entire root pool, not individual boot environments. Individual datasets within the pool can be excluded by using the `-D` option to the `flarcreate` and `flar` commands.

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

- Starting in the Oracle Solaris 10 8/11 release, you can use the interactive installation's flash archive option to install a system with a ZFS root file system. In addition, you use a flash archive to update an alternate ZFS BE by using the `luupgrade` command.
  - A system running the Solaris 10 9/10 release must add patch 124630-51 (SPARC) or patch 124631-51 (x86) to install a flash archive to an ABE.
  - The master system, where you are creating the flash archive and the clone system, where you are installing the flash archive must be at the same kernel patch level. For example, if you create a ZFS flash archive on a system running the Solaris 10 8/11 release, make sure that the clone system is also running the same Solaris 10 8/11 kernel patch level. Otherwise, the flash archive installation might fail with errors from the `zfs receive` command.
  - A master system running the Solaris 10 9/10 release with descendent root file systems, such as a separate `/var` file system, should be upgraded to the Solaris 10 8/11 release before the flash archive is created and applied to an ABE. Otherwise, the flash archive installation will fail.
- You can only install a flash archive on a system that has the same architecture as the system on which you created the ZFS flash archive. For example, an archive that is created on a `sun4v` system cannot be installed on a `sun4u` system.
- Only a full initial installation of a ZFS flash archive is supported. You cannot install a differential flash archive of a ZFS root file system or install a hybrid UFS/ZFS archive.
- Starting in the Solaris 10 8/11 release, you can use a UFS flash archive to install a ZFS root file system. For example:
  - If you use the `pool` keyword in JumpStart profile, the UFS flash archive installs into a ZFS root pool.

    ```
    pool rpool auto auto auto mirror c0t0d0s0 c0t1d0s0
    ```
  - During interactive installation of a UFS flash archive, select ZFS as the file system type.
- Although the entire root pool, except for any explicitly excluded datasets, is archived and installed, only the ZFS BE that is booted when the archive is created is usable after the flash archive is installed. However, pools that are archived with the `-R` *rootdir* option of the `flarcreate` or `flar` command can be used to archive a root pool other than the root pool that is currently booted.
- The `flarcreate` and `flar` command options that are used to include and exclude individual files are not supported in a ZFS flash archive. You can only exclude entire datasets from a ZFS flash archive.

- The `flar info` command is not supported for a ZFS flash archive. For example:

```
# flar info -l zfs10upflar
ERROR: archive content listing not supported for zfs archives.
```

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

- Create the ZFS flash archive with the `flarcreate` command on the master system. All datasets in the root pool, except for the swap and dump volumes, are included in the ZFS flash archive.
- Create a JumpStart profile to include the flash archive information on the installation server.
- Install the ZFS flash archive on the target system.

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

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

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

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

EXAMPLE 4–2 Installing a System With a ZFS Flash Archive (JumpStart Installation)

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

```
# flarcreate -n zfsBE zfs10upflar
Full Flash
Checking integrity...
Integrity OK.
Running precreation scripts...
Precreation scripts done.
Determining the size of the archive...
The archive will be approximately 6.77GB.
Creating the archive...
Archive creation complete.
Running postcreation scripts...
Postcreation scripts done.

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

EXAMPLE 4–2 Installing a System With a ZFS Flash Archive (JumpStart Installation) *(Continued)*

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

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

EXAMPLE 4–3 Initial Installation of a Bootable ZFS Root File System (Flash Archive Installation)

You can install a ZFS root file system by selecting the Flash installation option. This option assumes that a ZFS flash archive has already been created and is available.

1. From the Solaris Interactive Installation screen, select the F4_Flash option.
2. From the Reboot After Installation screen, select the Auto Reboot or Manual Reboot option.
3. From the Choose Filesystem Type screen, select ZFS.
4. From the Flash Archive Retrieval Method screen, select the retrieval method, such as HTTP, FTP, NFS, Local File, Local Tape, or Local Device.

   For example, select NFS if the ZFS flash archive is shared from an NFS server.
5. From the Flash Archive Addition screen, specify the location of the ZFS flash archive.

   For example, if the location is an NFS server, identify the server by its IP address and then specify the path to the ZFS flash archive.

   ```
   NFS Location: 12.34.567.890:/export/zfs10upflar
   ```
6. From the Flash Archive Selection screen, confirm the retrieval method and the ZFS BE name.

   ```
   Flash Archive Selection

     You selected the following Flash archives to use to install this system.  If
     you want to add another archive to install select "New".

           Retrieval Method      Name
           ========================================================================
           NFS                   zfsBE
   ```
7. Review the next set of screens, similar to an initial installation, and select the options that match your configuration:
   - Select Disks
   - Preserve Data?
   - Configure ZFS Settings

     Review the summary information and then select the Continue option.

     For example:

EXAMPLE 4–3 Initial Installation of a Bootable ZFS Root File System (Flash Archive Installation) *(Continued)*

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


             ZFS Pool Name: rpool
        ZFS Root Dataset Name: s10zfsBE
        ZFS Pool Size (in MB): 69995
  Size of Swap Area (in MB): 2048
  Size of Dump Area (in MB): 1024
        (Pool size must be between 7591 MB and 69995 MB)
```

If the flash archive is a ZFS send stream, the combined or separate `/var` file system options are not presented. In this case, whether `/var` is combined or not depends on how it is configured on the master system.

- Press Continue at the Mount Remote File Systems? screen.
- Review the Profile screen, and press F4 to make any changes. Otherwise, press Begin_Installation (F2).

  For example:

```
Profile

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

  ============================================================================

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

                            Software: 1 Flash Archive
                                        NFS: zfsBE
                           Pool Name: rpool
                 Boot Environment Name: s10zfsBE
                           Pool Size: 69995 MB
                       Devices in Pool: c1t0d0
```

Installing a ZFS Root File System (JumpStart Installation)

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

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

For general information about using JumpStart features, see *Oracle Solaris 10 1/13 Installation Guide: JumpStart Installations*.

If you will be configuring zones after the JumpStart installation of a ZFS root file system and you plan on patching or upgrading the system, see "Using Live Upgrade to Migrate or Upgrade a System With Zones (Solaris 10 10/08)" on page 131 or "Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (at Least Solaris 10 5/09)" on page 136.

JumpStart Keywords for ZFS

The following keywords are permitted in a ZFS specific JumpStart profile:

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

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

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

`bootenv` Identifies the boot environment characteristics.

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

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

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

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

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

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

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

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

`pool` *poolname poolsize swapsize dumpsize vdevlist*

poolname Identifies the name of the pool to be created. The pool is created with the specified pool *poolsize* and with the physical devices specified with one or more devices *vdevlist*). The *poolname* value should not identify the name of an existing pool because the existing pool will be overwritten.

poolsize Specifies the size of the pool to be created. The value can be `auto` or `existing`. The `auto` value allocates the largest possible pool size, given the constraints, such as size of the disks, and so on. The size is assumed to be in MB, unless specified by `g` (GB).

swapsize Specifies the size of the swap volume to be created. The `auto` value means that the default swap size is used. You can specify a size with a *size* value. The size is in MB, unless specified by `g` (GB).

dumpsize Specifies the size of the dump volume to be created. The `auto` value means that the default dump size is used. You can specify a size with a *size* value. The size is assumed to be in MB, unless specified by `g` (GB).

vdevlist Specifies one or more devices that are used to create the pool. The format of *vdevlist* is the same as the format of the `zpool create` command. At this time, only mirrored configurations are supported when multiple devices are specified. Devices in *vdevlist* must be slices for the root pool. The `any` value means that the installation software selects a suitable device.

You can mirror as many disks as you like, but the size of the pool that is created is determined by the smallest of the specified disks. For more information about creating mirrored storage pools, see "Mirrored Storage Pool Configuration" on page 45.

JumpStart Profile Examples for ZFS

This section provides examples of ZFS specific JumpStart profiles.

The following profile performs an initial installation specified with `install_type initial_install` in a new pool, identified with `pool newpool`, whose size is automatically set by the `auto` keyword to the size of the specified disks. The swap area and dump device are automatically sized with the `auto` keyword in a mirrored configuration of disks (with the

`mirror` keyword and disks specified as `c0t0d0s0` and `c0t1d0s0`). Boot environment characteristics are set with the `bootenv` keyword to install a new BE with the keyword `installbe`, and a BE named `s10-xx` is created.

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

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

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

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

JumpStart Issues for ZFS

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

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

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

- You must create your pool with disk slices rather than with whole disks as described in "Oracle Solaris Installation and Live Upgrade Requirements for ZFS Support" on page 105. For example, the bold syntax in the following example is not acceptable:

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

  The bold syntax in the following example is acceptable:

  ```
  install_type initial_install
  cluster SUNWCall
  ```

```
pool rpool all auto auto mirror c0t0d0s0 c0t1d0s0
bootenv installbe bename newBE
```

Migrating to a ZFS Root File System or Updating a ZFS Root File System (Live Upgrade)

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

The following features are available:

- **UFS BE to ZFS BE migration**
  - When you migrate your UFS root file system to a ZFS root file system, you must designate an existing ZFS storage pool with the `-p` option.
  - If the UFS root file system has components on different slices, they are migrated to the ZFS root pool.
  - In the Oracle Solaris 10 8/11 release, you can specify a separate `/var` file system when you migrate your UFS root file system to a ZFS root file system
  - The basic process for migrating a UFS root file system to a ZFS root file system follows:
    1. Install the required Live Upgrade patches, if needed.
    2. Install a current Oracle Solaris 10 release (Solaris 10 10/08 to Oracle Solaris 10 8/11), or use the standard upgrade program to upgrade from a previous Oracle Solaris 10 release on any supported SPARC based or x86 based system.
    3. When you are running at least the Solaris 10 10/08 release, create a ZFS storage pool for your ZFS root file system.
    4. Use Live Upgrade to migrate your UFS root file system to a ZFS root file system.
    5. Activate your ZFS BE with the `luactivate` command.
- **Patch or upgrade a ZFS BE**
  - You can use the `luupgrade` command to patch or upgrade an existing ZFS BE. You can also use `luupgrade` to upgrade an alternate ZFS BE with a ZFS flash archive. For information, see Example 4–8.
  - Live Upgrade can use the ZFS snapshot and clone features when you create a new ZFS BE in the same pool. So, BE creation is much faster than in previous releases.
- **Zone migration support**– You can migrate a system with zones but the supported configurations are limited in the Solaris 10 10/08 release. More zone configurations are supported starting in the Solaris 10 5/09 release. For more information, see the following sections:

- "Using Live Upgrade to Migrate or Upgrade a System With Zones (Solaris 10 10/08)" on page 131
- "Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (at Least Solaris 10 5/09)" on page 136

If you are migrating a system without zones, see "Using Live Upgrade to Migrate or Update a ZFS Root File System (Without Zones)" on page 124.

For detailed information about Oracle Solaris installation and Live Upgrade features, see the *Oracle Solaris 10 1/13 Installation Guide: Live Upgrade and Upgrade Planning*.

For information about ZFS and Live Upgrade requirements, see "Oracle Solaris Installation and Live Upgrade Requirements for ZFS Support" on page 105.

ZFS Migration Issues With Live Upgrade

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

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

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

  Before you create the new pool, ensure that the disks to be used in the pool have an SMI (VTOC) label instead of an EFI label. If the disk is relabeled with an SMI label, ensure that the labeling process did not change the partitioning scheme. In most cases, all of the disk's capacity should be in the slices that are intended for the root pool.
- You cannot use Oracle Solaris Live Upgrade to create a UFS BE from a ZFS BE. If you migrate your UFS BE to a ZFS BE and you retain your UFS BE, you can boot from either your UFS BE or your ZFS BE.
- Do not rename your ZFS BEs with the `zfs rename` command because Live Upgrade cannot detect the name change. Subsequent commands, such as `ludelete`, will fail. In fact, do not rename your ZFS pools or file systems if you have existing BEs that you want to continue to use.
- When creating an alternate BE that is a clone of the primary BE, you cannot use the `-f`, `-x`, `-y`, `-Y`, and `-z` options to include or exclude files from the primary BE. You can still use the inclusion and exclusion option set in the following cases:

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

- Although you can use Live Upgrade to upgrade your UFS root file system to a ZFS root file system, you cannot use Live Upgrade to upgrade non-root or shared file systems.
- You cannot use the `lu` command to create or migrate a ZFS root file system.
- If you want to have your system's swap and dump device in a non-root pool, see "Customizing ZFS Swap and Dump Volumes" on page 148.

Using Live Upgrade to Migrate or Update a ZFS Root File System (Without Zones)

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

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

- "Using Live Upgrade to Migrate or Upgrade a System With Zones (Solaris 10 10/08)" on page 131
- "Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (at Least Solaris 10 5/09)" on page 136

EXAMPLE 4–4 Using Live Upgrade to Migrate a UFS Root File System to a ZFS Root File System

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

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

```
# zpool create rpool mirror c1t2d0s0 c2t1d0s0
# lucreate -c ufsBE -n zfsBE -p rpool
Analyzing system configuration.
No name for current boot environment.
Current boot environment is named <ufsBE>.
Creating initial configuration for primary boot environment <ufsBE>.
The device </dev/dsk/c1t0d0s0> is not a root device for any boot environment; cannot get BE ID.
PBE configuration successful: PBE name <ufsBE> PBE Boot Device </dev/dsk/c1t0d0s0>.
Comparing source boot environment <ufsBE> file systems with the file
system(s) you specified for the new boot environment. Determining which
```

EXAMPLE 4–4 Using Live Upgrade to Migrate a UFS Root File System to a ZFS Root File System *(Continued)*

```
file systems should be in the new boot environment.
Updating boot environment description database on all BEs.
Updating system configuration files.
The device </dev/dsk/c1t2d0s0> is not a root device for any boot environment; cannot get BE ID.
Creating configuration for boot environment <zfsBE>.
Source boot environment is <ufsBE>.
Creating boot environment <zfsBE>.
Creating file systems on boot environment <zfsBE>.
Creating <zfs> file system for </> in zone <global> on <rpool/ROOT/zfsBE>.
Populating file systems on boot environment <zfsBE>.
Checking selection integrity.
Integrity check OK.
Populating contents of mount point </>.
Copying.
Creating shared file system mount points.
Creating compare databases for boot environment <zfsBE>.
Creating compare database for file system </rpool/ROOT>.
Creating compare database for file system </>.
Updating compare databases on boot environment <zfsBE>.
Making boot environment <zfsBE> bootable.
Creating boot_archive for /.alt.tmp.b-qD.mnt
updating /.alt.tmp.b-qD.mnt/platform/sun4u/boot_archive
Population of boot environment <zfsBE> successful.
Creation of boot environment <zfsBE> successful.
```

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

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

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

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

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

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

**********************************************************************
```

EXAMPLE 4–4 Using Live Upgrade to Migrate a UFS Root File System to a ZFS Root File System *(Continued)*

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

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

Next, reboot the system to the ZFS BE.

```
# init 6
```

Confirm that the ZFS BE is active.

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

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

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

EXAMPLE 4–5 Using Live Upgrade to Create a ZFS BE From a UFS BE (With a Separate `/var`)

In the Oracle Solaris 10 8/11 release, you can use the `lucreate -D` option to identify that you want a separate `/var` file system created when you migrate a UFS root file system to a ZFS root file system. In the following example, the existing UFS BE is migrated to a ZFS BE with a separate `/var` file system.

```
# lucreate -n zfsBE -p rpool -D /var
Determining types of file systems supported
Validating file system requests
Preparing logical storage devices
Preparing physical storage devices
Configuring physical storage devices
Configuring logical storage devices
Analyzing system configuration.
No name for current boot environment.
INFORMATION: The current boot environment is not named - assigning name <c0t0d0s0>.
Current boot environment is named <c0t0d0s0>.
Creating initial configuration for primary boot environment <c0t0d0s0>.
```

EXAMPLE 4–5 Using Live Upgrade to Create a ZFS BE From a UFS BE (With a Separate /var) *(Continued)*

```
INFORMATION: No BEs are configured on this system.
The device </dev/dsk/c0t0d0s0> is not a root device for any boot environment; cannot get BE ID.
PBE configuration successful: PBE name <c0t0d0s0> PBE Boot Device </dev/dsk/c0t0d0s0>.
Updating boot environment description database on all BEs.
Updating system configuration files.
The device </dev/dsk/c0t1d0s0> is not a root device for any boot environment; cannot get BE ID.
Creating configuration for boot environment <zfsBE>.
Source boot environment is <c0t0d0s0>.
Creating file systems on boot environment <zfsBE>.
Creating <zfs> file system for </> in zone <global> on <rpool/ROOT/zfsBE>.
Creating <zfs> file system for </var> in zone <global> on <rpool/ROOT/zfsBE/var>.
Populating file systems on boot environment <zfsBE>.
Analyzing zones.
Mounting ABE <zfsBE>.
Generating file list.
Copying data from PBE <c0t0d0s0> to ABE <zfsBE>
100% of filenames transferred
Finalizing ABE.
Fixing zonepaths in ABE.
Unmounting ABE <zfsBE>.
Fixing properties on ZFS datasets in ABE.
Reverting state of zones in PBE <c0t0d0s0>.
Making boot environment <zfsBE> bootable.
Creating boot_archive for /.alt.tmp.b-iaf.mnt
updating /.alt.tmp.b-iaf.mnt/platform/sun4u/boot_archive
Population of boot environment <zfsBE> successful.
Creation of boot environment <zfsBE> successful.
# luactivate zfsBE
A Live Upgrade Sync operation will be performed on startup of boot environment <zfsBE>.
.
.
.
Modifying boot archive service
Activation of boot environment <zfsBE> successful.
# init 6
```

Review the newly created ZFS file systems. For example:

```
# zfs list
NAME                   USED  AVAIL  REFER  MOUNTPOINT
rpool                 6.29G  26.9G  32.5K  /rpool
rpool/ROOT            4.76G  26.9G    31K  legacy
rpool/ROOT/zfsBE      4.76G  26.9G  4.67G  /
rpool/ROOT/zfsBE/var  89.5M  26.9G  89.5M  /var
rpool/dump             512M  26.9G   512M  -
rpool/swap            1.03G  28.0G    16K  -
```

EXAMPLE 4–6 Using Live Upgrade to Create a ZFS BE From a ZFS BE

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

EXAMPLE 4–6 Using Live Upgrade to Create a ZFS BE From a ZFS BE *(Continued)*

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

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

For more information, see Example 4–12.

```
# lucreate -n zfs2BE
Analyzing system configuration.
No name for current boot environment.
INFORMATION: The current boot environment is not named - assigning name <zfsBE>.
Current boot environment is named <zfsBE>.
Creating initial configuration for primary boot environment <zfsBE>.
The device </dev/dsk/c1t0d0s0> is not a root device for any boot environment; cannot get BE ID.
PBE configuration successful: PBE name <zfsBE> PBE Boot Device </dev/dsk/c1t0d0s0>.
Comparing source boot environment <zfsBE> file systems with the file
system(s) you specified for the new boot environment. Determining which
file systems should be in the new boot environment.
Updating boot environment description database on all BEs.
Updating system configuration files.
Creating configuration for boot environment <zfs2BE>.
Source boot environment is <zfsBE>.
Creating boot environment <zfs2BE>.
Cloning file systems from boot environment <zfsBE> to create boot environment <zfs2BE>.
Creating snapshot for <rpool/ROOT/zfsBE> on <rpool/ROOT/zfsBE@zfs2BE>.
Creating clone for <rpool/ROOT/zfsBE@zfs2BE> on <rpool/ROOT/zfs2BE>.
Setting canmount=noauto for </> in zone <global> on <rpool/ROOT/zfs2BE>.
Population of boot environment <zfs2BE> successful.
Creation of boot environment <zfs2BE> successful.
```

EXAMPLE 4–7 Update Your ZFS BE (`luupgrade`)

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

The basic process follows:

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

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

EXAMPLE 4–7 Update Your ZFS BE (`luupgrade`) *(Continued)*

```
Adding packages to the BE <zfsBE>.

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

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

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

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

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

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

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

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

Or, you can create a new BE to update to a later Oracle Solaris release. For example:

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

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

EXAMPLE 4–8 Creating a ZFS BE With a ZFS Flash Archive (`luupgrade`)

In the Oracle Solaris 10 8/11 release, you can use the `luupgrade` command to create a ZFS BE from an existing ZFS flash archive. The basic process is as follows:

1. Create a flash archive of a master system with a ZFS BE.

   For example:

   ```
   master-system# flarcreate -n s10zfsBE /tank/data/s10zfsflar
   Full Flash
   Checking integrity...
   Integrity OK.
   Running precreation scripts...
   Precreation scripts done.
   ```

EXAMPLE 4–8 Creating a ZFS BE With a ZFS Flash Archive (`luupgrade`) *(Continued)*

```
Determining the size of the archive...
The archive will be approximately 4.67GB.
Creating the archive...
Archive creation complete.
Running postcreation scripts...
Postcreation scripts done.

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

2. Make the ZFS flash archive that was created on the master system available to the clone system.

   Possible flash archive locations are a local file system, HTTP, FTP, NFS, and so on.

3. Create an empty alternate ZFS BE on the clone system.

   Use the `-s -` option to specify that this is an empty BE to be populated with the ZFS flash archive contents.

   For example:

```
clone-system# lucreate -n zfsflashBE -s - -p rpool
Determining types of file systems supported
Validating file system requests
Preparing logical storage devices
Preparing physical storage devices
Configuring physical storage devices
Configuring logical storage devices
Analyzing system configuration.
No name for current boot environment.
INFORMATION: The current boot environment is not named - assigning name <s10zfsBE>.
Current boot environment is named <s10zfsBE>.
Creating initial configuration for primary boot environment <s10zfsBE>.
INFORMATION: No BEs are configured on this system.
The device </dev/dsk/c0t0d0s0> is not a root device for any boot environment; cannot get BE ID.
PBE configuration successful: PBE name <s10zfsBE> PBE Boot Device </dev/dsk/c0t0d0s0>.
Updating boot environment description database on all BEs.
Updating system configuration files.
The device </dev/dsk/c0t1d0s0> is not a root device for any boot environment; cannot get BE ID.
Creating <zfs> file system for </> in zone <global> on <rpool/ROOT/zfsflashBE>.
Creation of boot environment <zfsflashBE> successful.
```

4. Install the ZFS flash archive into the alternate BE.

   For example:

```
clone-system# luupgrade -f -s /net/server/export/s10/latest -n zfsflashBE -a /tank/data/zfs10up2flar
miniroot filesystem is <lofs>
Mounting miniroot at </net/server/s10up/latest/Solaris_10/Tools/Boot>
Validating the contents of the media </net/server/export/s10up/latest>.
The media is a standard Solaris media.
Validating the contents of the miniroot </net/server/export/s10up/latest/Solaris_10/Tools/Boot>.
Locating the flash install program.
Checking for existence of previously scheduled Live Upgrade requests.
Constructing flash profile to use.
Creating flash profile for BE <zfsflashBE>.
```

EXAMPLE 4–8 Creating a ZFS BE With a ZFS Flash Archive (`luupgrade`) *(Continued)*

```
Performing the operating system flash install of the BE <zfsflashBE>.
CAUTION: Interrupting this process may leave the boot environment unstable or unbootable.
Extracting Flash Archive: 100% completed (of 5020.86 megabytes)
The operating system flash install completed.
updating /.alt.tmp.b-rgb.mnt/platform/sun4u/boot_archive

The Live Flash Install of the boot environment <zfsflashBE> is complete.
```

5. Activate the alternate BE.

```
clone-system# luactivate zfsflashBE
A Live Upgrade Sync operation will be performed on startup of boot environment <zfsflashBE>.
.
.
.
Modifying boot archive service
Activation of boot environment <zfsflashBE> successful.
```

6. Reboot the system.

```
clone-system# init 6
```

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

You can use Live Upgrade to migrate a system with zones, but the supported configurations are limited in the Solaris 10 10/08 release. If you are installing or upgrading to at least the Solaris 10 5/09 release, more zone configurations are supported. For more information, see "Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (at Least Solaris 10 5/09)" on page 136.

This section describes how to install and configure a system with zones so that it can be upgraded and patched with Live Upgrade. If you are migrating to a ZFS root file system without zones, see "Using Live Upgrade to Migrate or Update a ZFS Root File System (Without Zones)" on page 124.

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

- "How to Migrate a UFS Root File System With Zone Roots on UFS to a ZFS Root File System (Solaris 10 10/08)" on page 132
- "How to Configure a ZFS Root File System With Zone Roots on ZFS (Solaris 10 10/08)" on page 133
- "How to Upgrade or Patch a ZFS Root File System With Zone Roots on ZFS (Solaris 10 10/08)" on page 135
- "Resolving ZFS Mount-Point Problems That Prevent Successful Booting (Solaris 10 10/08)" on page 154

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

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

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

In the steps that follow the example pool name is `rpool`, and the example names of the active boot environment (BEs) begin with `s10BE*`.

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

   For more information about upgrading a system that is running the Solaris 10 release, see *Oracle Solaris 10 1/13 Installation Guide: Live Upgrade and Upgrade Planning*.

2. **Create the root pool.**

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

   For information about the root pool requirements, see "Oracle Solaris Installation and Live Upgrade Requirements for ZFS Support" on page 105.

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

4. **Create the new ZFS boot environment.**

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

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

5. **Activate the new ZFS boot environment.**

   ```
   # luactivate s10BE2
   ```

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

6. **Reboot the system.**

   ```
   # init 6
   ```

7. **Migrate the zones to a ZFS BE.**

   a. **Boot the zones.**

b. Create another ZFS BE within the pool.

```
# lucreate s10BE3
```

c. Activate the new boot environment.

```
# luactivate s10BE3
```

d. Reboot the system.

```
# init 6
```

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

8 Resolve any potential mount-point problems.

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

a. Review the `zfs list` output.

Look for incorrect temporary mount points. For example:

```
# zfs list -r -o name,mountpoint rpool/ROOT/s10up

NAME                                MOUNTPOINT
rpool/ROOT/s10up                    /.alt.tmp.b-VP.mnt/
rpool/ROOT/s10up/zones              /.alt.tmp.b-VP.mnt//zones
rpool/ROOT/s10up/zones/zonerootA    /.alt.tmp.b-VP.mnt/zones/zonerootA
```

The mount point for the root ZFS BE (`rpool/ROOT/s10up`) should be `/`.

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

For example:

```
# zfs inherit -r mountpoint rpool/ROOT/s10up
# zfs set mountpoint=/ rpool/ROOT/s10up
```

c. Reboot the system.

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

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

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

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

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

   Depending on which installation method you choose, see either "Installing a ZFS Root File System (Oracle Solaris Initial Installation)" on page 107 or "Installing a ZFS Root File System (JumpStart Installation)" on page 118.

2. **Boot the system from the newly created root pool.**

3. **Create a dataset for grouping the zone roots.**

   For example:

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

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

4. **Mount the newly created zones dataset.**

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

   The dataset is mounted at `/zones`.

5. **Create and mount a dataset for each zone root.**

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

6. **Set the appropriate permissions on the zone root directory.**

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

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

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

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

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

8. **Install the zone.**

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

9. **Boot the zone.**

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

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

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

In the steps that follow, `newBE` is the example name of the BE that is upgraded or patched.

1 Create the BE to upgrade or patch.

```
# lucreate -n newBE
```

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

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

- Upgrade the system.

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

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

- Apply patches to the new BE.

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

3 Activate the new BE.

```
# luactivate newBE
```

4 Boot from the newly activated BE.

```
# init 6
```

5 Resolve any potential mount-point problems.

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

a. Review the `zfs list` output.

Look for incorrect temporary mount points. For example:

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

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

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

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

For example:

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

c. Reboot the system.

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

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

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

This section describes how to configure a system with zones so that it can be upgraded and patched with Live Upgrade starting in the Solaris 10 5/09 release. If you are migrating to a ZFS root file system without zones, see "Using Live Upgrade to Migrate or Update a ZFS Root File System (Without Zones)" on page 124.

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

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

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

- "Supported ZFS with Zone Root Configuration Information (at Least Solaris 10 5/09)" on page 137
- "How to Create a ZFS BE With a ZFS Root File System and a Zone Root (at Least Solaris 10 5/09)" on page 138
- "How to Upgrade or Patch a ZFS Root File System With Zone Roots (at Least Solaris 10 5/09)" on page 140

- "How to Migrate a UFS Root File System With a Zone Root to a ZFS Root File System (at Least Solaris 10 5/09)" on page 143

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

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

- **Migrate a UFS root file system to a ZFS root file system** – The following configurations of zone roots are supported:
  - In a directory in the UFS root file system
  - In a subdirectory of a mount point in the UFS root file system
  - A UFS root file system with a zone root in a UFS root file system directory or in a subdirectory of a UFS root file system mount point and a ZFS non-root pool with a zone root

  A UFS root file system that has a zone root as a mount point is not supported.
- **Migrate or upgrade a ZFS root file system** – The following configurations of zone roots are supported:
  - In a file system in a ZFS root or a non-root pool. For example, `/zonepool/zones` is acceptable. In some cases, if a file system for the zone root is not provided before the Live Upgrade operation is performed, a file system for the zone root (`zoneds`) is created by Live Upgrade.
  - In a descendent file system or subdirectory of a ZFS file system as long as different zone paths are not nested. For example, `/zonepool/zones/zone1` and `/zonepool/zones/zone1_dir` are acceptable.

    In the following example, `zonepool/zones` is a file system that contains the zone roots, and `rpool` contains the ZFS BE:

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

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

    ```
    # lucreate -n newBE
    ```

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

In the preceding example, if `/zonepool/zones` were a subdirectory and not a separate file system, then Live Upgrade would migrate it as a component of the root pool, `rpool`.

- **The following ZFS and zone configuration is not supported:**
  - Live upgrade cannot be used to create an alternate BE when the source BE has a non-global zone with a zone path set to the mount point of a top-level pool file system. For example, if `zonepool` pool has a file system mounted as `/zonepool`, you cannot have a non-global zone with a zone path set to `/zonepool`.
  - Do not add an file system entry for a non-global zone in the global zone's `/etc/vfstab` file. Instead, use `zonecfg`'s `add fs` feature to add a file system to a non-global zone.

- **Zones migration or upgrade information with zones for both UFS and ZFS** – Review the following considerations that might affect a migration or an upgrade of either a UFS and ZFS environment:
  - If you configured your zones as described in "Using Live Upgrade to Migrate or Upgrade a System With Zones (Solaris 10 10/08)" on page 131 in the Solaris 10 10/08 release and have upgraded to at least the Solaris 10 5/09, you can migrate to a ZFS root file system or use Live Upgrade to upgrade to at least the Solaris 10 5/09 release.
  - Do not create zone roots in nested directories, for example, `zones/zone1` and `zones/zone1/zone2`. Otherwise, mounting might fail at boot time.

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

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

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

1 Review the existing ZFS file systems.

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

2 Ensure that the zones are installed and booted.

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

3 Create the ZFS BE.

```
# lucreate -n zfs2BE
Analyzing system configuration.
No name for current boot environment.
INFORMATION: The current boot environment is not named - assigning name <zfsBE>.
Current boot environment is named <zfsBE>.
Creating initial configuration for primary boot environment <zfsBE>.
The device </dev/dsk/c1t0d0s0> is not a root device for any boot environment; cannot get BE ID.
PBE configuration successful: PBE name <zfsBE> PBE Boot Device </dev/dsk/c1t0d0s0>.
Comparing source boot environment <zfsBE> file systems with the file
system(s) you specified for the new boot environment. Determining which
file systems should be in the new boot environment.
Updating boot environment description database on all BEs.
Updating system configuration files.
Creating configuration for boot environment <zfs2BE>.
Source boot environment is <zfsBE>.
Creating boot environment <zfs2BE>.
Cloning file systems from boot environment <zfsBE> to create boot environment <zfs2BE>.
Creating snapshot for <rpool/ROOT/zfsBE> on <rpool/ROOT/zfsBE@zfs2BE>.
Creating clone for <rpool/ROOT/zfsBE@zfs2BE> on <rpool/ROOT/zfs2BE>.
Setting canmount=noauto for </> in zone <global> on <rpool/ROOT/zfs2BE>.
Population of boot environment <zfs2BE> successful.
Creation of boot environment <zfs2BE> successful.
```

4 Activate the ZFS BE.

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

5 Boot the ZFS BE.

```
# init 6
```

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

```
# zfs list
NAME                              USED  AVAIL  REFER  MOUNTPOINT
rpool                            7.38G  59.6G    98K  /rpool
rpool/ROOT                       4.72G  59.6G    21K  legacy
rpool/ROOT/zfs2BE                4.72G  59.6G  4.64G  /
rpool/ROOT/zfs2BE@zfs2BE          74.0M      -  4.64G  -
```

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

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

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

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

1 Review the existing ZFS file systems.

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

2 Ensure that the zones are installed and booted.

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

3 Create the ZFS BE to upgrade or patch.

```
# lucreate -n zfs2BE
Analyzing system configuration.
Comparing source boot environment <zfsBE> file systems with the file
system(s) you specified for the new boot environment. Determining which
file systems should be in the new boot environment.
Updating boot environment description database on all BEs.
Updating system configuration files.
```

```
Creating configuration for boot environment <zfs2BE>.
Source boot environment is <zfsBE>.
Creating boot environment <zfs2BE>.
Cloning file systems from boot environment <zfsBE> to create boot environment <zfs2BE>.
Creating snapshot for <rpool/ROOT/zfsBE> on <rpool/ROOT/zfsBE@zfs2BE>.
Creating clone for <rpool/ROOT/zfsBE@zfs2BE> on <rpool/ROOT/zfs2BE>.
Setting canmount=noauto for </> in zone <global> on <rpool/ROOT/zfs2BE>.
Creating snapshot for <rpool/zones> on <rpool/zones@zfs10092BE>.
Creating clone for <rpool/zones@zfs2BE> on <rpool/zones-zfs2BE>.
Population of boot environment <zfs2BE> successful.
Creation of boot environment <zfs2BE> successful.
```

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

- Upgrade the system.

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

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

  This process can take a very long time.

  For a complete example of the `luupgrade` process, see Example 4–9.

- Apply patches to the new BE.

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

5 Activate the new boot environment.

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

6 Boot from the newly activated boot environment.

```
# init 6
```

Example 4–9 Upgrading a ZFS Root File System With a Zone Root to an Oracle Solaris 10 9/10 ZFS Root File System

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

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

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

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

# lucreate -n zfsBE
.
.
.
# luupgrade -u -n zfsBE -s /net/install/export/s10up/latest
40410 blocks
miniroot filesystem is <lofs>
Mounting miniroot at </net/system/export/s10up/latest/Solaris_10/Tools/Boot>
Validating the contents of the media </net/system/export/s10up/latest>.
The media is a standard Solaris media.
The media contains an operating system upgrade image.
The media contains <Solaris> version <10>.
Constructing upgrade profile to use.
Locating the operating system upgrade program.
Checking for existence of previously scheduled Live Upgrade requests.
Creating upgrade profile for BE <zfsBE>.
Determining packages to install or upgrade for BE <zfsBE>.
Performing the operating system upgrade of the BE <zfsBE>.
CAUTION: Interrupting this process may leave the boot environment unstable
or unbootable.
Upgrading Solaris: 100% completed
Installation of the packages from this media is complete.
Updating package information on boot environment <zfsBE>.
Package information successfully updated on boot environment <zfsBE>.
Adding operating system patches to the BE <zfsBE>.
The operating system patch installation is complete.
INFORMATION: The file </var/sadm/system/logs/upgrade_log> on boot
environment <zfsBE> contains a log of the upgrade operation.
INFORMATION: The file </var/sadm/system/data/upgrade_cleanup> on boot
environment <zfsBE> contains a log of cleanup operations required.
```

```
INFORMATION: Review the files listed above. Remember that all of the files
are located on boot environment <zfsBE>. Before you activate boot
environment <zfsBE>, determine if any additional system maintenance is
required or if additional media of the software distribution must be
installed.
The Solaris upgrade of the boot environment <zfsBE> is complete.
Installing failsafe
Failsafe install is complete.
# luactivate zfs2BE
# init 6
# lustatus
Boot Environment           Is       Active Active    Can    Copy
Name                       Complete Now    On Reboot Delete Status
-------------------------- -------- ------ --------- ------ ----------
zfsBE                      yes      no     no        yes    -
zfs2BE                     yes      yes    yes       no     -
# zoneadm list -cv
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              native   shared
   - zfszone          installed  /zonepool/zones                native   shared
```

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

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

In the steps that follow, the example UFS BE name is `c1t1d0s0`, the UFS zone root is `zonepool/zfszone`, and the ZFS root BE is `zfsBE`.

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

For information about upgrading a system that is running the Solaris 10 release, see *Oracle Solaris 10 1/13 Installation Guide: Live Upgrade and Upgrade Planning*.

2 Create the root pool.

For information about the root pool requirements, see "Oracle Solaris Installation and Live Upgrade Requirements for ZFS Support" on page 105.

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

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

4 Create the new ZFS BE.

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

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

5 Activate the new ZFS BE.

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

6 Reboot the system.

```
# init 6
```

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

```
# zfs list
NAME                              USED  AVAIL  REFER  MOUNTPOINT
rpool                            6.17G  60.8G    98K  /rpool
rpool/ROOT                       4.67G  60.8G    21K  /rpool/ROOT
rpool/ROOT/zfsBE                 4.67G  60.8G  4.67G  /
rpool/dump                       1.00G  60.8G  1.00G  -
rpool/swap                        517M  61.3G    16K  -
zonepool                          634M  7.62G    24K  /zonepool
zonepool/zones                    270K  7.62G   633M  /zonepool/zones
zonepool/zones-c1t1d0s0           634M  7.62G   633M  /zonepool/zones-c1t1d0s0
zonepool/zones-c1t1d0s0@zfsBE     262K      -   633M  -
# zoneadm list -cv
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              native   shared
   - zfszone          installed  /zonepool/zones                native   shared
```

Example 4–10 Migrating a UFS Root File System With a Zone Root to a ZFS Root File System

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

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

# lucreate -c ufsBE -n zfsBE -p rpool
Analyzing system configuration.
```

```
No name for current boot environment.
Current boot environment is named <zfsBE>.
Creating initial configuration for primary boot environment <zfsBE>.
The device </dev/dsk/c1t0d0s0> is not a root device for any boot environment; cannot get BE ID.
PBE configuration successful: PBE name <ufsBE> PBE Boot Device </dev/dsk/c1t0d0s0>.
Comparing source boot environment <ufsBE> file systems with the file
system(s) you specified for the new boot environment. Determining which
file systems should be in the new boot environment.
Updating boot environment description database on all BEs.
Updating system configuration files.
The device </dev/dsk/c1t1d0s0> is not a root device for any boot environment; cannot get BE ID.
Creating configuration for boot environment <zfsBE>.
Source boot environment is <ufsBE>.
Creating boot environment <zfsBE>.
Creating file systems on boot environment <zfsBE>.
Creating <zfs> file system for </> in zone <global> on <rpool/ROOT/zfsBE>.
Populating file systems on boot environment <zfsBE>.
Checking selection integrity.
Integrity check OK.
Populating contents of mount point </>.
Copying.
Creating shared file system mount points.
Copying root of zone <ufszone> to </.alt.tmp.b-EYd.mnt/uzone/ufszone>.
Creating snapshot for <pool/zones/zfszone> on <pool/zones/zfszone@zfsBE>.
Creating clone for <pool/zones/zfszone@zfsBE> on <pool/zones/zfszone-zfsBE>.
Creating compare databases for boot environment <zfsBE>.
Creating compare database for file system </rpool/ROOT>.
Creating compare database for file system </>.
Updating compare databases on boot environment <zfsBE>.
Making boot environment <zfsBE> bootable.
Creating boot_archive for /.alt.tmp.b-DLd.mnt
updating /.alt.tmp.b-DLd.mnt/platform/sun4u/boot_archive
Population of boot environment <zfsBE> successful.
Creation of boot environment <zfsBE> successful.
# lustatus
Boot Environment           Is       Active Active    Can    Copy
Name                       Complete Now    On Reboot Delete Status
-------------------------- -------- ------ --------- ------ ----------
ufsBE                      yes      yes    yes       no     -
zfsBE                      yes      no     no        yes    -
# luactivate zfsBE
.
.
.
# init 6
.
.
.
# zfs list
NAME                                 USED  AVAIL  REFER  MOUNTPOINT
pool                                 628M  66.3G    19K  /pool
pool/zones                           628M  66.3G    20K  /pool/zones
pool/zones/zfszone                  75.5K  66.3G   627M  /pool/zones/zfszone
pool/zones/zfszone-ufsBE             628M  66.3G   627M  /pool/zones/zfszone-ufsBE
pool/zones/zfszone-ufsBE@zfsBE        98K      -   627M  -
rpool                               7.76G  59.2G    95K  /rpool
rpool/ROOT                          5.25G  59.2G    18K  /rpool/ROOT
rpool/ROOT/zfsBE                    5.25G  59.2G  5.25G  /
```

```
rpool/dump                              2.00G  59.2G  2.00G  -
rpool/swap                               517M  59.7G    16K  -
# zoneadm list -cv
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              native   shared
   - ufszone          installed  /uzone/ufszone                 native   shared
   - zfszone          installed  /pool/zones/zfszone            native   shared
```

Managing Your ZFS Swap and Dump Devices

During an initial Oracle Solaris OS installation or after performing a Live Upgrade migration from a UFS file system, a swap area is created on a ZFS volume in the ZFS root pool. For example:

```
# swap -l
swapfile                  dev  swaplo  blocks   free
/dev/zvol/dsk/rpool/swap 256,1      16 4194288 4194288
```

During an initial Oracle Solaris OS installation or a Live Upgrade from a UFS file system, a dump device is created on a ZFS volume in the ZFS root pool. In general, a dump device requires no administration because it is set up automatically at installation time. For example:

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

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

For information about the swap and dump volume sizes that are created by the installation programs, see "Oracle Solaris Installation and Live Upgrade Requirements for ZFS Support" on page 105.

Both the swap volume size and the dump volume size can be adjusted during and after installation. For more information, see "Adjusting the Sizes of Your ZFS Swap Device and Dump Device" on page 147.

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

- Separate ZFS volumes must be used for the swap area and the dump device.
- Currently, using a swap file on a ZFS file system is not supported.

- If you need to change your swap area or dump device after the system is installed or upgraded, use the `swap` and `dumpadm` commands as in previous releases. For more information, see Chapter 16, "Configuring Additional Swap Space (Tasks)," in *System Administration Guide: Devices and File Systems* and Chapter 17, "Managing System Crash Information (Tasks)," in *System Administration Guide: Advanced Administration*.

See the following sections for more information:

- "Adjusting the Sizes of Your ZFS Swap Device and Dump Device" on page 147
- "Troubleshooting ZFS Dump Device Issues" on page 149

Adjusting the Sizes of Your ZFS Swap Device and Dump Device

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

- You can adjust the size of your swap and dump volumes during an initial installation. For more information, see Example 4–1.
- You can create and size your swap and dump volumes before you perform a Live Upgrade operation. For example:

  1. Create your storage pool.

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

  2. Create your dump device.

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

  3. Enable the dump device.

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

  4. Create your swap volume:

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

  5. You must enable the swap area when a new swap device is added or changed.

     ```
     # swap -a /dev/zvol/dsk/rpool/swap
     ```

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

  Live Upgrade does not resize existing swap and dump volumes.

- You can reset the `volsize` property of the dump device after a system is installed. For example:

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

- If the current swap area is not in use, you can resize the size of the current swap volume, but you must reboot the system to see the increased swap space size.

  ```
  # zfs get volsize rpool/swap
  NAME        PROPERTY  VALUE    SOURCE
  rpool/swap  volsize   4G       local
  # zfs set volsize=8g rpool/swap
  # zfs get volsize rpool/swap
  NAME        PROPERTY  VALUE    SOURCE
  rpool/swap  volsize   8G       local
  # init 6
  ```

- You can attempt to resize the swap volume, but it might be best to remove the swap device. Then, re-create it. For example:

  ```
  # swap -d /dev/zvol/dsk/rpool/swap
  # zfs create -V 2g rpool/swap
  # swap -a /dev/zvol/dsk/rpool/swap
  ```

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

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

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

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

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

  Then, activate the new swap volume. For example:

  ```
  # swap -a /dev/zvol/dsk/rpool/swap2
  # swap -l
  swapfile                  dev  swaplo   blocks   free
  /dev/zvol/dsk/rpool/swap  256,1      16 1058800 1058800
  /dev/zvol/dsk/rpool/swap2 256,3      16 4194288 4194288
  ```

  Finally, add an entry for the second swap volume to the `/etc/vfstab` file.

Customizing ZFS Swap and Dump Volumes

Keep the following points in the mind if you remove the default swap and dump volumes and recreate them in a non-root (data) pool:

- If you want to create swap and dump devices in a non-root pool, do not create swap and dump volumes in a RAIDZ pool. If a pool includes swap and dump volumes, it must be a one-disk pool or a mirrored pool.
- If you use Live Upgrade to update your system, use the `-P` option to preserve the dump device from the PBE to ABE. For example:

  ```
  # lucreate -n newBE -P
  ```

Troubleshooting ZFS Dump Device Issues

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

- If a crash dump was not created automatically, you can use the `savecore` command to save the crash dump.
- A dump volume is created automatically when you initially install a ZFS root file system or migrate to a ZFS root file system. In most cases, you only need to adjust the size of the dump volume if the default dump volume size is too small. For example, on a large-memory system, the dump volume size is increased to 40 GB as follows:

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

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

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

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

- A system with 128 GB or greater memory night need a larger dump device than the dump device that is created by default. If the dump device is too small to capture an existing crash dump, a message similar to the following is displayed:

  ```
  # dumpadm -d /dev/zvol/dsk/rpool/dump
  dumpadm: dump device /dev/zvol/dsk/rpool/dump is too small to hold a system dump
  dump size 36255432704 bytes, device size 34359738368 bytes
  ```

  For information about sizing the swap and dump devices, see "Planning for Swap Space" in *System Administration Guide: Devices and File Systems*.
- You cannot currently add a dump device to a pool with multiple top-level devices. You will see a message similar to the following:

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

Add the dump device to the root pool, which cannot have multiple top-level devices.

Booting From a ZFS Root File System

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

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

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

By default, the dataset selected for booting is identified by the pool's `bootfs` property. This default selection can be overridden by specifying an alternate bootable dataset with the `boot -Z` command.

Booting From an Alternate Disk in a Mirrored ZFS Root Pool

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

- "Installing a ZFS Root File System (Oracle Solaris Initial Installation)" on page 107
- "How to Create a Mirrored ZFS Root Pool (Postinstallation)" on page 113

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` command. 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` and `installgrub` command syntax follows:
  - SPARC:

    ```
    sparc# installboot -F zfs /usr/platform/'uname -i'/lib/fs/zfs/bootblk
    ```
  - x86:

```
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 the following pool:

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

        NAME          STATE     READ WRITE CKSUM
        rpool         ONLINE       0     0     0
          mirror-0    ONLINE       0     0     0
            c1t0d0s0  ONLINE       0     0     0
            c1t1d0s0  ONLINE       0     0     0
```

- SPARC: Specify the alternate disk at the `ok` prompt. For example:

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

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

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

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

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

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

SPARC: Booting From a ZFS Root File System

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

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

Multiple bootable datasets can exist within a pool. By default, the bootable dataset entry in the */pool-name*`/boot/menu.lst` file is identified by the pool's `bootfs` property. However, a `menu.lst` entry can contain a `bootfs` command, which specifies an alternate dataset in the pool. In this way, the `menu.lst` file can contain entries for multiple root file systems within the pool.

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

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

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

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

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

EXAMPLE 4–11 SPARC: Booting From a Specific ZFS Boot Environment

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

For example, the following `lustatus` output shows that two ZFS BEs are available:

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

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

For example:

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

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

Program terminated
ok boot -Z rpool/ROOT/zfsBE
```

EXAMPLE 4–12 SPARC: Booting a ZFS File System in Failsafe Mode

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

```
ok boot -F failsafe
```

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

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

x86: Booting From a ZFS Root File System

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

```
title Solaris 10 1/13  X86
findroot (rootfs0,0,a)
kernel$ /platform/i86pc/multiboot -B $ZFS-BOOTFS
module /platform/i86pc/boot_archive
title Solaris failsafe
findroot (rootfs0,0,a)
kernel /boot/multiboot kernel/unix -s -B console=ttya
module /boot/x86.miniroot-safe
```

If the device identified by GRUB as the boot device contains a ZFS storage pool, the `menu.lst` file is used to create the GRUB menu.

On an x86 based system with multiple ZFS BEs, you can select a BE from the GRUB menu. If the root file system corresponding to this menu entry is a ZFS dataset, the following option is added:

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

EXAMPLE 4–13 x86: Booting a ZFS File System

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

```
title Solaris 10 1/13  X86
findroot (pool_rpool,0,a)
kernel /platform/i86pc/multiboot -B $ZFS-BOOTFS
module /platform/i86pc/boot_archive
title Solaris failsafe
findroot (pool_rpool,0,a)
kernel /boot/multiboot kernel/unix -s -B console=ttya
module /boot/x86.miniroot-safe
```

EXAMPLE 4–14 x86: Booting a ZFS File System in Failsafe Mode

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

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

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

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

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

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

▼ How to Resolve ZFS Mount-Point Problems

1. **Boot the system from a failsafe archive.**

2. **Import the pool.**

   For example:

   ```
   # zpool import rpool
   ```

3. **Look for incorrect temporary mount points.**

   For example:

   ```
   # zfs list -r -o name,mountpoint rpool/ROOT/s10up

       NAME                              MOUNTPOINT
       rpool/ROOT/s10up                  /.alt.tmp.b-VP.mnt/
       rpool/ROOT/s10up/zones            /.alt.tmp.b-VP.mnt//zones
       rpool/ROOT/s10up/zones/zonerootA  /.alt.tmp.b-VP.mnt/zones/zonerootA
   ```

   The mount point for the root BE (`rpool/ROOT/s10up`) should be `/`.

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

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

For example:

```
# zfs inherit -r mountpoint rpool/ROOT/s10up
# zfs set mountpoint=/ rpool/ROOT/s10up
```

5 **Reboot the system.**

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

Booting for Recovery Purposes in a ZFS Root Environment

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

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

- "How to Boot ZFS Failsafe Mode" on page 155
- "How to Boot ZFS From Alternate Media" on page 156

If you need to recover a root pool or root pool snapshot, see "Recovering the ZFS Root Pool or Root Pool Snapshots" on page 157.

▼ How to Boot ZFS Failsafe Mode

1 **Boot failsafe mode.**

- On a SPARC based system, type the following at the `ok` prompt:

  ```
  ok boot -F failsafe
  ```

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

2 **Mount the ZFS BE on `/a` when prompted.**

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

3 **Change to the `/a/etc` directory.**

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

4 **If necessary, set the TERM type.**

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

5 **Correct the `passwd` or `shadow` file.**

```
# vi shadow
```

6 **Reboot the system.**

```
# init 6
```

How to Boot ZFS From Alternate Media

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

1 **Boot from an installation DVD or from the network.**

- SPARC - Select one of the following boot methods:

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

  If you don't use the `-s` option, you must exit the installation program.
- x86 – Select the network boot option or boot from local DVD.

2 **Import the root pool, and specify an alternate mount point. For example:**

```
# zpool import -R /a rpool
```

3 **Mount the ZFS BE. For example:**

```
# zfs mount rpool/ROOT/zfsBE
```

4 **Access the ZFS BE contents from the `/a` directory.**

```
# cd /a
```

5 **Reboot the system.**

```
# init 6
```

Recovering the ZFS Root Pool or Root Pool Snapshots

The following sections describe how to perform the following tasks:

▼ How to Replace a Disk in the ZFS Root Pool

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

- The root pool is too small and you want to replace a smaller disk with a larger disk.
- A root pool disk is failing. In a non-redundant pool, if the disk is failing such that the system won't boot, you must boot from alternate media, such as a DVD or the network, before you replace the root pool disk.

In a mirrored root pool configuration, you can attempt a disk replacement without booting from alternate media. You can replace a failed disk by using the `zpool replace` command. Or, if you have an additional disk, you can use the `zpool attach` command. See the procedure in this section for an example of attaching an additional disk and detaching a root pool disk.

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

```
# zpool offline rpool c1t0d0s0
# cfgadm -c unconfigure c1::dsk/c1t0d0
<Physically remove failed disk c1t0d0>
<Physically insert replacement disk c1t0d0>
# cfgadm -c configure c1::dsk/c1t0d0
# zpool replace rpool c1t0d0s0
# zpool online rpool c1t0d0s0
# zpool status rpool
<Let disk resilver before installing the boot blocks>
SPARC# installboot -F zfs /usr/platform/‘uname -i‘/lib/fs/zfs/bootblk /dev/rdsk/c1t0d0s0
x86# installgrub /boot/grub/stage1 /boot/grub/stage2 /dev/rdsk/c1t9d0s0
```

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

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

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

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

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

1. **Physically connect the replacement (or new) disk.**

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

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

   - SPARC: "How to Set Up a Disk for a ZFS Root File System" in *System Administration Guide: Devices and File Systems*
   - x86: "How to Set Up a Disk for a ZFS Root File System" in *System Administration Guide: Devices and File Systems*

3. **Attach the new disk to the root pool.**

   For example:

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

4. **Confirm the root pool status.**

   For example:

   ```
   # zpool status rpool
     pool: rpool
    state: ONLINE
   status: One or more devices is currently being resilvered.  The pool will
           continue to function, possibly in a degraded state.
   action: Wait for the resilver to complete.
    scrub: resilver in progress, 25.47% done, 0h4m to go
   config:

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

   errors: No known data errors
   ```

5. **If you are replacing a smaller root pool disk with a larger disk, set the pool's `autoexpand` property to expand the pool's size.**

   Determine the existing `rpool` pool size:

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

   # zpool set autoexpand=on rpool
   ```

Review the expanded `rpool` pool size:

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

6 Verify that you can boot from the new disk.

For example, on a SPARC based system, you would use syntax similar to the following:

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

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

For example:

```
# zpool detach rpool c1t10d0s0
```

8 Set up the system to boot automatically from the new disk by resetting the default boot device.

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

▼ How to Create Root Pool Snapshots

You can create root pool snapshots for recovery purposes. The best way to create root pool snapshots is to perform a recursive snapshot of the root pool.

The following procedure creates a recursive root pool snapshot and stores the snapshot as a file and as snapshots in a pool on a remote system. If a root pool fails, the remote dataset can be mounted by using NFS, and the snapshot file can be received into the re-created 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 Oracle Solaris OS miniroot.

Validating remotely stored snapshots as files or snapshots is an important step in root pool recovery. With either method, snapshots should be recreated on a routine basis, such as when the pool configuration changes or when the Solaris OS is upgraded.

In the following procedure, the system is booted from the `zfsBE` boot environment.

1 Create a pool and file system on a remote system to store the snapshots.

For example:

```
remote# zfs create rpool/snaps
```

2 Share the file system with the local system.

For example:

```
remote# zfs set sharenfs='rw=local-system,root=local-system' rpool/snaps
# share
-@rpool/snaps   /rpool/snaps   sec=sys,rw=local-system,root=local-system   ""
```

3 Create a recursive snapshot of the root pool.

```
local# zfs snapshot -r rpool@snap1
local# zfs list -r rpool
# NAME                       USED  AVAIL  REFER  MOUNTPOINT
rpool                     15.1G   119G   106K  /rpool
rpool@snap1                   0      -   106K  -
rpool/ROOT                5.00G   119G    31K  legacy
rpool/ROOT@snap1              0      -    31K  -
rpool/ROOT/zfsBE          5.00G   119G  5.00G  /
rpool/ROOT/zfsBE@snap1        0      -  5.00G  -
rpool/dump                2.00G   120G  1.00G  -
rpool/dump@snap1              0      -  1.00G  -
rpool/export                63K   119G    32K  /export
rpool/export@snap1            0      -    32K  -
rpool/export/home           31K   119G    31K  /export/home
rpool/export/home@snap1       0      -    31K  -
rpool/swap                8.13G   123G  4.00G  -
rpool/swap@snap1              0      -  4.00G  -
```

4 Send the root pool snapshots to the remote system.

For example, to send the root pool snapshots to a remote pool as a file, use syntax similar to the following:

```
local# zfs send -Rv rpool@snap1 > /net/remote-system/rpool/snaps/rpool.snap1
sending from @ to rpool@snap1
sending from @ to rpool/ROOT@snap1
sending from @ to rpool/ROOT/s10zfsBE@snap1
sending from @ to rpool/dump@snap1
sending from @ to rpool/export@snap1
sending from @ to rpool/export/home@snap1
sending from @ to rpool/swap@snap1

local# zfs send -Rv rpool@snap1 > /net/remote-system/rpool/snaps/rpool.snap1
sending from @ to rpool@snap1
sending from @ to rpool/export@snap1
sending from @ to rpool/export/home@snap1
sending from @ to rpool/ROOT@snap1
sending from @ to rpool/ROOT/zfsBE@snap1
sending from @ to rpool/dump@snap1
sending from @ to rpool/swap@snap1
```

To send the root pool snapshots to a remote pool as snapshots, use syntax similar to the following:

```
local# zfs send -Rv rpool@snap1 | ssh remote-system zfs receive
-Fd -o canmount=off tank/snaps
sending from @ to rpool@snap1
sending from @ to rpool/export@snap1
```

```
sending from @ to rpool/export/home@snap1
sending from @ to rpool/ROOT@snap1
sending from @ to rpool/ROOT/zfsBE@snap1
sending from @ to rpool/dump@snap1
sending from @ to rpool/swap@snap1
```

How to Re-create a ZFS Root Pool and Restore Root Pool Snapshots

In this procedure, assume the following conditions:

- The ZFS root pool cannot be recovered.
- The ZFS root pool snapshots are stored on a remote system and are shared over NFS.

All the steps are performed on the local system.

1. **Boot from an installation DVD or the network.**

   - SPARC - Select one of the following boot methods:

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

     If you don't use `-s` option, you'll need to exit the installation program.
   - x86 – Select the option for booting from the DVD or the network. Then, exit the installation program.

2. **Mount the remote snapshot file system if you have sent the root pool snapshots as a file to the remote system.**

   For example:

   ```
   # mount -F nfs remote-system:/rpool/snaps /mnt
   ```

   If your network services are not configured, you might need to specify the *remote-system*'s IP address.

3. **If the root pool disk is replaced and does not contain a disk label that is usable by ZFS, you must relabel the disk.**

   For more information about relabeling the disk, see these references:

   - SPARC: "How to Set Up a Disk for a ZFS Root File System" in *System Administration Guide: Devices and File Systems*
   - x86: "How to Set Up a Disk for a ZFS Root File System" in *System Administration Guide: Devices and File Systems*

4 **Re-create the root pool.**

For example:

```
# zpool create -f -o failmode=continue -R /a -m legacy -o cachefile=
/etc/zfs/zpool.cache rpool c1t1d0s0
```

5 **Restore the root pool snapshots.**

This step might take some time. For example:

```
# cat /mnt/rpool.snap1 | zfs receive -Fdu rpool
```

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

To restore the actual root pool snapshots that are stored in a pool on a remote system, use syntax similar to the following:

```
# ssh remote-system zfs send -Rb tank/snaps/rpool@snap1 | zfs receive -F rpool
```

6 **Verify that the root pool datasets are restored.**

For example:

```
# zfs list
```

7 **Set the `bootfs` property on the root pool BE.**

For example:

```
# zpool set bootfs=rpool/ROOT/zfsBE rpool
```

8 **Install the boot blocks on the new disk.**

- SPARC:

  ```
  # installboot -F zfs /usr/platform/‘uname -i‘/lib/fs/zfs/bootblk /dev/rdsk/c1t1d0s0
  ```

- x86:

  ```
  # installgrub /boot/grub/stage1 /boot/grub/stage2 /dev/rdsk/c1t1d0s0
  ```

9 **Reboot the system.**

```
# init 6
```

▼ How to Roll Back Root Pool Snapshots From a Failsafe Boot

This procedure assumes that existing root pool snapshots are available. In the example, they are available on the local system.

```
# zfs snapshot -r rpool@snap1
# zfs list -r rpool
NAME                        USED  AVAIL  REFER  MOUNTPOINT
rpool                      7.84G  59.1G   109K  /rpool
rpool@snap1                  21K      -   106K  -
rpool/ROOT                 4.78G  59.1G    31K  legacy
rpool/ROOT@snap1              0      -    31K  -
rpool/ROOT/s10zfsBE        4.78G  59.1G  4.76G  /
rpool/ROOT/s10zfsBE@snap1  15.6M      -  4.75G  -
rpool/dump                 1.00G  59.1G  1.00G  -
rpool/dump@snap1             16K      -  1.00G  -
rpool/export                 99K  59.1G    32K  /export
rpool/export@snap1           18K      -    32K  -
rpool/export/home            49K  59.1G    31K  /export/home
rpool/export/home@snap1      18K      -    31K  -
rpool/swap                 2.06G  61.2G    16K  -
rpool/swap@snap1              0      -    16K  -
```

1 Shut down the system and boot failsafe mode.

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

Starting shell.
```

2 Roll back each root pool snapshot.

```
# zfs rollback rpool@snap1
# zfs rollback rpool/ROOT@snap1
# zfs rollback rpool/ROOT/s10zfsBE@snap1
```

3 Reboot to multiuser mode.

```
# init 6
```

◆ ◆ ◆ CHAPTER 5

Managing Oracle Solaris ZFS File Systems

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

The following sections are provided in this chapter:

Managing ZFS File Systems (Overview)

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 6, "Working With Oracle Solaris ZFS Snapshots and Clones." For detailed information about ZFS volumes, see "ZFS Volumes" on page 259.

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

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 29.

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

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

ZFS automatically mounts the newly created file system if it is created successfully. By default, file systems are mounted as */dataset*, using the path provided for the file system name in the `create` subcommand. In this example, the newly created `jeff` file system is mounted at `/tank/home/jeff`. For more information about automatically managed mount points, see "Managing ZFS Mount Points" on page 188.

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 169.

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 189.

In the following example, the `tank/home/mark` file system is destroyed:

```
# zfs destroy tank/home/mark
```

Caution – No confirmation prompt appears with the `destroy` subcommand. Use it with extreme caution.

If the file system to be destroyed is busy and cannot be unmounted, the `zfs destroy` command fails. To destroy an active file system, use the `-f` option. Use this option with caution as it can unmount, unshare, and destroy active file systems, causing unexpected application behavior.

```
# zfs destroy tank/home/matt
cannot unmount 'tank/home/matt': Device busy

# zfs destroy -f tank/home/matt
```

The `zfs destroy` command also fails if a file system has descendents. To recursively destroy a file system and all its descendents, use the `-r` option. Note that a recursive destroy also destroys snapshots, so use this option with caution.

```
# zfs destroy tank/ws
cannot destroy 'tank/ws': filesystem has children
use '-r' to destroy the following datasets:
tank/ws/jeff
tank/ws/bill
tank/ws/mark
# zfs destroy -r tank/ws
```

If the file system to be destroyed has indirect dependents, even the recursive destroy command fails. To force the destruction of *all* dependents, including cloned file systems outside the target hierarchy, the `-R` option must be used. Use extreme caution with this option.

```
# zfs destroy -r tank/home/eric
cannot destroy 'tank/home/eric': filesystem has dependent clones
use '-R' to destroy the following datasets:
tank//home/eric-clone
# zfs destroy -R tank/home/eric
```

Caution – No confirmation prompt appears with the `-f`, `-r`, or `-R` options to the `zfs destroy` command, so use these options carefully.

For more information about snapshots and clones, see Chapter 6, "Working With Oracle Solaris ZFS Snapshots and Clones."

Renaming a ZFS File System

File systems can be renamed by using the `zfs rename` command. With the `rename` subcommand, you can perform the following operations:

- Change the name of a file system.
- Relocate the file system within the ZFS hierarchy.
- Change the name of a file system and relocate it within the ZFS hierarchy.

The following example uses the `rename` subcommand to rename of a file system from `eric` to `eric_old`:

```
# zfs rename tank/home/eric tank/home/eric_old
```

The following example shows how to use `zfs rename` to relocate a file system:

```
# zfs rename tank/home/mark tank/ws/mark
```

In this example, the `mark` file system is relocated from `tank/home` to `tank/ws`. When you relocate a file system through rename, the new location must be within the same pool and it must have enough disk space to hold this new file system. If the new location does not have enough disk space, possibly because it has reached its quota, the `rename` operation fails.

For more information about quotas, see "Setting ZFS Quotas and Reservations" on page 194.

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 204.

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.

Properties are divided into two types, native properties and user-defined properties. Native properties either provide internal statistics or control ZFS file system behavior. In addition, native properties are either settable or read-only. User properties have no effect on ZFS file system behavior, but you can use them to annotate datasets in a way that is meaningful in your environment. For more information about user properties, see "ZFS User Properties" on page 180.

Most settable properties are also inheritable. An inheritable property is a property that, when set on a parent file system, is propagated down to all of its descendents.

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

`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 183.
`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 180.

TABLE 5–1 ZFS Native Property Descriptions

Property Name	Type	Default Value	Description
`aclinherit`	String	`secure`	Controls how ACL entries are inherited when files and directories are created. The values are `discard`, `noallow`, `secure`, and `passthrough`. For a description of these values, see "ACL Properties" on page 227.

TABLE 5–1 ZFS Native Property Descriptions *(Continued)*

Property Name	Type	Default Value	Description
`aclmode`	String	`groupmask`	Controls how an ACL entry is modified during a `chmod` operation. The values are `discard`, `groupmask`, and `passthrough`. For a description of these values, see "ACL Properties" on page 227.
`atime`	Boolean	`on`	Controls whether the access time for files is updated when they are read. Turning this property off avoids producing write traffic when reading files and can result in significant performance gains, though it might confuse mailers and similar utilities.
`available`	Number	N/A	Read-only property that identifies the amount of disk space available to a file system and all its children, assuming no other activity in the pool. Because disk space is shared within a pool, available space can be limited by various factors including physical pool size, quotas, reservations, and other datasets within the pool. The property abbreviation is `avail`. For more information about disk space accounting, see "ZFS Disk Space Accounting" on page 30.
`canmount`	Boolean	`on`	Controls whether a file system can be mounted with the `zfs mount` command. This property can be set on any file system, and the property itself is not inheritable. However, when this property is set to `off`, a mount point can be inherited to descendent file systems, but the file system itself is never mounted. For more information, see "The `canmount` Property" on page 178.
`checksum`	String	`on`	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`, and `sha256`. A value of `off` disables integrity checking on user data. A value of `off` is not recommended.
`compression`	String	`off`	Enables or disables compression for a dataset. The values are `on`, `off`, `lzjb`, `gzip`, and `gzip-`*N*. Currently, setting this property to `lzjb`, `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`.

TABLE 5–1 ZFS Native Property Descriptions *(Continued)*

Property Name	Type	Default Value	Description
`compressratio`	Number	N/A	Read-only property that identifies the compression ratio achieved for a dataset, expressed as a multiplier. Compression can be enabled by the `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.
`copies`	Number	`1`	Sets the number of copies of user data per file system. Available values are `1`, `2`, or `3`. These copies are in addition to any pool-level redundancy. Disk space used by multiple copies of user data is charged to the corresponding file and dataset, and counts against quotas and reservations. In addition, the `used` property is updated when multiple copies are enabled. Consider setting this property when the file system is created because changing this property on an existing file system only affects newly written data.
`creation`	String	N/A	Read-only property that identifies the date and time that a dataset was created.
`devices`	Boolean	`on`	Controls whether device files in a file system can be opened.
`exec`	Boolean	`on`	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.
`mounted`	Boolean	N/A	Read-only property that indicates whether a file system, clone, or snapshot is currently mounted. This property does not apply to volumes. The value can be either `yes` or `no`.
`mountpoint`	String	N/A	Controls the mount point used for this file system. When the `mountpoint` 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 `legacy`, then they remain unmounted. Otherwise, they are automatically remounted in the new location if the property was previously `legacy` or `none`, or if they were mounted before the property was changed. In addition, any shared file systems are unshared and shared in the new location. For more information about using this property, see "Managing ZFS Mount Points" on page 188.

TABLE 5–1 ZFS Native Property Descriptions *(Continued)*

Property Name	Type	Default Value	Description
`primarycache`	String	`all`	Controls what is cached in the primary cache (ARC). 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. 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.
`origin`	String	N/A	Read-only property for cloned file systems or volumes that identifies the snapshot from which the clone was created. The origin cannot be destroyed (even with the `-r` or `-f` option) as long as a clone exists. Non-cloned file systems have an origin of `none`.
`quota`	Number (or `none`)	`none`	Limits the amount of disk space a file system and its descendents can consume. This property enforces a hard limit on the amount of disk space used, including all space consumed by descendents, such as file systems and snapshots. Setting a quota on a descendent of a file system that already has a quota does not override the ancestor's quota, but rather imposes an additional limit. Quotas cannot be set on volumes, as the `volsize` property acts as an implicit quota. For information about setting quotas, see "Setting Quotas on ZFS File Systems" on page 195.
`readonly`	Boolean	`off`	Controls whether a dataset can be modified. When set to `on`, no modifications can be made. The property abbreviation is `rdonly`.
`recordsize`	Number	`128K`	Specifies a suggested block size for files in a file system. The property abbreviation is `recsize`. For a detailed description, see "The `recordsize` Property" on page 179.

TABLE 5–1 ZFS Native Property Descriptions *(Continued)*

Property Name	Type	Default Value	Description
`referenced`	Number	N/A	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`.
`refquota`	Number (or none)	`none`	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.
`refreservation`	Number (or none)	`none`	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`.
`reservation`	Number (or none)	`none`	Sets the minimum amount of disk space guaranteed to a file system and its descendents. When the amount of disk space used is below this value, the file system is treated as if it were using the amount of space specified by its reservation. Reservations are accounted for in the parent file system's disk space used, and count against the parent file system's quotas and reservations. The property abbreviation is `reserv`. For more information, see "Setting Reservations on ZFS File Systems" on page 198.

TABLE 5–1 ZFS Native Property Descriptions *(Continued)*

Property Name	Type	Default Value	Description
`secondarycache`	String	`all`	Controls what is cached in the secondary cache (L2ARC). Possible values are `all`, `none`, and `metadata`. If set to `all`, both user data and metadata are cached. If set to `none`, neither user data nor metadata is cached. If set to `metadata`, only metadata is cached.
`setuid`	Boolean	`on`	Controls whether the `setuid` bit is honored in a file system.
`shareiscsi`	String	off	Controls whether a ZFS volume is shared as an iSCSI target. The property values are `on`, `off`, and `type=disk`. You might want to set `shareiscsi=on` for a file system so that all ZFS volumes within the file system are shared by default. However, setting this property on a file system has no direct effect.
`sharenfs`	String	`off`	Controls whether a file system is available over NFS and what options are used. If set to `on`, the `zfs share` command is invoked with no options. Otherwise, the `zfs 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 `share` and `unshare` commands and the `dfstab` file. For more information about sharing ZFS file systems, see "Sharing and Unsharing ZFS File Systems" on page 192.
`snapdir`	String	`hidden`	Controls whether the `.zfs` directory is hidden or visible in the root of the file system. For more information about using snapshots, see "Overview of ZFS Snapshots" on page 201.
`type`	String	N/A	Read-only property that identifies the dataset type as `filesystem` (file system or clone), `volume`, or `snapshot`.
`used`	Number	N/A	Read-only property that identifies the amount of disk space consumed by a dataset and all its descendents. For a detailed description, see "The used Property" on page 177.
`usedbychildren`	Number	`off`	Read-only property that identifies the amount of disk space that is used by children of this dataset, which would be freed if all the dataset's children were destroyed. The property abbreviation is `usedchild`.

TABLE 5–1 ZFS Native Property Descriptions *(Continued)*

Property Name	Type	Default Value	Description
`usedbydataset`	Number	`off`	Read-only property that identifies the amount of disk space that is used by a dataset itself, which would be freed if the dataset was destroyed, after first destroying any snapshots and removing any `refreservation` reservations. The property abbreviation is `usedds`.
`usedbyrefreservation`	Number	`off`	Read-only property that identifies the amount of disk space that is used by a `refreservation` set on a dataset, which would be freed if the `refreservation` was removed. The property abbreviation is `usedrefreserv`.
`usedbysnapshots`	Number	`off`	Read-only property that identifies the amount of disk space that is consumed by snapshots of a dataset. In particular, it is the amount of disk space that would be freed if all of this dataset's snapshots were destroyed. Note that this value is not simply the sum of the snapshots' `used` properties, because space can be shared by multiple snapshots. The property abbreviation is `usedsnap`.
`version`	Number	N/A	Identifies the on-disk version of a file system, which is independent of the pool version. This property can only be set to a later version that is available from the supported software release. For more information, see the `zfs upgrade` command.
`volsize`	Number	N/A	For volumes, specifies the logical size of the volume. For a detailed description, see "The `volsize` Property" on page 179.
`volblocksize`	Number	`8 KB`	For volumes, specifies the block size of the volume. The block size cannot be changed after the volume has been written, so set the block size at volume creation time. The default block size for volumes is 8 KB. Any power of 2 from 512 bytes to 128 KB is valid. The property abbreviation is `volblock`.

TABLE 5–1 ZFS Native Property Descriptions *(Continued)*

Property Name	Type	Default Value	Description
`zoned`	Boolean	N/A	Indicates whether a file system has been added to a non-global zone. If this property is set, then the mount point is not honored in the global zone, and ZFS cannot mount such a file system when requested. When a zone is first installed, this property is set for any added file systems. For more information about using ZFS with zones installed, see "Using ZFS on a Solaris System With Zones Installed" on page 262.
`xattr`	Boolean	`on`	Indicates whether extended attributes are enabled (`on`) or disabled (`off`) for this file system.

ZFS Read-Only Native Properties

Read-only native properties can be retrieved but not set. Read-only native properties are not inherited. Some native properties are specific to a particular type of dataset. In such cases, the dataset type is mentioned in the description in Table 5–1.

The read-only native properties are listed here and described in Table 5–1.

- `available`
- `compressratio`
- `creation`
- `mounted`
- `origin`
- `referenced`
- `type`
- `used`

  For detailed information, see "The `used` Property" on page 177.
- `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 30.

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 31.

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 5–1.

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 183 or by using the `zfs create` command as described in "Creating a ZFS File System" on page 166. 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 194.

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 5–1. 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 5–1.

- `aclinherit`

  For a detailed description, see "ACL Properties" on page 227.

- `aclmode`

  For a detailed description, see "ACL Properties" on page 227.

- atime
- canmount
- checksum
- compression
- copies
- devices
- exec
- mountpoint
- primarycache
- quota
- readonly
- recordsize

  For a detailed description, see "The recordsize Property" on page 179.
- refquota
- refreservation
- reservation
- secondarycache
- shareiscsi
- setuid
- snapdir
- version
- volsize

  For a detailed description, see "The volsize Property" on page 179.
- volblocksize
- zoned
- xattr

The canmount Property

If the canmount property is set to off, the file system cannot be mounted by using the zfs mount or zfs mount -a commands. Setting this property to off is similar to setting the mountpoint property to none, except that the file system still has a normal mountpoint property that can be inherited. For example, you can set this property to off, establish inheritable properties for descendent file systems, but the parent file system itself is never mounted nor is it accessible to users. In this case, the parent file system is serving as a *container* so that you can set properties on the container, but the container itself is never accessible.

In the following example, `userpool` is created, and its `canmount` property is set to `off`. Mount points for descendent user file systems are set to one common mount point, `/export/home`. Properties that are set on the parent file system are inherited by descendent file systems, but the parent file system itself is never mounted.

```
# zpool create userpool mirror c0t5d0 c1t6d0
# zfs set canmount=off userpool
# zfs set mountpoint=/export/home userpool
# zfs set compression=on userpool
# zfs create userpool/user1
# zfs create userpool/user2
# zfs mount
userpool/user1                  /export/home/user1
userpool/user2                  /export/home/user2
```

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 `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 259.

ZFS User Properties

In addition to the native properties, ZFS supports arbitrary user properties. User properties have no effect on ZFS behavior, but you can use them to annotate datasets with information that is meaningful in your environment.

User property names must conform to the following conventions:

- They must contain a colon (':') character to distinguish them from native properties.
- They must contain lowercase letters, numbers, or the following punctuation characters: ':', '+','.', '_'.
- The maximum length of a user property name is 256 characters.

The expected convention is that the property name is divided into the following two components but this namespace is not enforced by ZFS:

module:property

When making programmatic use of user properties, use a reversed DNS domain name for the *module* component of property names to reduce the chance that two independently developed packages will use the same property name for different purposes. Property names that begin with `com.oracle.` are reserved for use by Oracle Corporation.

The values of user properties must conform to the following conventions:

- They must consist of arbitrary strings that are always inherited and are never validated.
- The maximum length of the user property value is 1024 characters.

For example:

```
# zfs set dept:users=finance userpool/user1
# zfs set dept:users=general userpool/user2
# zfs set dept:users=itops userpool/user3
```

All of the commands that operate on properties, such as `zfs list`, `zfs get`, `zfs set`, and so on, can be used to manipulate both native properties and user properties.

For example:

```
zfs get -r dept:users userpool
NAME            PROPERTY    VALUE           SOURCE
userpool        dept:users  all             local
userpool/user1  dept:users  finance         local
userpool/user2  dept:users  general         local
userpool/user3  dept:users  itops           local
```

To clear a user property, use the `zfs inherit` command. For example:

```
# zfs inherit -r dept:users userpool
```

If the property is not defined in any parent dataset, it is removed entirely.

Querying ZFS File System Information

The `zfs list` command provides an extensible mechanism for viewing and querying dataset information. Both basic and complex queries are explained in this section.

Listing Basic ZFS Information

You can list basic dataset information by using the `zfs list` command with no options. This command displays the names of all datasets on the system and the values of their `used`, `available`, `referenced`, and `mountpoint` properties. For more information about these properties, see "Introducing ZFS Properties" on page 169.

For example:

```
# zfs list
users                   2.00G  64.9G    32K  /users
users/home              2.00G  64.9G    35K  /users/home
users/home/cindy         548K  64.9G   548K  /users/home/cindy
users/home/mark         1.00G  64.9G  1.00G  /users/home/mark
users/home/neil         1.00G  64.9G  1.00G  /users/home/neil
```

You can also use this command to display specific datasets by providing the dataset name on the command line. Additionally, use the `-r` option to recursively display all descendents of that dataset. For example:

```
# zfs list -t all -r users/home/mark
NAME                      USED  AVAIL  REFER  MOUNTPOINT
users/home/mark          1.00G  64.9G  1.00G  /users/home/mark
users/home/mark@yesterday     0      -  1.00G  -
users/home/mark@today         0      -  1.00G  -
```

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

```
# zfs list /user/home/mark
NAME              USED  AVAIL  REFER  MOUNTPOINT
users/home/mark  1.00G  64.9G  1.00G  /users/home/mark
```

The following example shows how to display basic information about `tank/home/gina` and all of its descendent file systems:

```
# zfs list -r users/home/gina
NAME                          USED  AVAIL  REFER  MOUNTPOINT
users/home/gina              2.00G  62.9G    32K  /users/home/gina
users/home/gina/projects     2.00G  62.9G    33K  /users/home/gina/projects
users/home/gina/projects/fs1  1.00G  62.9G  1.00G  /users/home/gina/projects/fs1
users/home/gina/projects/fs2  1.00G  62.9G  1.00G  /users/home/gina/projects/fs2
```

For additional information about the `zfs list` command, see `zfs(1M)`.

Creating Complex ZFS Queries

The `zfs list` output can be customized by using the `-o`, `-t`, and `-H` options.

You can customize property value output by using the `-o` option and a comma-separated list of desired properties. You can supply any dataset property as a valid argument. For a list of all supported dataset properties, see "Introducing ZFS Properties" on page 169. In addition to the properties defined, the `-o` option list can also contain the literal `name` to indicate that the output should include the name of the dataset.

The following example uses `zfs list` to display the dataset name, along with the `sharenfs` and `mountpoint` property values.

```
# zfs list -r -o name,sharenfs,mountpoint users/home
NAME                          SHARENFS  MOUNTPOINT
users/home                    on        /users/home
users/home/cindy              on        /users/home/cindy
users/home/gina               on        /users/home/gina
users/home/gina/projects      on        /users/home/gina/projects
users/home/gina/projects/fs1  on        /users/home/gina/projects/fs1
users/home/gina/projects/fs2  on        /users/home/gina/projects/fs2
users/home/mark               on        /users/home/mark
users/home/neil               on        /users/home/neil
```

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

TABLE 5–2 Types of ZFS Objects

Type	Description
`filesystem`	File systems and clones
`volume`	Volumes
`share`	File system share
`snapshot`	Snapshots

The `-t` options takes a comma-separated list of the types of datasets to be displayed. The following example uses the `-t` and `-o` options simultaneously to show the name and `used` property for all file systems:

```
# zfs list -r -t filesystem -o name,used users/home
NAME                           USED
users/home                    4.00G
users/home/cindy               548K
```

```
users/home/gina               2.00G
users/home/gina/projects      2.00G
users/home/gina/projects/fs1  1.00G
users/home/gina/projects/fs2  1.00G
users/home/mark               1.00G
users/home/neil               1.00G
```

You can use the `-H` option to omit the `zfs list` header from the generated output. With the `-H` option, all white space is replaced by the Tab character. This option can be useful when you need parseable output, for example, when scripting. The following example shows the output generated from using the `zfs list` command with the `-H` option:

```
# zfs list -r -H -o name users/home
users/home
users/home/cindy
users/home/gina
users/home/gina/projects
users/home/gina/projects/fs1
users/home/gina/projects/fs2
users/home/mark
users/home/neil
```

Managing ZFS Properties

Dataset properties are managed through the `zfs` command's `set`, `inherit`, and `get` subcommands.

- "Setting ZFS Properties" on page 183
- "Inheriting ZFS Properties" on page 184
- "Querying ZFS Properties" on page 185

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 177.

The `zfs set` command takes a property/value sequence in the format of *property=value* followed by a dataset name. Only one property can be set or modified during each `zfs set` invocation.

The following example sets the `atime` property to `off` for `tank/home`.

```
# zfs set atime=off tank/home
```

In addition, any file system property can be set when a file system is created. For example:

```
# zfs create -o atime=off tank/home
```

You can specify numeric property values by using the following easy-to-understand suffixes (in increasing sizes): BKMGTPEZ. Any of these suffixes can be followed by an optional b, indicating bytes, with the exception of the B suffix, which already indicates bytes. The following four invocations of `zfs set` are equivalent numeric expressions that set the `quota` property be set to the value of 20 GB on the `users/home/mark` file system:

```
# zfs set quota=20G users/home/mark
# zfs set quota=20g users/home/mark
# zfs set quota=20GB users/home/mark
# zfs set quota=20gb users/home/mark
```

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

```
# zfs set quota=20gb users/home/mark
cannot set property for '/users/home/mark': out of space
```

The values of non-numeric properties are case-sensitive and must be in lowercase letters, with the exception of `mountpoint` 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 file system, unless a quota or reservation is explicitly set on the descendent file system. If no ancestor has an explicit value set for an inherited property, the default value for the property is used. You can use the `zfs inherit` command to clear a property value, thus causing the value to be inherited from the parent file system.

The following example uses the `zfs set` command to turn on compression for the `tank/home/jeff` file system. Then, `zfs inherit` is used to clear the `compression` property, thus causing the property to inherit the default value of `off`. Because neither `home` nor `tank` has the `compression` property set locally, the default value is used. If both had compression enabled, the value set in the most immediate ancestor would be used (`home` in this example).

```
# zfs set compression=on tank/home/jeff
# zfs get -r compression tank/home
NAME                  PROPERTY     VALUE     SOURCE
tank/home             compression  off       default
tank/home/eric        compression  off       default
tank/home/eric@today  compression  -         -
tank/home/jeff        compression  on        local
# zfs inherit compression tank/home/jeff
# zfs get -r compression tank/home
NAME                  PROPERTY     VALUE     SOURCE
tank/home             compression  off       default
```

```
tank/home/eric        compression  off       default
tank/home/eric@today  compression  -         -
tank/home/jeff        compression  off       default
```

The `inherit` subcommand is applied recursively when the `-r` option is specified. In the following example, the command causes the value for the `compression` property to be inherited by `tank/home` and any descendents it might have:

```
# zfs inherit -r compression tank/home
```

Note – Be aware that the use of the `-r` option clears the current property setting for all descendent file systems.

For more information about the `zfs inherit` command, see `zfs(1M)`.

Querying ZFS Properties

The simplest way to query property values is by using the `zfs list` command. For more information, see "Listing Basic ZFS Information" on page 181. However, for complicated queries and for scripting, use the `zfs get` command to provide more detailed information in a customized format.

You can use the `zfs get` command to retrieve any dataset property. The following example shows how to retrieve a single property value on a dataset:

```
# zfs get checksum tank/ws
NAME            PROPERTY       VALUE                      SOURCE
tank/ws         checksum       on                         default
```

The fourth column, `SOURCE`, indicates the origin of this property value. The following table defines the possible source values.

TABLE 5–3 Possible SOURCE Values (`zfs get` Command)

Source Value	Description
`default`	This property value was never explicitly set for this dataset or any of its ancestors. The default value for this property is being used.
`inherited from` *dataset-name*	This property value is inherited from the parent dataset specified in *dataset-name*.
`local`	This property value was explicitly set for this dataset by using `zfs set`.
`temporary`	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 "Using Temporary Mount Properties" on page 191.

TABLE 5–3 Possible SOURCE Values (`zfs get` Command) *(Continued)*

Source Value	Description
- (none)	This property is read-only. Its value is generated by ZFS.

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

Note – The `casesensitivity`, `nbmand`, `normalization`, `sharesmb`, `utf8only`, and `vscan` properties are not fully operational in the Oracle Solaris 10 release because the Oracle Solaris SMB service is not supported in the Oracle Solaris 10 release.

```
# zfs get all tank/home
NAME       PROPERTY              VALUE                  SOURCE
tank/home  type                  filesystem             -
tank/home  creation              Mon Dec  3 13:10 2012  -
tank/home  used                  291K                   -
tank/home  available             58.7G                  -
tank/home  referenced            291K                   -
tank/home  compressratio         1.00x                  -
tank/home  mounted               yes                    -
tank/home  quota                 none                   default
tank/home  reservation           none                   default
tank/home  recordsize            128K                   default
tank/home  mountpoint            /tank/home             default
tank/home  sharenfs              off                    default
tank/home  checksum              on                     default
tank/home  compression           off                    default
tank/home  atime                 on                     default
tank/home  devices               on                     default
tank/home  exec                  on                     default
tank/home  setuid                on                     default
tank/home  readonly              off                    default
tank/home  zoned                 off                    default
tank/home  snapdir               hidden                 default
tank/home  aclmode               discard                default
tank/home  aclinherit            restricted             default
tank/home  canmount              on                     default
tank/home  shareiscsi            off                    default
tank/home  xattr                 on                     default
tank/home  copies                1                      default
tank/home  version               5                      -
tank/home  utf8only              off                    -
tank/home  normalization         none                   -
tank/home  casesensitivity       mixed                  -
tank/home  vscan                 off                    default
tank/home  nbmand                off                    default
tank/home  sharesmb              off                    default
tank/home  refquota              none                   default
tank/home  refreservation        none                   default
tank/home  primarycache          all                    default
tank/home  secondarycache        all                    default
tank/home  usedbysnapshots       0                      -
```

```
tank/home  usedbydataset         291K                -
tank/home  usedbychildren        0                   -
tank/home  usedbyrefreservation  0                   -
tank/home  logbias               latency             default
tank/home  sync                  standard            default
tank/home  rekeydate             -                   default
tank/home  rstchown              on                  default
```

The `-s` option to `zfs get` enables you to specify, by source type, the properties to display. This option takes a comma-separated list indicating the desired source types. Only properties with the specified source type are displayed. The valid source types are `local`, `default`, `inherited`, `temporary`, and `none`. The following example shows all properties that have been locally set on `tank/ws`.

```
# zfs get -s local all tank/ws
NAME     PROPERTY              VALUE                  SOURCE
tank/ws  compression           on                     local
```

Any of the above options can be combined with the `-r` option to recursively display the specified properties on all children of the specified file system. In the following example, all temporary properties on all file systems within `tank/home` are recursively displayed:

```
# zfs get -r -s temporary all tank/home
NAME             PROPERTY       VALUE                    SOURCE
tank/home          atime          off                    temporary
tank/home/jeff     atime          off                    temporary
tank/home/mark     quota          20G                    temporary
```

You can query property values by using the `zfs get` command without specifying a target file system, which means the command operates on all pools or file systems. For example:

```
# zfs get -s local all
tank/home               atime          off                    local
tank/home/jeff          atime          off                    local
tank/home/mark          quota          20G                    local
```

For more information about the `zfs get` command, see `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 169 section.
- A comma-separated list of literal fields, `name`, `value`, `property`, and `source`, to be output followed by a space and an argument, which is a comma-separated list of properties.

The following example shows how to retrieve a single value by using the `-H` and `-o` options of `zfs get`:

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

The `-p` option reports numeric values as their exact values. For example, 1 MB would be reported as 1000000. This option can be used as follows:

```
# zfs get -H -o value -p used tank/home
182983742
```

You can use the `-r` option, along with any of the preceding options, to recursively retrieve the requested values for all descendents. The following example uses the `-H`, `-o`, and `-r` options to retrieve the file system name and the value of the `used` property for `export/home` and its descendents, while omitting the header output:

```
# zfs get -H -o name,value -r used export/home
```

Mounting ZFS File Systems

This section describes how ZFS mounts file systems.

Managing ZFS Mount Points

By default, a ZFS file system is automatically mounted when it is created. You can determine specific mount-point behavior for a file system as described in this section.

You can also set the default mount point for a pool's file system at creation time by using `zpool create`'s `-m` option. For more information about creating pools, see "Creating ZFS Storage Pools" on page 48.

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

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

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

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

To prevent a file system from being mounted, set the mountpoint property to none. In addition, the canmount property can be used to control whether a file system can be mounted. For more information about the canmount property, see "The canmount Property" on page 178.

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 190.

Automatic Mount Points

- When you change the mountpoint property from legacy or none to a specific path, ZFS automatically mounts the file system.
- If ZFS is managing a file system but it is currently unmounted, and the mountpoint property is changed, the file system remains unmounted.

Any file system whose mountpoint property is not legacy is managed by ZFS. In the following example, a file system is created whose mount point is automatically managed by ZFS:

```
# zfs create pool/filesystem
# zfs get mountpoint pool/filesystem
NAME             PROPERTY      VALUE                     SOURCE
pool/filesystem  mountpoint    /pool/filesystem          default
# zfs get mounted pool/filesystem
NAME             PROPERTY      VALUE                     SOURCE
pool/filesystem  mounted       yes                       -
```

You can also explicitly set the mountpoint property as shown in the following example:

```
# zfs set mountpoint=/mnt pool/filesystem
# zfs get mountpoint pool/filesystem
NAME             PROPERTY      VALUE                     SOURCE
pool/filesystem  mountpoint    /mnt                      local
# zfs get mounted pool/filesystem
NAME             PROPERTY      VALUE                     SOURCE
pool/filesystem  mounted       yes                       -
```

When the mountpoint property is changed, the file system is automatically unmounted from the old mount point and remounted to the new mount point. Mount-point directories are created as needed. If ZFS is unable to unmount a file system due to it being active, an error is reported, and a forced manual unmount is necessary.

Legacy Mount Points

You can manage ZFS file systems with legacy tools by setting the `mountpoint` property to `legacy`. Legacy file systems must be managed through the `mount` and `umount` commands and the `/etc/vfstab` file. ZFS does not automatically mount legacy file systems at boot time, and the ZFS `mount` and `umount` commands do not operate on file systems of this type. The following examples show how to set up and manage a ZFS file system in legacy mode:

```
# zfs set mountpoint=legacy tank/home/eric
# mount -F zfs tank/home/eschrock /mnt
```

To automatically mount a legacy file system at boot time, you must add an entry to the `/etc/vfstab` file. The following example shows what the entry in the `/etc/vfstab` file might look like:

```
#device         device        mount           FS      fsck    mount   mount
#to mount       to fsck       point           type    pass    at boot options
#

tank/home/eric  -        /mnt           zfs        -        yes        -
```

The `device to fsck` and `fsck pass` entries are set to `-` because the `fsck` command is not applicable to ZFS file systems. For more information about ZFS data integrity, see "Transactional Semantics" on page 25.

Mounting ZFS File Systems

ZFS automatically mounts file systems when file systems are created or when the system boots. Use of the `zfs mount` command is necessary only when you need to change mount options, or explicitly mount or unmount file systems.

The `zfs mount` command with no arguments shows all currently mounted file systems that are managed by ZFS. Legacy managed mount points are not displayed. For example:

```
# zfs mount | grep tank/home
zfs mount | grep tank/home
tank/home                       /tank/home
tank/home/jeff                  /tank/home/jeff
```

You can use the `-a` option to mount all ZFS managed file systems. Legacy managed file systems are not mounted. For example:

```
# zfs mount -a
```

By default, ZFS does not allow mounting on top of a nonempty directory. For example:

```
# zfs mount tank/home/lori
cannot mount 'tank/home/lori': filesystem already mounted
```

Legacy mount points must be managed through legacy tools. An attempt to use ZFS tools results in an error. For example:

```
# zfs mount tank/home/bill
cannot mount 'tank/home/bill': legacy mountpoint
use mount(1M) to mount this filesystem
# mount -F zfs tank/home/billm
```

When a file system is mounted, it uses a set of mount options based on the property values associated with the file system. The correlation between properties and mount options is as follows:

TABLE 5–4 ZFS Mount-Related Properties and Mount Options

Property	Mount Option
atime	atime/noatime
devices	devices/nodevices
exec	exec/noexec
nbmand	nbmand/nonbmand
readonly	ro/rw
setuid	setuid/nosetuid
xattr	xattr/noaxttr

The mount option `nosuid` is an alias for `nodevices,nosetuid`.

Using Temporary Mount Properties

If any of the mount options described in the preceding section are set explicitly by using the `-o` option with the `zfs mount` command, the associated property value is temporarily overridden. These property values are reported as `temporary` by the `zfs get` command and revert back to their original values when the file system is unmounted. If a property value is changed while the file system is mounted, the change takes effect immediately, overriding any temporary setting.

In the following example, the read-only mount option is temporarily set on the `tank/home/neil` file system. The file system is assumed to be unmounted.

```
# zfs mount -o ro users/home/neil
```

To temporarily change a property value on a file system that is currently mounted, you must use the special `remount` option. In the following example, the `atime` property is temporarily changed to `off` for a file system that is currently mounted:

```
# zfs mount -o remount,noatime users/home/neil
NAME             PROPERTY  VALUE  SOURCE
users/home/neil  atime     off    temporary
# zfs get atime users/home/perrin
```

For more information about the `zfs mount` command, see `zfs(1M)`.

Unmounting ZFS File Systems

You can unmount ZFS file systems by using the `zfs unmount` subcommand. The `unmount` command can take either the mount point or the file system name as an argument.

In the following example, a file system is unmounted by its file system name:

```
# zfs unmount users/home/mark
```

In the following example, the file system is unmounted by its mount point:

```
# zfs unmount /users/home/mark
```

The `unmount` command fails if the file system is busy. To forcibly unmount a file system, you can use the `-f` option. Be cautious when forcibly unmounting a file system if its contents are actively being used. Unpredictable application behavior can result.

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

To provide for backward compatibility, the legacy `umount` command can be used to unmount ZFS file systems. For example:

```
# umount /tank/home/bob
```

For more information about the `zfs 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/eric
```

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/bill
# zfs create tank/home/mark
# zfs set sharenfs=ro tank/home/bob
```

Both `tank/home/bill` and `tank/home/mark` 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/mark` 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/mark
```

This command unshares the `tank/home/mark` 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/mark
```

You can also share all ZFS file systems on the system by using the `-a` option.

```
# zfs share -a
```

Legacy Share Behavior

If the `sharenfs` property is set to `off`, then ZFS does not attempt to share or unshare the file system at any time. This value enables you to administer file system sharing through traditional means, such as the `/etc/dfs/dfstab` file.

Unlike the legacy `mount` command, the legacy `share` and `unshare` commands can still function on ZFS file systems. As a result, you can manually share a file system with options that differ from the options of the `sharenfs` property. This administrative model is discouraged. Choose to manage NFS shares either completely through ZFS or completely through the `/etc/dfs/dfstab` file. The ZFS administrative model is designed to be simpler and less work than the traditional model.

Setting ZFS Quotas and Reservations

You can use the `quota` property to set a limit on the amount of disk space a file system can use. In addition, you can use the `reservation` property to guarantee that a specified amount of disk space is available to a file system. Both properties apply to the file system on which they are set and all descendents of that file system.

That is, if a quota is set on the `tank/home` file system, the total amount of disk space used by `tank/home` *and all of its descendents* cannot exceed the quota. Similarly, if `tank/home` is given a reservation, `tank/home` *and all of its descendents* draw from that reservation. The amount of disk space used by a file system and all of its descendents is reported by the `used` property.

The `refquota` and `refreservation` properties are used to manage file system space without accounting for disk space consumed by descendents, such as snapshots and clones.

In this Solaris release, you can set a *user* or a *group* quota on the amount of disk space consumed by files that are owned by a particular user or group. The user and group quota properties cannot be set on a volume, on a file system before file system version 4, or on a pool before pool version 15.

Consider the following points to determine which quota and reservation features might best help you manage your file systems:

- The `quota` and `reservation` properties are convenient for managing disk space consumed by file systems and their descendents.
- The `refquota` and `refreservation` properties are appropriate for managing disk space consumed by file systems.
- Setting the `refquota` or `refreservation` property higher than the `quota` or `reservation` property has no effect. If you set the `quota` or `refquota` property, operations that try to exceed either value fail. It is possible to a 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 195 and "Setting Reservations on ZFS File Systems" on page 198.

Setting Quotas on ZFS File Systems

Quotas on ZFS file systems can be set and displayed by using the `zfs set` and `zfs get` commands. In the following example, a quota of 10 GB is set on `tank/home/jeff`:

```
# zfs set quota=10G tank/home/jeff
# zfs get quota tank/home/jeff
NAME            PROPERTY  VALUE  SOURCE
tank/home/jeff  quota     10G    local
```

Quotas also affect the output of the `zfs list` and `df` commands. For example:

```
# zfs list -r tank/home
NAME                USED  AVAIL  REFER  MOUNTPOINT
tank/home          1.45M  66.9G    36K  /tank/home
tank/home/eric      547K  66.9G   547K  /tank/home/eric
tank/home/jeff      322K  10.0G   291K  /tank/home/jeff
tank/home/jeff/ws    31K  10.0G    31K  /tank/home/jeff/ws
tank/home/lori      547K  66.9G   547K  /tank/home/lori
tank/home/mark       31K  66.9G    31K  /tank/home/mark
# df -h /tank/home/jeff
Filesystem            Size  Used Avail Use% Mounted on
tank/home/jeff         10G  306K   10G   1% /tank/home/jeff
```

Note that although `tank/home` has 66.9 GB of disk space available, `tank/home/jeff` and `tank/home/jeff/ws` each have only 10 GB of disk space available, due to the quota on `tank/home/jeff`.

You cannot set a quota to an amount less than is currently being used by a file system. For example:

```
# zfs set quota=10K tank/home/jeff
cannot set property for 'tank/home/jeff':
size is less than current used or reserved space
```

You can set a `refquota` on a file system that limits the amount of disk space that the file system can consume. This hard limit does not include disk space that is consumed by descendents. For example, studentA's 10 GB quota is not impacted by space that is consumed by snapshots.

```
# zfs set refquota=10g students/studentA
# zfs list -t all -r students
NAME                         USED  AVAIL  REFER  MOUNTPOINT
students                     150M  66.8G    32K  /students
students/studentA            150M  9.85G   150M  /students/studentA
students/studentA@yesterday     0      -   150M  -
# zfs snapshot students/studentA@today
# zfs list -t all -r students
NAME                         USED  AVAIL  REFER  MOUNTPOINT
students                     150M  66.8G    32K  /students
students/studentA            150M  9.90G   100M  /students/studentA
students/studentA@yesterday  50.0M      -   150M  -
students/studentA@today          0      -   100M  -
```

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

```
# zfs set quota=20g students/studentA
# zfs list -t all -r students
NAME                          USED  AVAIL  REFER  MOUNTPOINT
students                      150M  66.8G    32K  /students
students/studentA             150M  9.90G   100M  /students/studentA
students/studentA@yesterday  50.0M      -   150M  -
students/studentA@today          0      -   100M  -
```

In this scenario, `studentA` might reach the `refquota` (10 GB) hard limit, but `studentA` can remove files to recover, even if snapshots exist.

In the preceding example, the smaller of the two quotas (10 GB as compared to 20 GB) is displayed in the `zfs list` output. To view the value of both quotas, use the `zfs get` command. For example:

```
# zfs get refquota,quota students/studentA
NAME               PROPERTY  VALUE              SOURCE
students/studentA  refquota  10G                local
students/studentA  quota     20G                local
```

Setting User and Group Quotas on a ZFS File System

You can set a user quota or a group quota by using the `zfs userquota` or `zfs groupquota` commands, respectively. For example:

```
# zfs create students/compsci
# zfs set userquota@student1=10G students/compsci
# zfs create students/labstaff
# zfs set groupquota@labstaff=20GB students/labstaff
```

Display the current user quota or group quota as follows:

```
# zfs get userquota@student1 students/compsci
NAME              PROPERTY            VALUE               SOURCE
students/compsci  userquota@student1  10G                 local
# zfs get groupquota@labstaff students/labstaff
NAME               PROPERTY             VALUE                SOURCE
students/labstaff  groupquota@labstaff  20G                  local
```

You can display general user or group disk space usage by querying the following properties:

```
# zfs userspace students/compsci
TYPE        NAME      USED  QUOTA
POSIX User  root      350M   none
POSIX User  student1  426M    10G
# zfs groupspace students/labstaff
TYPE         NAME      USED  QUOTA
POSIX Group  labstaff  250M    20G
POSIX Group  root      350M   none
```

To identify individual user or group disk space usage, query the following properties:

```
# zfs get userused@student1 students/compsci
NAME              PROPERTY           VALUE              SOURCE
students/compsci  userused@student1  550M               local
```

```
# zfs get groupused@labstaff students/labstaff
NAME               PROPERTY            VALUE               SOURCE
students/labstaff  groupused@labstaff  250                 local
```

The user and group quota properties are not displayed by using the `zfs get all` *dataset* command, which displays a list of all of the other file system properties.

You can remove a user quota or group quota as follows:

```
# zfs set userquota@student1=none students/compsci
# zfs set groupquota@labstaff=none students/labstaff
```

User and group quotas on ZFS file systems provide the following features:

- A user quota or group quota that is set on a parent file system is not automatically inherited by a descendent file system.
- However, the user or group quota is applied when a clone or a snapshot is created from a file system that has a user or group quota. Likewise, a user or group quota is included with the file system when a stream is created by using the `zfs send` command, even without the `-R` option.
- Unprivileged users can only access their own disk space usage. The root user or a user who has been granted the `userused` or `groupused` privilege, can access everyone's user or group disk space accounting information.
- The `userquota` and `groupquota` properties cannot be set on ZFS volumes, on a file system prior to file system version 4, or on a pool prior to pool version 15.

Enforcement of user and group quotas might be delayed by several seconds. This delay means that users might exceed their quota before the system notices that they are over quota and refuses additional writes with the `EDQUOT` error message.

You can use the legacy `quota` command to review user quotas in an NFS environment, for example, where a ZFS file system is mounted. Without any options, the `quota` command only displays output if the user's quota is exceeded. For example:

```
# zfs set userquota@student1=10m students/compsci
# zfs userspace students/compsci
TYPE        NAME      USED  QUOTA
POSIX User  root      350M   none
POSIX User  student1  550M    10M
# quota student1
Block limit reached on /students/compsci
```

If you reset the user quota and the quota limit is no longer exceeded, you can use the `quota -v` command to review the user's quota. For example:

```
# zfs set userquota@student1=10GB students/compsci
# zfs userspace students/compsci
TYPE        NAME      USED  QUOTA
POSIX User  root      350M   none
```

```
POSIX User  student1  550M    10G
# quota student1
# quota -v student1
Disk quotas for student1 (uid 102):
Filesystem     usage  quota  limit    timeleft  files  quota  limit    timeleft
/students/compsci
              563287 10485760 10485760           -      -      -      -           -
```

Setting Reservations on ZFS File Systems

A ZFS *reservation* is an allocation of disk space from the pool that is guaranteed to be available to a dataset. As such, you cannot reserve disk space for a dataset if that space is not currently available in the pool. The total amount of all outstanding, unconsumed reservations cannot exceed the amount of unused disk space in the pool. ZFS reservations can be set and displayed by using the `zfs set` and `zfs get` commands. For example:

```
# zfs set reservation=5G tank/home/bill
# zfs get reservation tank/home/bill
NAME            PROPERTY     VALUE   SOURCE
tank/home/bill  reservation  5G      local
```

Reservations can affect the output of the `zfs list` command. For example:

```
# zfs list -r tank/home
NAME               USED  AVAIL  REFER  MOUNTPOINT
tank/home          5.00G  61.9G    37K  /tank/home
tank/home/bill       31K  66.9G    31K  /tank/home/bill
tank/home/jeff      337K  10.0G   306K  /tank/home/jeff
tank/home/lori      547K  61.9G   547K  /tank/home/lori
tank/home/mark       31K  61.9G    31K  /tank/home/mark
```

Note that `tank/home` is using 5 GB of disk space, although the total amount of space referred to by `tank/home` and its descendents is much less than 5 GB. The used space reflects the space reserved for `tank/home/bill`. Reservations are considered in the used disk space calculation of the parent file system and do count against its quota, reservation, or both.

```
# zfs set quota=5G pool/filesystem
# zfs set reservation=10G pool/filesystem/user1
cannot set reservation for 'pool/filesystem/user1': size is greater than
available space
```

A dataset can use more disk space than its reservation, as long as unreserved space is available in the pool, and the dataset's current usage is below its quota. A dataset cannot consume disk space that has been reserved for another dataset.

Reservations are not cumulative. That is, a second invocation of `zfs set` to set a reservation does not add its reservation to the existing reservation. Rather, the second reservation replaces the first reservation. For example:

```
# zfs set reservation=10G tank/home/bill
# zfs set reservation=5G tank/home/bill
# zfs get reservation tank/home/bill
NAME            PROPERTY     VALUE   SOURCE
tank/home/bill  reservation  5G      local
```

You can set a `refreservation` reservation to guarantee disk space for a dataset that does not include disk space consumed by snapshots and clones. This reservation is accounted for in the parent dataset's space used calculation, and counts against the parent dataset's quotas and reservations. For example:

```
# zfs set refreservation=10g profs/prof1
# zfs list
NAME                      USED  AVAIL  REFER  MOUNTPOINT
profs                    10.0G  23.2G    19K  /profs
profs/prof1                10G  33.2G    18K  /profs/prof1
```

You can also set a reservation on the same dataset to guarantee dataset space and snapshot space. For example:

```
# zfs set reservation=20g profs/prof1
# zfs list
NAME                      USED  AVAIL  REFER  MOUNTPOINT
profs                    20.0G  13.2G    19K  /profs
profs/prof1                10G  33.2G    18K  /profs/prof1
```

Regular reservations are accounted for in the parent's used space calculation.

In the preceding example, the smaller of the two quotas (10 GB as compared to 20 GB) is displayed in the `zfs list` output. To view the value of both quotas, use the `zfs get` command. For example:

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

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

Upgrading ZFS File Systems

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

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

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

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

Use this command to identify the features that are available with each file system version.

```
# zfs upgrade -v
The following filesystem versions are supported:

VER  DESCRIPTION
---  --------------------------------------------------------
 1   Initial ZFS filesystem version
 2   Enhanced directory entries
 3   Case insensitive and File system unique identifier (FUID)
 4   userquota, groupquota properties
 5   System attributes

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

◆ ◆ ◆ CHAPTER 6

Working With Oracle Solaris ZFS Snapshots and Clones

This chapter describes how to create and manage Oracle Solaris ZFS snapshots and clones. Information about saving snapshots is also provided.

The following sections are provided in this chapter:

Overview of ZFS Snapshots

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 210.

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 29.

In the following example, a snapshot of `tank/home/cindy` that is named `friday` is created.

```
# zfs snapshot tank/home/cindy@friday
```

You can create snapshots for all descendent file systems by using the `-r` option. For example:

```
# zfs snapshot -r tank/home@snap1
# zfs list -t snapshot -r tank/home
NAME                   USED  AVAIL  REFER  MOUNTPOINT
tank/home@snap1           0      -  2.11G  -
tank/home/cindy@snap1     0      -   115M  -
tank/home/lori@snap1      0      -  2.00G  -
tank/home/mark@snap1      0      -  2.00G  -
tank/home/tim@snap1       0      -  57.3M  -
```

Snapshots have no modifiable properties. Nor can dataset properties be applied to a snapshot. For example:

```
# zfs set compression=on tank/home/cindy@friday
cannot set property for 'tank/home/cindy@friday':
this property can not be modified for snapshots
```

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

```
# zfs destroy tank/home/cindy@friday
```

A dataset cannot be destroyed if snapshots of the dataset exist. For example:

```
# zfs destroy tank/home/cindy
cannot destroy 'tank/home/cindy': filesystem has children
use '-r' to destroy the following datasets:
```

```
tank/home/cindy@tuesday
tank/home/cindy@wednesday
tank/home/cindy@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 167.

Holding ZFS Snapshots

If you have different automatic snapshot policies such that older snapshots are being inadvertently destroyed by `zfs receive` because they no longer exist on the sending side, you might consider using the snapshots hold feature.

Holding a snapshot prevents it from being destroyed. In addition, this feature allows a snapshot with clones to be deleted pending the removal of the last clone by using the `zfs destroy -d` command. Each snapshot has an associated user-reference count, which is initialized to zero. This count increases by 1 whenever a hold is put on a snapshot and decreases by 1 whenever a hold is released.

In the previous Oracle Solaris release, a snapshot could only be destroyed by using the `zfs destroy` command if it had no clones. In this Oracle Solaris release, the snapshot must also have a zero user-reference count.

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

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

You can use the `-r` option to recursively hold the snapshots of all descendent file systems. For example:

```
# zfs snapshot -r tank/home@now
# zfs hold -r keep tank/home@now
```

This syntax adds a single reference, `keep`, to the given snapshot or set of snapshots. Each snapshot has its own tag namespace and hold tags must be unique within that space. If a hold exists on a snapshot, attempts to destroy that held snapshot by using the `zfs destroy` command will fail. For example:

```
# zfs destroy tank/home/cindy@snap1
cannot destroy 'tank/home/cindy@snap1': dataset is busy
```

To destroy a held snapshot, use the `-d` option. For example:

```
# zfs destroy -d tank/home/cindy@snap1
```

Use the `zfs holds` command to display a list of held snapshots. For example:

```
# zfs holds tank/home@now
NAME           TAG   TIMESTAMP
tank/home@now  keep  Fri Aug  3 15:15:53 2012

# zfs holds -r tank/home@now
NAME                 TAG   TIMESTAMP
tank/home/cindy@now  keep  Fri Aug  3 15:15:53 2012
tank/home/lori@now   keep  Fri Aug  3 15:15:53 2012
tank/home/mark@now   keep  Fri Aug  3 15:15:53 2012
tank/home/tim@now    keep  Fri Aug  3 15:15:53 2012
tank/home@now        keep  Fri Aug  3 15:15:53 2012
```

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

```
# zfs release -r keep tank/home@now
```

If the snapshot is released, the snapshot can be destroyed by using the `zfs destroy` command. For example:

```
# zfs destroy -r tank/home@now
```

Two new properties identify snapshot hold information.

- The `defer_destroy` property is `on` if the snapshot has been marked for deferred destruction by using the `zfs destroy -d` command. Otherwise, the property is `off`.
- The `userrefs` property is set to the number of holds on this snapshot, also referred to as the *user-reference* count.

Renaming ZFS Snapshots

You can rename snapshots, but they must be renamed within the same pool and dataset from which they were created. For example:

```
# zfs rename tank/home/cindy@snap1 tank/home/cindy@today
```

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

```
# zfs rename tank/home/cindy@snap1 today
```

The following snapshot `rename` operation is not supported because the target pool and file system name are different from the pool and file system where the snapshot was created:

```
# zfs rename tank/home/cindy@today pool/home/cindy@saturday
cannot rename to 'pool/home/cindy@today': snapshots must be part of same
dataset
```

You can recursively rename snapshots by using the `zfs rename -r` command. For example:

```
# zfs list -t snapshot -r users/home
NAME                       USED  AVAIL  REFER  MOUNTPOINT
users/home@now            23.5K      -  35.5K  -
```

```
users/home@yesterday           0      -    38K  -
users/home/lori@yesterday      0      -  2.00G  -
users/home/mark@yesterday      0      -  1.00G  -
users/home/neil@yesterday      0      -  2.00G  -
# zfs rename -r users/home@yesterday @2daysago
# zfs list -t snapshot -r users/home
NAME                        USED  AVAIL  REFER  MOUNTPOINT
users/home@now             23.5K      -  35.5K  -
users/home@2daysago            0      -    38K  -
users/home/lori@2daysago       0      -  2.00G  -
users/home/mark@2daysago       0      -  1.00G  -
users/home/neil@2daysago       0      -  2.00G  -
```

Displaying and Accessing ZFS Snapshots

You can enable or disable the display of snapshot listings in the `zfs list` output by using the `listsnapshots` pool property. This property is enabled by default.

If you disable this property, you can use the `zfs list -t snapshot` command to display snapshot information. Or, enable the `listsnapshots` pool property. For example:

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

Snapshots of file systems are accessible in the `.zfs/snapshot` directory within the root of the file system. For example, if `tank/home/cindy` is mounted on `/home/cindy`, then the `tank/home/cindy@thursday` snapshot data is accessible in the `/home/cindy/.zfs/snapshot/thursday` directory.

```
# ls /tank/home/cindy/.zfs/snapshot
thursday   tuesday    wednesday
```

You can list snapshots as follows:

```
# zfs list -t snapshot -r tank/home
NAME                       USED  AVAIL  REFER  MOUNTPOINT
tank/home/cindy@tuesday     45K      -  2.11G  -
tank/home/cindy@wednesday   45K      -  2.11G  -
tank/home/cindy@thursday      0      -  2.17G  -
```

You can list snapshots that were created for a particular file system as follows:

```
# zfs list -r -t snapshot -o name,creation tank/home
NAME                       CREATION
tank/home/cindy@tuesday    Fri Aug  3 15:18 2012
tank/home/cindy@wednesday  Fri Aug  3 15:19 2012
tank/home/cindy@thursday   Fri Aug  3 15:19 2012
```

```
tank/home/lori@today       Fri Aug  3 15:24 2012
tank/home/mark@today       Fri Aug  3 15:24 2012
```

Disk Space Accounting for ZFS Snapshots

When a snapshot is created, its disk space is initially shared between the snapshot and the file system, and possibly with previous snapshots. As the file system changes, disk space that was previously shared becomes unique to the snapshot, and thus is counted in the snapshot's `used` property. Additionally, deleting snapshots can increase the amount of disk space unique to (and thus *used* by) other snapshots.

A snapshot's space `referenced` property value is the same as the file system's was when the snapshot was created.

You can identify additional information about how the values of the `used` property are consumed. New read-only file system properties describe disk space usage for clones, file systems, and volumes. For example:

```
$ zfs list -o space -r rpool
NAME                     AVAIL   USED  USEDSNAP  USEDDS  USEDREFRESERV  USEDCHILD
rpool                   59.1G  7.84G       21K    109K              0      7.84G
rpool@snap1                 -    21K         -       -              -          -
rpool/ROOT              59.1G  4.78G         0     31K              0      4.78G
rpool/ROOT@snap1            -      0         -       -              -          -
rpool/ROOT/zfsBE        59.1G  4.78G     15.6M   4.76G              0          0
rpool/ROOT/zfsBE@snap1      -  15.6M         -       -              -          -
rpool/dump              59.1G  1.00G       16K   1.00G              0          0
rpool/dump@snap1            -    16K         -       -              -          -
rpool/export            59.1G    99K       18K     32K              0        49K
rpool/export@snap1          -    18K         -       -              -          -
rpool/export/home       59.1G    49K       18K     31K              0          0
rpool/export/home@snap1     -    18K         -       -              -          -
rpool/swap              61.2G  2.06G         0     16K           2.06G          0
rpool/swap@snap1            -      0         -       -              -          -
```

For a description of these properties, see Table 5–1.

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/cindy` file system is rolled back to the `tuesday` snapshot:

```
# zfs rollback tank/home/cindy@tuesday
cannot rollback to 'tank/home/cindy@tuesday': more recent snapshots exist
use '-r' to force deletion of the following snapshots:
tank/home/cindy@wednesday
tank/home/cindy@thursday
# zfs rollback -r tank/home/cindy@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/cindy
NAME                     CREATION
tank/home/cindy@tuesday  Fri Aug  3 15:18 2012
```

Identifying ZFS Snapshot Differences (`zfs diff`)

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

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

```
$ ls /tank/home/tim
fileA
$ zfs snapshot tank/home/tim@snap1
$ ls /tank/home/tim
fileA  fileB
$ zfs snapshot tank/home/tim@snap2
```

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

```
$ zfs diff tank/home/tim@snap1 tank/home/tim@snap2
M       /tank/home/tim/
+       /tank/home/tim/fileB
```

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

The `R` in the following output indicates that a file in a snapshot has been renamed.

```
$ mv /tank/cindy/fileB /tank/cindy/fileC
$ zfs snapshot tank/cindy@snap2
$ zfs diff tank/cindy@snap1 tank/cindy@snap2
```

```
M       /tank/cindy/
R       /tank/cindy/fileB -> /tank/cindy/fileC
```

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

File or Directory Change	Identifier
File or directory has been modified or file or directory link has changed	`M`
File or directory is present in the older snapshot but not in the more recent snapshot	–
File or directory is present in the more recent snapshot but not in the older snapshot	+
File or directory has been renamed	`R`

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

Overview of ZFS Clones

A *clone* is a writable volume or file system whose initial contents are the same as the dataset from which it was created. As with snapshots, creating a clone is nearly instantaneous and initially consumes no additional disk space. In addition, you can snapshot a clone.

Clones can only be created from a snapshot. When a snapshot is cloned, an implicit dependency is created between the clone and snapshot. Even though the clone is created somewhere else in the file system hierarchy, the original snapshot cannot be destroyed as long as the clone exists. The `origin` property exposes this dependency, and the `zfs destroy` command lists any such dependencies, if they exist.

Clones do not inherit the properties of the dataset from which it was created. Use the `zfs get` and `zfs set` commands to view and change the properties of a cloned dataset. For more information about setting ZFS dataset properties, see "Setting ZFS Properties" on page 183.

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

To create a clone, use the `zfs clone` command, specifying the snapshot from which to create the clone, and the name of the new file system or volume. The new file system or volume can be located anywhere in the ZFS hierarchy. The new dataset is the same type (for example, file system or volume) as the snapshot from which the clone was created. You cannot create a clone of a file system in a pool that is different from where the original file system snapshot resides.

In the following example, a new clone named `tank/home/matt/bug123` with the same initial contents as the snapshot `tank/ws/gate@yesterday` is created:

```
# zfs snapshot tank/ws/gate@yesterday
# zfs clone tank/ws/gate@yesterday tank/home/matt/bug123
```

In the following example, a cloned workspace is created from the `projects/newproject@today` snapshot for a temporary user as `projects/teamA/tempuser`. Then, properties are set on the cloned workspace.

```
# zfs snapshot projects/newproject@today
# zfs clone projects/newproject@today projects/teamA/tempuser
# zfs set sharenfs=on projects/teamA/tempuser
# zfs set quota=5G projects/teamA/tempuser
```

Destroying a ZFS Clone

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

```
# zfs destroy tank/home/matt/bug123
```

Clones must be destroyed before the parent snapshot can be destroyed.

Replacing a ZFS File System With a ZFS Clone

You can use the `zfs promote` command to replace an active ZFS file system with a clone of that file system. This feature enables you to clone and replace file systems so that the *original* file system becomes the clone of the specified file system. In addition, this feature makes it possible to destroy the file system from which the clone was originally created. Without clone promotion, you cannot destroy an original file system of active clones. For more information about destroying clones, see "Destroying a ZFS Clone" on page 209.

In the following example, the `tank/test/productA` file system is cloned and then the clone file system, `tank/test/productAbeta`, becomes the original `tank/test/productA` file system.

```
# zfs create tank/test
# zfs create tank/test/productA
# zfs snapshot tank/test/productA@today
```

```
# zfs clone tank/test/productA@today tank/test/productAbeta
# zfs list -r tank/test
NAME                        USED  AVAIL  REFER  MOUNTPOINT
tank/test                   104M  66.2G    23K  /tank/test
tank/test/productA          104M  66.2G   104M  /tank/test/productA
tank/test/productA@today       0      -   104M  -
tank/test/productAbeta         0  66.2G   104M  /tank/test/productAbeta
# zfs promote tank/test/productAbeta
# zfs list -r tank/test
NAME                           USED  AVAIL  REFER  MOUNTPOINT
tank/test                      104M  66.2G    24K  /tank/test
tank/test/productA                0  66.2G   104M  /tank/test/productA
tank/test/productAbeta         104M  66.2G   104M  /tank/test/productAbeta
tank/test/productAbeta@today      0      -   104M  -
```

In this `zfs list` output, note that the disk space accounting information for the original `productA` file system has been replaced with the `productAbeta` file system.

You can complete the clone replacement process by renaming the file systems. For example:

```
# zfs rename tank/test/productA tank/test/productAlegacy
# zfs rename tank/test/productAbeta tank/test/productA
# zfs list -r tank/test
```

Optionally, you can remove the legacy file system. For example:

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

Sending and Receiving ZFS Data

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

- "Saving ZFS Data With Other Backup Products" on page 211
- "Sending a ZFS Snapshot" on page 213
- "Receiving a ZFS Snapshot" on page 214
- "Applying Different Property Values to a ZFS Snapshot Stream" on page 216
- "Sending and Receiving Complex ZFS Snapshot Streams" on page 218
- "Remote Replication of ZFS Data" on page 220

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 201.
- **Saving snapshots** – Use the `zfs send` and `zfs receive` commands to send and receive a ZFS snapshot. You can save incremental changes between snapshots, but you cannot restore files individually. You must restore the entire file system snapshot. These commands do not provide a complete backup solution for saving your ZFS data.
- **Remote replication** – Use the `zfs send` and `zfs receive` commands to copy a file system from one system to another system. This process is different from a traditional volume management product that might mirror devices across a WAN. No special configuration or hardware is required. The advantage of replicating a ZFS file system is that you can re-create a file system on a storage pool on another system, and specify different levels of configuration for the newly created pool, such as RAID-Z, but with identical file system data.
- **Archive utilities** – Save ZFS data with archive utilities such as `tar`, `cpio`, and `pax` or third-party backup products. Currently, both `tar` and `cpio` translate NFSv4-style ACLs correctly, but `pax` does not.

Saving ZFS Data With Other Backup Products

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

For up-to-date information about issues with ZFS and third-party backup products, see the Solaris 10 Release Notes.

Identifying ZFS Snapshot Streams

A snapshot of a ZFS file system or volume is converted into a snapshot stream by using the `zfs send` command. Then, you can use the snapshot stream to re-create a ZFS file system or volume by using the `zfs receive` command.

Depending on the `zfs send` options that were used to create the snapshot stream, different types of stream formats are generated.

- Full stream – Consists of all dataset content from the time that the dataset was created up to the specified snapshot.

The default stream generated by the `zfs send` command is a full stream. It contains one file system or volume, up to and including the specified snapshot. The stream does not contain snapshots other than the snapshot specified on the command line.

- Incremental stream – Consists of the differences between one snapshot and another snapshot.

A stream package is a stream type that contains one or more full or incremental streams. Three types of stream packages exist:

- Replication stream package – Consists of the specified dataset and its descendents. It includes all intermediate snapshots. If the origin of a cloned dataset is not a descendent of the snapshot specified on the command line, that origin dataset is not included in the stream package. To receive the stream, the origin dataset must exist in the destination storage pool.

  Consider the following list of datasets and their origins. Assume that they were created in the order that they appear below.

```
NAME                    ORIGIN
pool/a                  -
pool/a/1                -
pool/a/1@clone          -
pool/b                  -
pool/b/1                pool/a/1@clone
pool/b/1@clone2         -
pool/b/2                pool/b/1@clone2
pool/b@pre-send         -
pool/b/1@pre-send       -
pool/b/2@pre-send       -
pool/b@send             -
pool/b/1@send           -
pool/b/2@send           -
```

  A replication stream package that is created with the following syntax:

```
# zfs send -R pool/b@send ....
```

  Consists of the following full and incremental streams:

```
TYPE    SNAPSHOT                INCREMENTAL FROM
full    pool/b@pre-send         -
incr    pool/b@send             pool/b@pre-send
incr    pool/b/1@clone2         pool/a/1@clone
incr    pool/b/1@pre-send       pool/b/1@clone2
incr    pool/b/1@send           pool/b/1@send
incr    pool/b/2@pre-send       pool/b/1@clone2
incr    pool/b/2@send           pool/b/2@pre-send
```

  In the preceding output, the `pool/a/1@clone` snapshot is not included in the replication stream package. As such, this replication stream package can only be received in a pool that already has `pool/a/1@clone` snapshot.

- Recursive stream package – Consists of the specified dataset and its descendents. Unlike replication stream packages, intermediate snapshots are not included unless they are the origin of a cloned dataset that is included in the stream. By default, if the origin of a dataset is

not a descendent of the snapshot specified on the command line, the behavior is the similar to replication streams. However, a self-contained recursive stream, discussed below, are created in such a way that there are no external dependencies.

A recursive stream package that is created with the following syntax:

```
# zfs send -r pool/b@send ...
```

Consists of the following full and incremental streams:

```
TYPE    SNAPSHOT                INCREMENTAL FROM
full    pool/b@send             -
incr    pool/b/1@clone2         pool/a/1@clone
incr    pool/b/1@send           pool/b/1@clone2
incr    pool/b/2@send           pool/b/1@clone2
```

In the preceding output, the `pool/a/1@clone` snapshot is not included in the recursive stream package. As such, this recursive stream package can only be received in a pool that already has `pool/a/1@clone` snapshot. This behavior is similar to the replication stream package scenario described above.

- Self-contained recursive stream package - Is not dependent on any datasets that are not included in the stream package. This recursive stream package is created with the following syntax:

  ```
  # zfs send -rc pool/b@send ...
  ```

  Consists of the following full and incremental streams:

  ```
  TYPE    SNAPSHOT                INCREMENTAL FROM
  full    pool/b@send             -
  full    pool/b/1@clone2
  incr    pool/b/1@send           pool/b/1@clone2
  incr    pool/b/2@send           pool/b/1@clone2
  ```

  Notice that the self-contained recursive stream has a full stream of the `pool/b/1@clone2` snapshot, making it possible receive the `pool/b/1` snapshot with no external dependencies.

Sending a ZFS Snapshot

You can use the `zfs send` command to send a copy of a snapshot stream and receive the snapshot stream in another pool on the same system or in another pool on a different system that is used to store backup data. For example, to send the snapshot stream on a different pool to the same system, use syntax similar to the following:

```
# zfs send tank/dana@snap1 | zfs recv spool/ds01
```

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

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

```
sys1# zfs send tank/dana@snap1 | ssh sys2 zfs recv newtank/dana
```

When you send a full stream, the destination file system must not exist.

You can send incremental data by using the `zfs send -i` option. For example:

```
sys1# zfs send -i tank/dana@snap1 tank/dana@snap2 | ssh sys2 zfs recv newtank/dana
```

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.

Note – Accessing file information in the original received file system, can cause the incremental snapshot receive operation to fail with a message similar to this one:

```
cannot receive incremental stream of tank/dana@snap2 into newtank/dana:
most recent snapshot of tank/dana@snap2 does not match incremental source
```

Consider setting the `atime` property to off if you need to access file information in the original received file system and if you also need to receive incremental snapshots into the received file system.

The incremental *snap1* source can be specified as the last component of the snapshot name. This shortcut means you only have to specify the name after the `@` sign for *snap1*, which is assumed to be from the same file system as *snap2*. For example:

```
sys1# zfs send -i snap1 tank/dana@snap2 | ssh sys2 zfs recv newtank/dana
```

This shortcut syntax is equivalent to the incremental syntax in the preceding example.

The following message is displayed if you attempt to generate an incremental stream from a different file system `snapshot1`:

```
cannot send 'pool/fs@name': not an earlier snapshot from the same fs
```

If you need to store many copies, consider compressing a ZFS snapshot stream representation with the `gzip` command. For example:

```
# zfs send pool/fs@snap | gzip > backupfile.gz
```

Receiving a ZFS Snapshot

Keep the following key points in mind when you receive a file system snapshot:

- Both the snapshot and the file system are received.
- The file system and all descendent file systems are unmounted.

- The file systems are inaccessible while they are being received.
- The original file system to be received must not exist while it is being transferred.
- If the file system name already exists, you can use `zfs rename` command to rename the file system.

For example:

```
# zfs send tank/gozer@0830 > /bkups/gozer.083006
# zfs receive tank/gozer2@today < /bkups/gozer.083006
# zfs rename tank/gozer tank/gozer.old
# zfs rename tank/gozer2 tank/gozer
```

If you make a change to the destination file system and you want to perform another incremental send of a snapshot, you must first roll back the receiving file system.

Consider the following example. First, make a change to the file system as follows:

```
sys2# rm newtank/dana/file.1
```

Then, perform an incremental send of `tank/dana@snap3`. However, you must first roll back the receiving file system to receive the new incremental snapshot. Or, you can eliminate the rollback step by using the `-F` option. For example:

```
sys1# zfs send -i tank/dana@snap2 tank/dana@snap3 | ssh sys2 zfs recv -F newtank/dana
```

When you receive an incremental snapshot, the destination file system must already exist.

If you make changes to the file system and you do not roll back the receiving file system to receive the new incremental snapshot or you do not use the `-F` option, you see a message similar to the following:

```
sys1# zfs send -i tank/dana@snap4 tank/dana@snap5 | ssh sys2 zfs recv newtank/dana
cannot receive: destination has been modified since most recent snapshot
```

The following checks are performed before the `-F` option is successful:

- If the most recent snapshot doesn't match the incremental source, neither the roll back nor the receive is completed, and an error message is returned.
- If you accidentally provide the name of different file system that doesn't match the incremental source specified in the `zfs receive` command, neither the rollback nor the receive is completed, and the following error message is returned:

  ```
  cannot send 'pool/fs@name': not an earlier snapshot from the same fs
  ```

Applying Different Property Values to a ZFS Snapshot Stream

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

- Use the `zfs inherit -S` to revert a local property value to the received value, if any. If a property does not have a received value, the behavior of the `zfs inherit -S` command is the same as the `zfs inherit` command without the `-S` option. If the property does have a received value, the `zfs inherit` command masks the received value with the inherited value until issuing a `zfs inherit -S` command reverts it to the received value.
- You can use the `zfs get -o` to include the new non-default `RECEIVED` column. Or, use the `zfs get -o all` command to include all columns, including `RECEIVED`.
- You can use the `zfs send -p` option to include properties in the send stream without the `-R` option.
- You can use the `zfs receive -e` option to use the last element of the sent snapshot name to determine the new snapshot name. The following example sends the `poola/bee/cee@1` snapshot to the `poold/eee` file system and only uses the last element (`cee@1`) of the snapshot name to create the received file system and snapshot.

```
# zfs list -rt all poola
NAME              USED  AVAIL  REFER  MOUNTPOINT
poola             134K   134G    23K  /poola
poola/bee          44K   134G    23K  /poola/bee
poola/bee/cee      21K   134G    21K  /poola/bee/cee
poola/bee/cee@1      0      -    21K  -
# zfs send -R poola/bee/cee@1 | zfs receive -e poold/eee
# zfs list -rt all poold
NAME              USED  AVAIL  REFER  MOUNTPOINT
poold             134K   134G    23K  /poold
poold/eee          44K   134G    23K  /poold/eee
poold/eee/cee      21K   134G    21K  /poold/eee/cee
poold/eee/cee@1      0      -    21K  -
```

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

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

```
# zfs get compression tank/data
NAME       PROPERTY     VALUE     SOURCE
tank/data  compression  off       default
```

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

In the example, the `compression` property is enabled when the snapshot is received into `bpool`. So, for `bpool/data`, the `compression` value is on.

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

```
# zfs send -b bpool/data@snap1 | zfs recv -d restorepool
# zfs get -o all compression restorepool/data
NAME              PROPERTY     VALUE     RECEIVED  SOURCE
restorepool/data  compression  off       off       received
```

In the example, the compression value is `off`, which represents the snapshot compression value from the original `tank/data` file system.

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

```
# zfs send -R tank/home@snap1 | zfs recv -x quota bpool/home
# zfs get -r quota bpool/home
NAME                   PROPERTY  VALUE  SOURCE
bpool/home             quota     none   local
bpool/home@snap1       quota     -      -
bpool/home/lori        quota     none   default
bpool/home/lori@snap1  quota     -      -
bpool/home/mark        quota     none   default
bpool/home/mark@snap1  quota     -      -
```

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

```
# zfs send -R tank/home@snap1 | zfs recv bpool/home
# zfs get -r quota bpool/home
NAME                   PROPERTY  VALUE  SOURCE
bpool/home             quota     none   received
bpool/home@snap1       quota     -      -
bpool/home/lori        quota     10G    received
bpool/home/lori@snap1  quota     -      -
bpool/home/mark        quota     10G    received
bpool/home/mark@snap1  quota     -      -
```

Sending and Receiving Complex ZFS Snapshot Streams

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

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

- Use the `zfs send -I` option to send all incremental streams from one snapshot to a cumulative snapshot. Or, use this option to send an incremental stream from the original snapshot to create a clone. The original snapshot must already exist on the receiving side to accept the incremental stream.
- Use the `zfs send -R` option to send a replication stream of all descendent file systems. When the replication stream is received, all properties, snapshots, descendent file systems, and clones are preserved.
- When using the `zfs send -r` option without the `-c` option and when using the `zfs send -R` option stream packages omit the `origin` of clones in some circumstances. For more information, see "Identifying ZFS Snapshot Streams" on page 211.
- 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 6–1 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`.

EXAMPLE 6–1 Sending and Receiving Complex ZFS Snapshot Streams *(Continued)*

```
# zfs destroy pool/fs@snapB
# zfs destroy pool/fs@snapC
# zfs destroy pool/fs@snapD
```

To receive the combined snapshot, you would use the following command.

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

You can also use the `zfs send -I` command to combine a snapshot and a clone snapshot to create a combined dataset. For example:

```
# zfs create pool/fs
# zfs snapshot pool/fs@snap1
# zfs clone pool/fs@snap1 pool/clone
# zfs snapshot pool/clone@snapA
# zfs send -I pool/fs@snap1 pool/clone@snapA > /snaps/fsclonesnap-I
# zfs destroy pool/clone@snapA
# zfs destroy pool/clone
# zfs receive -F pool/clone < /snaps/fsclonesnap-I
```

You can use the `zfs send -R` command to replicate a ZFS file system and all descendent file systems, up to the named snapshot. When this stream is received, all properties, snapshots, descendent file systems, and clones are preserved.

In the following example, snapshots are created for user file systems. One replication stream is created for all user snapshots. Next, the original file systems and snapshots are destroyed and then recovered.

```
# zfs snapshot -r users@today
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
users               187K  33.2G    22K  /users
users@today            0      -    22K  -
users/user1          18K  33.2G    18K  /users/user1
users/user1@today      0      -    18K  -
users/user2          18K  33.2G    18K  /users/user2
users/user2@today      0      -    18K  -
users/user3          18K  33.2G    18K  /users/user3
users/user3@today      0      -    18K  -
# zfs send -R users@today > /snaps/users-R
# zfs destroy -r users
# zfs receive -F -d users < /snaps/users-R
# zfs list
NAME                USED  AVAIL  REFER  MOUNTPOINT
```

EXAMPLE 6–1 Sending and Receiving Complex ZFS Snapshot Streams *(Continued)*

```
users                196K  33.2G    22K  /users
users@today             0      -    22K  -
users/user1           18K  33.2G    18K  /users/user1
users/user1@today       0      -    18K  -
users/user2           18K  33.2G    18K  /users/user2
users/user2@today       0      -    18K  -
users/user3           18K  33.2G    18K  /users/user3
users/user3@today       0      -    18K  -
```

In the following example, the `zfs send -R` command was used to replicate the `users` file system and its descendents, and to send the replicated stream to another pool, `users2`.

```
# zfs create users2 mirror c0t1d0 c1t1d0
# zfs receive -F -d users2 < /snaps/users-R
# zfs list
NAME                  USED  AVAIL  REFER  MOUNTPOINT
users                 224K  33.2G    22K  /users
users@today              0      -    22K  -
users/user1            33K  33.2G    18K  /users/user1
users/user1@today      15K      -    18K  -
users/user2            18K  33.2G    18K  /users/user2
users/user2@today       0      -    18K  -
users/user3            18K  33.2G    18K  /users/user3
users/user3@today       0      -    18K  -
users2                188K  16.5G    22K  /users2
users2@today             0      -    22K  -
users2/user1           18K  16.5G    18K  /users2/user1
users2/user1@today       0      -    18K  -
users2/user2           18K  16.5G    18K  /users2/user2
users2/user2@today       0      -    18K  -
users2/user3           18K  16.5G    18K  /users2/user3
users2/user3@today       0      -    18K  -
```

Remote Replication of ZFS Data

You can use the `zfs send` and `zfs recv` commands to remotely copy a snapshot stream representation from one system to another system. For example:

```
# zfs send tank/cindy@today | ssh newsys zfs recv sandbox/restfs@today
```

This command sends the `tank/cindy@today` snapshot data and receives it into the `sandbox/restfs` file system. The command also creates a `restfs@today` snapshot on the `newsys` system. In this example, the user has been configured to use `ssh` on the remote system.

◆ ◆ ◆ CHAPTER 7

Using ACLs and Attributes to Protect Oracle Solaris ZFS Files

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

The following sections are provided in this chapter:

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 7–2.
- 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 226.

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 211.

Syntax Descriptions for Setting ACLs

Two basic ACL formats are provided as follows:

- **Trivial ACL** – Contains only traditional UNIX `user`, `group`, and `owner` entries.
- **Non-Trivial ACL** – Contains more entries than just owner, group, and everyone, or includes inheritance flags set, or the entries are ordered in a non-traditional way.

Syntax for Setting Trivial ACLs

```
chmod [options] A[index]{+|=}owner@ |group@
|everyone@:access-permissions/...[:inheritance-flags]:deny | allow file
```

```
chmod [options] A-owner@, group@,
everyone@:access-permissions/...[:inheritance-flags]:deny | allow file ...
```

```
chmod [options] A[index]- file
```

Syntax for Setting Non-Trivial ACLs

```
chmod [options]
A[index]{+|=}user|group:name:access-permissions/...[:inheritance-flags]:deny | allow file
```

```
chmod [options] A-user|group:name:access-permissions/...[:inheritance-flags]:deny |
allow file ...
```

```
chmod [options] A[index]- file
```

owner@, group@, everyone@
: Identifies the *ACL-entry-type* for trivial ACL syntax. For a description of *ACL-entry-types*, see Table 7–1.

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 7–1.

access-permissions/.../
: Identifies the access permissions that are granted or denied. For a description of ACL access privileges, see Table 7–2.

inheritance-flags
: Identifies an optional list of ACL inheritance flags. For a description of the ACL inheritance flags, see Table 7–4.

`deny` | `allow`
: Identifies whether the access permissions are granted or denied.

In the following example, no *ACL-entry-ID* value exists for owner@, group@, or everyone@..

```
group@:write_data/append_data/execute:deny
```

The following example includes an *ACL-entry-ID* because a specific user (*ACL-entry-type*) is included in the ACL.

```
0:user:gozer:list_directory/read_data/execute:allow
```

When an ACL entry is displayed, it looks similar to the following:

```
2:group@:write_data/append_data/execute:deny
```

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

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

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

TABLE 7–1 ACL Entry Types

ACL Entry Type	Description
`owner@`	Specifies the access granted to the owner of the object.
`group@`	Specifies the access granted to the owning group of the object.
`everyone@`	Specifies the access granted to any user or group that does not match any other ACL entry.
`user`	With a user name, specifies the access granted to an additional user of the object. Must include the *ACL-entry-ID*, which contains a *username* or *userID*. If the value is not a valid numeric UID or *username*, the ACL entry type is invalid.
`group`	With a group name, specifies the access granted to an additional group of the object. Must include the *ACL-entry-ID*, which contains a *groupname* or *groupID*. If the value is not a valid numeric GID or *groupname*, the ACL entry type is invalid.

ACL access privileges are described in the following table.

TABLE 7–2 ACL Access Privileges

Access Privilege	Compact Access Privilege	Description
`add_file`	`w`	Permission to add a new file to a directory.
`add_subdirectory`	`p`	On a directory, permission to create a subdirectory.
`append_data`	`p`	Not currently implemented.
`delete`	`d`	Permission to delete a file. For more information about specific `delete` permission behavior, see Table 7–3.
`delete_child`	`D`	Permission to delete a file or directory within a directory. For more information about specific `delete_child` permission behavior, see Table 7–3.

TABLE 7–2 ACL Access Privileges *(Continued)*

Access Privilege	Compact Access Privilege	Description
execute	x	Permission to execute a file or search the contents of a directory.
list_directory	r	Permission to list the contents of a directory.
read_acl	c	Permission to read the ACL (ls).
read_attributes	a	Permission to read basic attributes (non-ACLs) of a file. Think of basic attributes as the stat level attributes. Allowing this access mask bit means the entity can execute ls(1) and stat(2).
read_data	r	Permission to read the contents of the file.
read_xattr	R	Permission to read the extended attributes of a file or perform a lookup in the file's extended attributes directory.
synchronize	s	Not currently implemented.
write_xattr	W	Permission to create extended attributes or write to the extended attributes directory. Granting this permission to a user means that the user can create an extended attribute directory for a file. The attribute file's permissions control the user's access to the attribute.
write_data	w	Permission to modify or replace the contents of a file.
write_attributes	A	Permission to change the times associated with a file or directory to an arbitrary value.
write_acl	C	Permission to write the ACL or the ability to modify the ACL by using the chmod command.
write_owner	o	Permission to change the file's owner or group. Or, the ability to execute the chown or chgrp 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.

The following table provides additional details about ACL delete and delete_child behavior.

TABLE 7–3 ACL delete and delete_child Permission Behavior

Parent Directory Permissions	Target Object Permissions		
	ACL allows delete	ACL denies delete	Delete permission unspecified

TABLE 7–3 ACL delete and delete_child Permission Behavior *(Continued)*

Parent Directory Permissions	Target Object Permissions		
ACL allows `delete_child`	Permit	Permit	Permit
ACL denies `delete_child`	Permit	Deny	Deny
ACL allows only `write` and `execute`	Permit	Permit	Permit
ACL denies `write` and `execute`	Permit	Deny	Deny

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 7–4 ACL Inheritance Flags

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

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

ACL Properties

The ZFS file system includes the following ACL properties to determine the specific behavior of ACL inheritance and ACL interaction with `chmod` operations.

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

  The default mode for the `aclinherit` is `restricted`.
- `aclmode` – Modifies ACL behavior when a file is initially created or controls how an ACL is modified during a `chmod` operation. Values include the following:
  - `discard` – A file system with an `aclmode` property of `discard` deletes all ACL entries that do not represent the mode of the file. This is the default value.
  - `mask` – A file system with an `aclmode` property of `mask` reduces user or group permissions. The permissions are reduced, such that they are no greater than the group permission bits, unless it is a user entry that has the same UID as the owner of the file or directory. In this case, the ACL permissions are reduced so that they are no greater than owner permission bits. The mask value also preserves the ACL across mode changes, provided an explicit ACL set operation has not been performed.
  - `passthrough` – A file system with an `aclmode` property of `passthrough` indicates that no changes are made to the ACL other than generating the necessary ACL entries to represent the new mode of the file or directory.

  The default mode for the `aclmode` is `discard`.

For more information on using the `aclmode` property, see Example 7–13.

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 an 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 7–4 and "Setting ACL Inheritance on ZFS Files in Verbose Format" on page 234.

When you create a new file and depending on the `umask` value, a default trivial ACL, similar to the following, is applied:

```
$ ls -v file.1
-rw-r--r--   1 root     root      206663 Jun 23 15:06 file.1
     0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
     2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

Each user category (`owner@`, `group@`, `everyone@`) has an ACL entry in this example.

A description of this file ACL is as follows:

`0:owner@` The owner can read and modify the contents of the file (`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/synchronize:allow`). The `synchronize` access permission is not currently implemented.

When a new directory is created and depending on the `umask` value, a default directory ACL is similar to the following:

```
$ ls -dv dir.1
drwxr-xr-x   2 root     root           2 Jul 20 13:44 dir.1
     0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     1:group@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
     2:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

A description of this directory ACL is as follows:

`0:owner@` The owner can read and modify the directory contents (`list_directory/read_data/add_file/write_data/add_subdirectory /append_data`) and read and modify the file's attributes such as timestamps, extended attributes, and ACLs (`/read_xattr/write_xattr/read_attributes/write_attributes/read_acl/write_acl`). In addition, the owner can search the contents (`execute`), delete a file or directory (`delete_child`), and can modify the ownership of the directory (`write_owner:allow`).

The `synchronize` access permission is not currently implemented.

`1:group@` The group can list and read the directory contents and the directory's attributes. In addition, the group has execute permission to search the

directory contents (`list_directory/read_data/read_xattr/execute/read_attributes /read_acl/synchronize:allow`).

`2:everyone@` Everyone who is not user or group is granted read and execute permissions to the directory contents and the directory's attributes (`list_directory/read_data/read_xattr/execute/read_attributes/read_acl/synchronize:allow`). The `synchronize` access permission is not currently implemented.

Setting and Displaying ACLs on ZFS Files in Verbose Format

You can use the `chmod` command to modify ACLs on ZFS files. The following `chmod` syntax for modifying ACLs uses *acl-specification* to identify the format of the ACL. For a description of *acl-specification*, see "Syntax Descriptions for Setting ACLs" on page 222.

- Adding ACL entries
  - Adding an ACL entry for a user

    ```
    % chmod A+acl-specification filename
    ```
  - Adding an ACL entry by *index-ID*

    ```
    % chmod Aindex-ID+acl-specification filename
    ```

    This syntax inserts the new ACL entry at the specified *index-ID* location.
- Replacing an ACL entry

  ```
  % chmod A=acl-specification filename
  % chmod Aindex-ID=acl-specification filename
  ```
- Removing ACL entries
  - Removing an ACL entry by *index-ID*

    ```
    % chmod Aindex-ID- filename
    ```
  - Removing an ACL entry by user

    ```
    % chmod A-acl-specification filename
    ```
  - Removing all non-trivial ACEs from a file

    ```
    % chmod A- filename
    ```

Verbose ACL information is displayed by using the `ls -v` command. For example:

```
# ls -v file.1
-rw-r--r--   1 root     root      206695 Jul 20 13:43 file.1
     0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
```

```
     1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
     2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

For information about using the compact ACL format, see "Setting and Displaying ACLs on ZFS Files in Compact Format" on page 240.

EXAMPLE 7–1 Modifying Trivial ACLs on ZFS Files

This section provides examples of setting and displaying trivial ACLs, which means only the traditional UNIX entries, user, group, and other are included in the ACL.

In the following example, a trivial ACL exists on `file.1`:

```
# ls -v file.1
-rw-r--r--   1 root     root      206695 Jul 20 13:43 file.1
     0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
     2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

In the following example, `write_data` permissions are granted for `group@`.

```
# chmod A1=group@:read_data/write_data:allow file.1
# ls -v file.1
-rw-rw-r--   1 root     root      206695 Jul 20 13:43 file.1
     0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     1:group@:read_data/write_data:allow
     2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

In the following example, permissions on `file.1` are set back to 644.

```
# chmod 644 file.1
# ls -v file.1
-rw-r--r--   1 root     root      206695 Jul 20 13:43 file.1
     0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
     2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

EXAMPLE 7–2 Setting Non-Trivial ACLs on ZFS Files

This section provides examples of setting and displaying non-trivial ACLs.

In the following example, `read_data/execute` permissions are added for the user `gozer` on the `test.dir` directory.

EXAMPLE 7–2 Setting Non-Trivial ACLs on ZFS Files *(Continued)*

```
# chmod A+user:gozer:read_data/execute:allow test.dir
# ls -dv test.dir
drwxr-xr-x+  2 root     root           2 Jul 20 14:23 test.dir
     0:user:gozer:list_directory/read_data/execute:allow
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
     3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

In the following example, `read_data/execute` permissions are removed for user `gozer`.

```
# chmod A0- test.dir
# ls -dv test.dir
drwxr-xr-x   2 root     root           2 Jul 20 14:23 test.dir
     0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     1:group@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
     2:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

EXAMPLE 7–3 ACL Interaction With Permissions on ZFS Files

The following ACL examples illustrate the interaction between setting ACLs and then changing the file or directory's permission bits.

In the following example, a trivial ACL exists on `file.2`:

```
# ls -v file.2
-rw-r--r--   1 root     root        2693 Jul 20 14:26 file.2
     0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
     2:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

In the following example, ACL `allow` permissions are removed from `everyone@`.

```
# chmod A2- file.2
# ls -v file.2
-rw-r-----   1 root     root        2693 Jul 20 14:26 file.2
     0:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     1:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
```

EXAMPLE 7–3 ACL Interaction With Permissions on ZFS Files *(Continued)*

In this output, the file's permission bits are reset from 644 to 640. Read permissions for `everyone@` have been effectively removed from the file's permissions bits when the ACL allow permissions are removed for `everyone@`.

In the following example, the existing ACL is replaced with `read_data/write_data` permissions for `everyone@`.

```
# chmod A=everyone@:read_data/write_data:allow file.3
# ls -v file.3
-rw-rw-rw-   1 root     root        2440 Jul 20 14:28 file.3
     0:everyone@:read_data/write_data:allow
```

In this output, the `chmod` syntax effectively replaces the existing ACL with `read_data/write_data:allow` permissions to read/write permissions for owner, group, and `everyone@`. In this model, `everyone@` specifies access to any user or group. Since no `owner@` or `group@` ACL entry exists to override the permissions for owner and group, the permission bits are set to 666.

In the following example, the existing ACL is replaced with read permissions for user `gozer`.

```
# chmod A=user:gozer:read_data:allow file.3
# ls -v file.3
----------+  1 root     root        2440 Jul 20 14:28 file.3
     0:user:gozer:read_data:allow
```

In this output, the file permissions are computed to be 000 because no ACL entries exist for `owner@`, `group@`, or `everyone@`, which represent the traditional permission components of a file. The owner of the file can resolve this problem by resetting the permissions (and the ACL) as follows:

```
# chmod 655 file.3
# ls -v file.3
-rw-r-xr-x   1 root     root        2440 Jul 20 14:28 file.3
     0:owner@:execute:deny
     1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:read_data/read_xattr/execute/read_attributes/read_acl
         /synchronize:allow
     3:everyone@:read_data/read_xattr/execute/read_attributes/read_acl
         /synchronize:allow
```

EXAMPLE 7–4 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`.

EXAMPLE 7–4 Restoring Trivial ACLs on ZFS Files *(Continued)*

```
# ls -dv test5.dir
drwxr-xr-x+  2 root     root           2 Jul 20 14:32 test5.dir
     0:user:lp:read_data:file_inherit:deny
     1:user:gozer:read_data:file_inherit:deny
     2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     3:group@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
     4:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

In the following example, the non-trivial ACLs for users `gozer` and `lp` are removed. The remaining ACL contains the default values for `owner@`, `group@`, and `everyone@`.

```
# chmod A- test5.dir
# ls -dv test5.dir
drwxr-xr-x   2 root     root           2 Jul 20 14:32 test5.dir
     0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     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
```

Setting ACL Inheritance on ZFS Files in Verbose Format

You can determine how ACLs are inherited or not inherited on files and directories. By default, ACLs are not propagated. If you set a non-trivial ACL on a directory, the ACL is not inherited by any subsequent directory. You must specify the inheritance of an ACL on a file or directory.

The `aclinherit` property can be set globally on a file system. By default, `aclinherit` is set to `restricted`.

For more information, see "ACL Inheritance" on page 226.

EXAMPLE 7–5 Granting Default ACL Inheritance

By default, ACLs are not propagated through a directory structure.

In the following example, a non-trivial ACE of `read_data/write_data/execute` is applied for user `gozer` on `test.dir`.

```
# chmod A+user:gozer:read_data/write_data/execute:allow test.dir
# ls -dv test.dir
```

EXAMPLE 7–5 Granting Default ACL Inheritance *(Continued)*

```
drwxr-xr-x+  2 root     root           2 Jul 20 14:53 test.dir
     0:user:gozer:list_directory/read_data/add_file/write_data/execute:allow
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
     3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

If a `test.dir` subdirectory is created, the ACE for user `gozer` is not propagated. User `gozer` would only have access to `sub.dir` if the permissions on `sub.dir` granted him access as the file owner, group member, or `everyone@`.

```
# mkdir test.dir/sub.dir
# ls -dv test.dir/sub.dir
drwxr-xr-x   2 root     root           2 Jul 20 14:54 test.dir/sub.dir
     0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     1:group@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
     2:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

EXAMPLE 7–6 Granting ACL Inheritance on Files and Directories

This series of examples identify the file and directory ACEs that are applied when the `file_inherit` flag is set.

In the following example, `read_data/write_data` permissions are added for files in the `test2.dir` directory for user `gozer` so that he has read access on any newly created files.

```
# chmod A+user:gozer:read_data/write_data:file_inherit:allow test2.dir
# ls -dv test2.dir
drwxr-xr-x+  2 root     root           2 Jul 20 14:55 test2.dir
     0:user:gozer:read_data/write_data:file_inherit:allow
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:list_directory/read_data/read_xattr/execute/read_attributes
         /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.

EXAMPLE 7–6 Granting ACL Inheritance on Files and Directories *(Continued)*

```
# touch test2.dir/file.2
# ls -v test2.dir/file.2
-rw-r--r--+  1 root     root           0 Jul 20 14:56 test2.dir/file.2
     0:user:gozer:read_data:inherited:allow
     1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
     3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

Because the `aclinherit` property for this file system is set to the default mode, `restricted`, user `gozer` does not have `write_data` permission on `file.2` because the group permission of the file does not allow it.

Note the `inherit_only` permission, which is applied when the `file_inherit` or `dir_inherit` flags are set, is used to propagate the ACL through the directory structure. As such, user `gozer` is only granted or denied permission from `everyone@` permissions unless he is the file owner or is a member of the file's group owner. For example:

```
# mkdir test2.dir/subdir.2
# ls -dv test2.dir/subdir.2
drwxr-xr-x+  2 root     root           2 Jun 23 15:21 test2.dir/subdir.2
     0:user:gozer:list_directory/read_data/add_file/write_data:file_inherit
         /inherit_only: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.

```
# chmod A+user:gozer:read_data/write_data/execute:file_inherit/dir_inherit:allow
test3.dir
# ls -dv test3.dir
drwxr-xr-x+  2 root     root           2 Jul 20 15:00 test3.dir
     0:user:gozer:list_directory/read_data/add_file/write_data/execute
         :file_inherit/dir_inherit:allow
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

EXAMPLE 7–6 Granting ACL Inheritance on Files and Directories *(Continued)*

```
     3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow

# touch test3.dir/file.3
# ls -v test3.dir/file.3
-rw-r--r--+  1 root     root           0 Jun 23 15:25 test3.dir/file.3
     0:user:gozer:read_data:allow
     1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
     3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow

# mkdir test3.dir/subdir.1
# ls -dv test3.dir/subdir.1
drwxr-xr-x+  2 root     root           2 Jun 23 15:26 test3.dir/subdir.1
     0:user:gozer:list_directory/read_data/execute:file_inherit/dir_inherit
         :allow
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/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 above examples, because the permission bits of the parent directory for `group@` and `everyone@` deny write and execute permissions, user `gozer` is denied write and execute permissions. The default `aclinherit` property is `restricted`, which means that `write_data` and `execute` permissions are not inherited.

In the following example, user `gozer` is granted read, write, and execute permissions that are inherited for newly created files, but are not propagated to subsequent contents of the directory.

```
# chmod A+user:gozer:read_data/write_data/execute:file_inherit/no_propagate:allow
test4.dir
# ls -dv test4.dir
drwxr--r--+  2 root     root           2 Mar  1 12:11 test4.dir
     0:user:gozer:list_directory/read_data/add_file/write_data/execute
         :file_inherit/no_propagate:allow
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:list_directory/read_data/read_xattr/read_attributes/read_acl
         /synchronize:allow
     3:everyone@:list_directory/read_data/read_xattr/read_attributes/read_acl
         /synchronize:allow
```

As the following example illustrates, `gozer`'s `read_data/write_data/execute` permissions are reduced based on the owning group's permissions.

EXAMPLE 7–6 Granting ACL Inheritance on Files and Directories *(Continued)*

```
# touch test4.dir/file.4
# ls -v test4.dir/file.4
-rw-r--r--+  1 root     root           0 Jun 23 15:28 test4.dir/file.4
     0:user:gozer:read_data:allow
     1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
     3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

EXAMPLE 7–7 ACL Inheritance With ACL Inherit Mode Set to Pass Through

If the `aclinherit` property on the `tank/cindy` file system is set to `passthrough`, then user `gozer` would inherit the ACL applied on `test4.dir` for the newly created `file.5` as follows:

```
# zfs set aclinherit=passthrough tank/cindy
# touch test4.dir/file.4
# ls -v test4.dir/file.4
-rw-r--r--+  1 root     root           0 Jun 23 15:35 test4.dir/file.4
     0:user:gozer:read_data:allow
     1:owner@:read_data/write_data/append_data/read_xattr/write_xattr
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
     3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

EXAMPLE 7–8 ACL Inheritance With ACL Inherit Mode Set to Discard

If the `aclinherit` property on a file system is set to `discard`, then ACLs can potentially be discarded when the permission bits on a directory change. For example:

```
# zfs set aclinherit=discard tank/cindy
# chmod A+user:gozer:read_data/write_data/execute:dir_inherit:allow test5.dir
# ls -dv test5.dir
drwxr-xr-x+  2 root     root           2 Jul 20 14:18 test5.dir
     0:user:gozer:list_directory/read_data/add_file/write_data/execute
         :dir_inherit:allow
     1:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
     3:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

If, at a later time, you decide to tighten the permission bits on a directory, the non-trivial ACL is discarded. For example:

EXAMPLE 7–8 ACL Inheritance With ACL Inherit Mode Set to Discard *(Continued)*

```
# chmod 744 test5.dir
# ls -dv test5.dir
drwxr--r--   2 root     root           2 Jul 20 14:18 test5.dir
     0:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     1:group@:list_directory/read_data/read_xattr/read_attributes/read_acl
         /synchronize:allow
     2:everyone@:list_directory/read_data/read_xattr/read_attributes/read_acl
         /synchronize:allow
```

EXAMPLE 7–9 ACL Inheritance With ACL Inherit Mode Set to Noallow

In the following example, two non-trivial ACLs with file inheritance are set. One ACL allows `read_data` permission, and one ACL denies `read_data` permission. This example also illustrates how you can specify two ACEs in the same `chmod` command.

```
# zfs set aclinherit=noallow tank/cindy
# chmod A+user:gozer:read_data:file_inherit:deny,user:lp:read_data:file_inherit:allow
test6.dir
# ls -dv test6.dir
drwxr-xr-x+  2 root     root           2 Jul 20 14:22 test6.dir
     0:user:gozer:read_data:file_inherit:deny
     1:user:lp:read_data:file_inherit:allow
     2:owner@:list_directory/read_data/add_file/write_data/add_subdirectory
         /append_data/read_xattr/write_xattr/execute/delete_child
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     3:group@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
     4:everyone@:list_directory/read_data/read_xattr/execute/read_attributes
         /read_acl/synchronize:allow
```

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
         /read_attributes/write_attributes/read_acl/write_acl/write_owner
         /synchronize:allow
     2:group@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
     3:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize
         :allow
```

Setting and Displaying ACLs on ZFS Files in Compact Format

You can set and display permissions on ZFS files in a compact format that uses 14 unique letters to represent the permissions. The letters that represent the compact permissions are listed in Table 7–2 and Table 7–4.

You can display compact ACL listings for files and directories by using the `ls -V` command. For example:

```
# ls -V file.1
-rw-r--r--   1 root     root      206663 Jun 23 15:06 file.1
           owner@:rw-p--aARWcCos:------:allow
           group@:r-----a-R-c--s:------:allow
        everyone@:r-----a-R-c--s:------:allow
```

The compact ACL output is described as follows:

`owner@` The owner can read and modify the contents of the file (`rw=read_data/write_data`), (`p=append_data`). The owner can also modify the file's attributes such as timestamps, extended attributes, and ACLs (`a=read_attributes`, `W=write_xattr`, `R=read_xattr`, `A=write_attributes`, `c=read_acl`, `C=write_acl`). In addition, the owner can modify the ownership of the file (`o=write_owner`).

The `synchronize` (`s`) access permission is not currently implemented.

`group@` The group is granted read permissions to the file (`r=read_data`) and the file's attributes (`a=read_attributes`, `R=read_xattr`, `c=read_acl`).

The `synchronize` (`s`) access permission is not currently implemented.

`everyone@` Everyone who is not user or group is granted read permissions to the file and the file's attributes (`r=read_data`, `a=append_data`, `R=read_xattr`, `c=read_acl`, and `s=synchronize`).

The `synchronize` (`s`) access permission is not currently implemented.

Compact ACL format provides the following advantages over verbose ACL format:

- Permissions can be specified as positional arguments to the `chmod` command.
- The hyphen (-) characters, which identify no permissions, can be removed and only the required letters need to be specified.
- Both permissions and inheritance flags are set in the same fashion.

For information about using the verbose ACL format, see "Setting and Displaying ACLs on ZFS Files in Verbose Format" on page 230.

EXAMPLE 7–10 Setting and Displaying ACLs in Compact Format

In the following example, a trivial ACL exists on `file.1`:

```
# ls -V file.1
-rw-r--r--   1 root     root      206663 Jun 23 15:06 file.1
           owner@:rw-p--aARWcCos:------:allow
           group@:r-----a-R-c--s:------:allow
        everyone@:r-----a-R-c--s:------:allow
```

In this example, `read_data/execute` permissions are added for the user `gozer` on `file.1`.

```
# chmod A+user:gozer:rx:allow file.1
# ls -V file.1
-rw-r--r--+  1 root     root      206663 Jun 23 15:06 file.1
       user:gozer:r-x-----------:------:allow
           owner@:rw-p--aARWcCos:------:allow
           group@:r-----a-R-c--s:------:allow
        everyone@:r-----a-R-c--s:------:allow
```

In the following example, user `gozer` is granted read, write, and execute permissions that are inherited for newly created files and directories by using the compact ACL format.

```
# chmod A+user:gozer:rwx:fd:allow dir.2
# ls -dV dir.2
drwxr-xr-x+  2 root     root           2 Jun 23 16:04 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 `cindy` on `dir.2`, copy and paste the permission and inheritance flags (`rwx-----------:fd----:allow`) into your `chmod` command. For example:

```
# chmod A+user:cindy:rwx-----------:fd----:allow dir.2
# ls -dV dir.2
drwxr-xr-x+  2 root     root           2 Jun 23 16:04 dir.2
      user:cindy: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 7–11 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.

EXAMPLE 7–11 ACL Inheritance With ACL Inherit Mode Set to Pass Through *(Continued)*

The following examples use compact ACL syntax to show how to inherit permission bits by setting `aclinherit` mode to `passthrough`.

In this example, an ACL is set on `test1.dir` to force inheritance. The syntax creates an `owner@`, `group@`, and `everyone@` ACL entry for newly created files. Newly created directories inherit an `@owner`, `group@`, and `everyone@` ACL entry.

```
# zfs set aclinherit=passthrough tank/cindy
# pwd
/tank/cindy
# mkdir test1.dir

# chmod A=owner@:rwxpcCosRrWaAdD:fd:allow,group@:rwxp:fd:allow,everyone@::fd:allow
test1.dir
# ls -Vd test1.dir
drwxrwx---+  2 root     root           2 Jun 23 16:10 test1.dir
            owner@:rwxpdDaARWcCos:fd----:allow
            group@:rwxp----------:fd----:allow
         everyone@:--------------:fd----:allow
```

In this example, a newly created file inherits the ACL that was specified to be inherited to newly created files.

```
# cd test1.dir
# touch file.1
# ls -V file.1
-rwxrwx---+  1 root     root           0 Jun 23 16:11 file.1
            owner@:rwxpdDaARWcCos:------:allow
            group@:rwxp----------:------:allow
         everyone@:--------------:------:allow
```

In this example, a newly created directory inherits both ACEs that control access to this directory as well as ACEs for future propagation to children of the newly created directory.

```
# mkdir subdir.1
# ls -dV subdir.1
drwxrwx---+  2 root     root           2 Jun 23 16:13 subdir.1
            owner@:rwxpdDaARWcCos:fd----:allow
            group@:rwxp----------:fd----:allow
         everyone@:--------------:fd----: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.

```
# cd /tank/cindy
# mkdir test2.dir
# cd test2.dir
# touch file.2
```

EXAMPLE 7–11 ACL Inheritance With ACL Inherit Mode Set to Pass Through *(Continued)*

```
# ls -V file.2
-rw-r--r--   1 root     root           0 Jun 23 16:15 file.2
            owner@:rw-p--aARWcCos:------:allow
            group@:r-----a-R-c--s:------:allow
         everyone@:r-----a-R-c--s:------:allow
```

EXAMPLE 7–12 ACL Inheritance With ACL Inherit Mode Set to Pass Through-X

When `aclinherit=passthrough-x` is enabled, files are created with the execute (`x`) permission for `owner@`, `group@`, or `everyone@`, but only if execute permission is set in the file creation mode and in an inheritable ACE that affects the mode.

The following example shows how to inherit the execute permission by setting `aclinherit` mode to `passthrough-x`.

```
# zfs set aclinherit=passthrough-x tank/cindy
```

The following ACL is set on `/tank/cindy/test1.dir` to provide executable ACL inheritance for files for `owner@`.

```
# chmod A=owner@:rwxpcCosRrWaAdD:fd:allow,group@:rwxp:fd:allow,everyone@::fd:allow test1.dir
# ls -Vd test1.dir
drwxrwx---+  2 root     root           2 Jun 23 16:17 test1.dir
            owner@:rwxpdDaARWcCos:fd----:allow
            group@:rwxp----------:fd----:allow
         everyone@:--------------:fd----:allow
```

A file (`file1`) is created with requested permissions `0666`. The resulting permissions are `0660`. The execution permission was not inherited because the creation mode did not request it.

```
# touch test1.dir/file1
# ls -V test1.dir/file1
-rw-rw----+  1 root     root           0 Jun 23 16:18 test1.dir/file1
            owner@:rw-pdDaARWcCos:------:allow
            group@:rw-p----------:------:allow
         everyone@:--------------:------:allow
```

Next, an executable called `t` is generated by using the `cc` compiler in the `testdir` directory.

```
# cc -o t t.c
# ls -V t
-rwxrwx---+  1 root     root        7396 Dec  3 15:19 t
                owner@:rwxpdDaARWcCos:------:allow
                group@:rwxp----------:------:allow
             everyone@:--------------:------:allow
```

The resulting permissions are `0770` because `cc` requested permissions `0777`, which caused the execute permission to be inherited from the `owner@`, `group@`, and `everyone@` entries.

EXAMPLE 7–13 ACL Interaction With `chmod` Operations on ZFS Files

The following examples illustrate how specific `aclmode` and `aclinherit` property values influence the interaction of existing ACLs with a `chmod` operation that changes file or directory permissions to either reduce or expand any existing ACL permissions to be consistent with the owning group.

In this example, the `aclmode` property is set to `mask` and the `aclinherit` property is set to `restricted`. The ACL permissions in this example are displayed in compact mode, which more easily illustrates changing permissions.

The original file and group ownership and ACL permissions are as follows:

```
# zfs set aclmode=mask pond/whoville
# zfs set aclinherit=restricted pond/whoville

# ls -lV file.1
-rwxrwx---+  1 root     root      206695 Aug 30 16:03 file.1
              user:amy:r-----a-R-c---:-------:allow
             user:rory:r-----a-R-c---:-------:allow
         group:sysadmin:rw-p--aARWc---:-------:allow
            group:staff:rw-p--aARWc---:-------:allow
                owner@:rwxp--aARWcCos:-------:allow
                group@:rwxp--aARWc--s:-------:allow
             everyone@:------a-R-c--s:-------:allow
```

A `chown` operation changes the file ownership on `file.1` and the output is now seen by the owning user, `amy`. For example:

```
# chown amy:staff file.1
# su - amy
$ ls -lV file.1
-rwxrwx---+  1 amy      staff     206695 Aug 30 16:03 file.1
              user:amy:r-----a-R-c---:-------:allow
             user:rory:r-----a-R-c---:-------:allow
         group:sysadmin:rw-p--aARWc---:-------:allow
            group:staff:rw-p--aARWc---:-------:allow
                owner@:rwxp--aARWcCos:-------:allow
                group@:rwxp--aARWc--s:-------:allow
             everyone@:------a-R-c--s:-------:allow
```

The following `chmod` operation changes the permissions to a more restrictive mode. In this example, the modified `sysadmin` group's and `staff` group's ACL permissions do not exceed the owning group's permissions.

```
$ chmod 640 file.1
$ ls -lV file.1
-rw-r-----+  1 amy      staff     206695 Aug 30 16:03 file.1
              user:amy:r-----a-R-c---:-------:allow
             user:rory:r-----a-R-c---:-------:allow
         group:sysadmin:r-----a-R-c---:-------:allow
            group:staff:r-----a-R-c---:-------:allow
                owner@:rw-p--aARWcCos:-------:allow
```

EXAMPLE 7–13 ACL Interaction With chmod Operations on ZFS Files *(Continued)*

```
                group@:r-----a-R-c--s:-------:allow
             everyone@:------a-R-c--s:-------:allow
```

The following chmod operation changes the permissions to a less restrictive mode. In this example, the modified sysadmin group's and staff group's ACL permissions are restored to allow the same permissions as the owning group.

```
$ chmod 770 file.1
$ ls -lV file.1
-rwxrwx---+  1 amy      staff     206695 Aug 30 16:03 file.1
              user:amy:r-----a-R-c---:-------:allow
             user:rory:r-----a-R-c---:-------:allow
         group:sysadmin:rw-p--aARWc---:-------:allow
            group:staff:rw-p--aARWc---:-------:allow
                owner@:rwxp--aARWcCos:-------:allow
                group@:rwxp--aARWc--s:-------:allow
             everyone@:------a-R-c--s:-------:allow
```

◆ ◆ ◆ CHAPTER 8

Oracle Solaris ZFS Delegated Administration

This chapter describes how to use delegated administration to allow nonprivileged users to perform ZFS administration tasks.

The following sections are provided in this chapter:

Overview of ZFS Delegated Administration

ZFS delegated administration enables you to distribute refined permissions to specific users, groups, or everyone. Two types of delegated permissions are supported:

- Individual permissions can be explicitly delegated such as `create`, `destroy`, `mount`, `snapshot`, and so on.
- Groups of permissions called *permission sets* can be defined. A permission set can later be updated, and all of the consumers of the set automatically get the change. Permission sets begin with the `@` symbol and are limited to 64 characters in length. After the `@` symbol, the remaining characters in the set name have the same restrictions as normal ZFS file system names.

ZFS delegated administration provides features similar to the RBAC security model. ZFS delegation provides the following advantages for administering ZFS storage pools and file systems:

- Permissions follow the ZFS storage pool whenever a pool is migrated.
- Provides dynamic inheritance where you can control how the permissions propagate through the file systems.

- Can be configured so that only the creator of a file system can destroy the file system.
- You can delegate permissions to specific file systems. Newly created file systems can automatically pick up permissions.
- Provides simple NFS administration. For example, a user with explicit permissions can create a snapshot over NFS in the appropriate `.zfs/snapshot` directory.

Consider using delegated administration for distributing ZFS tasks. For information about using RBAC to manage general Oracle Solaris administration tasks, see Part III, "Roles, Rights Profiles, and Privileges," in *System Administration Guide: Security Services*.

Disabling ZFS Delegated Permissions

You control the delegated administration features by using a pool's `delegation` property. For example:

```
# zpool get delegation users
NAME  PROPERTY    VALUE       SOURCE
users  delegation  on          default
# zpool set delegation=off users
# zpool get delegation users
NAME  PROPERTY    VALUE       SOURCE
users  delegation  off         local
```

By default, the `delegation` property is enabled.

Delegating ZFS Permissions

You can use the `zfs allow` command to delegate permissions on ZFS file systems to non-root users in the following ways:

- Individual permissions can be delegated to a user, group, or everyone.
- Groups of individual permissions can be delegated as a *permission set* to a user, group, or everyone.
- Permissions can be delegated either locally to the current file system only or to all descendents of the current file system.

The following table describes the operations that can be delegated and any dependent permissions that are required to perform the delegated operations.

Permission (Subcommand)	Description	Dependencies
`allow`	The permission to grant permissions that you have to another user.	Must also have the permission that is being allowed.

Permission (Subcommand)	Description	Dependencies
`clone`	The permission to clone any of the dataset's snapshots.	Must also have the `create` permission and the `mount` permission in the original file system.
`create`	The permission to create descendent datasets.	Must also have the `mount` permission.
`destroy`	The permission to destroy a dataset.	Must also have the `mount` permission.
`diff`	The permission to identify paths within a dataset.	Non-root users need this permission to use the `zfs diff` command.
`hold`	The permission to hold a snapshot.	
`mount`	The permission to mount and unmount a file system, and create and destroy volume device links.	
`promote`	The permission to promote a clone to a dataset.	Must also have the `mount` permission and the `promote` permission in the original file system.
`receive`	The permission to create descendent file systems with the `zfs receive` command.	Must also have the `mount` permission and the `create` permission.
`release`	The permission to release a snapshot hold, which might destroy the snapshot.	
`rename`	The permission to rename a dataset.	Must also have the `create` permission and the `mount` permission in the new parent.
`rollback`	The permission to roll back a snapshot.	
`send`	The permission to send a snapshot stream.	
`share`	The permission to share and unshare a file system.	Must have both `share` and `sharenfs` to create an NFS share. Must have both `share` and `sharesmb` to create an SMB share.
`snapshot`	The permission to create a snapshot of a dataset.	

You can delegate the following set of permissions but a permission might be limited to access, read, or change permission:

- `groupquota`
- `groupused`
- `userprop`

- userquota
- userused

In addition, you can delegate administration of the following ZFS properties to non-root users:

- aclinherit
- aclmode
- atime
- canmount
- casesensitivity
- checksum
- compression
- copies
- devices
- exec
- logbias
- mountpoint
- nbmand
- normalization
- primarycache
- quota
- readonly
- recordsize
- refquota
- refreservation
- reservation
- rstchown
- secondarycache
- setuid
- sharenfs
- sharesmb
- snapdir
- sync
- utf8only
- version
- volblocksize
- volsize
- vscan
- xattr
- zoned

Some of these properties can be set only at dataset creation time. For a description of these properties, see "Introducing ZFS Properties" on page 169.

Delegating ZFS Permissions (`zfs allow`)

The `zfs allow` syntax follows:

`zfs allow -[ldugecs] everyone|`*user*|*group*[,...] *perm*|@*setname*,...] *filesystem*| *volume*

The following `zfs allow` syntax (in bold) identifies to whom the permissions are delegated:

`zfs allow [-uge]|user|group|everyone` `[,...]` *filesystem* | *volume*

Multiple entities can be specified as a comma-separated list. If no `-uge` options are specified, then the argument is interpreted preferentially as the keyword `everyone`, then as a user name, and lastly, as a group name. To specify a user or group named "everyone," use the `-u` or `-g` option. To specify a group with the same name as a user, use the `-g` option. The `-c` option delegates create-time permissions.

The following `zfs allow` syntax (in bold) identifies how permissions and permission sets are specified:

`zfs allow [-s] ... perm|@setname [,...]` *filesystem* | *volume*

Multiple permissions can be specified as a comma-separated list. Permission names are the same as ZFS subcommands and properties. For more information, see the preceding section.

Permissions can be aggregated into *permission sets* and are identified by the `-s` option. Permission sets can be used by other `zfs allow` commands for the specified file system and its descendents. Permission sets are evaluated dynamically, so changes to a set are immediately updated. Permission sets follow the same naming requirements as ZFS file systems, but the name must begin with an at sign (`@`) and can be no more than 64 characters in length.

The following `zfs allow` syntax (in bold) identifies how the permissions are delegated:

`zfs allow [-ld]` `... ...` *filesystem* | *volume*

The `-l` option indicates that the permissions are allowed for the specified file system and not its descendents, unless the `-d` option is also specified. The `-d` option indicates that the permissions are allowed for the descendent file systems and not for this file system, unless the `-l` option is also specified. If neither option is specified, then the permissions are allowed for the file system or volume and all of its descendents.

Removing ZFS Delegated Permissions (`zfs unallow`)

You can remove previously delegated permissions with the `zfs unallow` command.

For example, assume that you delegated `create`, `destroy`, `mount`, and `snapshot` permissions as follows:

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

To remove these permissions, you would use the following syntax:

```
# zfs unallow cindy tank/home/cindy
# zfs allow tank/home/cindy
```

Delegating ZFS Permissions (Examples)

EXAMPLE 8–1 Delegating Permissions to an Individual User

When you delegate `create` and `mount` permissions to an individual user, you must ensure that the user has permissions on the underlying mount point.

For example, to delegate user `mark` `create` and `mount` permissions on the `tank` file system, set the permissions first:

```
# chmod A+user:mark:add_subdirectory:fd:allow /tank/home
```

Then, use the `zfs allow` command to delegate `create`, `destroy`, and `mount` permissions. For example:

```
# zfs allow mark create,destroy,mount tank/home
```

Now, user `mark` can create his own file systems in the `tank/home` file system. For example:

```
# su mark
mark$ zfs create tank/home/mark
mark$ ^D
# su lp
$ zfs create tank/home/lp
cannot create 'tank/home/lp': permission denied
```

EXAMPLE 8–2 Delegating create and destroy Permissions to a Group

The following example shows how to set up a file system so that anyone in the `staff` group can create and mount file systems in the `tank/home` file system, as well as destroy their own file systems. However, `staff` group members cannot destroy anyone else's file systems.

```
# zfs allow staff create,mount tank/home
# zfs allow -c create,destroy tank/home
# zfs allow tank/home
---- Permissions on tank/home --------------------------------------
Create time permissions:
        create,destroy
Local+Descendent permissions:
```

EXAMPLE 8–2 Delegating create and destroy Permissions to a Group *(Continued)*

```
        group staff create,mount
# su cindy
cindy% zfs create tank/home/cindy/files
cindy% exit
# su mark
mark% zfs create tank/home/mark/data
mark% exit
cindy% zfs destroy tank/home/mark/data
cannot destroy 'tank/home/mark/data': permission denied
```

EXAMPLE 8–3 Delegating Permissions at the Correct File System Level

Ensure that you delegate users permission at the correct file system level. For example, user `mark` is delegated `create`, `destroy`, and `mount` permissions for the local and descendent file systems. User `mark` is delegated local permission to snapshot the `tank/home` file system, but he is not allowed to snapshot his own file system. So, he has not been delegated the `snapshot` permission at the correct file system level.

```
# zfs allow -l mark snapshot tank/home
# zfs allow tank/home
---- Permissions on tank/home --------------------------------------
Create time permissions:
        create,destroy
Local permissions:
        user mark snapshot
Local+Descendent permissions:
        group staff create,mount
# su mark
mark$ zfs snapshot tank/home@snap1
mark$ zfs snapshot tank/home/mark@snap1
cannot create snapshot 'tank/home/mark@snap1': permission denied
```

To delegate user `mark` permission at the descendent file system level, use the `zfs allow -d` option. For example:

```
# zfs unallow -l mark snapshot tank/home
# zfs allow -d mark snapshot tank/home
# zfs allow tank/home
---- Permissions on tank/home --------------------------------------
Create time permissions:
        create,destroy
Descendent permissions:
        user mark snapshot
Local+Descendent permissions:
        group staff create,mount
# su mark
$ zfs snapshot tank/home@snap2
cannot create snapshot 'tank/home@snap2': permission denied
$ zfs snapshot tank/home/mark@snappy
```

Now, user `mark` can only create a snapshot below the `tank/home` file system level.

EXAMPLE 8–4 Defining and Using Complex Delegated Permissions

You can delegate specific permissions to users or groups. For example, the following `zfs allow` command delegates specific permissions to the `staff` group. In addition, `destroy` and `snapshot` permissions are delegated after `tank/home` file systems are created.

```
# zfs allow staff create,mount tank/home
# zfs allow -c destroy,snapshot tank/home
# zfs allow tank/home
---- Permissions on tank/home --------------------------------------
Create time permissions:
        create,destroy,snapshot
Local+Descendent permissions:
        group staff create,mount
```

Because user `mark` is a member of the `staff` group, he can create file systems in `tank/home`. In addition, user `mark` can create a snapshot of `tank/home/mark2` because he has specific permissions to do so. For example:

```
# su mark
$ zfs create tank/home/mark2
$ zfs allow tank/home/mark2
---- Permissions on tank/home/mark2 --------------------------------
Local permissions:
        user mark create,destroy,snapshot
---- Permissions on tank/home --------------------------------------
Create time permissions:
        create,destroy,snapshot
Local+Descendent permissions:
        group staff create,mount
```

But, user `mark` cannot create a snapshot in `tank/home/mark` because he doesn't have specific permissions to do so. For example:

```
$ zfs snapshot tank/home/mark@snap1
cannot create snapshot 'tank/home/mark@snap1': permission denied
```

In this example, user `mark` has `create` permission in his home directory, which means he can create snapshots. This scenario is helpful when your file system is NFS mounted.

```
$ cd /tank/home/mark2
$ ls
$ cd .zfs
$ ls
shares snapshot
$ cd snapshot
$ ls -l
total 3
drwxr-xr-x   2 mark    staff          2 Sep 27 15:55 snap1
$ pwd
/tank/home/mark2/.zfs/snapshot
$ mkdir snap2
$ zfs list
# zfs list -r tank/home
```

EXAMPLE 8–4 Defining and Using Complex Delegated Permissions *(Continued)*

```
NAME                    USED  AVAIL  REFER  MOUNTPOINT
tank/home/mark           63K  62.3G    32K  /tank/home/mark
tank/home/mark2          49K  62.3G    31K  /tank/home/mark2
tank/home/mark2@snap1    18K      -    31K  -
tank/home/mark2@snap2      0      -    31K  -
$ ls
snap1  snap2
$ rmdir snap2
$ ls
snap1
```

EXAMPLE 8–5 Defining and Using a ZFS Delegated Permission Set

The following example shows how to create the permission set `@myset` and delegates the permission set and the `rename` permission to the group `staff` for the `tank` file system. User `cindy`, a `staff` group member, has the permission to create a file system in `tank`. However, user `lp` does not have permission to create a file system in `tank`.

```
# zfs allow -s @myset create,destroy,mount,snapshot,promote,clone,readonly tank
# zfs allow tank
---- Permissions on tank --------------------------------------------
Permission sets:
        @myset clone,create,destroy,mount,promote,readonly,snapshot
# zfs allow staff @myset,rename tank
# zfs allow tank
---- Permissions on tank --------------------------------------------
Permission sets:
        @myset clone,create,destroy,mount,promote,readonly,snapshot
Local+Descendent permissions:
        group staff @myset,rename
# chmod A+group:staff:add_subdirectory:fd:allow tank
# su cindy
cindy% zfs create tank/data
cindy% zfs allow tank
---- Permissions on tank --------------------------------------------
Permission sets:
        @myset clone,create,destroy,mount,promote,readonly,snapshot
Local+Descendent permissions:
        group staff @myset,rename
cindy% ls -l /tank
total 15
drwxr-xr-x   2 cindy   staff          2 Jun 24 10:55 data
cindy% exit
# su lp
$ zfs create tank/lp
cannot create 'tank/lp': permission denied
```

Displaying ZFS Delegated Permissions (Examples)

You can use the following command to display permissions:

```
# zfs allow dataset
```

This command displays permissions that are set or allowed on the specified dataset. The output contains the following components:

- Permission sets
- Individual permissions or create-time permissions
- Local dataset
- Local and descendent datasets
- Descendent datasets only

EXAMPLE 8–6 Displaying Basic Delegated Administration Permissions

The following output indicates that user `cindy` has `create`, `destroy`, `mount`, snapshot permissions on the `tank/cindy` file system.

```
# zfs allow tank/cindy
       -------------------------------------------------------------
       Local+Descendent permissions on (tank/cindy)
               user cindy create,destroy,mount,snapshot
```

EXAMPLE 8–7 Displaying Complex Delegated Administration Permissions

The output in this example indicates the following permissions on the `pool/fred` and `pool` file systems.

For the `pool/fred` file system:

- Two permission sets are defined:
  - `@eng` (`create`, `destroy`, `snapshot`, `mount`, `clone`, `promote`, `rename`)
  - `@simple` (`create`, `mount`)
- Create-time permissions are set for the `@eng` permission set and the `mountpoint` property. Create-time means that after a file system set is created, the `@eng` permission set and the permission to set the `mountpoint` property are delegated.
- User `tom` is delegated the `@eng` permission set, and user `joe` is granted `create`, `destroy`, and `mount` permissions for local file systems.
- User `fred` is delegated the `@basic` permission set, and `share` and `rename` permissions for the local and descendent file systems.
- User `barney` and the `staff` group are delegated the `@basic` permission set for descendent file systems only.

For the `pool` file system:

- The permission set `@simple` (create, destroy, mount) is defined.

EXAMPLE 8–7 Displaying Complex Delegated Administration Permissions *(Continued)*

- The group `staff` is granted the `@simple` permission set on the local file system.

Here is the output for this example:

```
$ zfs allow pool/fred
---- Permissions on pool/fred --------------------------------------
Permission sets:
        @eng create,destroy,snapshot,mount,clone,promote,rename
        @simple create,mount
Create time permissions:
        @eng,mountpoint
Local permissions:
        user tom @eng
        user joe create,destroy,mount
Local+Descendent permissions:
        user fred @basic,share,rename
        user barney @basic
        group staff @basic
---- Permissions on pool -------------------------------------------
Permission sets:
        @simple create,destroy,mount
Local permissions:
        group staff @simple
```

Removing ZFS Delegated Permissions (Examples)

You can use the `zfs unallow` command to remove delegated permissions. For example, user `cindy` has `create`, `destroy`, `mount`, and `snapshot` permissions on the `tank/cindy` file system.

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

The following `zfs unallow` syntax removes user `cindy`'s `snapshot` permission from the `tank/home/cindy` file system:

```
# zfs unallow cindy snapshot tank/home/cindy
# zfs allow tank/home/cindy
---- Permissions on tank/home/cindy ---------------------------------
Local+Descendent permissions:
        user cindy create,destroy,mount
cindy% zfs create tank/home/cindy/data
cindy% zfs snapshot tank/home/cindy@today
cannot create snapshot 'tank/home/cindy@today': permission denied
```

As another example, user `mark` has the following permissions on the `tank/home/mark` file system:

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

The following `zfs unallow` syntax removes all permissions for user `mark` from the `tank/home/mark` file system:

```
# zfs unallow mark tank/home/mark
```

The following `zfs unallow` syntax removes a permission set on the `tank` file system.

```
# zfs allow tank
---- Permissions on tank ------------------------------------------
Permission sets:
        @myset clone,create,destroy,mount,promote,readonly,snapshot
Create time permissions:
        create,destroy,mount
Local+Descendent permissions:
        group staff create,mount
# zfs unallow -s @myset tank
# zfs allow tank
---- Permissions on tank ------------------------------------------
Create time permissions:
        create,destroy,mount
Local+Descendent permissions:
        group staff create,mount
```

◆ ◆ ◆ CHAPTER 9

Oracle Solaris ZFS Advanced Topics

This chapter describes ZFS volumes, using ZFS on a Solaris system with zones installed, ZFS alternate root pools, and ZFS rights profiles.

The following sections are provided in this chapter:

ZFS Volumes

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 5–1.

You can display a ZFS volume's property information by using the `zfs get` or `zfs get all` command. For example:

```
# zfs get all tank/vol
```

A question mark (?) displayed for `volsize` in the `zfs get` output indicates an unknown value because an I/O error occurred. For example:

```
# zfs get -H volsize tank/vol
tank/vol        volsize ?       local
```

An I/O error generally indicates a problem with a pool device. For information about resolving pool device problems, see "Identifying Problems With ZFS Storage Pools" on page 272.

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 264.

Using a ZFS Volume as a Swap or Dump Device

During installation of a ZFS root file system or a migration from a UFS root file system, a swap device is created on a ZFS volume in the ZFS root pool. For example:

```
# swap -l
swapfile                  dev    swaplo   blocks     free
/dev/zvol/dsk/rpool/swap 253,3        16  8257520  8257520
```

During installation of a ZFS root file system or a migration from a UFS root file system, a dump device is created on a ZFS volume in the ZFS root pool. The dump device requires no administration after it is set up. For example:

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

If you need to change your swap area or dump device after the system is installed, use the `swap` and `dumpadm` commands as in previous Solaris releases. If you need to create an additional swap volume, create a ZFS volume of a specific size and then enable swap on that device. Then, add an entry for the new swap device in the `/etc/vfstab` file. For example:

```
# zfs create -V 2G rpool/swap2
# swap -a /dev/zvol/dsk/rpool/swap2
# swap -l
swapfile                   dev  swaplo blocks   free
/dev/zvol/dsk/rpool/swap  256,1      16 2097136 2097136
/dev/zvol/dsk/rpool/swap2 256,5      16 4194288 4194288
```

Do not swap to a file on a ZFS file system. A ZFS swap file configuration is not supported.

For information about adjusting the size of the swap and dump volumes, see "Adjusting the Sizes of Your ZFS Swap Device and Dump Device" on page 147.

Using a ZFS Volume as a Solaris iSCSI Target

You can easily create a ZFS volume as an iSCSI target by setting the `shareiscsi` property on the volume. For example:

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

After the iSCSI target is created, set up the iSCSI initiator. For more information about Solaris iSCSI targets and initiators, see Chapter 12, "Configuring Oracle Solaris iSCSI Targets (Tasks)," in *System Administration Guide: Devices and File Systems*.

Note – Solaris iSCSI targets can also be created and managed with the `iscsitadm` command. If you set the `shareiscsi` property on a ZFS volume, do not use the `iscsitadm` command to also create the same target device. Otherwise, you create duplicate target information for the same device.

A ZFS volume as an iSCSI target is managed just like any other ZFS dataset. However, the `rename`, `export`, and `import` operations work a little differently for iSCSI targets.

- When you rename a ZFS volume, the iSCSI target name remains the same. For example:

  ```
  # zfs rename tank/volumes/v2 tank/volumes/v1
  # iscsitadm list target
  Target: tank/volumes/v1
      iSCSI Name: iqn.1986-03.com.sun:02:984fe301-c412-ccc1-cc80-cf9a72aa062a
      Connections: 0
  ```

- Exporting a pool that contains a shared ZFS volume causes the target to be removed. Importing a pool that contains a shared ZFS volume causes the target to be shared. For example:

  ```
  # zpool export tank
  # iscsitadm list target
  # zpool import tank
  # iscsitadm list target
  Target: tank/volumes/v1
      iSCSI Name: iqn.1986-03.com.sun:02:984fe301-c412-ccc1-cc80-cf9a72aa062a
      Connections: 0
  ```

All iSCSI target configuration information is stored within the dataset. Like an NFS shared file system, an iSCSI target that is imported on a different system is shared appropriately.

Using ZFS on a Solaris System With Zones Installed

The following sections describe how to use ZFS on a system with Oracle Solaris zones:

For information about configuring zones on a system with a ZFS root file system that will be migrated or patched with Oracle Solaris Live Upgrade, see "Using Live Upgrade to Migrate or Upgrade a System With Zones (Solaris 10 10/08)" on page 131 or "Using Oracle Solaris Live Upgrade to Migrate or Upgrade a System With Zones (at Least Solaris 10 5/09)" on page 136.

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`.
- Do not add a ZFS dataset to a non-global zone when the non-global zone is configured. Instead, add a ZFS dataset after the zone is installed.
- When both a source `zonepath` and a target `zonepath` reside on a ZFS file system and are in the same pool, `zoneadm clone` will now automatically use the ZFS clone to clone a zone. The `zoneadm clone` command will create a ZFS snapshot of the source `zonepath` and set up the target `zonepath`. You cannot use the `zfs clone` command to clone a zone. For more

information, see Part II, "Zones," in *System Administration Guide: Oracle Solaris Containers-Resource Management and Oracle Solaris Zones*.

- If you delegate a ZFS file system to a non-global zone, you must remove that file system from the non-global zone before using Oracle Solaris Live Upgrade. Otherwise, Oracle Live Upgrade will fail due to a read-only file system error.

Adding ZFS File Systems to a Non-Global Zone

You can add a ZFS file system as a generic file system when the goal is solely to share space with the global zone. A ZFS file system that is added to a non-global zone must have its `mountpoint` property set to `legacy`. 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=/opt/data
zonecfg:zion:fs> end
```

This syntax adds the ZFS file system, `tank/zone/zion`, to the already configured `zion` zone, which is mounted at `/opt/data`. The `mountpoint` property of the file system must be set to `legacy`, and the file system cannot already be mounted in another location. The zone administrator can create and destroy files within the file system. The file system cannot be remounted in a different location, nor can the zone administrator change properties on the file system such as `atime`, `readonly`, `compression`, and so on. The global zone administrator is responsible for setting and controlling properties of the file system.

For more information about the `zonecfg` command and about configuring resource types with `zonecfg`, see Part II, "Zones," in *System Administration Guide: Oracle Solaris Containers-Resource Management and Oracle Solaris Zones*.

Delegating Datasets to a Non-Global Zone

To meet the primary goal of delegating the administration of storage to a zone, ZFS supports adding datasets to a non-global zone through the use of the `zonecfg` command's `add dataset` subcommand.

In the following example, a ZFS file system is delegated to a non-global zone by a global zone administrator from the global zone.

```
# zonecfg -z zion
zonecfg:zion> add dataset
zonecfg:zion:dataset> set name=tank/zone/zion
zonecfg:zion:dataset> end
```

Unlike adding a file system, this syntax causes the ZFS file system `tank/zone/zion` to be visible within the already configured `zion` zone. The zone administrator can set file system properties, as well as create descendent file systems. In addition, the zone administrator can create snapshots and clones, and otherwise control the entire file system hierarchy.

Unlike adding a file system, this syntax causes the ZFS file system `tank/zone/zion` to be visible within the already configured `zion` zone. Within the `zion` zone, this file system is not accessible as `tank/zone/zion`, but as a *virtual pool* named `tank`. The delegated file system alias provides a view of the original pool to the zone as a virtual pool. The alias property specifies the name of the virtual pool. If no alias is specified, a default alias matching the last component of the file system name is used. If a specific alias is not provided, the default alias in the above example would have been `zion`.

Within delegated datasets, the zone administrator can set file system properties, as well as create descendent file systems. In addition, the zone administrator can create snapshots and clones, and otherwise control the entire file system hierarchy. If ZFS volumes are created within delegated file systems, it is possible for them to conflict with ZFS volumes that are added as device resources. For more information, see the next section.

If you are using Oracle Solaris Live Upgrade to upgrade your ZFS BE with non-global zones, first remove any delegated datasets. Otherwise, Oracle Solaris Live Upgrade will fail with a read-only file system error. For example:

```
zonecfg:zion>
zonecfg:zion> remove dataset name=tank/zone/zion
zonecfg:zion1> exit
```

For more information about what actions are allowed within zones, see "Managing ZFS Properties Within a Zone" on page 265.

Adding ZFS Volumes to a Non-Global Zone

ZFS volumes cannot be added to a non-global zone by using the `zonecfg` command's `add dataset` subcommand. However, volumes can be added to a zone by using the `zonecfg` command's `add device` subcommand.

In the following example, a ZFS volume is added to a non-global zone by a global zone administrator from the global zone:

```
# zonecfg -z zion
zion: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:zion> create
zonecfg:zion> add device
zonecfg:zion:device> set match=/dev/zvol/dsk/tank/vol
zonecfg:zion:device> end
```

This syntax adds the `tank/vol` volume to the `zion` zone.

Note that adding a raw volume to a zone has implicit security risks, even if the volume doesn't correspond to a physical device. In particular, the zone administrator could create malformed file systems that would panic the system when a mount is attempted. For more information about adding devices to zones and the related security risks, see "Understanding the zoned Property" on page 266.

For more information about adding devices to zones, see Part II, "Zones," in *System Administration Guide: Oracle Solaris Containers-Resource Management and Oracle Solaris Zones*.

Using ZFS Storage Pools Within a Zone

ZFS storage pools cannot be created or modified within a zone. The delegated administration model centralizes control of physical storage devices within the global zone and control of virtual storage to non-global zones. Although a pool-level dataset can be added to a zone, any command that modifies the physical characteristics of the pool, such as creating, adding, or removing devices, is not allowed from within a zone. Even if physical devices are added to a zone by using the `zonecfg` command's `add device` subcommand, or if files are used, the `zpool` command does not allow the creation of any new pools within the zone.

Managing ZFS Properties Within a Zone

After a dataset is delegated to a zone, the zone administrator can control specific dataset properties. After a dataset is delegated to a zone, all its ancestors are visible as read-only datasets, while the dataset itself is writable, as are all of its descendents. For example, consider the following configuration:

```
global# zfs list -Ho name
tank
tank/home
tank/data
tank/data/matrix
tank/data/zion
tank/data/zion/home
```

If `tank/data/zion` were added to a zone, each dataset would have the following properties.

Dataset	Visible	Writable	Immutable Properties
tank	Yes	No	-
tank/home	No	-	-
tank/data	Yes	No	-
tank/data/matrix	No	-	-
tank/data/zion	Yes	Yes	sharenfs, zoned, quota, reservation
tank/data/zion/home	Yes	Yes	sharenfs, zoned

Note that every parent of tank/zone/zion is visible as read-only, all descendents are writable, and datasets that are not part of the parent hierarchy are not visible at all. The zone administrator cannot change the sharenfs property because non-global zones cannot act as NFS servers. The zone administrator cannot change the zoned property because doing so would expose a security risk as described in the next section.

Privileged users in the zone can change any other settable property, except for quota and reservation properties. This behavior allows the global zone administrator to control the disk space consumption of all datasets used by the non-global zone.

In addition, the sharenfs and mountpoint properties cannot be changed by the global zone administrator after a dataset has been delegated to a non-global zone.

Understanding the zoned Property

When a dataset is delegated to a non-global zone, the dataset must be specially marked so that certain properties are not interpreted within the context of the global zone. After a dataset has been delegated to a non-global zone and is under the control of a zone administrator, its contents can no longer be trusted. As with any file system, there might be setuid binaries, symbolic links, or otherwise questionable contents that might adversely affect the security of the global zone. In addition, the mountpoint property cannot be interpreted in the context of the global zone. Otherwise, the zone administrator could affect the global zone's namespace. To address the latter, ZFS uses the zoned property to indicate that a dataset has been delegated to a non-global zone at one point in time.

The zoned property is a boolean value that is automatically turned on when a zone containing a ZFS dataset is first booted. A zone administrator does not need to manually turn on this property. If the zoned property is set, the dataset cannot be mounted or shared in the global zone. In the following example, tank/zone/zion has been delegated to a zone, while tank/zone/global has not:

```
# zfs list -o name,zoned,mountpoint -r tank/zone
NAME                  ZONED  MOUNTPOINT
tank/zone/global        off  /tank/zone/global
tank/zone/zion           on  /tank/zone/zion
# zfs mount
tank/zone/global           /tank/zone/global
tank/zone/zion             /export/zone/zion/root/tank/zone/zion
```

Note the difference between the `mountpoint` property and the directory where the `tank/zone/zion` dataset is currently mounted. The `mountpoint` property reflects the property as it is stored on disk, not where the dataset is currently mounted on the system.

When a dataset is removed from a zone or a zone is destroyed, the `zoned` property is *not* automatically cleared. This behavior is due to the inherent security risks associated with these tasks. Because an untrusted user has had complete access to the dataset and its descendents, the `mountpoint` property might be set to bad values, or `setuid` binaries might exist on the file systems.

To prevent accidental security risks, the `zoned` property must be manually cleared by the global zone administrator if you want to reuse the dataset in any way. Before setting the `zoned` property to `off`, ensure that the `mountpoint` property for the dataset and all its descendents are set to reasonable values and that no `setuid` binaries exist, or turn off the `setuid` property.

After you have verified that no security vulnerabilities are left, the `zoned` property can be turned off by using the `zfs set` or `zfs inherit` command. If the `zoned` property is turned off while a dataset is in use within a zone, the system might behave in unpredictable ways. Only change the property if you are sure the dataset is no longer in use by a non-global zone.

Using ZFS Alternate Root Pools

When a pool is created, it is intrinsically tied to the host system. The host system maintains information about the pool so that it can detect when the pool is unavailable. Although useful for normal operations, this information can prove a hindrance when you are booting from alternate media or creating a pool on removable media. To solve this problem, ZFS provides an *alternate root* pool feature. An alternate root pool does not persist across system reboots, and all mount points are modified to be relative to the root of the pool.

Creating ZFS Alternate Root Pools

The most common reason for creating an alternate root pool is for use with removable media. In these circumstances, users typically want a single file system, and they want it to be mounted wherever they choose on the target system. When an alternate root pool is created by using the `zpool create -R` option, the mount point of the root file system is automatically set to `/`, which is the equivalent of the alternate root value.

In the following example, a pool called `morpheus` is created with `/mnt` as the alternate root path:

```
# zpool create -R /mnt morpheus c0t0d0
# zfs list morpheus
NAME                   USED  AVAIL  REFER  MOUNTPOINT
morpheus              32.5K  33.5G     8K  /mnt
```

Note the single file system, `morpheus`, whose mount point is the alternate root of the pool, `/mnt`. The mount point that is stored on disk is `/` and the full path to `/mnt` is interpreted only in this initial context of the pool creation. This file system can then be exported and imported under an arbitrary alternate root pool on a different system by using `-R` *alternate root value* syntax.

```
# zpool export morpheus
# zpool import morpheus
cannot mount '/': directory is not empty
# zpool export morpheus
# zpool import -R /mnt morpheus
# zfs list morpheus
NAME                   USED  AVAIL  REFER  MOUNTPOINT
morpheus              32.5K  33.5G     8K  /mnt
```

Importing Alternate Root Pools

Pools can also be imported using an alternate root. This feature allows for recovery situations, where the mount points should not be interpreted in context of the current root, but under some temporary directory where repairs can be performed. This feature also can be used when you are mounting removable media as described in the preceding section.

In the following example, a pool called `morpheus` is imported with `/mnt` as the alternate root path. This example assumes that `morpheus` was previously exported.

```
# zpool import -R /a pool
# zpool list morpheus
NAME   SIZE   ALLOC  FREE    CAP  HEALTH  ALTROOT
pool  44.8G    78K  44.7G     0%  ONLINE  /a
# zfs list pool
NAME   USED  AVAIL  REFER  MOUNTPOINT
pool  73.5K  44.1G    21K  /a/pool
```

◆ ◆ ◆ CHAPTER 10

Oracle Solaris ZFS Troubleshooting and Pool Recovery

This chapter describes how to identify and recover from ZFS failures. Information for preventing failures is provided as well.

The following sections are provided in this chapter:

For information about complete root pool recovery, see "Recovering the ZFS Root Pool or Root Pool Snapshots" on page 157.

Identifying ZFS Problems

As a combined file system and volume manager, ZFS can exhibit many different failures. This chapter outlines how to diagnose general hardware failures and then how to resolve pool device and file system problems. You can encounter the following types of problems:

- **General Hardware Problems** – Hardware problems can impact your pool performance and the availability of your pool data. Rule out general hardware problems, such as faulty components and memory, before determining problems at a higher level, such as your pools and file systems.
- **ZFS Storage Pool Problems**
  - "Identifying Problems With ZFS Storage Pools" on page 272
  - "Resolving ZFS Storage Device Problems" on page 276
- "Resolving ZFS File System Problems" on page 289

- "Repairing a Damaged ZFS Configuration" on page 298
- "Repairing an Unbootable System" on page 298

Note that a single pool can experience all three errors, so a complete repair procedure involves finding and correcting one error, proceeding to the next error, and so on.

Resolving General Hardware Problems

Review the following sections to determine whether pool problems or file system unavailability is related to a hardware problem, such as faulty system board, memory, device, HBA, or a misconfiguration.

For example, a failing or faulty disk on a busy ZFS pool can greatly degrade overall system performance.

If you start by diagnosing and identifying hardware problems first, which can be easier to detect and all your hardware checks out, you can then move on to diagnosing pool and file system problems as described in the rest of this chapter. If your hardware, pool, and file system configurations are healthy, consider diagnosing application problems, which are generally more complex to unravel and are not covered in this guide.

Identifying Hardware and Device Faults

The Solaris Fault Manager tracks software, hardware and specific device problems by identifying error telemetry information that indicate a specific symptom in an error log and then reporting actual fault diagnosis when the error symptom results in an actual fault.

The following command identifies any software or hardware related fault.

```
# fmadm faulty
```

Use the above command routinely to identify failed services or devices.

Use the following command routinely to identify hardware or device related errors.

```
# fmdump -eV | more
```

Error messages in this log file that describe `vdev.open_failed`, `checksum`, or `io_failure` issues need your attention or they might evolve into actual faults that are displayed with the `fmadm` faulty command.

If the above indicates that a device is failing, then this is a good time to make sure you have a replacement device available.

You can also track additional device errors by using `iostat` command. Use the following syntax to identify a summary of error statistics.

```
# iostat -en
  ---- errors ---
  s/w h/w trn tot device
    0   0   0   0 c0t5000C500335F95E3d0
    0   0   0   0 c0t5000C500335FC3E7d0
    0   0   0   0 c0t5000C500335BA8C3d0
    0  12   0  12 c2t0d0
    0   0   0   0 c0t5000C500335E106Bd0
    0   0   0   0 c0t50015179594B6F11d0
    0   0   0   0 c0t5000C500335DC60Fd0
    0   0   0   0 c0t5000C500335F907Fd0
    0   0   0   0 c0t5000C500335BD117d0
```

In the above output, errors are reported on an internal disk `c2t0d0`. Use the following syntax to display more detailed device errors.

```
# iostat -En
c0t5000C500335F95E3d0 Soft Errors: 0 Hard Errors: 0 Transport Errors: 0
Vendor: SEAGATE  Product: ST930003SSUN300G Revision: 0B70 Serial No: 110672QFSB
Size: 300.00GB <300000000000 bytes>
Media Error: 0 Device Not Ready: 0 No Device: 0 Recoverable: 0
Illegal Request: 0 Predictive Failure Analysis: 0
c0t5000C500335FC3E7d0 Soft Errors: 0 Hard Errors: 0 Transport Errors: 0
Vendor: SEAGATE  Product: ST930003SSUN300G Revision: 0B70 Serial No: 110672TE67
Size: 300.00GB <300000000000 bytes>
Media Error: 0 Device Not Ready: 0 No Device: 0 Recoverable: 0
Illegal Request: 0 Predictive Failure Analysis: 0
c0t5000C500335BA8C3d0 Soft Errors: 0 Hard Errors: 0 Transport Errors: 0
Vendor: SEAGATE  Product: ST930003SSUN300G Revision: 0B70 Serial No: 110672SDF4
Size: 300.00GB <300000000000 bytes>
Media Error: 0 Device Not Ready: 0 No Device: 0 Recoverable: 0
Illegal Request: 0 Predictive Failure Analysis: 0
c2t0d0           Soft Errors: 0 Hard Errors: 12 Transport Errors: 0
Vendor: AMI      Product: Virtual CDROM    Revision: 1.00 Serial No:
Size: 0.00GB <0 bytes>
Media Error: 0 Device Not Ready: 12 No Device: 0 Recoverable: 0
Illegal Request: 2 Predictive Failure Analysis: 0
```

System Reporting of ZFS Error Messages

In addition to persistently tracking errors within the pool, ZFS also displays `syslog` messages when events of interest occur. The following scenarios generate notification events:

- **Device state transition** – If a device becomes `FAULTED`, ZFS logs a message indicating that the fault tolerance of the pool might be compromised. A similar message is sent if the device is later brought online, restoring the pool to health.
- **Data corruption** – If any data corruption is detected, ZFS logs a message describing when and where the corruption was detected. This message is only logged the first time it is detected. Subsequent accesses do not generate a message.
- **Pool failures and device failures** – If a pool failure or a device failure occurs, the fault manager daemon reports these errors through `syslog` messages as well as the `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.

Identifying Problems With ZFS Storage Pools

The following sections describe how to identify and resolve problems with your ZFS file systems or storage pools:

- "Determining If Problems Exist in a ZFS Storage Pool" on page 273
- "Reviewing `zpool status` Output" on page 273
- "System Reporting of ZFS Error Messages" on page 271

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':
2012-11-12.13:01:31 zpool create tank mirror c0t1d0 c0t2d0 c0t3d0
2012-11-12.13:28:10 zfs create tank/eric
2012-11-12.13:37:48 zfs set checksum=off tank/eric
```

  In this output, note that checksums are disabled for the `tank/eric` file system. This configuration is not recommended.

- Identify the errors through the `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 unavailable or missing device and bring it online.
  - Restoring the faulted configuration or corrupted data from a backup.
  - Verifying the recovery by using the `zpool status -x` command.
  - Backing up your restored configuration, if applicable.

This section describes how to interpret `zpool status` output in order to diagnose the type of failures that can occur. Although most of the work is performed automatically by the command, it is important to understand exactly what problems are being identified in order to diagnose the failure. Subsequent sections describe how to repair the various problems that you might encounter.

Determining If Problems Exist in a ZFS Storage Pool

The easiest way to determine if any known problems exist on a system is to use the `zpool status -x` command. This command describes only pools that are exhibiting problems. If no unhealthy pools exist on the system, then the command displays the following:

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

Without the `-x` flag, the command displays the complete status for all pools (or the requested pool, if specified on the command line), even if the pools are otherwise healthy.

For more information about command-line options to the `zpool status` command, see "Querying ZFS Storage Pool Status" on page 81.

Reviewing `zpool status` Output

The complete `zpool status` output looks similar to the following:

```
# zpool status tank
   pool: tank
 state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist for
        the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://www.sun.com/msg/ZFS-8000-2Q
 scan: scrub repaired 0 in 0h3m with 0 errors on Mon Nov 12 15:17:02 2012
config:

        NAME        STATE     READ WRITE CKSUM
```

```
        tank        DEGRADED     0     0     0
          mirror-0  DEGRADED     0     0     0
            c1t1d0  ONLINE       0     0     0
            c1t2d0  UNAVAIL      0     0     0  cannot open

errors: No known data errors
```

This output is described in the following section.

Overall Pool Status Information

This section in the `zpool status` output contains the following fields, some of which are only displayed for pools exhibiting problems:

`pool` Identifies the name of the pool.

`state` Indicates the current health of the pool. This information refers only to the ability of the pool to provide the necessary replication level.

`status` Describes what is wrong with the pool. This field is omitted if no errors are found.

`action` A recommended action for repairing the errors. This field is omitted if no errors are found.

`see` Refers to a knowledge article containing detailed repair information. Online articles are updated more often than this guide can be updated. So, always reference them for the most up-to-date repair procedures. This field is omitted if no errors are found.

`scrub` Identifies the current status of a scrub operation, which might include the date and time that the last scrub was completed, a scrub is in progress, or if no scrub was requested.

`errors` Identifies known data errors or the absence of known data errors.

ZFS Storage Pool Configuration Information

The `config` field in the `zpool status` output describes the configuration of the devices in the pool, as well as their state and any errors generated from the devices. The state can be one of the following: `ONLINE`, `FAULTED`, `DEGRADED`, or `SUSPENDED`. If the state is anything but `ONLINE`, the fault tolerance of the pool has been compromised.

The second section of the configuration output displays error statistics. These errors are divided into three categories:

- `READ` – I/O errors that occurred while issuing a read request
- `WRITE` – I/O errors that occurred while issuing a write request
- `CKSUM` – Checksum errors, meaning that the device returned corrupted data as the result of a read request

These errors can be used to determine if the damage is permanent. A small number of I/O errors might indicate a temporary outage, while a large number might indicate a permanent problem with the device. These errors do not necessarily correspond to data corruption as interpreted by applications. If the device is in a redundant configuration, the devices might show uncorrectable errors, while no errors appear at the mirror or RAID-Z device level. In such cases, ZFS successfully retrieved the good data and attempted to heal the damaged data from existing replicas.

For more information about interpreting these errors, see "Determining the Type of Device Failure" on page 279.

Finally, additional auxiliary information is displayed in the last column of the `zpool status` output. This information expands on the `state` field, aiding in the diagnosis of failures. If a device is `UNAVAIL`, this field indicates whether the device is inaccessible or whether the data on the device is corrupted. If the device is undergoing resilvering, this field displays the current progress.

For information about monitoring resilvering progress, see "Viewing Resilvering Status" on page 288.

ZFS Storage Pool Scrubbing Status

The scrub section of the `zpool status` output describes the current status of any explicit scrubbing operations. This information is distinct from whether any errors are detected on the system, though this information can be used to determine the accuracy of the data corruption error reporting. If the last scrub ended recently, most likely, any known data corruption has been discovered.

Scrub completion messages persist across system reboots.

For more information about the data scrubbing and how to interpret this information, see "Checking ZFS File System Integrity" on page 289.

ZFS Data Corruption Errors

The `zpool status` command also shows whether any known errors are associated with the pool. These errors might have been found during data scrubbing or during normal operation. ZFS maintains a persistent log of all data errors associated with a pool. This log is rotated whenever a complete scrub of the system finishes.

Data corruption errors are always fatal. Their presence indicates that at least one application experienced an I/O error due to corrupt data within the pool. Device errors within a redundant pool do not result in data corruption and are not recorded as part of this log. By default, only the number of errors found is displayed. A complete list of errors and their specifics can be found by using the `zpool status -v` option. For example:

```
# zpool status -v
  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 294.

Resolving ZFS Storage Device Problems

Review the following sections to resolve a missing, removed or faulted device.

Resolving a Missing or Removed Device

If a device cannot be opened, it displays the `UNAVAIL` state in the `zpool status` output. This state means that ZFS was unable to open the device when the pool was first accessed, or the device has since become unavailable. If the device causes a top-level virtual device to be unavailable, then nothing in the pool can be accessed. Otherwise, the fault tolerance of the pool might be compromised. In either case, the device just needs to be reattached to the system to restore normal operations. If you need to replace a device that is `UNAVAIL` because it has failed, see "Replacing a Device in a ZFS Storage Pool" on page 281.

If a device is `UNAVAIL` in a root pool or a mirrored root pool, see the following references:

- **Mirrored root pool disk failed** – "Booting From an Alternate Disk in a Mirrored ZFS Root Pool" on page 150
- **Replacing a disk in a root pool**
  - "How to Replace a Disk in the ZFS Root Pool" on page 157

- **Full root pool disaster recovery** – "Recovering the ZFS Root Pool or Root Pool Snapshots" on page 157

For example, you might see a message similar to the following from `fmd` after a device failure:

```
SUNW-MSG-ID: ZFS-8000-FD, TYPE: Fault, VER: 1, SEVERITY: Major
EVENT-TIME: Thu Jun 24 10:42:36 PDT 2010
PLATFORM: SUNW,Sun-Fire-T200, CSN: -, HOSTNAME: daleks
SOURCE: zfs-diagnosis, REV: 1.0
EVENT-ID: a1fb66d0-cc51-cd14-a835-961c15696fcb
DESC: The number of I/O errors associated with a ZFS device exceeded
acceptable levels.  Refer to http://sun.com/msg/ZFS-8000-FD for more information.
AUTO-RESPONSE: The device has been offlined and marked as faulted.  An attempt
will be made to activate a hot spare if available.
IMPACT: Fault tolerance of the pool may be compromised.
REC-ACTION: Run 'zpool status -x' and replace the bad device.
```

To view more detailed information about the device problem and the resolution, use the `zpool status -x` command. For example:

```
# zpool status -x
  pool: tank
 state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist for
        the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://www.sun.com/msg/ZFS-8000-2Q
  scan: scrub repaired 0 in 0h0m with 0 errors on Tue Sep 27 16:59:07 2011
config:

        NAME        STATE     READ WRITE CKSUM
        tank        DEGRADED     0     0     0
          mirror-0  DEGRADED     0     0     0
            c2t2d0  ONLINE       0     0     0
            c2t1d0  UNAVAIL      0     0     0  cannot open

errors: No known data errors
```

You can see from this output that the `c2t1d0` device is not functioning. If you determine that the device is faulty, replace it.

If necessary, use the `zpool online` command to bring the replaced device online. For example:

```
# zpool online tank c2t1d0
```

Let FMA know that the device has been replaced if the output of the `fmadm faulty` identifies the device error. For example:

```
# fmadm faulty
--------------- ------------------------------------  -------------- ---------
TIME            EVENT-ID                              MSG-ID         SEVERITY
--------------- ------------------------------------  -------------- ---------
Sep 27 16:58:50 e6bb52c3-5fe0-41a1-9ccc-c2f8a6b56100  ZFS-8000-D3    Major
```

```
Host        : neo
Platform    : SUNW,Sun-Fire-T200        Chassis_id  :
Product_sn  :

Fault class : fault.fs.zfs.device
Affects     : zfs://pool=tank/vdev=c75a8336cda03110
                  faulted and taken out of service
Problem in  : zfs://pool=tank/vdev=c75a8336cda03110
                  faulted and taken out of service

Description : A ZFS device failed.  Refer to http://sun.com/msg/ZFS-8000-D3 for
              more information.

Response    : No automated response will occur.

Impact      : Fault tolerance of the pool may be compromised.

Action      : Run 'zpool status -x' and replace the bad device.

# fmadm repaired zfs://pool=tank/vdev=c75a8336cda03110
```

As a last step, confirm that the pool with the replaced device is healthy. For example:

```
# zpool status -x tank
pool 'tank' is healthy
```

Resolving a Removed Device

If a device is completely removed from the system, ZFS detects that the device cannot be opened and places it in the `REMOVED` state. Depending on the data replication level of the pool, this removal might or might not result in the entire pool becoming unavailable. If one disk in a mirrored or RAID-Z device is removed, the pool continues to be accessible. A pool might become `UNAVAIL`, which means no data is accessible until the device is reattached, under the following conditions:

- If 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

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 279.

A pool might be SUSPENDED if device connectivity is compromised. A SUSPENDED pool remains in the wait state until the device issue is resolved. For example:

```
# zpool status cybermen
  pool: cybermen
 state: SUSPENDED
status: One or more devices are unavailable in response to IO failures.
        The pool is suspended.
action: Make sure the affected devices are connected, then run 'zpool clear' or
        'fmadm repaired'.
    see: http://www.sun.com/msg/ZFS-8000-HC
  scan: none requested
config:

        NAME           STATE     READ WRITE CKSUM
        cybermen       UNAVAIL      0    16     0
            c8t3d0     UNAVAIL      0     0     0
            c8t1d0     UNAVAIL      0     0     0
```

After device connectivity is restored, clear the pool or device errors.

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

Notifying ZFS of Device Availability

After a device is reattached to the system, ZFS might or might not automatically detect its availability. If the pool was previously UNAVAIL or SUSPENDED, or the system was rebooted as part of the attach procedure, then ZFS automatically rescans all devices when it tries to open the pool. If the pool was degraded and the device was replaced while the system was running, you must notify ZFS that the device is now available and ready to be reopened by using the zpool online command. For example:

```
# zpool online tank c0t1d0
```

For more information about bringing devices online, see "Bringing a Device Online" on page 69.

Replacing or Repairing a Damaged Device

This section describes how to determine device failure types, clear transient errors, and replacing a device.

Determining the Type of Device Failure

The term *damaged device* is rather vague and can describe a number of possible situations:

- **Bit rot** – Over time, random events such as magnetic influences and cosmic rays can cause bits stored on disk to flip. These events are relatively rare but common enough to cause potential data corruption in large or long-running systems.
- **Misdirected reads or writes** – Firmware bugs or hardware faults can cause reads or writes of entire blocks to reference the incorrect location on disk. These errors are typically transient, though a large number of them might indicate a faulty drive.
- **Administrator error** – Administrators can unknowingly overwrite portions of a disk with bad data (such as copying `/dev/zero` over portions of the disk) that cause permanent corruption on disk. These errors are always transient.
- **Temporary outage**– A disk might become unavailable for a period of time, causing I/Os to fail. This situation is typically associated with network-attached devices, though local disks can experience temporary outages as well. These errors might or might not be transient.
- **Bad or flaky hardware** – This situation is a catch-all for the various problems that faulty hardware exhibits, including consistent I/O errors, faulty transports causing random corruption, or any number of failures. These errors are typically permanent.
- **Offline device** – If a device is offline, it is assumed that the administrator placed the device in this state because it is faulty. The administrator who placed the device in this state can determine if this assumption is accurate.

Determining exactly what is wrong with a device can be a difficult process. The first step is to examine the error counts in the `zpool status` output. For example:

```
# zpool status -v tank
  pool: tank
 state: ONLINE
status: One or more devices has experienced an error resulting in data
        corruption.  Applications may be affected.
action: Restore the file in question if possible.  Otherwise restore the
        entire pool from backup.
   see: http://www.sun.com/msg/ZFS-8000-8A
  scan: scrub in progress since Tue Sep 27 17:12:40 2011
    63.9M scanned out of 528M at 10.7M/s, 0h0m to go
    0 repaired, 12.11% done
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       2     0     0
          mirror-0  ONLINE       2     0     0
            c2t2d0  ONLINE       2     0     0
            c2t1d0  ONLINE       2     0     0

errors: Permanent errors have been detected in the following files:

        /tank/words
```

The errors are divided into I/O errors and checksum errors, both of which might indicate the possible failure type. Typical operation predicts a very small number of errors (just a few over long periods of time). If you are seeing a large number of errors, then this situation probably

indicates impending or complete device failure. However, an administrator error can also result in large error counts. The other source of information is the `syslog` system log. If the log shows a large number of SCSI or Fibre Channel driver messages, then this situation probably indicates serious hardware problems. If no `syslog` messages are generated, then the damage is likely transient.

The goal is to answer the following question:

Is another error likely to occur on this device?

Errors that happen only once are considered *transient* and do not indicate potential failure. Errors that are persistent or severe enough to indicate potential hardware failure are considered *fatal*. The act of determining the type of error is beyond the scope of any automated software currently available with ZFS, and so much must be done manually by you, the administrator. After determination is made, the appropriate action can be taken. Either clear the transient errors or replace the device due to fatal errors. These repair procedures are described in the next sections.

Even if the device errors are considered transient, they still might have caused uncorrectable data errors within the pool. These errors require special repair procedures, even if the underlying device is deemed healthy or otherwise repaired. For more information about repairing data errors, see "Repairing Damaged Data" on page 293.

Clearing Transient Device Errors

If the device errors are deemed transient, in that they are unlikely to affect the future health of the device, they can be safely cleared to indicate that no fatal error occurred. To clear error counters for RAID-Z or mirrored devices, use the `zpool clear` command. For example:

```
# zpool clear tank c1t1d0
```

This syntax clears any device errors and clears any data error counts associated with the device.

To clear all errors associated with the virtual devices in a pool, and to clear any data error counts associated with the pool, use the following syntax:

```
# zpool clear tank
```

For more information about clearing pool errors, see "Clearing Storage Pool Device Errors" on page 70.

Replacing a Device in a ZFS Storage Pool

If device damage is permanent or future permanent damage is likely, the device must be replaced. Whether the device can be replaced depends on the configuration.

- "Determining If a Device Can Be Replaced" on page 282

Determining If a Device Can Be Replaced

If the device to be replaced is part of a redundant configuration, sufficient replicas from which to retrieve good data must exist. For example, if two disks in a four-way mirror are `UNAVAIL`, then either disk can be replaced because healthy replicas are available. However, if two disks in a four-way RAID-Z (`raidz1`) virtual device are `UNAVAIL`, then neither disk can be replaced because insufficient replicas from which to retrieve data exist. If the device is damaged but otherwise online, it can be replaced as long as the pool is not in the `UNAVAIL` state. However, any corrupted data on the device is copied to the new device, unless sufficient replicas with good data exist.

In the following configuration, the `c1t1d0` disk can be replaced, and any data in the pool is copied from the healthy replica, `c1t0d0`:

```
    mirror            DEGRADED
    c1t0d0             ONLINE
    c1t1d0             FAULTED
```

The `c1t0d0` disk can also be replaced, though no self-healing of data can take place because no good replica is available.

In the following configuration, neither `UNAVAIL` disk can be replaced. The `ONLINE` disks cannot be replaced either because the pool itself is `UNAVAIL`.

```
    raidz              FAULTED
    c1t0d0             ONLINE
    c2t0d0             FAULTED
    c3t0d0             FAULTED
    c4t0d0             ONLINE
```

In the following configuration, either top-level disk can be replaced, though any bad data present on the disk is copied to the new disk.

```
c1t0d0         ONLINE
c1t1d0         ONLINE
```

If either disk is `UNAVAIL`, then no replacement can be performed because the pool itself is `UNAVAIL`.

Devices That Cannot be Replaced

If the loss of a device causes the pool to become `UNAVAIL` or the device contains too many data errors in a non-redundant configuration, then the device cannot be safely replaced. Without

sufficient redundancy, no good data with which to heal the damaged device exists. In this case, the only option is to destroy the pool and re-create the configuration, and then to restore your data from a backup copy.

For more information about restoring an entire pool, see "Repairing ZFS Storage Pool-Wide Damage" on page 296.

Replacing a Device in a ZFS Storage Pool

After you have determined that a device can be replaced, use the `zpool replace` command to replace the device. If you are replacing the damaged device with different device, use syntax similar to the following:

```
# zpool replace tank c1t1d0 c2t0d0
```

This command migrates data to the new device from the damaged device or from other devices in the pool if it is in a redundant configuration. When the command is finished, it detaches the damaged device from the configuration, at which point the device can be removed from the system. If you have already removed the device and replaced it with a new device in the same location, use the single device form of the command. For example:

```
# zpool replace tank c1t1d0
```

This command takes an unformatted disk, formats it appropriately, and then resilvers data from the rest of the configuration.

For more information about the `zpool replace` command, see "Replacing Devices in a Storage Pool" on page 70.

EXAMPLE 10–1 Replacing a SATA Disk in a ZFS Storage Pool

The following example shows how to replace a device (`c1t3d0`) in a mirrored storage pool `tank` on a system with SATA devices. To replace the disk `c1t3d0` with a new disk at the same location (`c1t3d0`), then you must unconfigure the disk before you attempt to replace it. If the disk to be replaced is not a SATA disk, then see "Replacing Devices in a Storage Pool" on page 70.

The basic steps follow:

- Take offline the disk (`c1t3d0`)to be replaced. You cannot unconfigure a SATA disk that is currently being used.
- Use the `cfgadm` command to identify the SATA disk (`c1t3d0`) to be unconfigured and unconfigure it. The pool will be degraded with the offline disk in this mirrored configuration, but the pool will continue to be available.
- Physically replace the disk (`c1t3d0`). Ensure that the blue `Ready to Remove` LED is illuminated before you physically remove the `UNAVAIL` drive, if available.

EXAMPLE 10–1 Replacing a SATA Disk in a ZFS Storage Pool *(Continued)*

- Reconfigure the SATA disk (`c1t3d0`).
- Bring the new disk (`c1t3d0`) online.
- Run the `zpool replace` command to replace the disk (`c1t3d0`).

Note – If you had previously set the pool property `autoreplace` to `on`, then any new device, found in the same physical location as a device that previously belonged to the pool is automatically formatted and replaced without using the `zpool replace` command. This feature might not be supported on all hardware.

- If a failed disk is automatically replaced with a hot spare, you might need to detach the hot spare after the failed disk is replaced. For example, if `c2t4d0` is still an active hot spare after the failed disk is replaced, then detach it.

  ```
  # zpool detach tank c2t4d0
  ```

- If FMA is reporting the failed device, then you should clear the device failure.

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

The following example walks through the steps to replace a disk in a ZFS storage pool.

```
# zpool offline tank c1t3d0
# cfgadm | grep c1t3d0
sata1/3::dsk/c1t3d0            disk         connected    configured   ok
# cfgadm -c unconfigure sata1/3
Unconfigure the device at: /devices/pci@0,0/pci1022,7458@2/pci11ab,11ab@1:3
This operation will suspend activity on the SATA device
Continue (yes/no)? yes
# cfgadm | grep sata1/3
sata1/3                        disk         connected    unconfigured ok
<Physically replace the failed disk c1t3d0>
# cfgadm -c configure sata1/3
# cfgadm | grep sata1/3
sata1/3::dsk/c1t3d0            disk         connected    configured   ok
# zpool online tank c1t3d0
# zpool replace tank c1t3d0
# zpool status tank
  pool: tank
 state: ONLINE
 scrub: resilver completed after 0h0m with 0 errors on Tue Feb  2 13:17:32 2010
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            c0t1d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
```

EXAMPLE 10–1 Replacing a SATA Disk in a ZFS Storage Pool *(Continued)*

```
            c0t2d0  ONLINE       0     0     0
            c1t2d0  ONLINE       0     0     0
          mirror-2  ONLINE       0     0     0
            c0t3d0  ONLINE       0     0     0
            c1t3d0  ONLINE       0     0     0

errors: No known data errors
```

Note that the preceding `zpool output` might show both the new and old disks under a *replacing* heading. For example:

```
replacing     DEGRADED     0     0    0
  c1t3d0s0/o  FAULTED      0     0    0
  c1t3d0      ONLINE       0     0    0
```

This text means that the replacement process is in progress and the new disk is being resilvered.

If you are going to replace a disk (`c1t3d0`) with another disk (`c4t3d0`), then you only need to run the `zpool replace` command. For example:

```
# zpool replace tank c1t3d0 c4t3d0
# zpool status
  pool: tank
 state: DEGRADED
 scrub: resilver completed after 0h0m with 0 errors on Tue Feb  2 13:35:41 2010
config:

        NAME             STATE     READ WRITE CKSUM
        tank             DEGRADED     0     0     0
          mirror-0       ONLINE       0     0     0
            c0t1d0       ONLINE       0     0     0
            c1t1d0       ONLINE       0     0     0
          mirror-1       ONLINE       0     0     0
            c0t2d0       ONLINE       0     0     0
            c1t2d0       ONLINE       0     0     0
          mirror-2       DEGRADED     0     0     0
            c0t3d0       ONLINE       0     0     0
            replacing    DEGRADED     0     0     0
              c1t3d0     OFFLINE      0     0     0
              c4t3d0     ONLINE       0     0     0

errors: No known data errors
```

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:
```

EXAMPLE 10–1 Replacing a SATA Disk in a ZFS Storage Pool *(Continued)*

```
        NAME          STATE     READ WRITE CKSUM
        tank          ONLINE       0     0     0
          mirror-0    ONLINE       0     0     0
            c0t1d0    ONLINE       0     0     0
            c1t1d0    ONLINE       0     0     0
          mirror-1    ONLINE       0     0     0
            c0t2d0    ONLINE       0     0     0
            c1t2d0    ONLINE       0     0     0
          mirror-2    ONLINE       0     0     0
            c0t3d0    ONLINE       0     0     0
            c4t3d0    ONLINE       0     0     0
```

EXAMPLE 10–2 Replacing a Failed Log Device

ZFS identifies intent log failures in the `zpool status` command output. Fault Management Architecture (FMA) reports these errors as well. Both ZFS and FMA describe how to recover from an intent log failure.

The following example shows how to recover from a failed log device (`c0t5d0`) in the storage pool (`pool`). The basic steps follow:

- Review the `zpool status -x` output and FMA diagnostic message, described here:

  https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&doctype=REFERENCE&alias=EVENT:ZFS-8000-K4

- Physically replace the failed log device.
- Bring the new log device online.
- Clear the pool's error condition.
- Clear the FMA error.

```
# zpool status -x
  pool: pool
 state: FAULTED
status: One or more of the intent logs could not be read.
        Waiting for adminstrator 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
<Physically replace the failed log device>
```

EXAMPLE 10–2 Replacing a Failed Log Device *(Continued)*

```
# zpool online pool c0t5d0
# zpool clear pool
```

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 adminstrator intervention to fix the faulted pool.
action: Either restore the affected device(s) and run 'zpool online',
        or ignore the intent log records by running 'zpool clear'.
 scrub: none requested
config:

        NAME          STATE     READ WRITE CKSUM
        pool          FAULTED      0     0     0 bad intent log
          mirror-0    ONLINE       0     0     0
            c0t1d0    ONLINE       0     0     0
            c0t4d0    ONLINE       0     0     0
        logs          FAULTED      0     0     0 bad intent log
          c0t5d0      UNAVAIL      0     0     0 cannot open
<Physically replace the failed log device>
# zpool online pool c0t5d0
# zpool clear pool
# fmadm faulty
# fmadm repair zfs://pool=name/vdev=guid
```

You can resolve the log device failure in the following ways:

- Replace or recover the log device. In this example, the log device is `c0t5d0`.
- Bring the log device back online.

  ```
  # zpool online pool c0t5d0
  ```

- Reset the failed log device error condition.

  ```
  # zpool clear pool
  ```

To recover from this error without replacing the failed log device, you can clear the error with the `zpool clear` command. In this scenario, the pool will operate in a degraded mode and the log records will be written to the main pool until the separate log device is replaced.

Consider using mirrored log devices to avoid the log device failure scenario.

Viewing Resilvering Status

The process of replacing a device can take an extended period of time, depending on the size of the device and the amount of data in the pool. The process of moving data from one device to another device is known as *resilvering* and can be monitored by using the `zpool status` command.

Traditional file systems resilver data at the block level. Because ZFS eliminates the artificial layering of the volume manager, it can perform resilvering in a much more powerful and controlled manner. The two main advantages of this feature are as follows:

- ZFS only resilvers the minimum amount of necessary data. In the case of a short outage (as opposed to a complete device replacement), the entire disk can be resilvered in a matter of minutes or seconds. When an entire disk is replaced, the resilvering process takes time proportional to the amount of data used on disk. Replacing a 500-GB disk can take seconds if a pool has only a few gigabytes of used disk space.
- Resilvering is interruptible and safe. If the system loses power or is rebooted, the resilvering process resumes exactly where it left off, without any need for manual intervention.

To view the resilvering process, use the `zpool status` command. For example:

```
# zpool status tank
  pool: tank
 state: DEGRADED
status: One or more devices is currently being resilvered.  The pool will
        continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
 scrub: resilver in progress for 0h0m, 22.60% done, 0h1m to go
config:
        NAME                  STATE     READ WRITE CKSUM
        tank                DEGRADED     0     0     0
          mirror-0          DEGRADED     0     0     0
            replacing-0     DEGRADED     0     0     0
              c1t0d0        UNAVAIL      0     0     0  cannot open
              c2t0d0        ONLINE       0     0     0  85.0M resilvered
            c1t1d0          ONLINE       0     0     0

errors: No known data errors
```

In this example, the disk `c1t0d0` is being replaced by `c2t0d0`. This event is observed in the status output by the presence of the `replacing` virtual device in the configuration. This device is not real, nor is it possible for you to create a pool by using it. The purpose of this device is solely to display the resilvering progress and to identify which device is being replaced.

Note that any pool currently undergoing resilvering is placed in the `ONLINE` or `DEGRADED` state because the pool cannot provide the desired level of redundancy until the resilvering process is completed. Resilvering proceeds as fast as possible, though the I/O is always scheduled with a lower priority than user-requested I/O, to minimize impact on the system. After the resilvering is completed, the configuration reverts to the new, complete, configuration. For example:

```
# zpool status tank
  pool: tank
 state: ONLINE
 scrub: resilver completed after 0h1m with 0 errors on Tue Feb  2 13:54:30 2010
config:

        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            c2t0d0  ONLINE       0     0     0  377M resilvered
            c1t1d0  ONLINE       0     0     0

errors: No known data errors
```

The pool is once again `ONLINE`, and the original failed disk (`c1t0d0`) has been removed from the configuration.

Resolving ZFS File System Problems

Resolving Data Problems in a ZFS Storage Pool

Examples of data problems include the following:

- Transient I/O errors due to a bad disk or controller
- On-disk data corruption due to cosmic rays
- Driver bugs resulting in data being transferred to or from the wrong location
- A user overwriting portions of the physical device by accident

In some cases, these errors are transient, such as a random I/O error while the controller is having problems. In other cases, the damage is permanent, such as on-disk corruption. Even still, whether the damage is permanent does not necessarily indicate that the error is likely to occur again. For example, if you accidentally overwrite part of a disk, no type of hardware failure has occurred, and the device does not need to be replaced. Identifying the exact problem with a device is not an easy task and is covered in more detail in a later section.

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 296.

If your pool is not redundant, the risk that file system corruption can render some or all of your data inaccessible is always present.

File System Validation

In addition to performing file system repair, the `fsck` utility validates that the data on disk has no problems. Traditionally, this task requires unmounting the file system and running the `fsck` utility, possibly taking the system to single-user mode in the process. This scenario results in downtime that is proportional to the size of the file system being checked. Instead of requiring an explicit utility to perform the necessary checking, ZFS provides a mechanism to perform routine checking of all inconsistencies. This feature, known as *scrubbing*, is commonly used in memory and other systems as a method of detecting and preventing errors before they result in a hardware or software failure.

Controlling ZFS Data Scrubbing

Whenever ZFS encounters an error, either through scrubbing or when accessing a file on demand, the error is logged internally so that you can obtain a quick overview of all known errors within the pool.

Explicit ZFS Data Scrubbing

The simplest way to check data integrity is to initiate an explicit scrubbing of all data within the pool. This operation traverses all the data in the pool once and verifies that all blocks can be read. Scrubbing proceeds as fast as the devices allow, though the priority of any I/O remains below that of normal operations. This operation might negatively impact performance, though the pool's data should remain usable and nearly as responsive while the scrubbing occurs. To initiate an explicit scrub, use the `zpool scrub` command. For example:

```
# zpool scrub tank
```

The status of the current scrubbing operation can be displayed by using the `zpool status` command. For example:

```
# zpool status -v tank
  pool: tank
 state: ONLINE
 scrub: scrub completed after 0h7m with 0 errors on Tue Tue Feb  2 12:54:00 2010
config:
        NAME        STATE     READ WRITE CKSUM
        tank        ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            c1t0d0  ONLINE       0     0     0
            c1t1d0  ONLINE       0     0     0

errors: No known data errors
```

Only one active scrubbing operation per pool can occur at one time.

You can stop a scrubbing operation that is in progress by using the `-s` option. For example:

```
# zpool scrub -s tank
```

In most cases, a scrubbing operation to ensure data integrity should continue to completion. Stop a scrubbing operation at your own discretion if system performance is impacted by the operation.

Performing routine scrubbing guarantees continuous I/O to all disks on the system. Routine scrubbing has the side effect of preventing power management from placing idle disks in low-power mode. If the system is generally performing I/O all the time, or if power consumption is not a concern, then this issue can safely be ignored.

For more information about interpreting `zpool status` output, see "Querying ZFS Storage Pool Status" on page 81.

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 288.

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.

Resolving ZFS Space Issues

Review the following sections if you are unsure how ZFS reports file system and pool space accounting. Also review "ZFS Disk Space Accounting" on page 30.

ZFS File System Space Reporting

The `zpool list` and `zfs list` commands are better than the previous `df` and `du` commands for determining your available pool and file system space. With the legacy commands, you cannot easily discern between pool and file system space, nor do the legacy commands account for space that is consumed by descendent file systems or snapshots.

For example, the following root pool (`rpool`) has 5.46 GB allocated and 68.5 GB free.

```
# zpool list rpool
NAME   SIZE  ALLOC   FREE  CAP  DEDUP  HEALTH  ALTROOT
rpool   74G  5.46G  68.5G   7%  1.00x  ONLINE  -
```

If you compare the pool space accounting with the file system space accounting by reviewing the `USED` column of your individual file systems, you can see that the pool space that is reported in `ALLOC` is accounted for in the file systems' `USED` total. For example:

```
# zfs list -r rpool
NAME                      USED  AVAIL  REFER  MOUNTPOINT
rpool                    5.41G  67.4G  74.5K  /rpool
rpool/ROOT               3.37G  67.4G    31K  legacy
rpool/ROOT/solaris       3.37G  67.4G  3.07G  /
rpool/ROOT/solaris/var    302M  67.4G   214M  /var
rpool/dump               1.01G  67.5G  1000M  -
rpool/export             97.5K  67.4G    32K  /rpool/export
rpool/export/home        65.5K  67.4G    32K  /rpool/export/home
rpool/export/home/admin  33.5K  67.4G  33.5K  /rpool/export/home/admin
rpool/swap               1.03G  67.5G  1.00G  -
```

ZFS Storage Pool Space Reporting

The SIZE value that is reported by the `zpool list` command is generally the amount of physical disk space in the pool, but varies depending on the pool's redundancy level. See the examples below. The `zfs list` command lists the usable space that is available to file systems, which is disk space minus ZFS pool redundancy metadata overhead, if any.

- **Non-redundant storage pool** – When a pool is created with one 136-GB disk, the `zpool list` command reports `SIZE` and initial `FREE` values as 136 GB. The initial `AVAIL` space reported by the `zfs list` command is 134 GB, due to a small amount of pool metadata overhead. For example:

```
# zpool create tank c0t6d0
# zpool list tank
NAME   SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT
tank   136G  95.5K   136G     0%  1.00x  ONLINE  -
# zfs list tank
NAME   USED  AVAIL  REFER  MOUNTPOINT
tank    72K   134G    21K  /tank
```

- **Mirrored storage pool** – When a pool is created with two 136-GB disks, `zpool list` command reports `SIZE` as 136 GB and initial `FREE` value as 136 GB. This reporting is referred to as the *deflated* space value. The initial `AVAIL` space reported by the `zfs list` command is 134 GB, due to a small amount of pool metadata overhead. For example:

```
# zpool create tank mirror c0t6d0 c0t7d0
# zpool list tank
NAME   SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT
tank   136G  95.5K   136G     0%  1.00x  ONLINE  -
# zfs list tank
NAME   USED  AVAIL  REFER  MOUNTPOINT
tank    72K   134G    21K  /tank
```

- **RAID-Z storage pool** – When a `raidz2` pool is created with three 136-GB disks, the `zpool list` commands reports `SIZE` as 408 GB and initial `FREE` value as 408 GB. This reporting is referred to as the *inflated* disk space value, which includes redundancy overhead, such as parity information. The initial `AVAIL` space reported by the `zfs list` command is 133 GB, due to the pool redundancy overhead. The space discrepancy between the `zpool list` and the `zfs list` output for a RAID-Z pool is because `zpool list` reports the inflated pool space.

```
# zpool create tank raidz2 c0t6d0 c0t7d0 c0t8d0
# zpool list tank
NAME   SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT
tank   408G   286K   408G     0%  1.00x  ONLINE  -
# zfs list tank
NAME   USED  AVAIL  REFER  MOUNTPOINT
tank  73.2K   133G  20.9K  /tank
```

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 294
- "Repairing a Corrupted File or Directory" on page 295
- "Repairing ZFS Storage Pool-Wide Damage" on page 296

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 289.

Identifying the Type of Data Corruption

By default, the `zpool status` command shows only that corruption has occurred, but not where this corruption occurred. For example:

```
# zpool status monkey
  pool: monkey
 state: ONLINE
status: One or more devices has experienced an error resulting in data
        corruption.  Applications may be affected.
action: Restore the file in question if possible.  Otherwise restore the
        entire pool from backup.
   see: http://www.sun.com/msg/ZFS-8000-8A
 scrub: scrub completed after 0h0m with 8 errors on Tue Jul 13 13:17:32 2010
config:

        NAME        STATE     READ WRITE CKSUM
        monkey      ONLINE       8     0     0
          c1t1d0    ONLINE       2     0     0
          c2t5d0    ONLINE       6     0     0

errors: 8 data errors, use '-v' for a list
```

Each error indicates only that an error occurred at a given point in time. Each error is not necessarily still present on the system. Under normal circumstances, this is the case. Certain temporary outages might result in data corruption that is automatically repaired after the outage ends. A complete scrub of the pool is guaranteed to examine every active block in the pool, so the error log is reset whenever a scrub finishes. If you determine that the errors are no longer present, and you don't want to wait for a scrub to complete, reset all errors in the pool by using the `zpool online` command.

If the data corruption is in pool-wide metadata, the output is slightly different. For example:

```
# zpool status -v morpheus
  pool: morpheus
    id: 1422736890544688191
 state: FAULTED
status: The pool metadata is corrupted.
action: The pool cannot be imported due to damaged devices or data.
   see: http://www.sun.com/msg/ZFS-8000-72
config:

        morpheus    FAULTED   corrupted data
          c1t10d0   ONLINE
```

In the case of pool-wide corruption, the pool is placed into the `FAULTED` state because the pool cannot provide the required redundancy level.

Repairing a Corrupted File or Directory

If a file or directory is corrupted, the system might still function, depending on the type of corruption. Any damage is effectively unrecoverable if no good copies of the data exist on the system. If the data is valuable, you must restore the affected data from backup. Even so, you might be able to recover from this corruption without restoring the entire pool.

If the damage is within a file data block, then the file can be safely removed, thereby clearing the error from the system. Use the `zpool status -v` command to display a list of file names with persistent errors. For example:

```
# zpool status -v
  pool: monkey
 state: ONLINE
status: One or more devices has experienced an error resulting in data
        corruption.  Applications may be affected.
action: Restore the file in question if possible.  Otherwise restore the
        entire pool from backup.
   see: http://www.sun.com/msg/ZFS-8000-8A
 scrub: scrub completed after 0h0m with 8 errors on Tue Jul 13 13:17:32 2010
config:

        NAME        STATE     READ WRITE CKSUM
        monkey      ONLINE       8     0     0
          c1t1d0    ONLINE       2     0     0
          c2t5d0    ONLINE       6     0     0

errors: Permanent errors have been detected in the following files:

/monkey/a.txt
/monkey/bananas/b.txt
/monkey/sub/dir/d.txt
monkey/ghost/e.txt
/monkey/ghost/boo/f.txt
```

The list of file names with persistent errors might be described as follows:

- If the full path to the file is found and the dataset is mounted, the full path to the file is displayed. For example:

  ```
  /monkey/a.txt
  ```

- If the full path to the file is found, but the dataset is not mounted, then the dataset name with no preceding slash (/), followed by the path within the dataset to the file, is displayed. For example:

  ```
  monkey/ghost/e.txt
  ```

- If the object number to a file path cannot be successfully translated, either due to an error or because the object doesn't have a real file path associated with it, as is the case for a `dnode_t`, then the dataset name followed by the object's number is displayed. For example:

  ```
  monkey/dnode:<0x0>
  ```

- If an object in the metaobject set (MOS) is corrupted, then a special tag of `<metadata>`, followed by the object number, is displayed.

If the corruption is within a directory or a file's metadata, the only choice is to move the file elsewhere. You can safely move any file or directory to a less convenient location, allowing the original object to be restored in its place.

Repairing Corrupted Data With Multiple Block References

If a damaged file system has corrupted data with multiple block references, such as snapshots, the `zpool status -v` command cannot display **all** corrupted data paths. The current `zpool status` reporting of corrupted data is limited by the amount of metadata corruption and if any blocks have been reused after the `zpool status` command is executed. Deduplicated blocks makes reporting all corrupted data even more complicated.

If you have corrupted data and the `zpool status -v` command identifies that snapshot data is impacted, then considering running the following command to identify additional corrupted paths:

Repairing ZFS Storage Pool-Wide Damage

If the damage is in pool metadata and that damage prevents the pool from being opened or imported, then the following options are available to you:

- You can attempt to recover the pool by using the `zpool clear -F` command or the `zpool import -F` command. These commands attempt to roll back the last few pool transactions to an operational state. You can use the `zpool status` command to review a damaged pool and the recommended recovery steps. For example:

  ```
  # zpool status
    pool: tpool
   state: FAULTED
  status: The pool metadata is corrupted and the pool cannot be opened.
  ```

```
action: Recovery is possible, but will result in some data loss.
        Returning the pool to its state as of Wed Jul 14 11:44:10 2010
        should correct the problem.  Approximately 5 seconds of data
        must be discarded, irreversibly.  Recovery can be attempted
        by executing 'zpool clear -F tpool'.  A scrub of the pool
        is strongly recommended after recovery.
   see: http://www.sun.com/msg/ZFS-8000-72
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
         tpool      FAULTED      0     0     1  corrupted data
          c1t1d0    ONLINE       0     0     2
          c1t3d0    ONLINE       0     0     4
```

The recovery process as described in the preceding output is to use the following command:

```
# zpool clear -F tpool
```

If you attempt to import a damaged storage pool, you will see messages similar to the following:

```
# zpool import tpool
cannot import 'tpool': I/O error
        Recovery is possible, but will result in some data loss.
        Returning the pool to its state as of Wed Jul 14 11:44:10 2010
        should correct the problem.  Approximately 5 seconds of data
        must be discarded, irreversibly.  Recovery can be attempted
        by executing 'zpool import -F tpool'.  A scrub of the pool
        is strongly recommended after recovery.
```

The recovery process as described in the preceding output is to use the following command:

```
# zpool import -F tpool
Pool tpool returned to its state as of Wed Jul 14 11:44:10 2010.
Discarded approximately 5 seconds of transactions
```

If the damaged pool is in the `zpool.cache` file, the problem is discovered when the system is booted, and the damaged pool is reported in the `zpool status` command. If the pool isn't in the `zpool.cache` file, it won't successfully import or open and you will see the damaged pool messages when you attempt to import the pool.

- You can import a damaged pool in read-only mode. This method enables you to import the pool so that you can access the data. For example:

  ```
  # zpool import -o readonly=on tpool
  ```

  For more information about importing a pool read-only, see "Importing a Pool in Read-Only Mode" on page 97.

- You can import a pool with a missing log device by using the `zpool import -m` command. For more information, see "Importing a Pool With a Missing Log Device" on page 96.

- If the pool cannot be recovered by either pool recovery method, you must restore the pool and all its data from a backup copy. The mechanism you use varies widely depending on the pool configuration and backup strategy. First, save the configuration as displayed by the `zpool status` command so that you can re-create it after the pool is destroyed. Then, use the `zpool destroy -f` command to destroy the pool.

  Also, keep a file describing the layout of the datasets and the various locally set properties somewhere safe, as this information will become inaccessible if the pool is ever rendered inaccessible. With the pool configuration and dataset layout, you can reconstruct your complete configuration after destroying the pool. The data can then be populated by using whatever backup or restoration strategy you use.

Repairing 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 91.

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.

◆ ◆ ◆ CHAPTER 11

Recommended Oracle Solaris ZFS Practices

This chapter describes recommended practices for creating, monitoring, and maintaining your ZFS storage pools and file systems.

The following sections are provided in this chapter:

Recommended Storage Pool Practices

The following sections provide recommended practices for creating and monitoring ZFS storage pools. For information about troubleshooting storage pool problems, see Chapter 10, "Oracle Solaris ZFS Troubleshooting and Pool Recovery."

General System Practices

- Keep system up-to-date with latest Solaris releases and patches
- Confirm that your controller honors cache flush commands so that you know your data is safely written, which is important before changing the pool's devices or splitting a mirrored storage pool. This is generally not a problem on Oracle/Sun hardware, but it is good practice to confirm that your hardware's cache flushing setting is enabled.
- Size memory requirements to actual system workload
  - With a known application memory footprint, such as for a database application, you might cap the ARC size so that the application will not need to reclaim its necessary memory from the ZFS cache.
  - Consider deduplication memory requirements
  - Identify ZFS memory usage with the following command:

```
# mdb -k
> ::memstat
Page Summary                Pages                MB  %Tot
------------     ----------------  ----------------  ----
Kernel                     388117              1516   19%
ZFS File Data               81321               317    4%
Anon                        29928               116    1%
Exec and libs                1359                 5    0%
Page cache                   4890                19    0%
Free (cachelist)             6030                23    0%
Free (freelist)           1581183              6176   76%

Total                     2092828              8175
Physical                  2092827              8175
> $q
```

  - Consider using ECC memory to protect against memory corruption. Silent memory corruption can potentially damage your data.

- Perform regular backups – Although a pool that is created with ZFS redundancy can help reduce down time due to hardware failures, it is not immune to hardware failures, power failures, or disconnected cables. Make sure you backup your data on a regular basis. If your data is important, it should be backed up. Different ways to provide copies of your data are:
  - Regular or daily ZFS snapshots
  - Weekly backups of ZFS pool data. You can use the `zpool split` command to create an exact duplicate of ZFS mirrored storage pool.
  - Monthly backups by using an enterprise-level backup product
- Hardware RAID
  - Consider using JBOD-mode for storage arrays rather than hardware RAID so that ZFS can manage the storage and the redundancy.
  - Use hardware RAID or ZFS redundancy or both
  - Using ZFS redundancy has many benefits – For production environments, configure ZFS so that it can repair data inconsistencies. Use ZFS redundancy, such as RAID-Z, RAID-Z-2, RAID-Z-3, mirror, regardless of the RAID level implemented on the underlying storage device. With such redundancy, faults in the underlying storage device or its connections to the host can be discovered and repaired by ZFS.

  Also see "Pool Creation Practices on Local or Network Attached Storage Arrays" on page 305.
- Crash dumps consume more disk space, generally in the 1/2-3/4 size of physical memory range.

ZFS Storage Pool Creation Practices

The following sections provide general and more specific pool practices.

General Storage Pool Practices

- Use whole disks to enable disk write cache and provide easier maintenance. Creating pools on slices adds complexity to disk management and recovery.
- Use ZFS redundancy so that ZFS can repair data inconsistencies.
  - The following message is displayed when a non-redundant pool is created:

    ```
    # zpool create tank c4t1d0 c4t3d0
    'tank' successfully created, but with no redundancy; failure
    of one device will cause loss of the pool
    ```
  - For mirrored pools, use mirrored disk pairs
  - For RAID-Z pools, group 3-9 disks per VDEV
  - Do not mix RAID-Z and mirrored components within the same pool. These pools are harder to manage and performance might suffer.
- Use hot spares to reduce down time due to hardware failures
- Use similar size disks so that I/O is balanced across devices
  - Smaller LUNs can be expanded to large LUNs
  - Do not expand LUNs from extremely varied sizes, such as 128 MB to 2 TB, to keep optimal metaslab sizes
- Consider creating a small root pool and larger data pools to support faster system recovery
- Recommended minimum pool size is 8 GB. Although the minimum pool size is 64 MB, anything less than 8 GB makes allocating and reclaiming free pool space more difficult.
- Recommended maximum pool size should comfortably fit your workload or data size. Do not try to store more data than you can routinely back up on a regular basis. Otherwise, your data is at risk due to some unforeseen event.

Root Pool Creation Practices

- Create root pools with slices by using the `s*` identifier. Do not use the `p*` identifier. In general, a system's ZFS root pool is created when the system is installed. If you are creating a second root pool or re-creating a root pool, use syntax similar to the following:

  ```
  # zpool create rpool c0t1d0s0
  ```

  Or, create a mirrored root pool. For example:

  ```
  # zpool create rpool mirror c0t1d0s0 c0t2d0s0
  ```
  - The root pool must be created as a mirrored configuration or as a single-disk configuration. Neither a RAID-Z nor a striped configuration is supported. You cannot add additional disks to create multiple mirrored top-level virtual devices by using the `zpool add` command, but you can expand a mirrored virtual device by using the `zpool attach` command.
  - The root pool cannot have a separate log device.

- Pool properties can be set during an AI installation, but the `gzip` compression algorithm is not supported on root pools.
- Do not rename the root pool after it is created by an initial installation. Renaming the root pool might cause an unbootable system.
- Do not create a root pool on a USB stick for a production system because root pool disks are critical for continuous operation, particularly in an enterprise environment. Consider using a system's internal disks for the root pool, or at least use, the same quality disks that you would use for your non-root data. In addition, a USB stick might not be large enough to support a dump volume size that is equivalent to at least 1/2 the size of physical memory.

Non-Root Pool Creation Practices

- Create non-root pools with whole disks by using the `d*` identifier. Do not use the `p*` identifier.
  - ZFS works best without any additional volume management software.
  - For better performance, use individual disks or at least LUNs made up of just a few disks. By providing ZFS with more visibility into the LUNs setup, ZFS is able to make better I/O scheduling decisions.
  - Create redundant pool configurations across multiple controllers to reduce down time due to a controller failure.
  - **Mirrored storage pools** – Consume more disk space but generally perform better with small random reads.

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

  - **RAID-Z storage pools** – Can be created with 3 parity strategies, where parity equals 1 (`raidz`), 2 (`raidz2`), or 3 (`raidz3`). A RAID-Z configuration maximizes disk space and generally performs well when data is written and read in large chunks (128K or more).
    - Consider a single-parity RAID-Z (`raidz`) configuration with 2 VDEVs of 3 disks (2+1) each.

      ```
      # zpool create rzpool raidz1 c1t0d0 c2t0d0 c3t0d0 raidz1 c1t1d0 c2t1d0 c3t1d0
      ```

    - A RAIDZ-2 configuration offers better data availability, and performs similarly to RAID-Z. RAIDZ-2 has significantly better mean time to data loss (MTTDL) than either RAID-Z or 2-way mirrors. Create a double-parity RAID-Z (`raidz2`) configuration at 6 disks (4+2).

      ```
      # zpool create rzpool raidz2 c0t1d0 c1t1d0 c4t1d0 c5t1d0 c6t1d0 c7t1d0
      raidz2 c0t2d0 c1t2d0 c4t2d0 c5t2d0 c6t2d0 c7t2d
      ```

    - A RAIDZ-3 configuration maximizes disk space and offers excellent availability because it can withstand 3 disk failures. Create a triple-parity RAID-Z (`raidz3`) configuration at 9 disks (6+3).

```
# zpool create rzpool raidz3 c0t0d0 c1t0d0 c2t0d0 c3t0d0 c4t0d0
c5t0d0 c6t0d0 c7t0d0 c8t0d0
```

Pool Creation Practices on Local or Network Attached Storage Arrays

Consider the following storage pool practices when creating an a ZFS storage pool on a storage array that is connected locally or remotely.

- If you create an pool on SAN devices and the network connection is slow, the pool's devices might be `UNAVAIL` for a period of time. You need to assess whether the network connection is appropriate for providing your data in a continuous fashion. Also, consider that if you are using SAN devices for your root pool, they might not be available as soon as the system is booted and the root pool's devices might also be `UNAVAIL`.
- Confirm with your array vendor that the disk array is not flushing its cache after a flush write cache request is issued by ZFS.
- Use whole disks, not disk slices, as storage pool devices so that Oracle Solaris ZFS activates the local small disk caches, which get flushed at appropriate times.
- For best performance, create one LUN for each physical disk in the array. Using only one large LUN can cause ZFS to queue up too few read I/O operations to actually drive the storage to optimal performance. Conversely, using many small LUNs could have the effect of swamping the storage with a large number of pending read I/O operations.
- A storage array that uses dynamic (or thin) provisioning software to implement virtual space allocation is not recommended for Oracle Solaris ZFS. When Oracle Solaris ZFS writes the modified data to free space, it writes to the entire LUN. The Oracle Solaris ZFS write process allocates all the virtual space from the storage array's point of view, which negates the benefit of dynamic provisioning.

  Consider that dynamic provisioning software might be unnecessary when using ZFS:

  - You can expand a LUN in an existing ZFS storage pool and it will use the new space.
  - Similar behavior works when a smaller LUN is replaced with a larger LUN.
  - If you assess the storage needs for your pool and create the pool with smaller LUNs that equal the required storage needs, then you can always expand the LUNs to a larger size if you need more space.
- If the array can present individual devices (JBOD-mode), then consider creating redundant ZFS storage pools (mirror or RAID-Z) on this type of array so that ZFS can report and correct data inconsistencies.

Pool Creation Practices for an Oracle Database

Consider the following storage pool practices when creating an Oracle database.

- Use a mirrored pool or hardware RAID for pools
- RAID-Z pools are generally not recommended for random read workloads
- Create a small separate pool with a separate log device for database redo logs

- Create a small separate pool for the archive log

For more information, see the following white paper:

http://blogs.oracle.com/storage/entry/new_white_paper_configuring_oracle

Using ZFS Storage Pools in VirtualBox

- Virtual Box is configured to ignore cache flush commands from the underlying storage by default. This means that in the event of a system crash or a hardware failure, data could be lost.
- Enable cache flushing on Virtual Box by issuing the following command:

```
VBoxManage setextradata <VM_NAME> "VBoxInternal/Devices/<type>/0/LUN#<n>/Config/IgnoreFlush" 0
```

  - <VM_NAME> is the name of the virtual machine
  - <type> is the controller type, either piix3ide (if you're using the usual IDE virtual controller) or ahci, if you're using a SATA controller
  - <n> is the disk number

Storage Pool Practices for Performance

- Keep pool capacity below 80% for best performance
- Mirrored pools are recommended over RAID-Z pools for random read/write workloads
- Separate log devices
  - Recommended to improve synchronous write performance
  - With a high synchronous write load, prevents fragmentation of writing many log blocks in the main pool
- Separate cache devices are recommended to improve read performance
- Scrub/resilver - A very large RAID-Z pool with lots of devices will have longer scrub and resilver times
- Pool performance is slow – Use the `zpool status` command to rule out any hardware problems that are causing pool performance problems. If no problems show up in the `zpool status` command, use the `fmdump` command to display hardware faults or use the `fmdump -eV` command to review any hardware errors that have not yet resulted in a reported fault.

ZFS Storage Pool Maintenance and Monitoring Practices

- Make sure that pool capacity is below 80% for best performance.

Pool performance can degrade when a pool is very full and file systems are updated frequently, such as on a busy mail server. Full pools might cause a performance penalty, but no other issues. If the primary workload is immutable files, then keep pool in the 95-96% utilization range. Even with mostly static content in the 95-96% range, write, read, and resilvering performance might suffer.

- Monitor pool and file system space to make sure that they are not full.
- Consider using ZFS quotas and reservations to make sure file system space does not exceed 80% pool capacity.

- Monitor pool health
  - Redundant pools, monitor pool with `zpool status` and `fmdump` on a weekly basis
  - Non-redundant pools, monitor pool with `zpool status` and `fmdump` on a biweekly basis
- Run `zpool scrub` on a regular basis to identify data integrity problems.
  - If you have consumer-quality drives, consider a weekly scrubbing schedule.
  - If you have datacenter-quality drives, consider a monthly scrubbing schedule.
  - You should also run a scrub prior to replacing devices or temporarily reducing a pool's redundancy to ensure that all devices are currently operational.
- Monitoring pool or device failures - Use `zpool status` as described below. Also use `fmdump` or `fmdump -eV` to see if any device faults or errors have occurred.
  - Redundant pools, monitor pool health with `zpool status` and `fmdump` on a weekly basis
  - Non-redundant pools, monitor pool health with `zpool status` and `fmdump` on a biweekly basis
- Pool device is `UNAVAIL` or `OFFLINE` – If a pool device is not available, then check to see if the device is listed in the `format` command output. If the device is not listed in the `format` output, then it will not be visible to ZFS.

  If a pool device has `UNAVAIL` or `OFFLINE`, then this generally means that the device has failed or cable has disconnected, or some other hardware problem, such as a bad cable or bad controller has caused the device to be inaccessible.
- Monitor your storage pool space – Use the `zpool list` command and the `zfs list` command to identify how much disk is consumed by file system data. ZFS snapshots can consume disk space and if they are not listed by the `zfs list` command, they can silently consume disk space. Use the `zfs list -t` snapshot command to identify disk space that is consumed by snapshots.

Recommended File System Practices

The following sections describe recommended file system practices.

File System Creation Practices

The following sections describe ZFS file system creation practices.

- Create one file system per user for home directories
- Consider using file system quotas and reservations to manage and reserve disk space for important file systems
- Consider using user and group quotas to manage disk space in an environment with many users
- Use ZFS property inheritance to apply properties to many descendent file systems

File System Creation Practices for an Oracle Database

Consider the following file system practices when creating an Oracle database.

- Match the ZFS `recordsize` property to the Oracle `db_block_size`.
- Create database table and index file systems in main database pool, using an 8 KB `recordsize` and the default `primarycache` value.
- Create temp data and undo table space file systems in the main database pool, using default `recordsize` and `primarycache` values.
- Create archive log file system in the archive pool, enabling compression and default `recordsize` value and `primarycache` set to `metadata`.

For more information, see the following white paper:

`http://blogs.oracle.com/storage/entry/new_white_paper_configuring_oracle`

Monitoring ZFS File System Practices

You should monitor your ZFS file systems to ensure they are available and to identify space consumption issues.

- Weekly, monitor file system space availability with the `zpool list` and `zfs list` commands rather than the `du` and `df` commands because legacy commands do not account for space that is consumed by descendent file systems or snapshots.

  For more information, see "Resolving ZFS Space Issues" on page 292.
- Display file system space consumption by using the `zfs list -o space` command.
- File system space can be unknowingly consumed by snapshots. You can display all dataset information by using the following syntax:

```
# zfs list -t all
```

- A separate `/var` file system is created automatically when a system is installed, but you should set a quota and reservation on this file system to ensure that it does not unknowingly consume root pool space.
- In addition, you can use the `fsstat` command to display file operation activity of ZFS file systems. Activity can be reported by mount point or by file system type. The following example shows general ZFS file system activity:

```
# fsstat /
 new  name   name  attr  attr lookup rddir  read read  write write
 file remov  chng   get   set    ops   ops   ops bytes   ops bytes
  832   589   286  837K 3.23K  2.62M 20.8K 1.15M 1.75G 62.5K  348M /
```

- Backups
  - Keep file system snapshots
  - Consider enterprise-level software for weekly and monthly backups
  - Store root pool snapshots on a remote system for bare metal recovery

◆ ◆ ◆ APPENDIX A

Oracle Solaris ZFS Version Descriptions

This appendix describes available ZFS versions, features of each version, and the Solaris OS that provides the ZFS version and feature.

The following sections are provided in this appendix:

Overview of ZFS Versions

New ZFS pool and file system features are introduced and accessible by using a specific ZFS version that is available in Solaris releases. You can use the `zpool upgrade` or `zfs upgrade` to identify whether a pool or file system is at lower version than the currently running Solaris release provides. You can also use these commands to upgrade your pool and file system versions.

For information about using the `zpool upgrade` and `zfs upgrade` commands, see "Upgrading ZFS File Systems" on page 199 and "Upgrading ZFS Storage Pools" on page 100.

ZFS Pool Versions

The following table provides a list of ZFS pool versions that are available in the Oracle Solaris release.

Version	Solaris 10	Description
1	Solaris 10 6/06	Initial ZFS version
2	Solaris 10 11/06	Ditto blocks (replicated metadata)

Version	Solaris 10	Description
3	Solaris 10 11/06	Hot spares and double parity RAID-Z
4	Solaris 10 8/07	`zpool history`
5	Solaris 10 10/08	`gzip` compression algorithm
6	Solaris 10 10/08	`bootfs` pool property
7	Solaris 10 10/08	Separate intent log devices
8	Solaris 10 10/08	Delegated administration
9	Solaris 10 10/08	`refquota` and `refreservation` properties
10	Solaris 10 5/09	Cache devices
11	Solaris 10 10/09	Improved scrub performance
12	Solaris 10 10/09	Snapshot properties
13	Solaris 10 10/09	`snapused` property
14	Solaris 10 10/09	`aclinherit passthrough-x` property
15	Solaris 10 10/09	user and group space accounting
16	Solaris 10 9/10	`stmf` property support
17	Solaris 10 9/10	Triple-parity RAID-Z
18	Solaris 10 9/10	Snapshot user holds
19	Solaris 10 9/10	Log device removal
20	Solaris 10 9/10	`Compression using zle` (zero-length encoding)
21	Solaris 10 9/10	Reserved
22	Solaris 10 9/10	Received properties
23	Solaris 10 8/11	Slim ZIL
24	Solaris 10 8/11	System attributes
25	Solaris 10 8/11	Improved scrub stats
26	Solaris 10 8/11	Improved snapshot deletion performance
27	Solaris 10 8/11	Improved snapshot creation performance
28	Solaris 10 8/11	Multiple vdev replacements
29	Solaris 10 8/11	RAID-Z/mirror hybrid allocator
30	Solaris 10 1/13	Reserved

Version	Solaris 10	Description
31	Solaris 10 1/13	Improved `zfs list` performance
32	Solaris 10 1/13	One MB blocksize

ZFS File System Versions

The following table lists the ZFS file system versions that are available in the Oracle Solaris release. Keep in mind that a feature that is available in a specific file system version requires a specific pool version.

Version	Solaris 10	Description
1	Solaris 10 6/06	Initial ZFS file system version
2	Solaris 10 10/08	Enhanced directory entries
3	Solaris 10 10/08	Case insensitivity and file system unique identifier (FUID)
4	Solaris 10 10/09	`userquota` and `groupquota` properties
5	Solaris 10 8/11	System attributes

Index

B

C

D

E

M

N

O

P

Q

R

S

T

U

V

W

X

Z