Chapter W22

ZFS Administration and Use

Objectives:

* To describe and give an overview of the Zettabyte File System (ZFS)
* To illustrate the use of the **zpool** and **zfs** commands in the context of system administration
* To give a brief ZFS commands and operations reference encyclopedia
* To give a complete example of file system backups using the **zfs snapshot** command in a Bash Shell script
* To cover the commands and primitives  
  **zpool**, **zfs**

W22.1 Introduction

This chapter will detail the hands-on mechanics of a modern Linux file system commonly known as the Zettabyte File System (ZFS).

ZFS has the following attributes: it corrects itself at the bit level, it is very secure, it is a volume manager, and it provides its own real-time file backup system procedures. It is sometimes called a “user” file system because ordinary privileged users of the system have control over ZFS operations through the use of typed-in commands. We show examples of using the two most important ZFS commands, **zpool** and **zfs**.

At the time of the writing of this book, ZFS was a kernel-loadable module in the Debian-family (Ubuntu 16.04, and Linux Mint 18.2, Sonya), but special instructions for building it on Debian 9.1 Stretch are given in Appendix A in the printed book. Additionally, instructions for building ZFS on CentOS 7.4 are given in Appendix A as well.

In all systems, ZFS could not easily or reliably be made the root file system at installation, or using some post-installation procedure. We do provide instructions in Section 4.7 that illustrate how to make ZFS the root file system on Ubuntu 18.04. We also pose a problem at the end of the chapter that deals with utilizing the mirroring capability of ZFS, when it is the root file system. Traditionally, the root file system that you boot into in Linux is an EXT4 file system. To download and install ZFS on Ubuntu 16.04 and Linux Mint 18.2 , we used the following commands:

$ **sudo apt-get update**

$ **sudo apt-get upgrade**

$ **sudo apt-get install zfsutils-linux**

In order to do anything in this chapter, you must first use these three commands to download and install ZFS on your Ubuntu 16.04 and Linux Mint 18.2 system.

**\*\*\*Warning\*\*\*:** This warning applies, at the time of the writing of this book, to Debian 9.1, Ubuntu 16.04, Linux Mint 18.2, and CentOS 7.4. It is important to realize that if you want to create and use ZFS on your Linux system other than for the practice Examples W22.1 and W22.2 in Section W22.2, and Example W22.8 (applicable to Ubuntu 18.04) in Section W22.4.7, you must have an additional storage device, or devices, attached to your hardware. This could be additional internally-mounted SATA hard drives, an externally-mounted USB or ESATA hard drive in an enclosure, or USB thumbdrives. We have found that if you attempt to create zpools on your system or boot disk, this will render that disk unbootable! See the specific advisory on this given at the beginning of Section W22.2.

A drawback of the following practical worked examples in this chapter is that ideally, in a commercial production situation where you would use ZFS to full advantage on Linux or UNIX systems, the hardware would have Error Correction Code (ECC) system memory only. That type of memory is available exclusively on server-class machines, but is also available for desktop systems. For learning purposes, we do not assume you are using ECC memory.

As discussed in Section 19.2.1 in the printed book, we highly recommend a disk storage model for desktop computers capable of effectively using two or more hard drives, that puts the operating system on a single disk (a Solid State Drive or SSD), and user data files on a second, larger capacity hard drive. What that storage model allows you to accomplish is keeping the system operable, maintained, and upgradable on its own discrete storage device; user data is securely archived on another, possibly redundant, device or devices. That model implementation, and its effectiveness for archiving user data files, is made possible by what we show in this chapter. We also show techniques for implementing this storage model when your computer can only have a single hard drive mounted internally. This model strictly conforms to the warning we previously gave you, and must accommodate the ECC memory drawback also mentioned.

This chapter assumes that you have already looked at Chapter 17, Section 17.4.3, “Adding a New Disk to the System.”

W22.1.1 **zpool** and **zfs** Command Syntax

The following are the general syntax forms for the **zpool** and **zfs** commands. For a more complete description of these two important commands, see the man pages for zfs and zpool on your Linux system (after you have installed ZFS according to the instructions given above).

|  |
| --- |
| **SYNTAX**  **zpool subcommand [options] [option arguments] [command arguments]**  **Purpose:** To create and manage storage pools of virtual devices such as disk drives  **Commonly used options/features:**  zpool create name vdev Creates a new pool with name on the specified vdev  zpool create –o copies=2 nameCreates a new pool name with the property copies set to 2  zpool destroy name Destroys, or removes, a pool name  zpool list name Lists storage space and health of pool name  zpool scrub name Verifies that the checksums on pool name are correct  zpool status name Displays the status of pool name |

|  |
| --- |
| **SYNTAX**  **zfs subcommand [options] [option arguments] [command arguments]**  **Purpose:** To create and manage datasets or file systems mapped to devices such as disk drives  **Commonly used options/features:**  zfs create name Creates a dataset with name  zfs create –o copies=2 name Creates a dataset name with the property copies set to 2  zfs destroy name Destroys, or removes, a dataset name  zfs list Lists all datasets  zfs rollback name Returns dataset name to a previous snapshot state |

W22.1.2 ZFS Terminology

The following describes the basic terminology used throughout this chapter, and as it applies to ZFS practice in general as well:

*Boot environment*: A boot environment is a bootable environment consisting of a ZFS root file system and, optionally, other file systems mounted underneath it. Exactly one boot environment can be active at a time. Heed the Warning given in Section W22.1!

*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 ZFS snapshot.

*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 name space. Datasets are identified using the following format:

pool/path[@snapshot]

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

path A slash-delimited pathname for the dataset component

snapshot An optional component that identifies a snapshot of a dataset

*Deduplication*:Data deduplication is a method of reducing storage capacity needs by eliminating redundant data. Only one unique instance of the data is actually retained on storage media. Redundant data is replaced with a pointer to the unique data copy.

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

*Mirror*: A vdev that stores identical copies of data on two or more disks, in a variety of ways defined by Redundant Array of Independent Disks (RAID) specifications. If any disk in a mirror fails, any other disk in that mirror can provide the same data, according to those RAID specifications.

*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.

*RAIDZ*: A virtual device that stores data and parity on multiple disks, using the RAID specifications.

*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.

*Slice*: A disk partition created with partitioning software.

*Snapshot*:A read-only copy of a file system or volume at a given point in time.

*Vdev (virtual device)*:A whole disk, a disk partition, a file, or a collection of the previous, usually all of the same type. There is no performance penalty for using disk partitions rather than entire disks.The write cache is disabled for partitions, thus incurring a performance penalty. Using files as vdevs is discouraged, except for experimenting and testing purposes as we do in this chapter for beginners! A collection of vdevs is a mirror.

*Volume*: A dataset that represents a block device. For example, you can create a ZFS volume as a swap device.

W22.1.3 How ZFS Works

Create zpool mapped to vdev > Create ZFS file system(s) on zpool > Add files

Simply stated, you create a named zpool first, which at the time it is created is mapped or associated with a vdev, such as a hard disk drive. Then you create one or more file systems in that zpool. Then you add files to the file system(s). Finally, you manage the files, file systems, pools, and vdevs using the appropriate ZFS commands.

Working with ZFS in Linux is a matter of efficiently and easily managing zpools that have vdevs “mapped” to them, and then managing file systems, and their files, in those zpools.

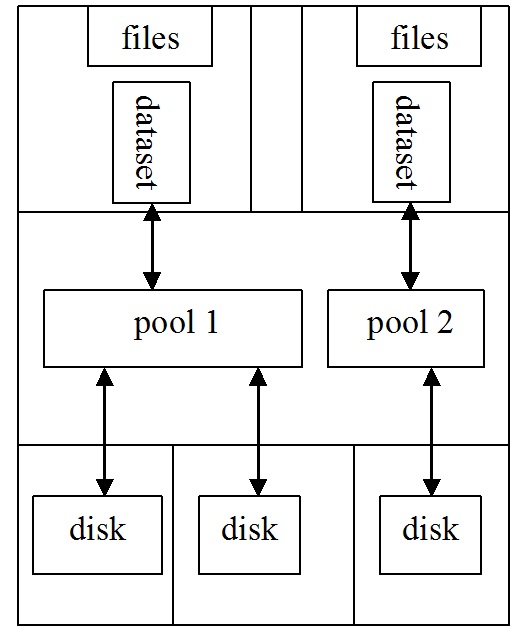


Figure W22.1 ZFS Components

Figure W22.1 shows this relationship between files, datasets (file systems), pools, and disks. **Pool 1** has two disks mapped to it, and a dataset with a number of files in it. **Pool 2** has a single disk mapped to it, and has a dataset in it. This layering of files and datasets, pools, and disks is the basic structure of ZFS.

W22.1.4 Important ZFS Concepts

Some very important points have to be made here:

1. Only one zpool can be mapped or associated with any vdev. So if you want to create a zpool on a physical hard disk or one of its slices, no other existing active zpool can be mapped to that vdev!

2. There are seven types of vdev in ZFS:

* Disk (default): The physical hard drives in your system, usually the whole drive or primary slice
* File: The absolute path of preallocated files/images, similar to the Section W22.2.2, Example W22.1
* Mirror: Standard software RAID1 mirror
* Raidz1/2/3: Nonstandard distributed parity-based software RAID levels
* Spare: Hard drives marked as a *hot spare* for ZFS software RAID
* Cache: Device used for a level-2 adaptive read cache (L2ARC)
* Log: A separate log (SLOG) called the ZFS intent log (ZIL)

3. Unlike a traditional file system, where the mount point of the file system begins at a particular logical drive letter, the default mount point for a zpool is root (**/**).

This is how the path to a file named **test.txt** appears when it is in the zpool named **data1** on the file system **sarwar**:

**/data1/sarwar/test.txt**

Here’s how the path to a file named **test.txt** appears on a traditional file system:

**C:\Users\Robert\Desktop\test.txt**

When you want a ZFS file system to expand onto more than one disk, for example, you add more disks to the zpool.

4. A zpool can be enlarged by adding more devices, but it cannot be shrunk (at least not at this time)!

W22.2 Example ZFS Pools and File systems: Using the zpool and zfs Commands

Advisory:

As stated in Section W22.1, it is important to realize that if you want to create and use ZFS on a Linux system other than for the practice Examples W22.1 and W22.2 we show in this section, you must have an additional storage device, or devices, attached to your hardware. This could be additional internally-mounted hard drives, an externally-mounted USB or ESATA hard drive in an enclosure, or USB thumbdrives.

Per the Warning given in Section W22.1, we have found that if you attempt to create zpools on your Linux system disk or boot disk, this will render that disk unbootable!

For example, the following command line session illustrates what you should not do:

$ **sudo zpool create test /dev/sda**  
[sudo] password for bob: **QQQ**  
invalid vdev specification  
use '-f' to override the following errors:  
/dev/sda does not contain an EFI label but it may contain partition information in the MBR.  
$ **sudo zpool create test /dev/sda1**  
invalid vdev specification  
use '-f' to override the following errors:  
/dev/sda1 contains a filesystem of type 'ext4'  
$

If you force the override in either of the above two commands, you will render your Linux system disk unbootable.

In this section, we first reiterate a set of simple methods given in Chapter 19, to allow you to quickly determine the logical device names of disks attached to your Linux system. We then present five examples that will give you some basic experience in using the **zpool** and **zfs** commands.

W22.2.1 A Quick and Easy Way to Find Out the Logical Device Names of Disks Actually Installed on Your System

These techniques, which were presented in Chapter 19, are worth repeating here in preparation for using the **zpool** and **zfs** commands.

It is important to know how to determine, in a very quick and easy manner, what the currently installed logical device names of disk drives actually attached and useable on your system are. What we mean by “attached and useable” is that the disk drive is properly connected and recognized by the system and has a device driver that the system can use to communicate with it.

The simple method that follows shows how to determine what disk drives are attached and useable on your system, and what the logical device names of those and any others you might want to add to your system are.

Change your current working directory to **/dev**. Type **ls**. Hard drives, for example, show up in the **ls** listing as **sda**, **sdb**, and so on. The full path to the first slice, or partition, on one of these disks is specified as **/dev/sda1**. A USB device, like a thumbdrive, would show up in the **ls** listing as **sdc**, and the full path to the first slice on it would be **/dev/sdc1**.

You can use the Gparted Partition Editor, a GUI app available on your Linux system, to view all usable hard disks or USB thumbdrives, and their logical device names. In the Examples in this chapter, we use Gparted to partition, and put file systems on disks that we are going to use as vdevs for ZFS.

W22.2.2 Basic ZFS Examples

In this section we present five instructive, introductory examples of how to work with ZFS. It is expected that for you to get the full benefit from them, you do them and their attendant In-Chapter Exercises in the order presented.

**Example W22.1: The zpool Command: Using Files Instead of Disks as Vdevs**

Objective: To introduce the **zpool** command, implemented on files instead of disks, and to show forms of ZFS pool creation and mirroring.

Introduction: A vdev, as defined previously, can be a physical device such as a disk drive, a file, a single slice on a hard disk drive, or a collection of devices. Before beginning to use ZFS on physical devices, and to practice using ZFS on an existing file system instead of deploying ZFS on actual SATA hard disk drives, we will create and manipulate files with the important ZFS commands.

Also, if you do not have a second hard disk drive in your computer, you can do this example to gain an appreciation of what ZFS is.

In case you want to use four real disks mounted and partitioned in this preliminary introductory example, make a note of the full path to their device names (e.g.**/dev/sdb1** ). You will be destroying all the partition information and data on these disks, so be sure they’re not needed!

If you make a mistake anywhere along the way, you can always start over by executing the cleanup steps shown at the end of the example and begin again.

Prerequisites: Installation of ZFS on your Linux system, as shown in Appendix A of the printed book.

Procedures: Follow the steps in the order shown to complete this example.

1. Become root, and then create four 128 MB files as follows (the files must be a minimum of 64 MB in size):

$ **sudo -i**

[sudo] password for bob: **QQQ**

~ #

~# **truncate --size 128m /home/bob/disk1**

~# **truncate --size 128m /home/bob/disk2**

~# **truncate --size 128m /home/bob/disk3**

~# **truncate --size 128m /home/bob/disk4**

Check the /home/bob directory with the following command:

~ # **ls -lh /home/bob**

total 48K

...

-rw-r--r-- 1 root root 128M Nov 22 12:36 disk1

-rw-r--r-- 1 root root 128M Nov 22 12:37 disk2

-rw-r--r-- 1 root root 128M Nov 22 12:37 disk3

-rw-r--r-- 1 root root 128M Nov 22 12:37 disk4

Output truncated…

In this example, we initially create and use files to simulate disks on an already existing file system, and we named them **disk1**, **disk2**, **disk3**, and **disk4** to enhance that illusion.

Also, it is assumed in the example code that the current working directory is **/home/bob** unless otherwise noted.

2. Before creating new pools you should check for existing pools to avoid confusing them with the example pools we create here. You can check what pools exist with **zpool list**:

~ # **zpool list**

no pools available

~ #

3. Pools are created using **zpool create**. We can create a single disk pool using a file as follows (you must use the absolute path to the file), and check the zpools that now exist:

**~ # zpool create data /home/bob/disk1**

~ # **zpool list**

NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALTROOT

data 112M 61K 112M - 0% 0% 1.00x ONLINE -

4. Now we will create an actual file in the new pool, check its size, and get a zpool listing of it:

~ # **truncate --size 32m /data/data20file**

~ # **ls -lh /data/data20file**

-rw-r--r-- 1 root root 32M Nov 22 12:43 /data/data20file

~ # **zpool list**

NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALTROOT

data 112M 58K 112M - 0% 0% 1.00x ONLINE -

5. We will now destroy the pool data with zpool destroy, and check on zpools now available:

**~ # zpool destroy data**

~ # **sudo zpool list**

No pools avaulable

6. Creating a Mirrored Pool with Files

A pool composed of a single disk doesn’t offer any redundancy! One way of providing protection against physical disk failure is to use a mirrored pair of disks in a pool:

**~ # zpool create data2 mirror /home/bob/disk1 /home/bob/disk2**

~ # **zpool list**

NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALTROOT

data2 112M 61K 112M - 0% 0% 1.00x ONLINE -

7. To get more information about the pool **data2**, we use **zpool status**:

**~ # zpool status data2**

pool: data2

state: ONLINE

scan: none requested

config:

NAME STATE READ WRITE CKSUM

data2 ONLINE 0 0 0

mirror-0 ONLINE 0 0 0

/home/bob/disk1 ONLINE 0 0 0

/home/bob/disk2 ONLINE 0 0 0

errors: No known data errors

8. Create a file in the **data2** pool.

**~ # truncate --size 32m /data2/data2file**

Note the change in the pool after we have added a file to it, using the following command:

**~ # zpool list**

NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALTROOT

data2 112M 58K 112M - 0% 0% 1.00x ONLINE -

A fraction of the disk has been used, but more importantly the data is now stored redundantly over two disks.

9. Let’s test that redundancy by overwriting the first “disk” label with random data. If you are using real hard disks, you could physically remove the disk from the computer.

**~ # dd if=/dev/random of=/home/bob/disk1 bs=512 count=1**

0+1 records in

0+1 records out

113 bytes copied, 0.000695301 s, 163 kB/s

10. ZFS automatically checks for errors when it reads/writes files, but we can force a check with the **zfs scrub** command.

**~ # zpool scrub data2**

11. Let’s check the status of the pool:

**~ # zpool status**

pool: data2

state: DEGRADED

status: One or more devices could not be used because the label is missing or

invalid. Sufficient replicas exist for the pool to continue

functioning in a degraded state.

action: Replace the device using 'zpool replace'.

see: http://zfsonlinux.org/msg/ZFS-8000-4J

scan: scrub repaired 0 in 0h0m with 0 errors on Tue Nov 22 13:02:19 2016

config:

NAME STATE READ WRITE CKSUM

data2 DEGRADED 0 0 0

mirror-0 DEGRADED 0 0 0

/home/bob/disk1 UNAVAIL 0 0 0 corrupted data

/home/bob/disk2 ONLINE 0 0 0

errors: No known data errors

12. The disk we used dd on is showing as UNAVAIL (unavailable) with corrupted data, but no data errors are reported for the pool as a whole, and we can still read and write to the pool:

**~ # truncate --size 32m /data2/data2file2**

**~ # ls -l /data2/**

total 1

-rw-r--r-- 1 root root 33554432 Nov 22 12:57 data2file

-rw-r--r-- 1 root root 33554432 Nov 22 13:05 data2file2

13. To maintain redundancy we should replace the broken disk with another. If you are using a physical disk you can use the **zpool replace** command (the zpool man page has details). However, in this file-based example we will just remove the disk file from the mirror and recreate it.

Devices are detached with **zpool detach**:

**~ # zpool detach data2 /home/bob/disk1**

14. Let’s check the status of the pool:

**~ # zpool status data2**

pool: data2

state: ONLINE

scan: scrub repaired 0 in 0h0m with 0 errors on Tue Nov 22 13:02:19 2016

config:

NAME STATE READ WRITE CKSUM

data2 ONLINE 0 0 0

/home/bob/disk2 ONLINE 0 0 0

errors: No known data errors

15. Let’s remove the disk, and then try to replace it, to simulate a failure:

**~ # rm /home/bob/disk1**

**~ # truncate --size 128m /home/bob/disk1**

16. In order to replace it in the mirror, we need to do the following. To attach another device we specify an existing device in the mirror to attach it to with **zpool attach**:

**~ # zpool attach data2 /home/bob/disk2 /home/bob/disk1**

17. Check the status of the pool:

**~ # zpool status data2**

pool: data2

state: ONLINE

scan: resilvered 73K in 0h0m with 0 errors on Tue Nov 22 13:13:25 2016

config:

NAME STATE READ WRITE CKSUM

data2 ONLINE 0 0 0

mirror-0 ONLINE 0 0 0

/home/bob/disk2 ONLINE 0 0 0

/home/bob/disk1 ONLINE 0 0 0

errors: No known data errors

18. Adding to a Mirrored Pool

A very critical systems administration procedure accomplished by ZFS is to add disks to a pool without taking it offline. Let’s double the size of our **data2** pool:

**~ # zpool list**

NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALTROOT

data2 112M 89.5K 112M - 0% 0% 1.00x ONLINE -

19. We can use the zpool add command to add disks to the existing pool.

**~ # zpool add data2 mirror /home/bob/disk3 /home/bob/disk4**

**~ # zpool list**

NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALTROOT

data2 224M 95.5K 224M - 0% 0% 1.00x ONLINE -

20. The file systems within the pool are always available. If we look at the status now, it shows the pool consists of two mirrors:

**~ # zpool status data2**

pool: data2

state: ONLINE

scan: resilvered 73K in 0h0m with 0 errors on Tue Nov 22 13:13:25 2016

config:

NAME STATE READ WRITE CKSUM

data2 ONLINE 0 0 0

mirror-0 ONLINE 0 0 0

/home/bob/disk2 ONLINE 0 0 0

/home/bob/disk1 ONLINE 0 0 0

mirror-1 ONLINE 0 0 0

/home/bob/disk3 ONLINE 0 0 0

/home/bob/disk4 ONLINE 0 0 0

errors: No known data errors

21. We can see where the data is currently written in our pool using **zpool iostat -v**:

**~ # zpool iostat -v data2**

capacity operations bandwidth

pool alloc free read write read write

------------------- ----- ----- ----- ----- ----- -----

data2 95.5K 224M 0 0 16 300

mirror 88K 112M 0 1 64 1.10K

/home/bob/disk2 - - 0 0 406 3.26K

/home/bob/disk1 - - 0 0 56 6.61K

mirror 7.50K 112M 0 0 0 74

/home/bob/disk3 - - 0 0 108 10.5K

/home/bob/disk4 - - 0 0 108 10.5K

------------------- ----- ----- ----- ----- ----- -----

22. All the data is currently written on the first mirror pair and none on the second. This makes sense, as the second pair of disks was added after the data was written. If we write some new data to the pool, the new mirror will be used:

**~ # truncate --size 64m /data2/data2file3**

**~ # zpool iostat -v data2**

capacity operations bandwidth

pool alloc free read write read write

------------------- ----- ----- ----- ----- ----- -----

data2 83.5K 224M 0 0 15 312

mirror 68.5K 112M 0 0 48 918

/home/bob/disk2 - - 0 0 374 3.03K

/home/bob/disk1 - - 0 0 42 5.05K

mirror 15K 112M 0 0 0 108

/home/bob/disk3 - - 0 0 66 6.50K

/home/bob/disk4 - - 0 0 66 6.50K

------------------- ----- ----- ----- ----- ----- -----

23. We see how a little more of the data has been written to the new mirror than to the old: ZFS tries to make the best use of all the resources in the pool. Now do these in-chapter exercises, and then continue onto the next step.

In-Chapter Exercise W22.1

If you have not already done so, execute all of the steps of Example W22.1 using proper commands and pathnames.

In-Chapter Exercise W22.2

In Example W22.1, step 4, what is the pathname to **datafile20**?

In-Chapter Exercise W22.3

If you were to use a text editor like emacs to create a text file named **text1.txt** in the file system named **data**, how would you designate the complete pathname to that text file?

In-Chapter Exercise W22.4

In Example W22.1, after step 6 was executed correctly, and you created a text file with emacs in the **data2** file system, would the pathnames to the two mirrored versions of that text file be different? In other words, could you edit each one of them separately by designating different pathnames to them?

In-Chapter Exercise W22.5

In Example W22.1, step 19, could you add a single disk into the mirrored **data2** zpool, instead of the two disks specified?

In-Chapter Exercise W22.6

In Example W22.1, step 20, are the mirrors named **mirror-0** and **mirror-1** mirrors of each other?

W22. To clean up after doing our work, let’s delete everything we created in this example.

From the root directory, destroy the **data2** file system and its files.

**~ # zfs destroy -r data2**

25. Next, destroy the **data2** zpool.

**~ # zpool destroy data2**

26. Finally, destroy the disk simulation files, and leave root.

**~ # rm /home/bob/disk\***

~ # exit

logout

$

Conclusion: We can use the **zpool** command and its **create** subcommand to associate or map file systems to vdevs, whether the vdev is a file itself or a disk drive.

**Example W22.2: The zfs Command: Send and Receive, Snapshot**

Objectives: The following is a complete example of using the commands **zfs send** and **zfs receive**. Its objective is to show how to create a file system with **zfs** and work with file systems.

Introduction: In the example, we backup a file system with an incremental update, from one file system to another, on the same zpool and vdev.

As with Example W22.1, this example creates a file in your home directory that *emulates* a vdev, so you don’t have to have a second hard disk available! This is the easiest, most cost-effective technique, and the best for practicing and developing your basic skills with ZFS.

What the example does: It backs up a file system named **data** in the zpool named **sender** to another file system named **backup** in the same pool. The data file system contains a file we create named **test.txt**. It uses the **snapshot** subcommand of the **zfs** command to achieve this.

If you make a mistake anywhere along the way, you can always start over by executing the cleanup steps shown at the end of the example and begin again.

Prerequisites: Installation of ZFS on your Linux system, as shown in Appendix A of the printed book, having completed Example W22.1.

Procedure: To accomplish the objectives of this example, do the following steps in the order presented.

1. Become root and then list the zpools that exist currently on the system. On Debian-family Linux systems, this can be done with the **sudo -i** command, on CentOS 7 with the **su** command

$ **sudo -i**

[sudo] password for bob: **QQQ**

~ # **zpool list**

no pools available

~ #

2. Create the vdev as a file.

~ # **truncate --size 100m /home/bob/master**

3. Create a zpool in that vdev named **sender**.

~ # **zpool create sender /home/bob/master**

4. Create a ZFS file system, named data, in the sender zpool.

~ # **zfs create sender/data**

5. Create a test file in the sender/data ZFS file system.

~ # **echo "created: 09:58" > /sender/data/test.txt**

6. Create a snapshot of the ZFS file system named sender/data.

**~ # zfs snapshot sender/data@1**

7. Examine the location where the snapshot has been saved. First, use the zfs list command with the snapshot command argument as follows:

~ # **zfs list -t snapshot**

NAME USED AVAIL REFER MOUNTPOINT

sender/data@1 0 - 1 9.5K -

8. By default the snapshot location is hidden. To unhide it, use the **zfs set** command.

~ # **zfs set snapdir=visible sender/data**

9. See what the contents of the data file system are, using the ls -la command as follows:

~ # ls -la /sender/data

total 2

drwxr-xr-x 2 root root 3 Nov 22 10:52 .

drwxr-xr-x 3 root root 3 Nov 22 10:50 ..

-rw-r--r-- 1 root root 15 Nov 22 10:52 test.txt

dr-xr-xr-x 1 root root 0 Nov 22 10:50 .zfs

10. The snapshot directory that contains the first snapshot is under **.zfs**, as shown. So let’s change to the directory that contains it, and use ls –la to see what is in that directory.

~ # **cd /sender/data/.zfs/snapshot/1**

# ls -la

total 2

drwxr-xr-x 2 root root 3 Nov 22 10:52 .

dr-xr-xr-x 2 root root 2 Nov 22 10:54 ..

-rw-r--r-- 1 root root 15 Nov 22 10:52 test.txt

The file **test.txt** in this directory is a “frozen” picture of what was contained in the **/sender/data** file system at the time we did step 6.

11. Return to your home directory.

# **cd**

~ #

12.Create a ZFS file system named **backup** in the **sender** zpool.

~ # **zfs create sender/backup**

13. Send the snapshot to the backup file system.

~ # **zfs send sender/data@1 | zfs receive -F sender/backup**

After this command executes, the file **test.txt** is in the backup file system.

14. Set the sender/backup file system to read only to prevent data corruption. Make sure to do this before accessing anything in the **sender/backup** file system.

~ # **zfs set readonly=on sender/backup**

15. Now we will make some changes in the original file. Become root and use the echo command to update the original **test.txt** file to simulate changes in the data file system.

# **echo "`date`" >> /sender/data/test.txt**

16. Create a second snapshot of **sender/data**.

**~** # **zfs snapshot sender/data@2**

17. Send the differences. You may get an error message saying that the destination has been modified if you did not set the **sender/data** file system to read only three commands previously in step 14.

~ # **zfs send -i sender/data@1 sender/data@2 | zfs receive sender/backup**

18. *Optional Step:* At this point you could use ssh to send the file system to another zpool on another machine, such as **backup\_server** (where you need to supply the IP address and have root privileges on that system), as follows:

~ # **zfs send sender/data@1 | ssh backup\_server zfs receive backup/data@1**

Example W22.6 is a complete example of achieving this optional approach to using send and receive between two different systems.

19. Now let’s take a look at what is in the second snapshot directory.

~ # **cd /sender/data/.zfs/snapshot/2**

# **ls -la**

total 2

drwxr-xr-x 2 root root 3 Nov 22 10:52 .

dr-xr-xr-x 2 root root 2 Nov 22 11:02 ..

-rw-r--r-- 1 root root 44 Nov 22 11:01 test.txt

20. Let’s look at the contents of the **test.txt** file.

# **more test.txt**

created: 09:58

Tue Nov 22 11:01:48 PST 2016

21. Now let’s compare what is in the second snapshot directory to the **sender** and **backup** file systems.

# **cd**

~ # **cd /sender/data**

# **ls**

test.txt

# **more test.txt**

created: 09:58

Tue Nov 22 11:01:48 PST 2016

# **cd ..**

# **cd backup**

# **ls**

test.txt

**# more test.txt**

created: 09:58

Tue Nov 22 11:01:48 PST 2016

22. Return to your home directory.

# **cd**

~ #

Now do these In-Chapter Exercises, and then continue onto the next step.

In-Chapter Exercise W22.7

If you have not already done so, execute all the steps of Example W22.2 using proper commands and pathnames.

In-Chapter Exercise W22.8

What commands would you use to make the current working directory the one that contains the second snapshot?

In-Chapter Exercise W22.9

What are the contents of the first snapshot file, **test.txt**?

In-Chapter Exercise W22.10

Redo Example W22.2 using two different zpools named **source** and **target**. Create a file system on a pool source named **origin**, and create a file system on a pool target named **destination**. Instead of using the echo command to create the file **test.txt** in the **source/origin** file system, use your favorite text editor, like emacs. Then create a couple of snapshots of **origin** and **destination**, making some changes in **test.txt** with emacs. Finally, use the techniques shown in Example W22.2 to verify that the snapshots indeed contain the changes you made with emacs in **test.txt**.

23. To clean up after doing our work, let’s delete everything we created in this example. Notice that destroying the datasets destroys the snapshots.

From the root directory, destroy the **backup** file system and its data.

~ # **zfs destroy -r sender/backup**

W22. Next, destroy the **data** file system and its data.

~ # **zfs destroy -r sender/data**

25. Next, destroy the **sender** zpool.

~ # **zpool destroy sender**

26. Finally, delete the disk simulation file and exit root.

~ # **rm /home/bob/master**

~ # **exit**

logout

$

Conclusion: You can use zfs send/receive as a backup mechanism, either locally between two hard disks attached to the system, or between systems over a network.

**Example W22.3: Mirroring of Physical User Data Hard Disks**

Objectives: To mirror a physical user data hard disk onto another hard disk that is added to the system sometime after the initial build of the system.

Introduction: The following example illustrates one of the most important disk maintenance procedures a user can perform: the mirroring of a physical device using the **zpool attach** command. In the example, we mirror a user data disk onto another hard disk of equal size. This is a very important system administration task, because if one of the hard disks fails, you have an exact duplicate of it attached to your machine which archives the user data.

The resilvering operation for the 1 TB hard disk in the example, with very little data on it, takes about five to ten minutes.

This is important because if one of the hard disks with vital user data on it fails, you have an exact usable duplicate of it attached to your machine, that you can operate with and not lose service. This original disk has all of your user datasets on it. You can then replace the failed disk and in a few simple ZFS steps, restoring the integrity and redundancy of your system without taking it offline! This is an operation that a commercial system administrator would perform, as well as an ordinary, single-user of a desktop Linux system.

This example accomplishes the same thing that the Clonezilla Live program illustrated in Chapter 19 does, but with one significant difference. Whereas Clonezilla Live “clones” an entire disk, including the boot sectors, at only one discrete instant in time, ZFS **zpool attach** applied to a mirrored pair creates a constantly mirrored “clone” of a user data disk and all datasets on it. This is critical, because you never know when your user data disk is going to fail. You can have any number of backup schemes in place, as shown in Chapter 19 and this chapter as well, to save user datasets with rolling, incremental backups using **rsync** or **zfs snapshot**. But this example’s methodology allows you to constantly have an exact clone of your user data disk available as long as it is running and active.

Of course, a more advanced and necessarily complex technique for doing what is shown here would involve multiple disks, including a separate system disk, higher-level ZRAID dataset disks, and even the disk that holds the ZIL.

If you make a mistake anywhere along the way, you can always start over by executing the cleanup step shown at the end of the example and begin again. Depending on how far you go in the procedure, you can also reformat the new disk with the gparted program and restart from the beginning.

Prerequisites:

1. Installation of ZFS on your Linux system, as shown in Appendix A in the printed book.

2. That you have previously completed Example W22.2.

3. That you have correctly connected and put a single primary partition on a third hard disk on your system (either internally-mounted on the SATA bus, or externally-mounted on the USB bus), using gparted, or a similar facility such as **fdisk**. Additionally, we have found that unmounting this third hard disk, with the **umount** command, is necessary before executing the steps of the procedures below. It will still show up in a listing of /dev, even though it’s unmounted.

4. That you have previously determined the logical device name and the full path to the new hard disk using the methods “A Quick and Easy Way to Find Out the Logical Device Names of Disks Actually Installed on Your System” from Section W22.2.1. The complete logical device names of our primary user data hard disk (not our root, or system, disk!), and the second hard disk drive we want to mirror it to are **/dev/sdb1** and **/dev/sdc1**. On your system they may not be exactly the same, but they will be very similar.

5. You have already created **rpool** on your user data disk’s primary partition (in our case, /dev/sdb1). And there are already ZFS filesystems, and user data existing on this user data disk.

If you have not fulfilled this Prerequisite, do so now before proceeding with this example!

6. The size in bytes of **rpool,** the name of the original user data disk zpool, is smaller than or equal to the size of the primary partition on the new hard disk drive you will mirror **rpool** to.

7. That you have done Example 17.6 in Chapter 17, Section 17.4.3 “Adding a New Hard Disk to the System”.

Procedure: Do the following steps in the order shown to meet the objectives.

1. List the zpools currently on the system with the **zpool list** command. **rpool** must show up in this listing, be online, and healthy (according to ZFS!)

2. Use the **attach** sub-command of **zpool** to create a mirror of your original user data disk. Be sure to specify the complete pathname to the devices, as shown.

$ **sudo** **zpool attach rpool /dev/sdb1 /dev/sdc1**

Make sure to wait until resilver is done before rebooting.

3. While the resilvering is happening, check the status of the resilvering process.

$ **sudo** **zpool status**

pool: rpool

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.

Run 'zpool status -v' to see device specific details.

scan: resilver in progress since Fri Oct 27 06:30:26 2017

32.4G scanned

9.98G resilvered at 102M/s, 30.80% done, 0h3m to go

config:

NAME STATE READ WRITE CKSUM

rpool DEGRADED 0 0 0

mirror-0 DEGRADED 0 0 0

sdb1 ONLINE 0 0 0

sdc1 DEGRADED 0 0 0 (resilvering)

errors: No known data errors

4. Check the status of the resilvering again. This time, it is done and **rpool** is back online as a two-disk mirror.

$ **sudo** **zpool status**

pool: rpool

state: ONLINE

scan: resilvered 32.4G in 0h9m with 0 errors on Fri Oct 27 06:40:15 2017

config:

NAME STATE READ WRITE CKSUM

rpool ONLINE 0 0 0

mirror-0 ONLINE 0 0 0

sdb1 ONLINE 0 0 0

sdc1 ONLINE 0 0 0

errors: No known data errors

$

In-Chapter Exercise W22.11

To test the usability of the new hard disks in the mirrored pair, shut down your machine gracefully. Then disconnect and remove the first disk from the machine. Finally, reboot the machine with only the root, or system disk, and the new user data hard disk in the system. What is the status of the pool you attached the second hard disk to as a mirror after a successful reboot? After doing this exercise, you may replace the original disks and boot into it normally. Do the remaining step at your discretion.

5. If you want to retain this two-disk mirror, stop. If you want to detach the second hard disk from the pool, thus destroying the mirror, do the following:

$ **sudo** **zpool detach rpool /dev/sdc**

$

Conclusion: You have created a post-installation two-disk mirror of a user data disk, with another appropriately sized hard disk you have added to the system. In addition, you have used the gparted program, or a similar program, to prepare that new disk for ZFS mirroring.

This example has also shown that to get maximum control over the whole range of sub-commands and options of the **zpool** and **zfs** commands, and to be able to integrate that control with other Linux commands, the command line is the most inclusive, efficient, and reliable method of working with ZFS.

**Example W22.4: Creating a ZFS Pool on an Externally-Mounted USB Hard Drive**

Objectives: To use zpool create to create a ZFS pool on an externally-mounted hard drive drive.

Introduction:

Question: Where would this example be most useful?

Answer: On a laptop or desktop computer which could only accommodate a single hard drive internally, but has USB 3.0 capabilities for mounting hard drives.

The following example uses the most essential command in ZFS, the zpool command, with it’s sub-command create, applied to an externally-mounted hard drive on your Linux system.

The real power of ZFS as a file system/volume manager/file backup system is most obvious in multi-disk computer systems, where redundancy for files and directories can be established across two or more different physical disks. In addition, this redundancy can be extended to single-disk systems with the use of the zfs set property command, as shown in Problem W22.9 at the end of the chapter.

Our Linux Mint 18.2 system automatically mounted an external USB hard drive that was in its own enclosure.

If the only thing you want to do is use the USB hard drive to transfer or archive files (e.g., text files, C program source code, LibreOffice documents, and so on) to and from the computer, you do not have to use the procedures of this example to accomplish that.

If you make a mistake anywhere along the way, you can always start over by executing the cleanup step shown at the end of the example and begin again.

Prerequisites:

0. You have installed ZFS on your Linux system, as shown in Appendix A of the printed book.

1. That you have an expendable externally-mounted USB hard drive that is usable on your system, and perhaps already has one partition and no data on it. More specifically, if the USB hard drive is by default formatted to FAT32, then there is a high probability that it will automatically mount and show up as an icon on your Linux system desktop.

2. That there is no pool named **backup** (or any other pool) already defined on the primary partition of the USB hard drive, and there is no pool named **test1** on your Linux Mint system. To find this out, type zpool list on the command line when the USB thumbdrive has been inserted and automounts.

3. That you can determine the logical device name and the full path to it using the methods “A Quick and Easy Way to Find Out the Logical Device Names of Disks Actually Installed on Your System” from Section W22.2.1. The full path to our USB hard drive is **/dev/sdb** , and the path to the first partition on it is **/dev/sdb1**. This may not be the same designation as the USB hard drive on your system, but you can substitute your device designation for them.

4. That you have done Example 19.6 in Chapter 19, Section 19.4.3 “Adding a New Hard Disk to the System”.

Procedures: To complete the objectives of this example, do the following steps in the order presented.

1. Check the names of the existing zpools on your system with **sudo zpool list** to verify that you do not have existing zpools on your system with the names “backup” or “test1”.

2. Plug the hard drive into a USB port on the computer. It should automatically mount and appear as an icon on your desktop in Linux Mint. Determine the logical device name of the USB thumbdrive, as shown in Section W22.2.1. Then, clear any ZFS information off the externally-mounted hard drive using the following command:

$ **sudo zpool labelclear -f /dev/sdb**

If for some reason, this hard disk had been used in a ZFS pool before, we need to clear the ZFS information from the hard disk so that the old ZFS information does not conflict with what we are going to do in this Example.

3. Use the gparted program on your Linux system to-

a. Unmount the externally-mounted hard disk. The icon for it disappears from the desktop. Then delete all partitions on it.

b. In the resulting unallocated space, create a gpt partition table on it.

c. Add a new single primary partition with an ext4 file system on it.

d. Quit the gparted program.

e. At this point, a desktop icon for the USB drive should not appear!

But, it must still show up in an **ls** listing of /dev to be useable for this example.

f. If the external hard drive is not recognized in /dev after the above steps have been done, you need to use another hard drive.

4. Create the pool, which we will name **backup**, on the USB hard drive, with the following command:

$ **sudo** **zpool create -f backup /dev/sdb1**

When the shell prompt reappears, the pool has been built.

Possible error messages you might get from executing the above command:

\* The USB hard drive is still mounted, you have to unmount it with the **umount /dev/sdb1** command before the above **zpool create** command will work.

\* Not designating the proper full path name to the USB thumbdrive as a vdev. To create a zpool you must designate the full path to the drive.

\* In practice, you should designate the entire disk, not just a single partition on it, as the vdev for zpool create. The most effective use of ZFS on disks is when you use entire disk drives as vdevs.

4. List the zpools on your system.

$ **sudo** **zpool list**

NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALTROOT

backup 149G 64K 149G - 0% 0% 1.00x ONLINE -

You should see your new zpool **backup** listed as **backup**, along with any other pools that have been created previously. The listing for your new pool should also show the amount of disk space used on the drive, and the total available disk available on it.

5. Check the listing using the **df -h** command.

$ **sudo** **df -h**

Filesystem Size Used Avail Use% Mounted on

udev 2.9G 0 2.9G 0% /dev

tmpfs 585M 1.2M 583M 1% /run

/dev/sda1 105G 48G 52G 49% /

tmpfs 2.9G 1.5M 2.9G 1% /dev/shm

tmpfs 5.0M 0 5.0M 0% /run/lock

tmpfs 2.9G 0 2.9G 0% /sys/fs/cgroup

cgmfs 100K 0 100K 0% /run/cgmanager/fs

tmpfs 585M 64K 585M 1% /run/user/1000

backup 145G 0 145G 0% /backup

As shown in the above listing, you should see the new pool you created shown as **backup**, right at the bottom of all the other pools created on your system by default at installation. You can also see that it is mounted at /backup. So to add files onto it, you would put them in /backup.

If you want to retain this USB hard drive as a ZFS vdev permanently, along with the pool you created on it in the preceding steps so that you can do Problem W22.9 at the end of the chapter, do not continue onto the next step. If you want to reuse this USB hard drive for another purpose, and do not want to do Problem W22.9 at the end of the chapter, proceed to the next step.

6. Now destroy the pool and exit as superuser.

$ **sudo** **zpool destroy backup**

$

Be aware that destroying the zpool will then allow you to remove the USB-mounted drive from the machine. If you want to use that drive for another purpose, you will have to reformat it to a useful format, such as EXT4 or FAT32, with the gparted program, and erase the data on it.

Conclusion: You have created and destroyed a zpool on an externally-mounted USB hard drive. If you did not destroy the zpool, you can now have datasets placed on it, and directories and files. All of the ZFS commands in this chapter can be used to manage zpools and datasets on this drive. You can also use this drive as a backup for user data files. See Problems W22.9 and W22.10 at the end of the chapter.

**Example W22.5 : Creating a ZFS Pool on a USB thumbdrive**

Objectives: To create a zpool and a ZFS dataset in that pool on a USB thumbdrive.

Introduction:

Question: Where would this example be most useful?

Answer: On a laptop or desktop computer which could only accommodate a single hard drive internally, and on which you want to learn the basics of ZFS.

The following example uses the most essential command in ZFS, the **zpool** command, with the subcommand **create** applied to a USB thumbdrive on a Linux system. The real power of ZFS can be harnessed to common, readily available hardware vdevs that are available on the system, by an ordinary user.

If the only thing you want to do is use a USB thumbdrive to transfer files (such as text files, C program source code, LibreOffice documents, and so on) to and from the computer, you do not have to use the procedures of this example to accomplish that.

If you make a mistake anywhere along the way, you can always start over by executing the cleanup step shown at the end of the example, and then begin again.

Prerequisites:

0. You have installed ZFS on your Linux system, as shown in Appendix A in the printed book.

1. That you have an expendable USB thumbdrive that is usable on your system, and already has only one partition and no data you want to keep on it. More specifically, if the USB thumbdrive is by default formatted to FAT32 (which most commercially-availble thumbdrives are), then there is a high probability that it will automatically mount and show up as an icon on your desktop in Linux system.

2. That there is no pool named **test3** already defined on the primary partition of the USB thumbdrive, and there is no pool named **test3** on your Linux system. To find this out, in a terminal window type **zpool list** on the command line, after the USB thumbdrive has been inserted and auto-mounts.

3. That you can determine the logical device name and the full path to the thumbdrive using the methods “A Quick and Easy Way to Find Out the Logical Device Names of Disks Actually Installed on Your System” from Section W22.2.1. The full path to our USB thumbdrive is /dev/sdc , and the path to the first partition on it is /dev/sdc1. This may not be the same designations as the USB thumbdrive on your system, but you can substitute your logical device designations for them.

4. That you have done Example 19.6 in Chapter 19, Section 19.4.3 “Adding a New Hard Disk to the System”.

Procedure: All commands are shown in **bold** type and are typed at the command line.

1. Plug the thumbdrive into a USB port on the computer. It should automatically mount and appear as an icon on your Linux system desktop. Determine the logical device name of the USB thumbdrive, as shown in Section W22.2.1. Then, clear any ZFS information off the thumbdrive using the following command:

$ **sudo zpool labelclear -f /dev/sdb**

This step is necessary because, if for some reason, this thumbdrive had been used in a ZFS pool before, you need to clear the ZFS meta-data information from the thumbdrive so that the old ZFS meta-data information does not conflict with what we are going to do in this example.

2. Unmount the thumbdrive using the **umount** command, in our case **sudo umount /dev/sdb1**. At this point, the desktop icon for the USB thumbdrive disappears!

It must still show up in an **ls** listing of /dev to be useable. If the thumbdrive is not recognized in /dev after the above step has been done, you need to use another thumbdrive!

3. You can now create a zpool named **test3** on the primary partition on the thumbdrive you worked on in step 2.. To create a zpool on the thumbdrive, use the following command:

$ **sudo** **zpool create -f test3 /dev/sdb1**

We included the **-f** option on **zpool create** to force creation of a ZFS file system on partition 1 of the thumbdrive, effectively eradicating whatever file system was on it.

4. You can now create a file system named **newfilesystem** on the zpool **test3**, using the following command:

$ **sudo** **zfs create test3/newfilesystem**

5. Obtain a listing of the datasets on your computer, using the following command. Several datasets may be listed, for example the backup dataset we created in Example 23.5. Notice in the listing that the mount point of each dataset is given in the last column. This is the path that you designate to the datasets’ directory. If you put files in that directory, they are part of that file system.

$ **sudo** **zfs list**

NAME USED AVAIL REFER MOUNTPOINT

test3 81.5K 7.02G 19K /test3

test3/newfilesystem 19K 7.02G 19K /test3/newfilesystem

To put files in the ZFS dataset **newfilesystem**, put them in the directory **/test3/newfilesystem**.

6. If you want to continue using this thumbdrive as an additional ZFS drive on your system, stop here. Be aware that if you remove the thumbdrive without taking it offline according to ZFS protocols, unexpected results will occur. If you want to use the thumbdrive for other purposes, continue.

The following three steps allow you to undo everything you have done in this example (except, of course, the deletion of any data that was on the thumbdrive before you began this example!).

7. To begin, first destroy the file system.

$ **sudo** **zfs destroy test3/newfilesystem**

8. Then destroy the zpool.

$ **sudo** **zpool destroy test3**

9. Pull the thumbdrive out of the USB port. To reuse the thumbdrive for non-ZFS, purely data storage purposes, you will need to use gparted or other facilities to delete the partition on it, and put a new partition table and partition on it. This effectively destroys the ZFS partition we created in this example.

Conclusion: You have created and destroyed a zpool and a ZFS dataset on a USB thumbdrive that you attached to your Linux system. If you did not destroy the zpool and dataset on it, you can now create additional datasets on it and place files in those datasets.

All of the ZFS commands in this chapter can be used to manage zpools and datasets on this drive. You can also use this drive as a backup for user data files, and use all of the ZFS functionality to do backups.

W22.3 ZFS Commands and Operations

The following section is an abbreviated encyclopedia, or reference manual, that illustrates many uses of the two important ZFS commands, **zfs** and **zpool**. It shows the kinds of operations you can perform with those two commands and with their options and subcommands. In order to get a complete listing, with examples, of the commands, subcommands, and options, consult the man pages for **zfs** or **zpool** on your system.

We first present a summary of the command categories and basic definitions for **zpool** and **zfs**. We then show several examples of **zpool** and **zfs** command, subcommand, option, and command argument usage.

This section assumes that you have done at least one or more of the previous examples in Section W22.2 to get a feel for what ZFS can do.

All sample code you type on the command line is shown in **bold** text and is always followed by pressing **<Enter>** on the keyboard. Comments specific to a command, operation, or term usually appear after the item of interest.

W22.3.1 Command Categories and Basic Definitions

1. Directories and Files

Where error messages appear: **/var/adm/messages** , **console**

2. ZFS States

DEGRADED One or more top-level devices is in the degraded state because they have become offline. Sufficient replicas exist to keep functioning.

FAULTED One or more top-level devices is in the faulted state because they have become offline. Insufficient replicas exist to keep functioning.

OFFLINE The device was explicitly taken offline by the **zpool offline** command.

ONLINE The device is online and functioning.

REMOVED The device was physically removed while the system was running.

UNAVAIL The device could not be opened.

3. Scrubbing and Resilvering

Scrubbing: Examines all data to discover hardware faults or disk failures. Only one scrub may be running at one time, and you can manually scrub.

Resilvering: The same concept as rebuilding or resyncing data on to new disks into an array. The smart thing resilvering does is it does not rebuild the whole disk, only the data that is required (the data blocks not the free blocks), thus reducing the time to *resync* a disk. Resilvering is automatic when you replace disks and so on. If a scrub is already running, it is suspended until the resilvering has finished, then the scrubbing will continue.

4. ZFS Devices and Device Terminology

Disk: A physical disk drive.

File: The absolute path of preallocated files/images.

Mirror: Standard RAID1 mirror.

Raidz1/2/3: Nonstandard distributed parity-based software RAID levels. Basically, if a power failure occurs in the middle of a write then you have the data plus the parity, or you don’t. Also, ZFS supports *self-healing*, which means that if it cannot read a bad block it will reconstruct it using the parity, and repair or indicate that this block should not be used.

Raidz1: 3, 5, 9 disks

Raidz2: 4, 6, 8, 10, 18 disks

Raidz3: 5, 7, 11, 19 disks

The more parity bits, the longer it takes to resilver an array. Standard mirroring does not have the problem of creating the parity, so it is quicker in resilvering. Raidz is more like RAID3 than RAID5, but does use parity to protect from disk failures.

Raidz/Raidz1: A minimum of three devices (one parity disk); you can suffer a one-disk loss.

Raidz2: A minimum of four devices (two parity disks); you can suffer a two-disk loss.

Raidz3: A minimum of five devices (three parity disks); you can suffer a three-disk loss.

Spare: hard drives marked as *hot spare* for ZFS RAID. By default, hot spares are not used in a disk failure; you must turn on the *autoreplace* feature.

5. Cache

A zfs cache caches both the least recently used (LRU) and least frequently used (LFU) block requests; the cache device uses level-2 adaptive read cache (L2ARC).

6. Log

There are two log types used:

ZFS intent log (ZIL): A logging mechanism where all the data to be written is stored, then later flushed, as a transactional write; this is similar to a journal file system (**ext3** or **ext4**).

Separate intent log (SLOG): A separate logging device that caches the synchronous parts of the ZIL before flushing them to the slower disk; it does not cache asynchronous data (asynchronous data is flushed directly to the disk). If the SLOG exists, the ZIL will be moved to it rather than residing on the platter disk; everything in the SLOG will always be in the system memory. Basically, the SLOG is the device and the ZIL is data on the device.

W22.3.2 ZFS Storage Poolsand the **zpool** Command

The sub-commands and options shown in this section are presented in this general way:

x. What the command, subcommand, and options accomplish.

**The command, subcommand, options, and command arguments**

Commentary or explanation.

Further examples:

**More variations of the command, subcommand, options and command arguments**

Commentary or explanation.

1. How to display zpools:

**zpool list**

Further examples:

**zpool list -o poolname, size, altroot**

There are a number of properties that you can select, the default is: name, size, used, available, capacity, health, altroot.

2. How to display zpool status:

**zpool status**

Further examples:

**zpool status -xv**

Shows only errored pools with more verbosity.

3. How to show zpool statistics:

**zpool iostat -v 5 5**

Use this command like you would **iostat**

4. How to show zpool history:

**zpool history -il**

Once a pool has been removed, the history is gone!

5. a) How to create a zpool:

**zpool create -n data2 /dev/sdb1**

The -n option performs a dry run but doesn’t actually perform the creation.

Further examples:

b) **zpool create data2 /dev/sdb1 /dev/sdc1**

You cannot shrink a pool, only grow it! Assumes there are two disks called **/dev/sdb1** and **/dev/sdc1** .

c) **zpool create data2a /dev/sdb1**

Using a standard disk slice on **/dev/sdb**, the first partition, numbered 1.

d) **zpool create -m /zfspool data2a /dev/sdb1**

Using a different mount point than the default **/<pool name>**.

e) **zpool create data3 mirror /dev/sdb1 /dev/sdc1 mirror /dev/sdd1 /dev/sde1**

**zpool create data4 mirror /dev/sdb1 /dev/sdc1 spare /dev/sdd1**

Mirror and hot spare disk examples. “Hot spares” are not used by default, so you need to turn on the auto-replace feature with **zpool setautoreplace=on** for each pool!

f) **zpool create data5 mirror /dev/sdb1 /dev/sdc1 log mirror /dev/sdd1 /dev/sde1**

Setting up a log device and mirroring it.

g) **zpool create data6 mirror /dev/sdb1 /dev/sdc1 cache /dev/sdd1 /dev/sde1**

Setting up a cache device.

h) **zpool create data7 raidz2 /dev/sdb1 /dev/sdc1 /dev/sdd1 /dev/sde1 /dev/sdf1**

You can also create RAID pools (RAIDZ/RAIDZ1: mirror; RAIDZ2: single parity; RAIDZ3: double parity).

In-Chapter Exercise

W22.12. How many discrete disks would you need to be able to actually implement the three levels of RAIDZ that you can create?

6. How to destroy a zpool:

**zpool destroy data2**

Further examples:

**zpool import -f -D -d /mypool/data2**

You can re-import a destroyed pool.

Another very powerful use of zpool importing, as described in Chapter 19 in the printed book, is to enable you to completely replace your bootable system disk with the Linux operating system built on it (which cannot at the current time be a ZFS vdev!) by keeping your user data on a second hard disk with a ZFS zpool on it (possibly mirrored onto other disks in RAID arrays). Then, when you are ready to replace the system disk, you export the user data disk(s) with the **zpool export** command, and then once you’ve replaced the system disk and installed ZFS on it, you can simply **import** the user data disk.

7. How to add a device to a zpool:

**zpool add data01 /dev/sdc1**

The **zpool** command only supports the removal of hot spares and cache disks! Therefore, be sure you want to add the device to the pool, because you cannot ordinarily remove it with the **zpool** remove command. For adding to mirrors, see the **attach** and **detach** subcommands that follow.

8. How to resize a zpool:

**zpool set autoreplace=on pool\_name**

**zpool set autoexpand=on pool\_name**

This is not about resizing partitions on an existing vdev that is contained in a zpool. It is about replacing an entire existing vdev, of smaller capacity, with another piece of hardware that has a larger capacity. When replacing a smaller disk with a larger one you must enable the autoreplace and autoexpand features to allow you to use the larger space. You must do this before replacing the first smaller capacity disk with the larger capacity second disk.

9. How to remove a zpool:

**zpool remove data01 /dev/sdb1**

**zpool** only supports the removal of hot spares and cache disks! Therefore, be sure you want to add the device to the pool, because you cannot ordinarily remove it with the **zpool remove** command. For adding to mirrors, see the **attach** and **detach** sub-commands that follow.

10. How to clear faults:

**zpool clear data01**

Further examples:

**zpool clear data01 /dev/sdb1**

Clears a specific disk fault.

11. Attaching additional drives as a mirror:

**zpool attach data01 /dev/sdb1 /dev/sdc1**

**/dev/sdb1** is an existing disk that is not mirrored, so by attaching **/dev/sdc1** to the pool **data01**, both disks will become a mirrored pair.

12. How to detach a mirror disk:

**zpool detach data01 /dev/sdb1**

See the previous note on attaching additional drives as a mirror.

13. How to *online* a zpool (put the pool online):

**zpool online data01 /dev/sdb1**

14. How to *offline* a zpool (take the pool offline):

**zpool offline data01 /dev/sdb1**

Further examples:

**zpool offline data01 -t /dev/sdb1**

This achieves temporary offlining using **-t** (will revert back to online after a reboot).

15. How to replace pools:

**zpool replace data03 /dev/sdb1**

Replaces one disk that uses the same designation in **/dev** as another disk.

Further examples:

**zpool replace data03 /dev/sdb1 /dev/sdc1**

Replaces one disk with another diskin **/dev** that has a different designation. As mentioned above, make sure to set the autoreplace and autoexpand features on before doing this operation.

16. How to do scrubbing:

**zpool scrub data01**

Further examples:

**zpool scrub -s data01**

Stop a scrubbing in progress; check the scrub line using zpool status data01 to see any errors.

17. How to do exporting.

As mentioned above, and detailed in our recommended storage model in Chapter 19 of the printed book, this command, in combination with the **zpool import** command, enables you to completely replace your bootable system disk with the Linux operating system built on it (which cannot at the current time be a ZFS vdev!) by keeping your user data on a second hard disk with a ZFS zpool on it (possibly mirrored onto other disks in RAID arrays). Then, when you are ready to replace the system disk, you export the user data disk(s) with the **zpool export** command. Once you’ve replaced the system disk and installed ZFS on it, you can simply **import** the user data disk. This is a very powerful commercial system administration technique.

**zpool export data01**

You can list exported pools using the import command **zpool import** to find what the names of exported zpools are, if any.

18. How to do importing:

**zpool import data01**

When using standard disk devices—that is, **/dev/sdb1**.

Further examples:

**zpool import -d /zfs**

If using files in the **/zfs** file system

**zpool import -f -D -d /zfs1 data2**

Imports a destroyed pool.

19. Getting zpool parameters:

**zpool get all data01**

The source column denotes if the value has been changed from its default value; a dash in this column means it is a read-only value.

20. Setting zpool parameters:

**zpool set autoreplace=on data01**

Use the command **zpool get all <pool>** to obtain a list of current settings.

21. How to upgrade pools:

**zpool upgrade –v**

Lists upgrade paths.

Further examples:

**zpool upgrade -a**

Upgrades all pools.

**zpool upgrade data01**

Upgrades a specific pool; use **zpool get all poolname** to obtain the version number of a pool.

**zpool upgrade -V 10 data01**

Upgrades to a specific version.

22. Replace a failed disk:

**zpool list**

Lists the zpools and identifies the failed disk.

Further examples:

**zpool replace data01 /dev/sdb1**

**zpool replace data01 /dev/sdb1 /dev/sdc1**

Replaces the disk. You can use the same capacity disk or a new disk of equal or larger capacity. As mentioned above, make sure to set the autoreplace and autoexpand features on before doing this operation.

**zpool clear data01**

Clears any existing errors.

**zpool scrub data01**

Scrub the pool to check for any more errors (this depends on the size of the zpool, as it can take a long time to complete).You can now remove the failed disk in the normal way, depending on your hardware.

23. How to expand a pool’s capacity:

**zpool set autoexpand=on data01**

**zpool set autoreplace=on data01**

**zpool replace data01 /dev/sdb1 /dev/sdc1**

You cannot remove a disk from a pool and you cannot shrink the pool, but you can enlarge it by replacing existing disks with larger disks!

W22.3.3 ZFS File System Commands and the **zfs** Command

The subcommands and options shown in this section are presented in the following general way:

x. What the command, subcommand, and options accomplish.

**The command, subcommand, options, and command arguments**

Commentary or explanation.

Further examples:

**More variations of the command, subcommand, options and command arguments**

Commentary or explanation.

1. Displaying ZFS file systems:

**zfs list**

Lists all ZFS file systems

Further examples:

**zfs list -t filesystem**

**zfs list -t snapshot**

**zfs list -t volume**

**zfs list -t all -r poolname**

Lists different types (file system, snapshot, volume) by **poolname**.

**zfs list -r data01/sarwar**

Recursive display.

**zfs list -o poolname,mounted,sharenfs,mountpoint**

Complex listing:there are a number of attributes that you can use in a complex listing; see the man page for zfs.

2. How to create a file system:

**zfs create data01/sarwar**

Assumes a pool exists named **data01**, and creates a **/data01/sarwar** ZFS file systemon that pool**.**

Further examples:

**zfs create -o mountpoint=/users/data01/users**

Creates the ZFS filesystem at a different mount point.

3. How to destroy a file system:

**zfs destroy data01/sarwar**

Further examples:

**zfs destroy -r data01/sarwar**

**zfs destroy -R data01/sarwar**

Uses the recursive options **-r** (all children), **-R** (all dependents).

4. How to mount a file system:

**zfs mount data01**

Further examples:

**zfs mount -o mountpoint=/tmpmnt data01/sarwar**

You can create temporary mount that expires after unmounting. You can apply all the normal mount options, i.e., **ro/rw, setuid,** etc..

5. How to unmount a file system:

**zfs umount data01**

6. How to share a file system:

**zfs share data01**

Further examples:

**zfs set sharenfs=on data01**

This file system persists after reboots!

**zfs set sharenfs="rw=@192.168.0.13/24" data01/sarwar**

Shares with specific hosts.

7. How to unshare a file system:

**zfs unshare data01**

Further examples:

**zfs set sharenfs=off data01**

This file system persists after reboot!

8. How to take snapshots of file systems:

Taking a “snapshot” of a file system is like taking a picture: changes are recorded to the snapshot when the original file system changes; to remove a dataset all previous snapshots have to be removed. You can also rename snapshots. You cannot destroy a snapshot if it has a clone.

**zfs snapshot data01@10022010**

Creates a snapshot.

Further examples:

**zfs snapshot rename data01@10022010 data01@mybackup**

Renames a snapshot.

**zfs destroy data01@10022010**

Destroys a snapshot.

9. How to roll back a file system:

By default, you can only roll back to the latest snapshot. To roll back to older ones, you must delete all newer snapshots.

**zfs rollback data01@10022010**

10. Cloning/promoting file systems:

Clones are writable file systems that have been upgraded from a snapshot. A dependency will remain on the snapshot as long as the clone exists. A clone uses the data from the snapshot to exist. As you use the clone, it uses space separate from the snapshot. Clones cannot be created across zpools, you need to use the **zfs send/receive** commands to do this, as shown in Example W22.2.

**zfs clone data7@10022010 data8/clone**

**zfs clone -o mountpoint=/clone data7@10022010 data8/clone**

Clones, changes the mount point of the clone.

Further examples:

**zfs promote data8/clone**

Promotes a clone. This allows you to destroy the original file system that the clone is attached to.The clone must reside in the same pool!

11. Renaming a file system:

**zfs rename data03/koretsky\_disk01 data03/koretsky\_d01**

The dataset must be kept within the same pool**.** There are two options on this command: **-p** creates all the nonexistent parent datasets; **-r** recursively rename the snapshots of all descendent datasets (used with snapshots only).

12. Compression of file systems:

**zfs set compression=lzjb data03/sarwar**

You enable compression by setting a feature. Compressions are on, off, lzjb, gzip, gzip[1–9] and zle. Compression only starts when you turn it on; other existing data will not be compressed.

Further examples:

**zfs get compressratio data03/sarwar**

You can get the compression ratio.

13. Deduplication:

You can save disk space using *deduplication*, which can be done on the level of a file, a block, or a byte. For example, at the file level, each file is hashed with a cryptographic hashing algorithm such as SHA-256; if two files match, then just point to the existing file rather than storing a whole new file. This is ideal for small files, but for large files, a single character change would mean that all the data has to be copied over again! Block deduplication allows you to share all the same blocks in a file minus the blocks that are different; this allows the sharing of unique blocks on disk and the reference-shared blocks in RAM. However, a lot of RAM is necessary to keep track of which blocks are shared and which are not. This is the preferred option rather than file or byte deduplication. Shared blocks are stored in what is called a *deduplication table*; the more deduplicated blocks there are, the larger the table. The table is read every time to make a block change, thus the table should be held in fast RAM. If you run out of RAM, then the table will be saved onto disk. So how much RAM do you need? You can use the zdb command to check and take the *bp count*. A good rule of thumb is to allow 5 GB of RAM for every 1 TB of disk.

**zdb -b data01**

Use this command to see the block the dataset consumes.

Further examples:

**zfs set dedup=on data01/myfiles**

To turn on deduplicate.

**zfs get dedupratio data01/myfiles**

To see the deduplication ratio.

**zdb -DD poolname**

To see a histogram of how many blocks are referenced how many times.

14. Getting file system parameters:

**zfs get all data03/sarwar**

Lists all the properties.

Further examples:

**zfs get setuid data03/sarwar**

Gets a specific property.

**zfs get compression**

Gets a list of a specific properties for all datasets**.** The source column denotes if the value has been changed from its default value; a dash in this column means it is a read-only value.

15. Setting file system parameters:

**zfs set copies=2 data03/sarwar**

Sets the number of copies of dataset **sarwar** in the pool **data03** to 2; the default number of copies is 1. This is probably the most useful and important way to ensure redundancy on a nonredundant vdev, such as a single hard disk in a laptop computer. Although it doubles the storage space required to contain the dataset, error correction with **zpool scrub** can be achieved on the nonredundant vdev that contains the pool and its datasets that have copies set to 2.

Further examples:

**zfs set quota=50M data03/sarwar**

**zfs set quota=none data03/sarwar**

Sets and unsets the disk usage quota.Use the command **zfs get all <dataset>** to obtain a list of current settings.

16. How to have a file system inherit attributes:

**zfs inherit compression data03/sarwar**

Sets back to the default value.

17. How to upgrade the ZFS version:

**zfs upgrade –v**

Lists the upgrade paths.

Further examples:

**zfs upgrade**

Lists all the datasets that are not at the current level.

**upgrade -V <version> data03/linuxthetextbook2**

Upgrades a specific dataset.

18. How to use allow/unallow:

**zfs allow master**

Displays the permissions set and any user permissions.

Further examples:

**zfs allow -s @permset1 create,mount,snapshot,clone,promote master**

Creates a permission set.

**zfs unallow -s @permset1 master**

Deletes a permission set.

**zfs allow vallep @permset1 master**

Grants a user permissions.

**zfs unallow vallep @permset1 master**

Revokes a user’s permissions.There are many permissions that you can set. Refer to the zfs man page, or just use the zfs allow command, to get help.

W22.4 File System Backups Using **zfs snapshot**

Snapshots are the ZFS way of creating archives and backups automatically or with very simple operations and commands. As stated previously, taking a snapshot of a file system is like taking a picture; changes are recorded to the snapshot when the original file system changes.

Here are some important things to remember about snapshots:

* To remove a dataset, all previous snapshots have to be removed.
* You can rename snapshots.
* You cannot destroy a snapshot if it has a clone.

W22.4.1 Examples of **snapshot**

An example of creating a snapshot:

**zfs snapshot data01@10022010**

An example of renaming a snapshot:

**zfs snapshot data01@10022010 data01@mybackup**

An example of destroying a snapshot:

**zfs destroy data01@10022010**

W22.4.2 **zfs rollback**

It is possible to roll back a file system, or return it to a previous state. You must use the **zfs rollback** command. By default you can only roll back to the latest snapshot, to roll back to an older one you must delete all newer snapshots!

An example of rolling back to a snapshot is:

**zfs rollback data01@10022010**

W22.4.3 Cloning/Promoting

As stated previously, clones are writable file systems that have been upgraded from a snapshot, and a dependency will remain on the snapshot as long as the clone exists. A clone uses the data from the snapshot to exist. As you use the clone it uses space separate from the snapshot. Clones cannot be created across zpools, you need to use the **zfs send/receive** commands to do this.

Two examples of cloning are:

**zfs clone data7@10022010 data8/clone**

**zfs clone -o mountpoint=/clone data7@10022010 data8/clone**

Promoting a clone allows you to destroy the original file system that the clone is attached to. An example of this is:

**zfs promote data8/clone**

The clone must reside in the same pool.

W22.4.4 Renaming a Filesystem

The dataset must be kept within the same zpool! An example of this is:

**zfs rename data03/koretsky\_disk01 data03/koretsky\_d01**

There are two options on this command: **-p** creates all the nonexistent parent datasets; **-r** recursively renames the snapshots of all descendent datasets (used with snapshots only).

W22.4.5 Compression of Filesystems

You enable compression by setting a feature. Compressions are on, off, lzjb, gzip, gzip[1–9] and zle. Compression starts when you turn it on; other existing data will not be compressed. An example of this is:

**zfs set compression=lzjb data03/sarwar**

You can get the compression ratio by using the following example:

**zfs get compressratio data03/sarwar**

W22.4.6 Bourne Shell Script Example for Incremental ZFS Backups

In this section we give examples of how to utilize zfs to accomplish an important system administration task for the ordinary user, backup of file systems and the files in them. The first example uses zfs send/receive to accomplish this backup. The second example uses a Bourne shell script to automate the process, in order to make it faster and more efficient.

Example W22.6 Sending and Receiving ZFS Snapshots Across a LAN

Objectives: To create two zpools and their default datasets on two different thumbdrives mounted on two different systems on a LAN. To then take zfs snapshots (backups) of the dataset on one system, and use zfs send/receive to transmit those snapshots to the other system.

Pre-Requisites:

1. Completion of Examples W22.1 through W22.5

2. Installation of ZFS on both systems Linux, as shown in Appendix A in the printed book

3. Having root access privileges on both systems.

Step 1. Prepare the two thumbdrives as shown in Example W22.5

Step 2. Mount the thumbdrives on the two systems, and create one zpool on each as follows-

sender , receiver

Step 3. In the dataset named sender (by default mounted at /sender), which was created by default in the zpool created in Step 2., use a text editor to create a file named “newfile”.

Step 4. Take a zfs snapshot of the sender dataset with the following command:

$ **sudo zfs snapshot sender@2016-11-20**

Step 5. Send the snapshot stream to the second system, as seen in the previous section, using ssh, as follows:

$ **zfs send sender@2016-11-20 | ssh root@192.168.0.8 zfs receive -F receiver**

Step 6. On the second system, the file named newfile should be in the directory /receiver.

When you send a full stream, the destination dataset must not exist.

Step 7. Add another file using your text editor to /sender named newfile2. Take another snapshot of sender with the following command:

$ **zfs snapshot sender@2016-11-21**

Step 8. You can send incremental data by using the **zfs send -i** option. To send incremental sanpshots to the second machine, use the following command:

$ **zfs send -i sender@2016-11-20 sender@2016-11-21 | ssh root@192.168.0.8 zfs \ receive -F receiver**

Note that the first argument, sender@2016 -11-20, is the earlier snapshot and the second argument, sender@2016-11-21, is the second snapshot. In this case, the dataset receiver must already exist for the incremental receive to be successful.

Step 9. On the second system, the files named newfile and newfile2 should be in the directory /receiver.

10. To clean up, use the sudo zfs destroy command to delete the sanpshots on both systems. Then use the sudo zpool destroy command on both systems to destroy the datasets and pools created on the thumbdrives. Finally, use the gparted program on your Linux system to reinitialize the thumbdrives so that they can be used outside of zfs.

Example W22.7 : zfs snapshot Command Automation in a Bourne Shell Script

The following Bourne shell script achieves the incremental backing up of a file system on one computer to a remote host system on a LAN, using zfs snapshot send/receive. It is very similar to, and a further extension of, the zfs send/receive example code shown in Examples W22.2 and W22.6

The prerequisites and advisories for running this script are:

0. Installation of ZFS on both Linux systems, as shown in Appendix A of the printed book.

1. The host receiving the snapshot must be running the same or a higher version of ZFS than the sender.

2. You must be able to login as root via ssh on the destination host receiving the backup. This involves changing your sshd\_config file, and is a security risk if your machine has a public-facing IP address.

3. You must be sending to an account that has ZFS create/receive properties.

4. You have previously created a snapshot of the source, and then renamed it to yesterday’s date.

5. You have previously created a ZFS dataset on the destination host named **/mnt/receive**.

6. The zfs dataset names for source and destination, and the LAN IP address shown, are for our system. You need to change these appropriately for your system.

**#!/bin/sh**

**# This assigns a local filesystem as the source to be transmitted**

**pool="/sender/send\_dir"**

**# This assigns a remote destination**

**destination="/mnt/receive"**

**# This names the IP address of the remote target host**

**host="192.168.0.8"**

**# Sets the date format for today**

**today=`date +"%Y-%m-%d"`**

**# This sets the date format for yesterday**

**yesterday=`date +"%Y-%m-%d" -d"-1 day"`**

**# Create today’s snapshot**

**snapshot\_today="$pool@$today"**

**# look for a snapshot with this name, and if none exists, take the snapshot**

**if zfs list -H -o name -t snapshot | sort | grep "$snapshot\_today$" > /dev/null**

**then**

**echo " snapshot, $snapshot\_today, already exists"**

**exit 1**

**else**

**echo " taking todays snapshot, $snapshot\_today"**

**zfs snapshot -r $snapshot\_today**

**fi**

**# look for yesterdays snapshot**

**snapshot\_yesterday="$pool@$yesterday"**

**# If it exists, zfs send todays snapshot**

**if zfs list -H -o name -t snapshot | sort | grep "$snapshot\_yesterday$" > /dev/null**

**then**

**echo " yesterday snapshot, $snapshot\_yesterday, exists, send todays backup"**

**zfs send -R -i $snapshot\_yesterday $snapshot\_today | ssh root@$host**

**zfs receive –Fduv $destination**

**echo " backup complete destroying yesterdays snapshot"**

**zfs destroy -r $snapshot\_yesterday**

**exit 0**

**else**

**echo " missing yesterday snapshot aborting, $snapshot\_yesterday"**

**exit 1**

**fi**

W22.4.7 Making ZFS the Root File System in Linux

As stated in Section W22.1, at the time of the writing of this book, you cannot easily and reliably use ZFS as the root, or boot, disk on three out of the four representative Linux systems we illustrate everything in this book with. But in order to gain an appreciation of the utility of having ZFS as the root, do the instructions for installing Ubuntu 18.04 to a root ZFS filesystem, as shown in Example W22.8.

Example W22.8 Installing Ubuntu 18.04 as the Root File System on a Single Disk Computer

Objectives: To do a bare-metal install of a fresh copy of 64-bit Ubuntu 18.04 Desktop, using ISO DVD or USB install media (not server, netboot, or alternative install editions), and create ZFS as the root, or system disk, file system.

Pre-Requisites:

1. Completion of Examples W22.1 through W22.7.

2. Having the ISO DVD or USB install media available for Ubuntu 18.04 Desktop Edition.

3. Having a single hard drive system available onto which you can build the Ubuntu 18.04 system, not in a Virtual Machine.

Background: In all of the previous ZFS examples, we used either a file on the system disk, or a separate hard disk or USB thumb drive, to install our ZFS file system on. In this Example, we will directly install ZFS as the root, or system disk, file system on a single-disk computer. The default installer, Ubiquity, for release 18.04 does not support the ZFS filesystem, nor are the ZFS tools pre-installed in the LiveCD environment. The Example explains how to install ZFS to the LiveCD environment ZFS, how to install Ubuntu 18.04 to a ZFS zvol formatted as ext4, then finally transfers (via the rsync command) this ZVOL into a bootable hard disk environment.

System Requirements

16Gb or larger hard drive that can be devoted exclusively to this install, possibly the only hard disk available on the system.

An Internet connection usable while in the LiveCD environment

4GB memory recommended, non-ECC memory OK

Strategic Overview

The Ubiquity installer for 18.04 does not recognize the ZFS filesystem as a usable target, however it can be installed to a ZFS ZVOL then manually copied to the ZFS filesystem.

We accomplish this by using the ZVOL as a vdev. The ZVOL then becomes a block device that can be used just like a physical drive. Additionally, you can create mirror(s) or raidz(x) configurations starting at Step 1.c. below. We use the Ubiquity installer to transfer the ZVOL to the actually-installed system as part of the root file system that Ubiquity installs.

An important difference between this example and the previous ones is that it specifies devices found in "/dev/disk/by-id/", not "/dev/", as vdevs when creating pools. The first step of the Requirements below lists the possible choices in /dev/disk/by-id/ for our system. In the Requirements steps, we'll use a single disk, "/dev/disk/by-id/ata-VB0250EAVER\_Z2AEN9TX". Additionally, the ZFS pool name we create is named "test1".

Requirements: Do the following steps, in the order presented below, to fulfill the objectives of this Example-

0. Boot your computer to the LiveCd medium, and when presented with the choice of “Installing” or “Trying” Ubuntu, make the “Trying” menu choice. Proceed until the desktop environment is running.

1. The purpose of this step is to install the ZFS packages to the LiveCD install environment, create a zpool in that environment, and finally a ZVOL in that zpool which can be used for installation of the Ubuntu 18.04 system.

a. Open a terminal with the keystroke combination **<Ctrl>+<Alt>+T**

Then make note of the disks in /dev/disk/by-id/

ubuntu@ubuntu:~$ **cd /dev/disk/by-id/**

/dev/disk/by-id$ **ls**

ata-hp\_DVD-RAM\_GH60L\_K2SB68J5149 wwn-0x5000c5003f0445d8

ata-VB0250EAVER\_Z2AEN9TX

ubuntu@ubuntu:/dev/disk/by-id$ cd

ubuntu@ubuntu:~$

b. Become the superuser, and then install the zfsutils package.

ubuntu@ubuntu:~$ **sudo su**

root@ubuntu:/home/ubuntu# **apt install -y zfsutils**

Reading package lists... Done

Building dependency tree

Reading state information... Done

Note, selecting 'zfsutils-linux' instead of 'zfsutils'

The following additional packages will be installed:

libnvpair1linux libuutil1linux libzfs2linux libzpool2linux zfs-zed

Output truncated…

root@ubuntu:/home/ubuntu#

c. Create a zpool named test 1 with the options shown, onto the disk from sub-step a. above. In that zpool, create a 10 Gigabyte ZVOL named ubuntu-temp.

root@ubuntu:/home/ubuntu# **zpool create -f -o ashift=12 -O atime=off -O compression=lz4 -O normalization=formD -O recordsize=1M -O xattr=sa test1 /dev/disk/by-id/ata-VB0250EAVER\_Z2AEN9TX**

root@ubuntu:/home/ubuntu# **zfs create -V 10G test1/ubuntu-temp**

root@ubuntu:/home/ubuntu#

2. Use the Ubiquity Installer launched from the terminal command line to install Ubuntu 18.04 to the ZVOL from sub-step 1.c. above.

root@ubuntu:/home/ubuntu# **ubiquity –no-bootloader**

Make any option choices you need or want in the installer, until you get to the "Installation Type" screen. Then select "Something Else".

a. The drive section lists "/dev/zd0". Select it and choose "New Partition Table...".

b. Choose "Continue" until the "Create new empty partition table on this device?" menu choice.

c. Select /dev/zd0 Free Space and press the "+" button.

d. Select "Use As: Ext4 journaling file system" and "Mount point: /", then click "OK",

e. Click "Install Now" and acknowledge "Write the changes to disks?" prompt by clicking "Continue".

f. Select the options and choices that apply to you on the additional installation screens appear, such as language, timezone, user account creation, and computer name.

g. At the end of the install select "Continue testing"from the menu choices that appeared in Step 0..

3. Copy your Ubuntu image to the ZFS filesystem using the **rsync** command.

Ubiquity has not unmounted the ZVOL after it finishes installation of the system on the ZVOL. Therefore, its mountpoint is "/target". To create ZFS as the Ubuntu 18.04 filesystem, you must use **rsync** to move your Ubuntu install from the ZVOL to the installed Ubuntu 18.04 filesystem. Be careful to type the rsync command shown below using the exact format shown! This example shows how to create a single filesystem. If you wish to have additional filesystems, for example /home or /var, you can create them, and set their mountpoints before the **rsync**.

Continuing in the terminal:

root@ubuntu:/home/ubuntu# **zfs create test1/ROOT**

root@ubuntu:/home/ubuntu# **zfs create test1/ROOT/ubuntu-1**

root@ubuntu:/home/ubuntu# **rsync -avPX /target/ ./test1/ROOT/ubuntu-1/ .**

This last command takes awhile!

Output truncated...

sent 4,231,494,705 bytes received 2,031,763 bytes 14,880,585.12 bytes/sec

total size is 4,222,703,710 speedup is 1.00

root@ubuntu:/home/ubuntu#

4. Preparing your ZFS filesystem copy of Ubuntu 18.04 to utilize ZFS

The ZFS filesystem needs to be configured so that it becomes available after reboot. To accomplish this configuration, we do the following-

a. Connect the active /proc, /dev, and /sys mounts to the ZFS filesystem copy and chroot into it.

b. Give ZFS a nameserver, update the available repositories, and install the ZFS binaries.

c. Remove the root filesystem and /swapfile from fstab, which are automatically mounted after reboot. d. \*Optional\* If you want to have swap space, you can create a ZVOL for swap and reference that in the fstab.

e. Remove the unused "/swapfile".

root@ubuntu:/home/ubuntu# **for d in proc sys dev; do mount --bind /$d /test1/ROOT/ubuntu-1/$d; done**

root@ubuntu:/home/ubuntu# **chroot /test1/ROOT/ubuntu-1**

root@ubuntu:/# **echo "nameserver 8.8.8.8" | tee -a /etc/resolv.conf**

nameserver 8.8.8.8

root@ubuntu:/# **apt update**

Hit:1 http://us.archive.ubuntu.com/ubuntu bionic InRelease

Hit:2 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease

Hit:3 http://security.ubuntu.com/ubuntu bionic-security InRelease

Hit:4 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease

Reading package lists... Done

Building dependency tree

Reading state information... Done

14 packages can be upgraded. Run 'apt list --upgradable' to see them.

root@ubuntu:/# **apt install -y zfs-initramfs**

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following additional packages will be installed:

libnvpair1linux libuutil1linux libzfs2linux libzpool2linux zfs-zed zfsutils-linux

Output truncated...

root@ubuntu:/# **nano /etc/fstab** ## comment out the lines for the mountpoint "/" and "/swapfile" and exit

root@ubuntu:/# **rm /swapfile**

5. Create a BIOS Grub partition and install Grub

ZFS whole disk formatting uses GPT partitioning, and ZFS whole disk formatting will only allow for booting from a GPT disk partition table, with EFI boot and an EFI partition present, or legacy (MBR) boot with a GRUB BIOS partition present. Therefore, we create a GRUB BIOS partition in an unused section at the beginning of the drive, then update and install grub to the disk.

root@ubuntu:/# **sgdisk -a1 -n2:512:2047 -t2:EF02 /dev/disk/by-id/ata-VB0250EAVER\_Z2AEN9TX**

Warning: The kernel is still using the old partition table.

The new table will be used at the next reboot or after you

run partprobe(8) or kpartx(8)

The operation has completed successfully.

root@ubuntu:/# **update-grub**

Generating grub configuration file ...

Warning: Setting GRUB\_TIMEOUT to a non-zero value when GRUB\_HIDDEN\_TIMEOUT is set is no longer supported.

Found linux image: /boot/vmlinuz-4.15.0-20-generic

Found initrd image: /boot/initrd.img-4.15.0-20-generic

Found memtest86+ image: /ROOT/ubuntu-1@/boot/memtest86+.elf

Found memtest86+ image: /ROOT/ubuntu-1@/boot/memtest86+.bin

done

root@ubuntu:/# **grub-install /dev/disk/by-id/ata-VB0250EAVER\_Z2AEN9TX**

Installing for i386-pc platform.

Installation finished. No error reported.

root@ubuntu:/#

6. Set Mountpoint and reboot

Exit the chroot, unmount the /dev, /sys, and /proc from /test1/ROOT/ubuntu-1, as well as /test1/ROOT/ubuntu-1 itself. We set our ZFS filesystem's mountpoint variable to /, snapshot our filesystem , unmount our Zvol, export our pool, then reboot.

root@ubuntu:/# **exit**

root@ubuntu:/home/ubuntu# **umount -R /test1/ROOT/ubuntu-1**

root@ubuntu:/home/ubuntu# **zfs snapshot test1/ROOT/ubuntu-1@pre-reboot**

root@ubuntu:/home/ubuntu# **umount /target**

umount: /target: target is busy.

root@ubuntu:/home/ubuntu# **zpool export test1**

cannot export 'test1': pool is busy

root@ubuntu:/home/ubuntu# **shutdown -r 0**

At this point you should have successfully booted Ubuntu 18.04 using ZFS as a root filesystem.

Conclusion: This example created ZFS as the root, or system disk, file system with a bare-metal install of a fresh copy of 64-bit Ubuntu 18.04 Desktop Edition, using ISO DVD or USB install media (not server, netboot, or alternative install editions). This installation was done on a single-disk system, such as found on laptop computers.

Summary

This chapter provided a common user of Linux Mint with the basic techniques of working with the Zettabyte File System (ZFS).

We first went over some basic terms used in ZFS and then described what ZFS is from the user perspective. What differentiates ZFS from other file systems is that ZFS file systems are mapped onto pool storage facilities, known as zpools, rather than onto physical storage media like disk drives. The zpools are then mapped onto physical media. That means that the file system’s storage requirements can grow as more physical media devices are added to the zpools.

We then provided six working examples of using the two ZFS commands, **zpool** and **zfs**. These working examples illustrate for the beginner some of the basic operations that can be used to create pools and file systems.

We provided a command reference section that gives many ZFS command usage examples.

We provided file system backup procedures and a Bourne shell script example that used the **zfs snapshot, zfs send**,and **zfs receive** commands.

Finally, we provided a worked example (Example W22.8) of how to install a Linux system, Ubuntu 18.04 , with ZFS as the root file system. This procedure is particularly useful and applicable to single-disk systems, such as laptop computers.

**Questions and Problems**

1. Is it possible to create a zpool using only a single slice on a vdev, and if so, what would the advantages and disadvantages of doing this be? Does your answer reflect the fact that only one file system can exist on that slice?

To follow up on this question, is it possible to create a volume with a ZFS file system on it? What would be the advantage of doing this?

2. List the advantages and disadvantages that ZFS has for you on your Linux system.

3. Give a brief description of the **zdb** command.

4. Similar to Example W22.3, create a mirrored zpool using two files that simulate disk drives, and are 256 MB in size each. Name the files **disk1** and **disk2**. Then answer the following questions:

a. What is the pathname to any file you can create in the zpool?

b. If you create a 32 MB file in your zpool, what size increase do you see in the files **disk1** and **disk2**?

c. How much free space is now in the zpool?

5. Define the following terms in ZFS, in your own words:

*Scrubbing*, *resilvering*, *slicing*, *mirroring*, *zpool*, *vdev*.

6. For Ubuntu Releases 18.04 and above-

As stated in Section W22.1, at the time of the writing of this book, you cannot easily and reliably use ZFS as the root, or boot, disk on three out of the four representative Linux systems we illustrate everything in this book with. But in order to gain an appreciation of the utility of having ZFS as the root, do the instructions for installing Ubuntu 18.04 to a root ZFS filesystem, as shown in Example W22.8.

The prerequisites for doing this problem are:

\* That you have an expendable, single-hard disk computer and an external USB hard disk in an enclosure. A laptop computer would serve very well.

\* You can download and create a DVD or USB thumbdrive version of Ubuntu 18.04 (or for the latest version of Ubuntu) at www.ubuntu.com.

This problem assumes that the external USB hard disk in its enclosure, and the internal hard disk in your computer are the same capacity, and can be easily removed, swapped, and reinstalled. The instructions given in Example W22.8 make allowance for mirroring of the ZFS root disk, which is the main objective of this problem.

Also, an important difference between the Example W22.8 instructions, and what we have shown in this chapter, is that we used logical device names for vdev specifications. In Example W22.8, we use by-id reference to hard disks. For example, where we normally refer to a hard disk to be included as a vdev for a zpool as /dev/sda, Example W22.8 uses a designation like /dev/disk/by-id/disk\_name, such as-

/dev/disk/by-id/ata-Samsung\_SSD\_850\_EVO\_120GB\_S21TNXAG719325D

To find out the disk name, or alias, on your system, use the command ls -la /dev/disk/by-id

Again, the most important use of the Example W22.8 procedure that you will implement in this problem is to enable you to create a ZFS mirror of your system disk. That gives you redundancy of your system disk, and user data if it is stored on that disk as well. A single, internally mounted hard disk vdev may be inside of a laptop computer that can only have a single hard disk internally mounted. The mirror you will create in this problem is maintained over the USB bus. If the hard drive in the external (but openable) enclosure we designate here is a SATA drive, then that drive can also be used as a replacement for the internal, single SATA hard disk drive if that drive fails. They must also both be SATA drives of equal capacity, either spinning disk or SSD. This problem achieves some of the same objectives as those illustrated in Chapter 19 for Clonezilla Live.

Use an externally mounted USB hard drive in an openable enclosure to accomplish a hybridized set of the operations given in ExamplesW22.4 and W22.8. You must make sure that the capacity of this USB external drive is large enough to accommodate the creation of the mirror.

To test whether or not you have correctly solved this problem, shut down the system. Then remove the external USB hard drive from its enclosure, and replace the internally mounted hard drive in your computer with it. Your system should then boot from the new hard drive, but the mirror will show as being in a degraded state. Wait until the mirror is resilvered.

To repeat this procedure when your internal hard drive actually fails, follow the processes shown previously in the chapter for replacements of vdevs in a mirror.

7. If you create a zpool named **pool1**, and a file system on that zpool named **bobsfiles** with the zfs command, what is the pathname to a file named **data27** in that file system? What is the exact syntax of the command you used to create the file system?

8. List eight of the basic zfs subcommands that allow you to do file and file system backups and archiving.

9. Following the completion of the first five steps of Example W22.5, do the following:

a. Use the zfs command to create a dataset named **usbdrive** on the zpool **test3** located on the thumbdrive. The name of the dataset would be **test3/usbdrive**.

b. Type the command zfs set copies=2 test3/usbdrive.

c. What you have achieved with this command is a signature validation of using ZFS and creating a zpool on the USB thumbdrive. The USB thumbdrive is a redundant device to the extent shown. But more importantly, by setting the property of **copies=2** on this dataset, you have made the USB thumbdrive redundant to itself, because ZFS now keeps two copies of everything you put in the dataset **backup/usbdrive**. And you can use ZFS facilities to ensure integrity of the data to the bit level on the USB thumbdrive. Given how inexpensive USB thumbdrives are, even in larger capacities, having two automatically created copies of your files on this thumbdrive is not prohibitive.

d. Copy a number of important files into this new dataset from your systems hard drive using either the cp or rsync commands shown in Chapter 23.

e. Instead of proceeding onto step 6 of the example, retain the zpool and dataset you have created on the thumbdrive, and use it as a backup drive for your important files. You can periodically use **rsync** to keep the backup files synchronized to the important files on your hard drive. You may even decide that the important files you want to back up to the thumbdrive are in a single directory or multiple directories. You can then use rsync to copy directories over to the thumbdrive.

f. To remove the USB thumbdrive temporarily at any time, use the zfs unmount command. Then you can remove it from the computer. Remember to use the zfs mount command when you want to reinsert the USB thumbdrive and archive or backup files to it.

10. Repeat Problem W22.9, but instead of using **cp** or **rsync** to move important files to the USB thumbdrive from some arbitrary place(s), do the following:

a. Create a new zfs dataset on a hard drive, perhaps your only hard drive and system disk, which will be the source of your important files to be backed up.

b. Fill the dataset with important files you want to backup.

c. Use the methods of Example W22.2 to create a ZFS snapshot of the new dataset created on your hard drive pool. Then, continuing the methods shown in Example W22.2, use **zfs send** and **zfs receive** to copy the snapshot from the hard drive to the USB thumbdrive.

11. Back up your home directory (or a selected subdirectory of your home directory) to a USB thumbdrive using the commands zfs snapshot, zfs send, and zfs receive. Be sure to insert and test the USB thumbdrive you want to use for this problem, to find out if it is usable on your system. If it doesn’t show up in /dev after insertion, it is not usable!

After testing the USB thumbdrive for usability, prepare the USB thumbdrive using the gparted program to destroy any partition table on it, create a GPT partitioning table on it, and create one primary partition with an EXT4 file system.

12. Using the techniques shown in the five examples of Section W22.2, create a mirrored pair of USB thumbdrives on you computer. Then create a single dataset on the mirrored pair. Finally, using the following Bash shell script, backup an important data file directory in your home directory, in tar format, to the dataset on the mirrored-pair of USB thumbdrives. Substitute your pathnames for source and destination directories on your system.

Example W22.9 bash Backup to a Mirrored Pair

#!/bin/bash

# A time stamp variable for logging

TIMESTAMP=`date +%Y%m%d.%H%M`

# Destination directory location on the mirrored pair as a variable

DEST\_DIR="/media/bob/Sony\_16SA1"

# Source directory as a variable

SRC\_DIR="/home/bob/nweb23"

# Variable for the backup file name file

FNAME="MyBackup"

# Variable for a log file, in an already created sub-directory in your home

# directory, and name the log file with the file name and time stamp

LOG="/home/bob/log/$FNAME-$TIMESTAMP.log"

# Message that the backup is started

echo -e "Starting backup of $SRC\_DIR directory" >> ${LOG}

# Compress the source directory and files, copy the tar.gz file to

# your destination directory

tar -vczf ${DEST\_DIR}/${FNAME}-${TIMESTAMP}.tar.gz ${SRC\_DIR} >> ${LOG}

# Message that the backup has ended, and append to log file

echo -e "Ending backup of $SRC\_DIR" >> ${LOG}

13. Use the procedures of Example W22.6 to send incremental backup snapshots of an important user dataset, from a USB thumbdrive over a LAN to another USB thumbdrive mounted on a second Linux system.

14. Use the procedures and Bourne shell script of Example W22.7 to send incremental backup snapshots of an important user dataset, over a LAN to another Linux system.