# Oracle Solaris 11 ZFS File System

## 1 Introduction

Oracle Solaris ZFS is a revolutionary file system that changes the way we manage storage. Participants in this lab will gain awareness through example of devices, storage pools, and performance and availability. We will learn about the various types of ZFS datasets and when to use each type. We will examine snapshots, cloning, allocation limits, and recovering from common errors.

We will cover the following areas around ZFS:

- Zpools
- Vdevs
- ZFS datasets
- Snapshots / Clones
- ZFS properties
- ZFS updates

These exercises are meant to provide a primer into the value and flexibility of Oracle Solaris 11 ZFS for the enterprise. Upon completion of this lab, the learner will understand the simplicity and power of the ZFS file system and how it can help address business requirements with Oracle Solaris 11 storage technology and will be well on their way to mastering this powerful technology.

## 2 Overview

ZFS is the default file system in Oracle Solaris 11. This lab will follow the basic system administration duties revolving around storage in a basic system. As in any installation or implementation we'll follow a basic path for building our storage infrastructure

1) Hardware setup and initial storage connection and assignment. (VirtualBox, virtual disks, and files)
2) Creating pools. Storage devices in ZFS are grouped into pools. A pool provides all of the storage allocations that are used by the file systems and volumes that an installation will require.
3) Creating file systems which can be assigned to users and applications and

manipulated to fit the needs of each.

4) Details around each of the above resulting in a complete storage picture for Oracle Solaris 11 from which the learner can begin to develop more expertise.
5) We will use the VirtualBox application to create virtual SAS disks that Oracle Solaris 11 can work with just like real disks.
6) We will create files within the operating system to work with for simplicity. The files are treated just like disks and working with them would be no different than working with a large storage array connected to a customer system.

# 3 Pre-requisites

This lab requires the use of the following elements:

- A current laptop with at least 3GB memory and 100GB free disk space
- Oracle VirtualBox Software (4.0.16 with Extension Pack installed)
- Oracle Solaris 11 11/11 PreBuilt VM for Oracle VM VirtualBox

The following assumptions have been made regarding the environment where this lab is being performed:

1. Network connectivity to the Internet is not necessary
2. We will only work with a single Solaris 11 Virtual Instance

# 4 Lab Setup

## 4.1 Oracle VirtualBox Hypervisor Software basics

Your system should already have Oracle VirtualBox hypervisor software installed and ready to use. For this lab we will require a GUI interface and will be using the pre-built Oracle Solaris 11 VM image. We only need to acquire it and import it to get running quickly.

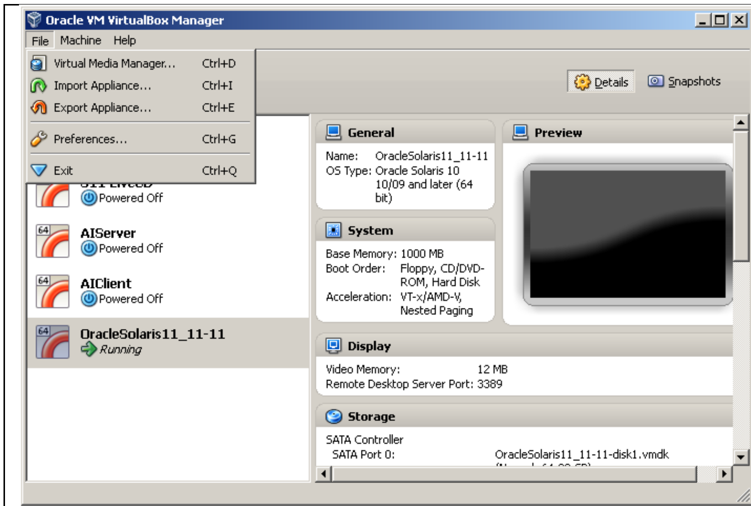
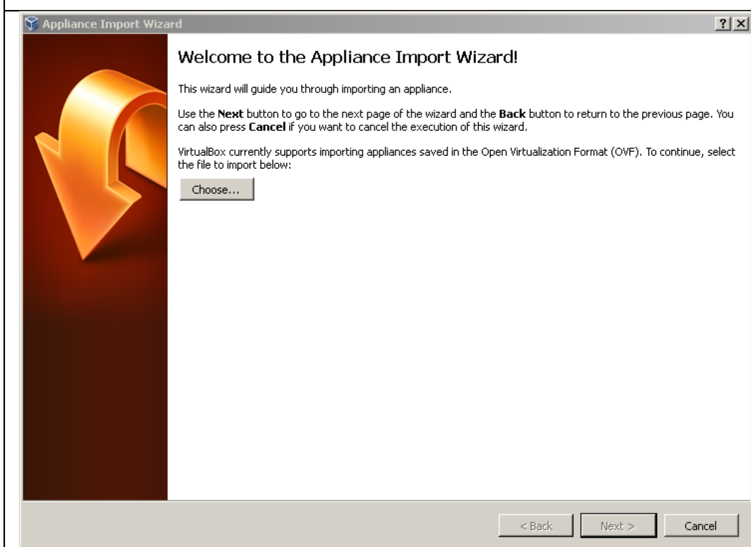
**Download Virtual Box**

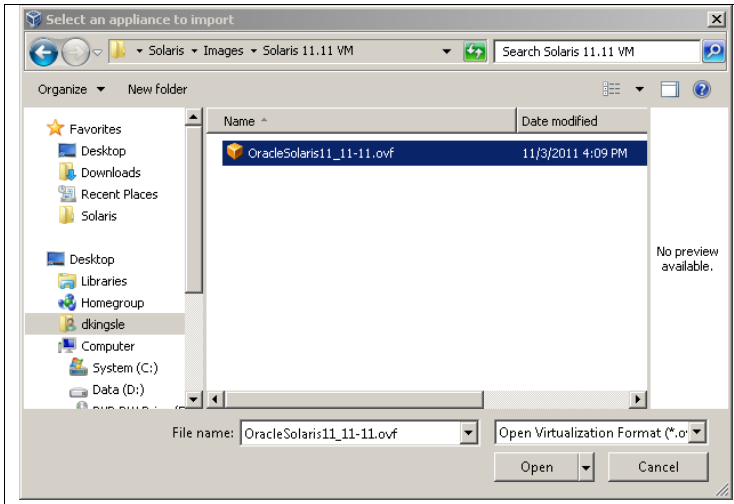
**VirtualBox Installation Notes:**

- VirtualBox mouse capture can sometimes be frustrating in the way it handles mouse interaction between VBox and your OS. Use the right control key on your keyboard to return mouse control back to your default environment. You can change this in VirtualBox preferences.

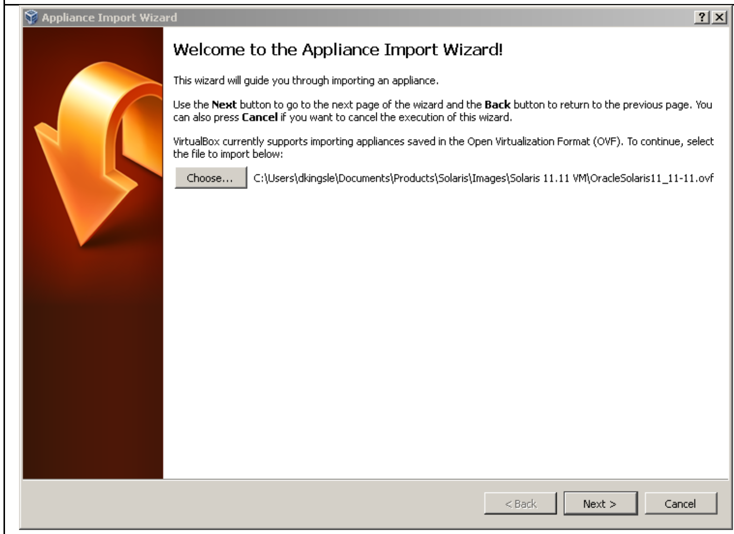
## 4.2 Oracle Solaris 11 VM Image installation

- Make sure that you have the Oracle Solaris 11 VM Image copied to your laptop hard disk.
- Unzip the Solaris 11 image to your hard disk
- In the VirtualBox Manager Screen click 'New'
- Click Next on the welcome screen

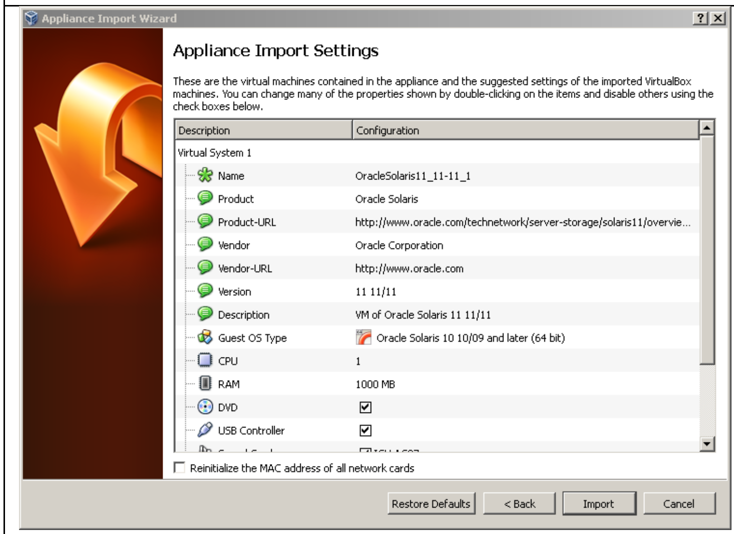
<table>
<tr><td>
</td><td>Click on the File Menu and choose "Import Appliance …"</td></tr>
<tr><td>
</td><td>Welcome to the Appliance Import Wizard! Click the Choose … button.</td></tr>
</table>



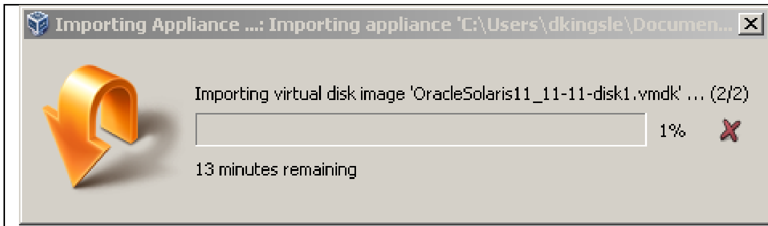
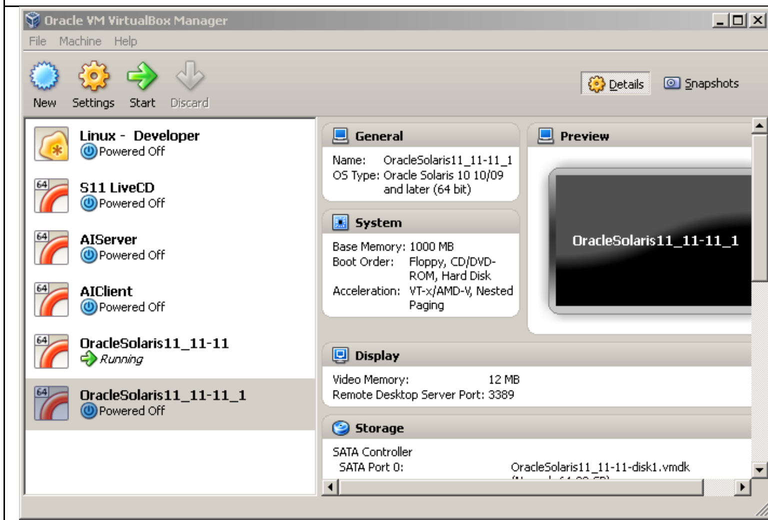
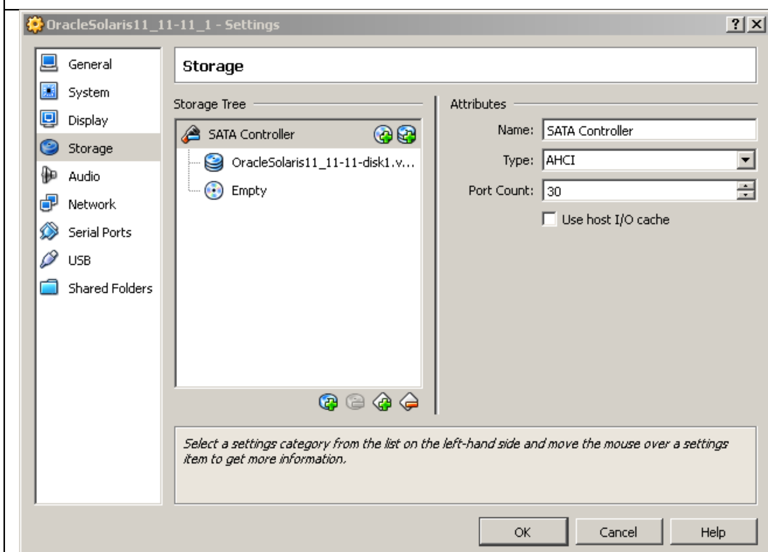

Navigate to the folder where you downloaded or copied the Oracle Solaris 11 PreBuilt image and click Open. The file is named OracleSolaris11_11-11.ovf. The .ovf extension indicates that it's a virtual box export.

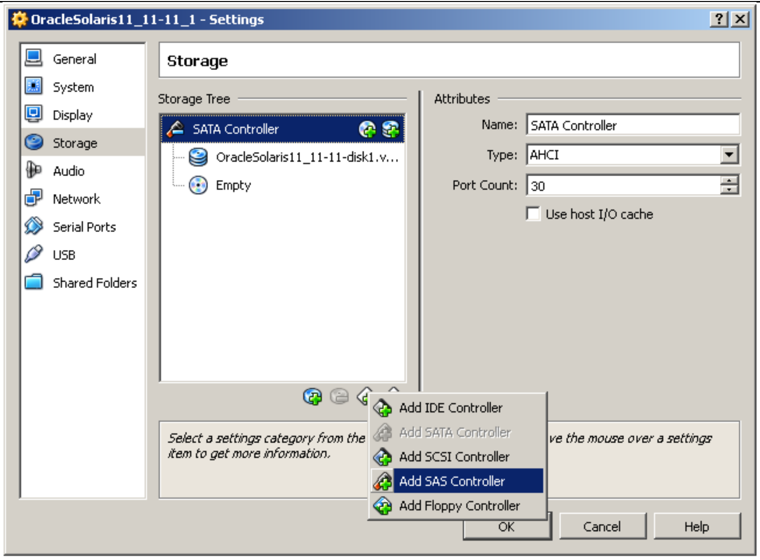
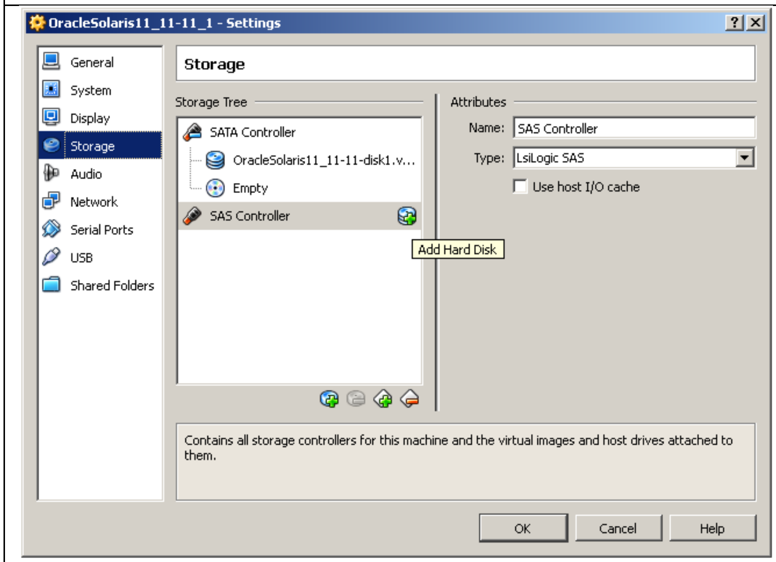


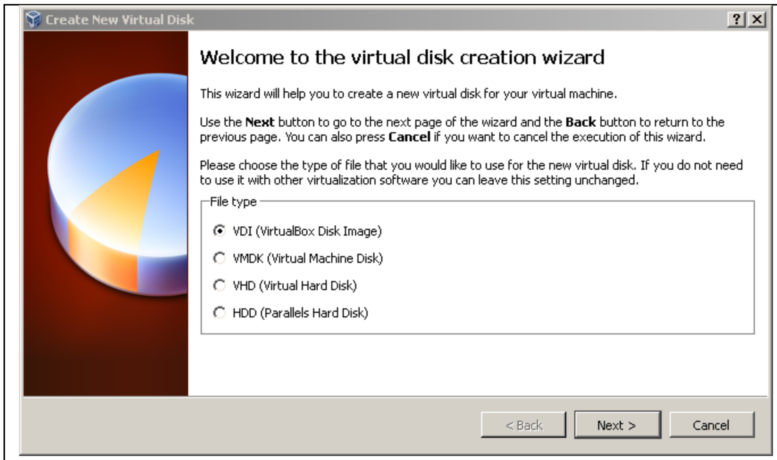

Next to the 'Choose' button it will indicate the file you've chosen to import. Click Next >



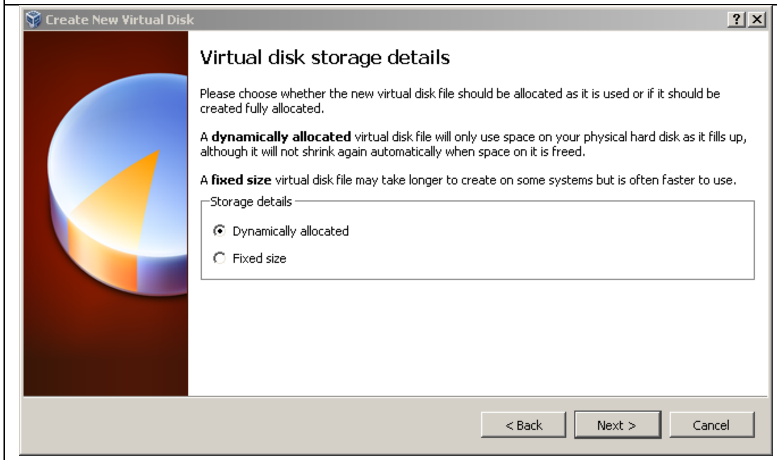

You'll be presented with the Appliance Import settings. We will accept the defaults. Click Import.

| | |
|---|---|
|  | The progress bar will show the import progress. Usually looks slow in the beginning but this shouldn't take more than a few minutes. |
|  | Your new image has been imported and is ready for use. Let's just make some simple changes, add some disks, and we'll be ready for the labs. |
|  | Make sure your new image is selected and choose "Settings", "Storage" |

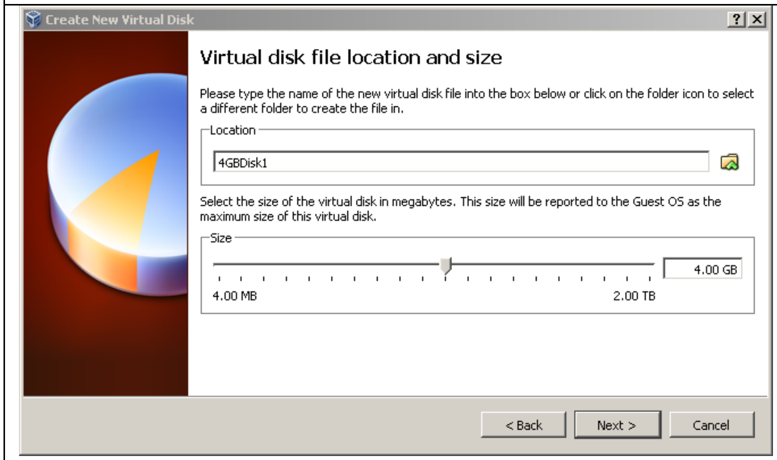
<table>
<tr>
<td>
</td>
<td>On the bottom of  the ‘Storage Tree’ section click on the green plus sign to “Add a new SAS controller’ to our virtual image.</td>
</tr>
<tr>
<td>
</td>
<td>Click on the disk icon to the right of the SAS controller you just added to add a new hard disk.</td>
</tr>
<tr>
<td>
</td>
<td>In the resulting dialog box, choose ‘Create new disk’</td>
</tr>
</table>



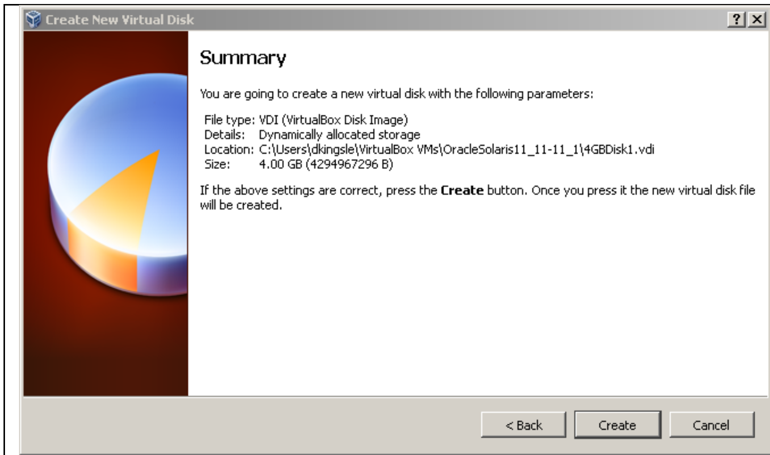

The virtual disk creation wizard will start up. Choose the default selection of VDI and click Next >



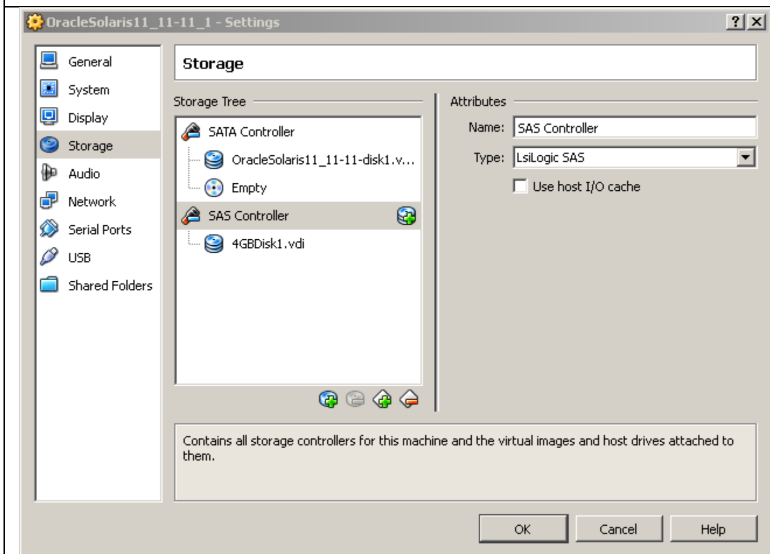

Click Next > for Dynamically Allocated



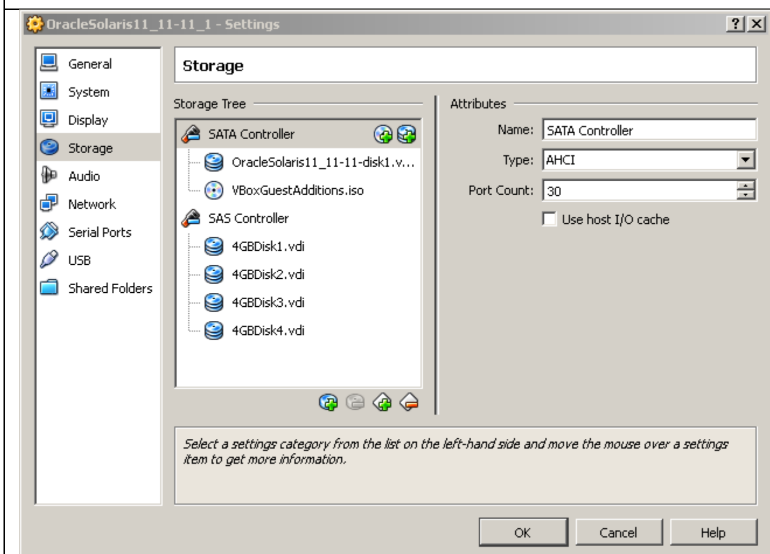

Name the disk '4GBDisk1' and either choose 4GB with the slider or type in the number 4.00 GB into the text box. Choose Next >




You'll receive a Summary of your disk selection. If all looks good, click Create.




After your disk is created you should now have a virtual disk named 4GBDisk1 underneath your new SAS controller.

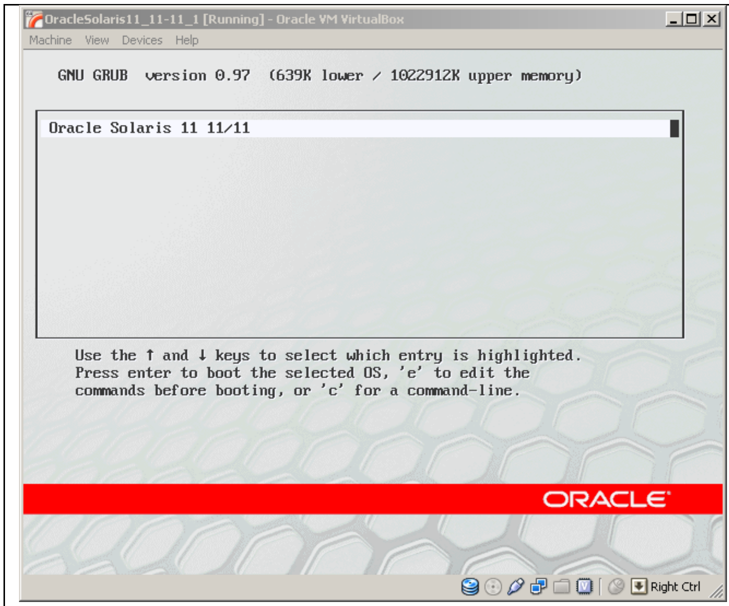


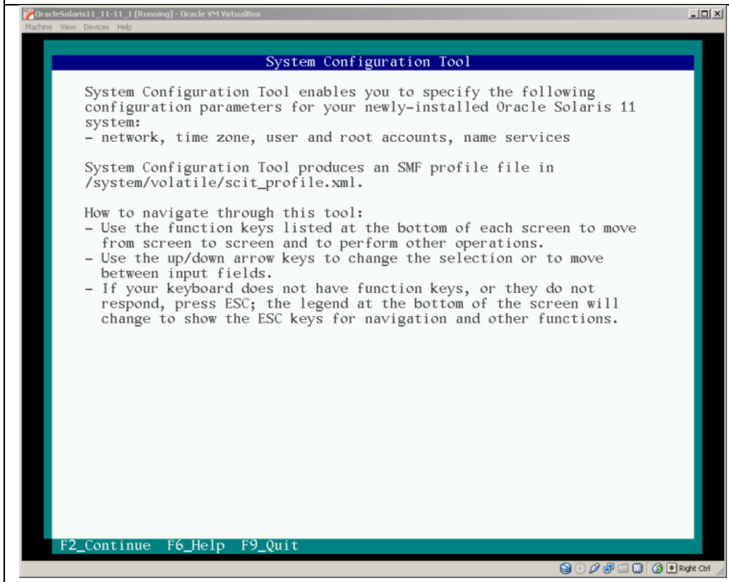

Repeat the above steps to create 3 more 4GB virtual disks under the same SAS controller and name them 4GBDisk2, 4GBDisk3, and 4GBDisk4 respectively. Verify your settings match the scre

en to the left.

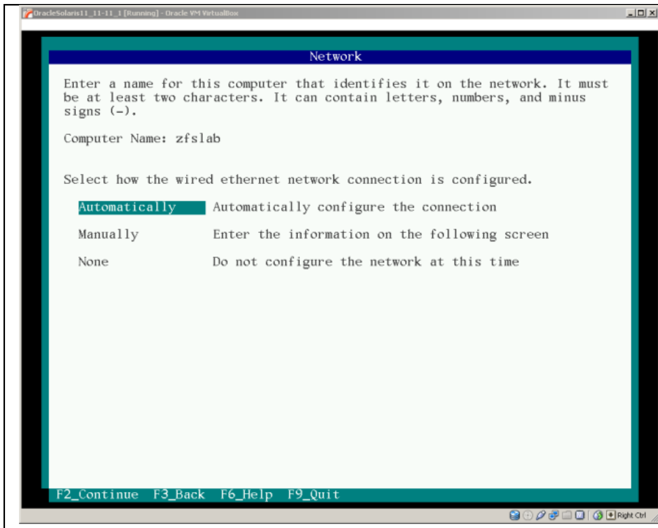
## 4.3 Install and Configure Oracle Solaris 11 Virtual Image

In this lab we are utilizing a pre-built Oracle Solaris Image. This image is based on the desktop version and we will be running in the Gnome environment. Even though the system has been pre-installed, we still have to answer the basic installation questions in order to get things running for the ZFS lab. The system will run through the basic set up dialog as illustrated below.

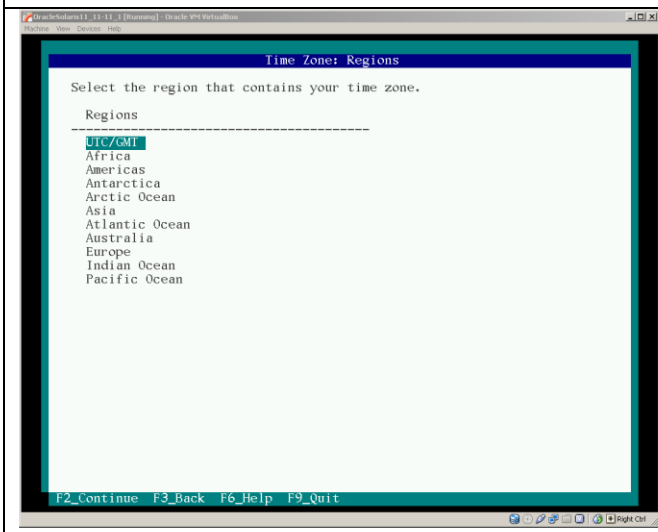
<table>
<tr>
<td>


</td>
<td>Upon startup you'll be presented with the GRUB menu. Select Oracle Solaris 11 11/11 and press the return/enter key</td>
</tr>
<tr>
<td>


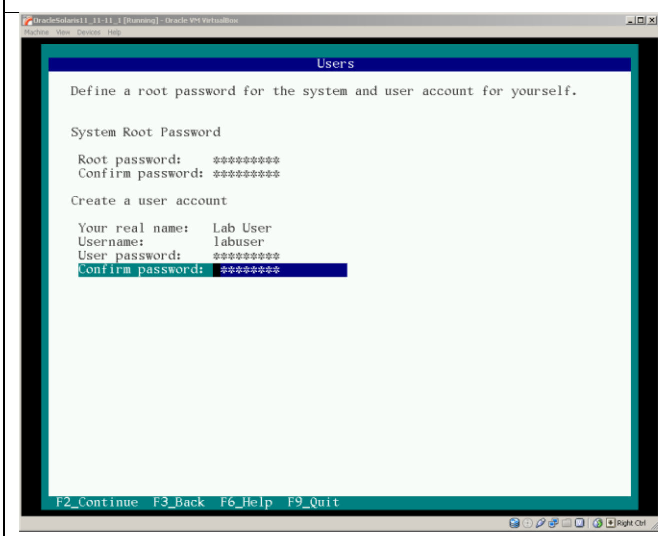
</td>
<td>After a few minutes you should see the SCI welcome screen where you'll answer a few simple questions in order to bring the system up. You should recognize this step from earlier installation labs. Press the <F2> key.</td>
</tr>
</table>




In the next screen we'll name our system and choose how we want to configure networking. Name your system 'zfslab' and choose 'Automatically' for network configuration. We will not use any networking for this lab so don't worry about this selection.




The next screens will ask about your region, time zone, etc. Choose UTC/GMT to avoid the time zone screens or select your local time zone.



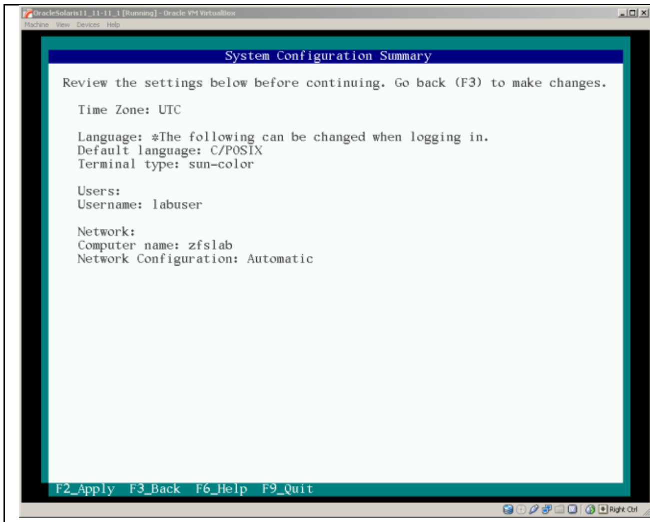

The next screen will ask for the Root password, and to create a user account. Use the below values if you wish for consistency in the labs.

Root password: **solaris11**

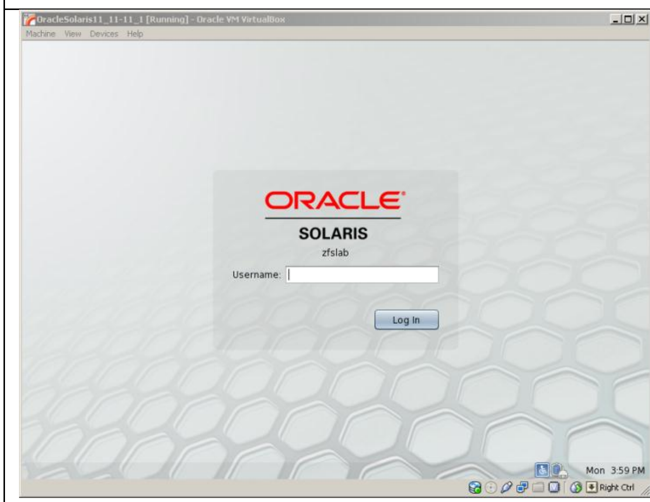
Your real name: **Lab User**

Username: **labuser**

User password: **solaris11**

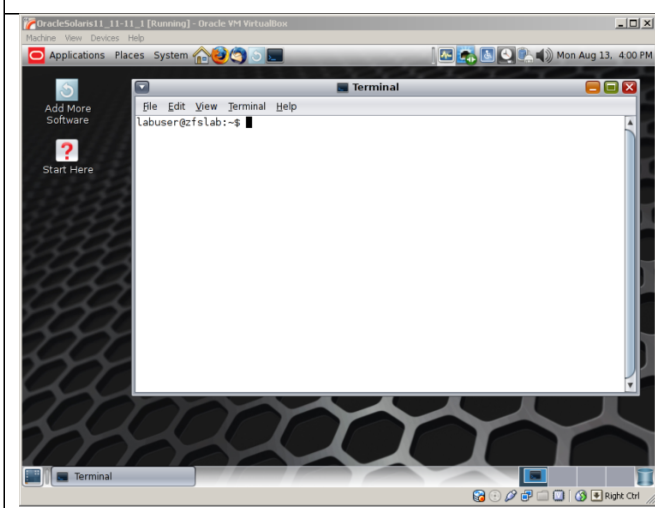



You should see a confirmation screen. Review and press <F2> to proceed with configuration.




The System Configuration Tool will exit and the system will come up with your new configuration parameters.

Wait for the GNOME login screen and login as **labuser**, password, **solaris11**.




We'll do our labs from terminal windows within the Oracle Solaris 11 GUI. Congratulations, you're up and running! Now let's get started with ZFS.

# 5 Working with Pools

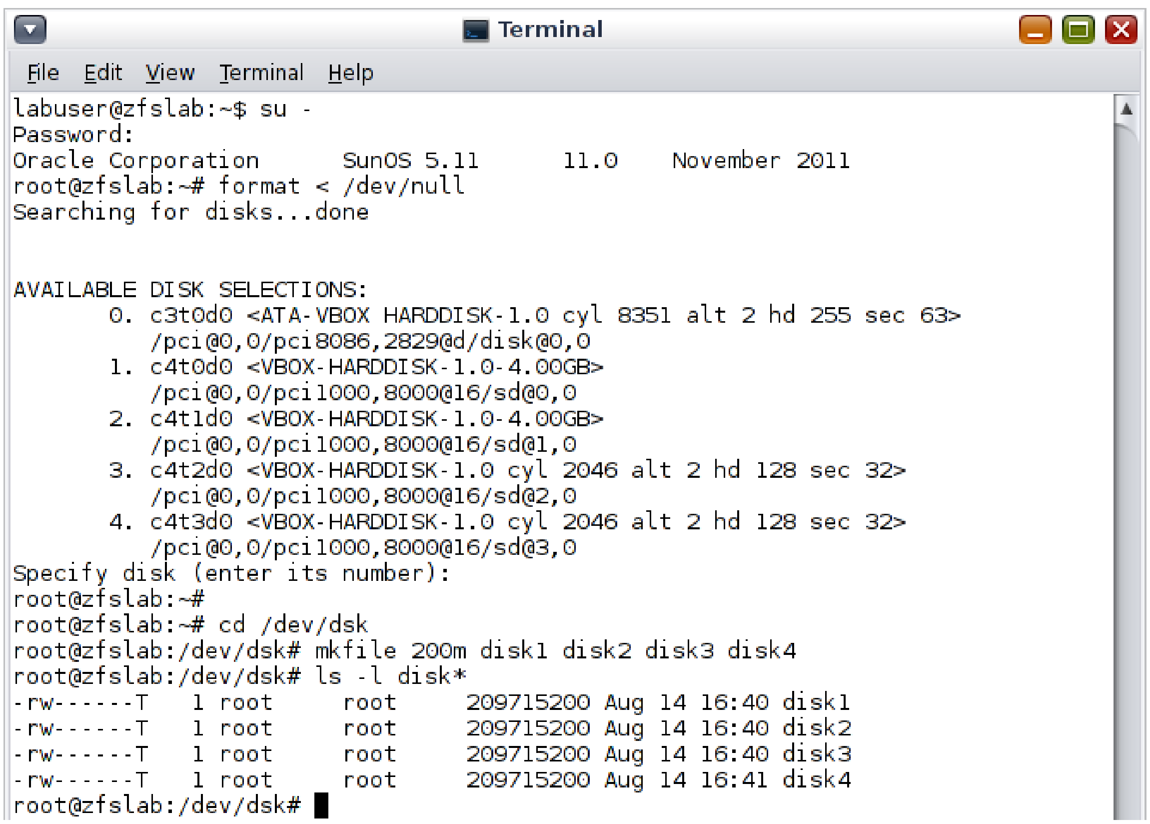
## 5.1 Verifying the SAS and flat file disk devices

It's easier to work as root during the labs, remember to su - to root when first logging in because root is a role and not a user. Let's 'su' to root, confirm our environment, and create some disk files before we get started with ZFS.

'`su -`' to root,

to confirm that our 4 SAS disks are available to the system. `cd` to the /dev/dsk directory and create 4 disk files using the mkfile command.

```
# format < /dev/null
# cd /dev/dsk
# mkfile 200m disk1 disk2 disk3 disk4
# ls -l disk*
```



```
labuser@zfslab:~$ su -
Password:
Oracle Corporation      SunOS 5.11      11.0    November 2011
root@zfslab:~# format < /dev/null
Searching for disks...done


AVAILABLE DISK SELECTIONS:
       0. c3t0d0 <ATA-VBOX HARDDISK-1.0 cyl 8351 alt 2 hd 255 sec 63>
          /pci@0,0/pci8086,2829@d/disk@0,0
       1. c4t0d0 <VBOX-HARDDISK-1.0-4.00GB>
          /pci@0,0/pci1000,8000@16/sd@0,0
       2. c4t1d0 <VBOX-HARDDISK-1.0-4.00GB>
          /pci@0,0/pci1000,8000@16/sd@1,0
       3. c4t2d0 <VBOX-HARDDISK-1.0 cyl 2046 alt 2 hd 128 sec 32>
          /pci@0,0/pci1000,8000@16/sd@2,0
       4. c4t3d0 <VBOX-HARDDISK-1.0 cyl 2046 alt 2 hd 128 sec 32>
          /pci@0,0/pci1000,8000@16/sd@3,0
Specify disk (enter its number):
root@zfslab:~#
root@zfslab:~# cd /dev/dsk
root@zfslab:/dev/dsk# mkfile 200m disk1 disk2 disk3 disk4
root@zfslab:/dev/dsk# ls -l disk*
-rw------T   1 root     root     209715200 Aug 14 16:40 disk1
-rw------T   1 root     root     209715200 Aug 14 16:40 disk2
-rw------T   1 root     root     209715200 Aug 14 16:40 disk3
-rw------T   1 root     root     209715200 Aug 14 16:41 disk4
root@zfslab:/dev/dsk#
```

Confirm you've created the four disk files.

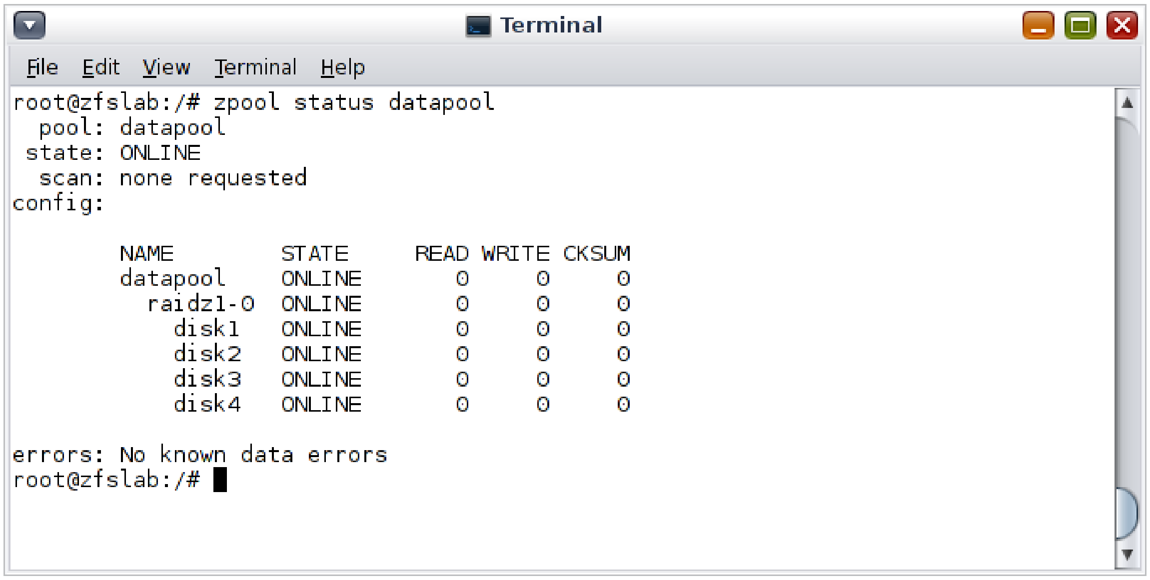
## 5.2 Creating and destroying pools

To create your first pool ...

```
# zpool create datapool raidz disk1 disk2 disk3 disk4
```

That's all there is to it. We can use *zpool status* to see what our first pool looks like.

```
# zpool status datapool
```




Note from this output that the pool names *datapool* has a single ZFS virtual device (vdev) called ***raidz1-0***. That vdev is comprised of our four disk files that we created in the previous step.
The RAIDZ1-0 type vdev provides single device parity protection, meaning that if one device develops an error, no data is lost because it can be reconstructed using the remaining disk devices. This organization is commonly called a 3+1, 3 data disks plus one parity.

ZFS provides additional types of availability: raidz2 (2 device protection), raidz3 (3 device protection), mirroring and none. We will look at some of these in later exercises.

Before continuing, let's take a look at the currently mounted file systems.

```
# zfs list
```

```
root@zfslab:/# zfs list
NAME                          USED  AVAIL  REFER  MOUNTPOINT
datapool                     97.2K   551M  44.9K  /datapool
rpool                        6.02G  56.5G    39K  /rpool
rpool/ROOT                   4.48G  56.5G    31K  legacy
rpool/ROOT/solaris           4.48G  56.5G  4.05G  /
rpool/ROOT/solaris/var        332M  56.5G   182M  /var
rpool/dump                    516M  56.5G   500M  -
rpool/export                 1.77M  56.5G    32K  /export
rpool/export/home            1.74M  56.5G   720K  /export/home
rpool/export/home/labuser    1.00M  56.5G  1.00M  /export/home/labuser
rpool/export/home/releng       31K  56.5G    31K  /export/home/releng
rpool/swap                   1.03G  56.5G  1.00G  -
root@zfslab:/#
```

One thing to notice in the ZFS makes things easier category is that when we created the ZFS pool with one simple command, ZFS also created the first file system and also mounted it. The default mountpoint is derived from the name of the pool but can be changed easily.

> ***Note:***
> Things we no longer have to do with ZFS are …
> - Create a filesystem
> - Make a directory to mount the filesystem
> - Add entries to /etc/vfstab

We've decided that we need a different type of vdev for our datapool example. Let's destroy this pool and create another.

```
# zpool destroy datapool
# zfs list
```

```
Terminal
File  Edit  View  Terminal  Help
root@zfslab:/# zpool destroy datapool
root@zfslab:/# zfs list
NAME                          USED  AVAIL  REFER  MOUNTPOINT
rpool                        6.02G  56.5G    39K  /rpool
rpool/ROOT                   4.48G  56.5G    31K  legacy
rpool/ROOT/solaris           4.48G  56.5G  4.05G  /
rpool/ROOT/solaris/var        332M  56.5G   182M  /var
rpool/dump                    516M  56.5G   500M  -
rpool/export                 1.77M  56.5G    32K  /export
rpool/export/home            1.74M  56.5G   720K  /export/home
rpool/export/home/labuser    1.00M  56.5G  1.00M  /export/home/labuser
rpool/export/home/releng       31K  56.5G    31K  /export/home/releng
rpool/swap                   1.03G  56.5G  1.00G  -
```

All file systems in the pool have been unmounted and the pool has been destroyed. The devices in the vdev have also been marked as free so they can be used again. Notice how easy it is to destroy and there's no 'destroy? Are you sure?' warning.

Next, let's create a simple pool using a 2 way mirror instead of raidz.

```
# zpool create datapool mirror disk1 disk2
```

```
Terminal
File  Edit  View  Terminal  Help
root@zfslab:/# zpool create datapool mirror disk1 disk2
root@zfslab:/# zpool status datapool
  pool: datapool
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        datapool    ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            disk1   ONLINE       0     0     0
            disk2   ONLINE       0     0     0

errors: No known data errors
```

Now the vdev name has changed to *mirror-0* to indicate that data redundancy is provided by mirroring (redundant copies of the data).

What happens if you try to use a disk device that is already being used by another pool?

```
# zpool create datapool2 mirror disk1 disk2
```

```
Terminal
File  Edit  View  Terminal  Help
root@zfslab:/# zpool create datapool2 mirror disk1 disk2
invalid vdev specification
use '-f' to override the following errors:
/dev/dsk/disk1 is part of active pool 'datapool'
```

The usage error indicates that */dev/dsk/disk1* has been identified as being part of an existing pool called *datapool*. The -f flag to the *zpool create* command can override the failsafe in case datapool is no longer being used, but use that option with caution.

## 5.3 Adding capacity to a pool

Our application has made it necessary to add more space the the 'datapool'. The next exercise will show you how simple it is to add capacity to an existing pool.

```
# zpool list datapool
# zpool add datapool mirror disk3 disk4
# zpool status datapool
```

```
Terminal
File  Edit  View  Terminal  Help
root@zfslab:/# zpool list datapool
NAME       SIZE  ALLOC  FREE  CAP  DEDUP  HEALTH  ALTROOT
datapool  195M    94K  195M   0%  1.00x  ONLINE  -
root@zfslab:/# zpool add datapool mirror disk3 disk4
root@zfslab:/# zpool status datapool
  pool: datapool
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        datapool    ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            disk1   ONLINE       0     0     0
            disk2   ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            disk3   ONLINE       0     0     0
            disk4   ONLINE       0     0     0

errors: No known data errors
```

Note that a second vdev (mirror-1) has been added to the pool.

To see if your pool has actually grown, do another # `zfs list` command.

```
# zfs list datapool
```

```
root@zfslab:/# zfs list datapool
NAME       USED  AVAIL  REFER  MOUNTPOINT
datapool   318K   358M    31K  /datapool
```

Datapool has grown from 195MB to 358MB

Notice that you don't have to grow file systems when the pool capacity increases. File systems can use whatever space is available in the pool, subject to quota limitations, which we will examine in a later exercise.

## 5.4 Importing and exporting pools

ZFS zpools can also be exported, allowing all of the data and associated configuration information to be moved from one system to another. For this example, use the two SAS disks (*c4t0d0* and *c4t1d0*).

```
# zpool create pool2 mirror c4t0d0 c4t1d0
# zpool status pool2
```

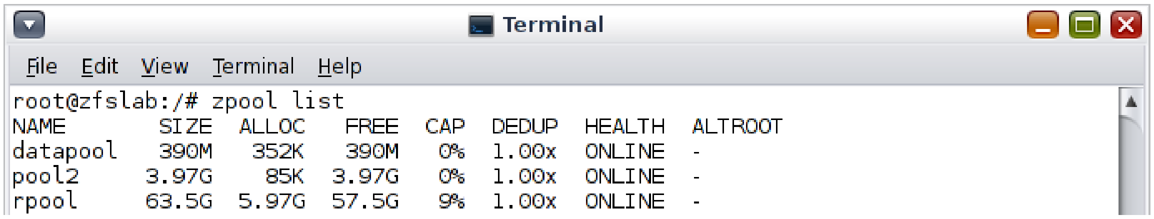
```
root@zfslab:/# zpool create pool2 mirror c4t0d0 c4t1d0
root@zfslab:/# zpool status pool2
  pool: pool2
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        pool2       ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            c4t0d0  ONLINE       0     0     0
            c4t1d0  ONLINE       0     0     0

errors: No known data errors
```

As before, we have created a simple mirrored pool of two disks. In this case, the disk devices are real disks, not files. We've told ZFS to use the entire disk (no slice number was included) and if the disk was not labeled, ZFS will write a default label.

ZFS Storage pools can be exported in order to migrate them easily to other system. Storage pools should be explicitly exported to indicate that they are ready to be migrated. This operation flushes any unwritten data to disk, writes data to the disk indicating that the export was done, and removes all knowledge of the pool from the system.

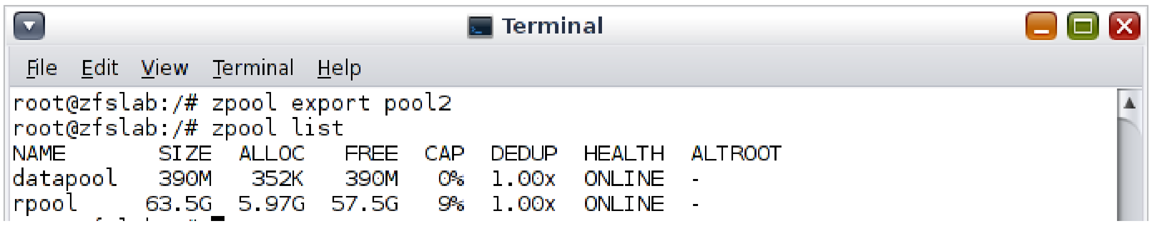
Let's export pool2 so that another system can use it.

```
# zpool list
```




```
# zpool export pool2
```

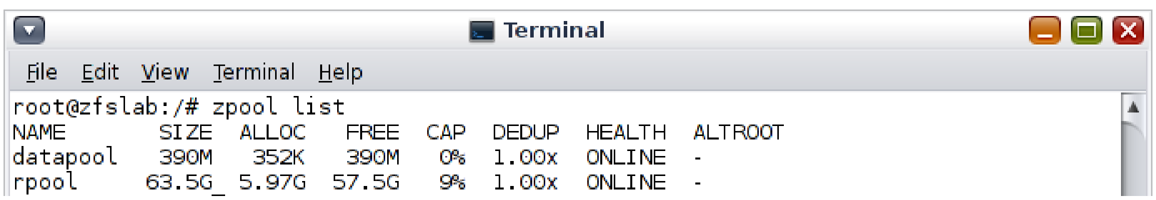
```
# zpool list
```



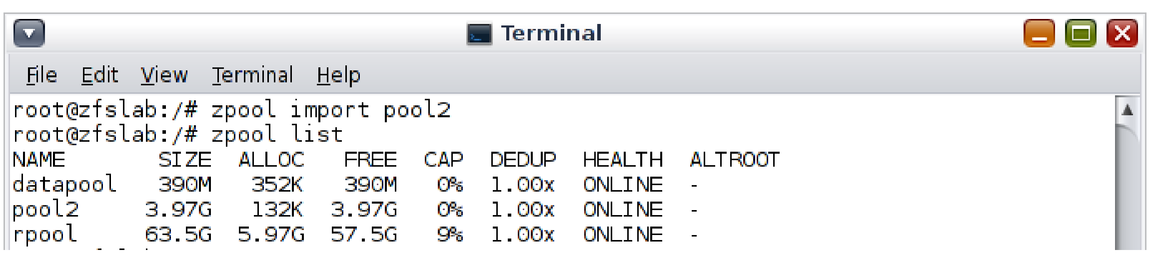

Note that our pool, 'pool2' is no longer in our list of available pools.

The next step will be to import the pool, again showing how easy ZFS is to use.

```
# zpool list
```



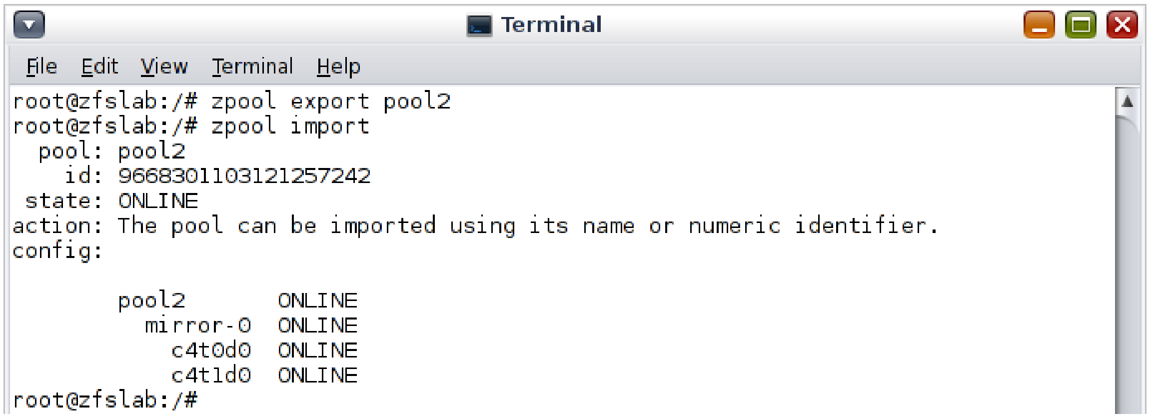

```
# zpool import pool2
# zpool list
```




Notice that we didn't have to tell ZFS where the disks were located. All we told ZFS was the name of the pool. ZFS looked through all of the available disk devices and reassembled the pool, even if the device names had been changed.

If you don't know the name of the pool ZFS will provide the names of available pools.

```
# zpool export pool2
# zpool import
```




Import your pool.

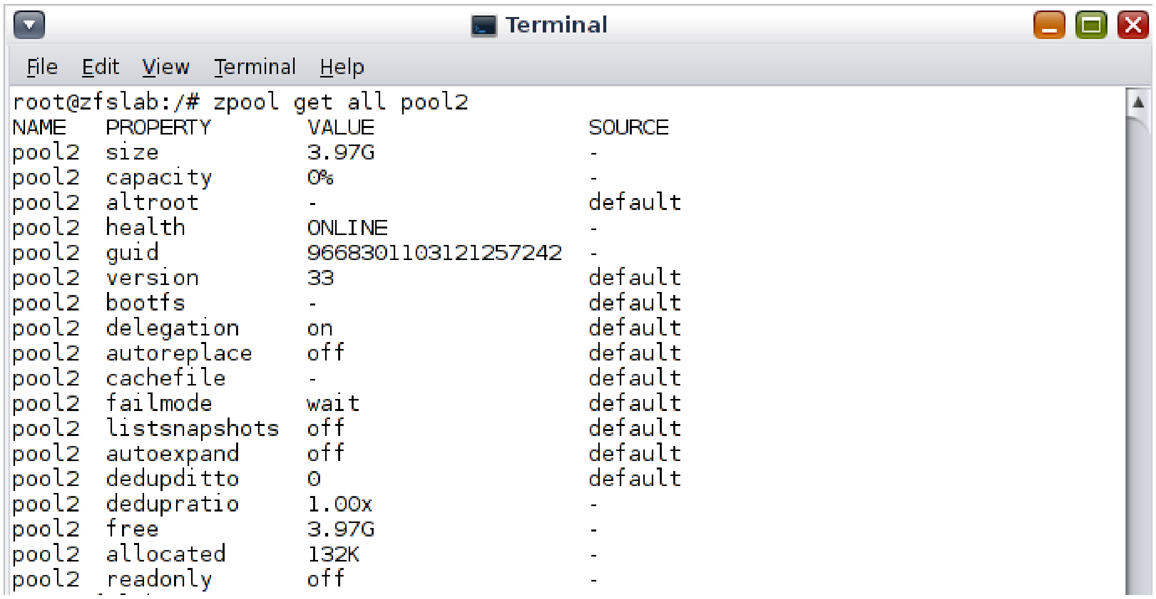
```
# zpool import pool2
```

Without an argument, ZFS will look at all of the disks attached to the system and will provide a list of pool names that it can import. If it finds two pools of the same name, the unique identifier can be used to select which pool you want imported.

## 5.5 Pool properties

There are many pool properties that can be customized for your environment. To see a list of these properties type the following command.

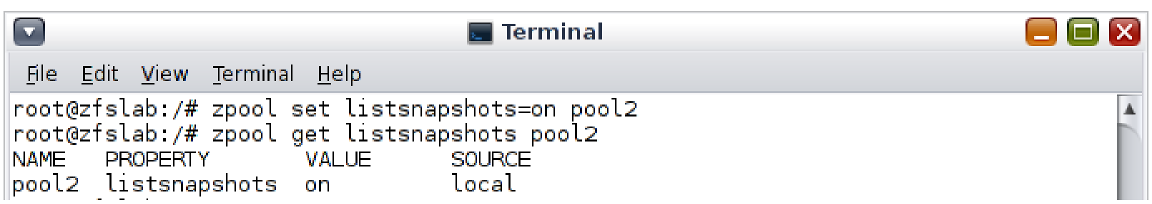
```
# zpool get all pool2
```




Pool properties are described in the zpool(1M) man page. Pool properties provide information about the pool, effect performance, security, and availability. To set a pool property, use `zpool set`. Note that not all properties can be changed (ex. version, free, allocated).

Set the 'listsnapshot' property to 'on'. The listsnapshot (also listsnaps) controls whether information about snapshots is displayed when the 'zfs list' command is run without the –t option. The default value is 'off'.

```
# zpool set listsnapshots=on pool2
# zpool get listsnapshots pool2
```




## 5.6 Upgrading pools

As with any software package, ZFS goes through upgrades over time. Let's take a look at how we can find the zpool version number, features provided by that version, and how we can potentially upgrade, or even downgrade our ZFS Pool version to accommodate potential compatibility scenarios. To find out what features have been added over time, use the below command.

```
# zpool upgrade -v
```

```
root@zfslab:/# zpool upgrade -v
This system is currently running ZFS pool version 33.

The following versions are supported:

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

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

When you patch or upgrade Oracle Solaris, a new version of zpool may be available. It is simple to upgrade or downgrade an existing pool. We'll create a pool using an older version number, and then upgrade the pool.

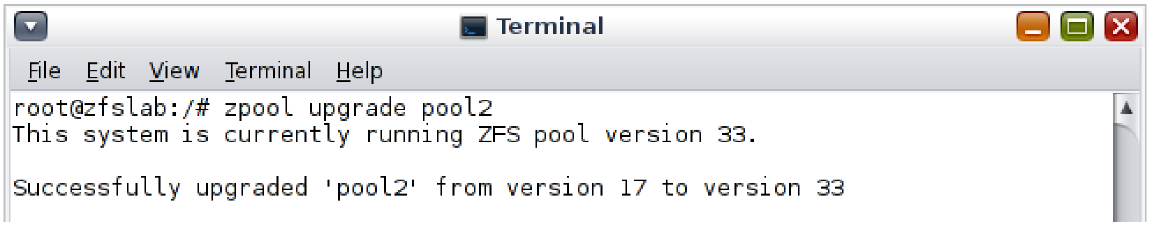
```
# zpool destroy pool2
# zpool create -o version=17 pool2 mirror c4t0d0 c4t1d0
# zpool get version pool2
```

```
root@zfslab:/# zpool destroy pool2
root@zfslab:/# zpool create -o version=17 pool2 mirror c4t0d0 c4t1d0
root@zfslab:/# zpool get version pool2
NAME   PROPERTY  VALUE    SOURCE
pool2  version   17       local
```

Note that your pool is now at version 17.

The next step would be to upgrade the old pool to the latest version. Execute the following commands.

```
# zpool upgrade pool2
```




It's that simple. Now you can use features provided in the newer zpool version, like log device removal (19), snapshot user holds (18), etc.

This concludes the section on pools. There is a wealth of features that we haven't explored yet. Check out the man page for many other features that you can take advantage of.

Let's clean up before proceeding to the next lab.

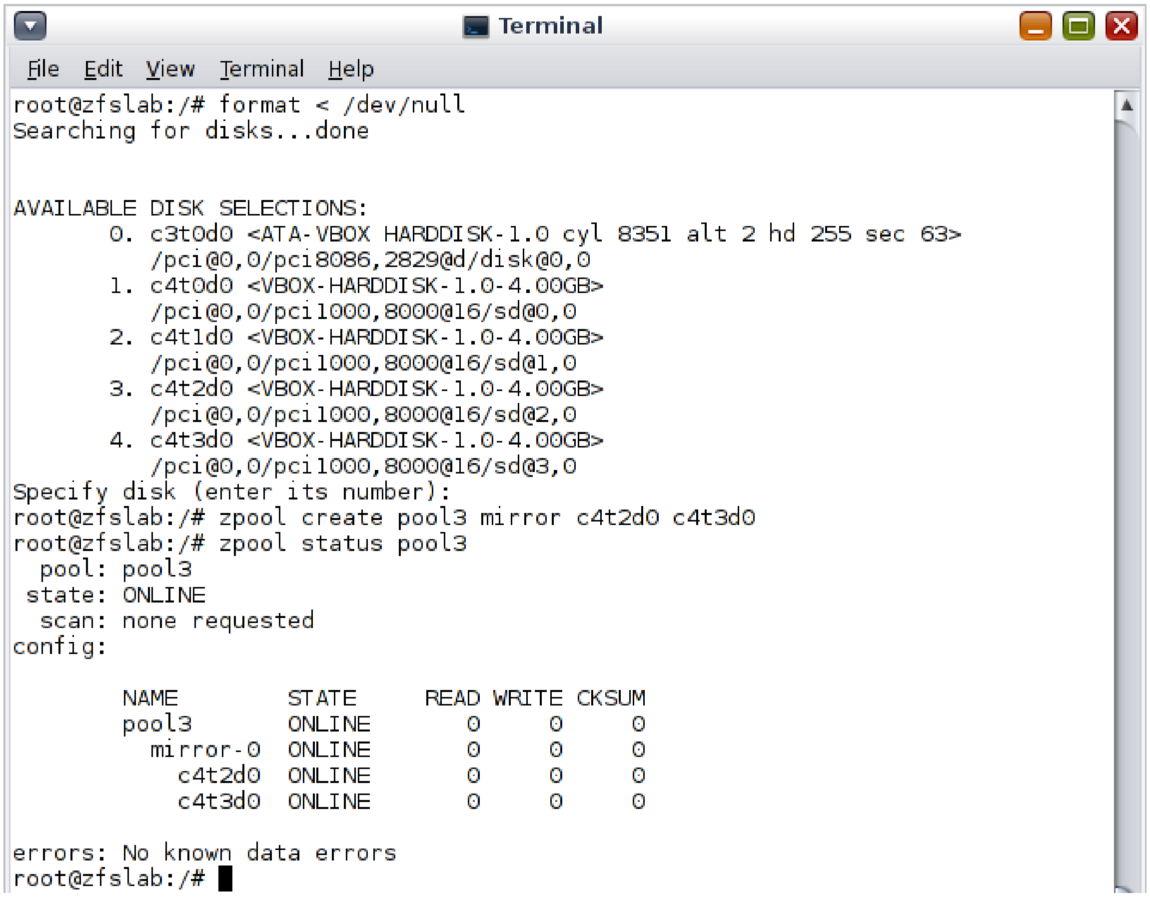
```
# zpool destroy pool2
# zpool destroy datapool
```

## 5.1 ZFS split command

A mirrored ZFS storage pool can be quickly cloned as a backup pool by using the zpool split command.

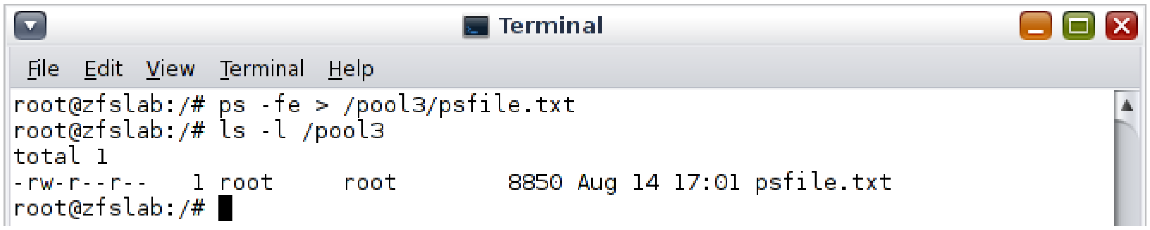
First let's create a mirrored ZFS pool named pool3 with two of our SAS disks.

```
# format < /dev/null
# zpool create pool3 mirror c4t2d0 c4t3d0
# zpool status pool3
```




Remember that our pool is automatically mounted so let's go ahead and create some data and store it in the resulting file system.

```
# ps –fe > /pool3/psfile.txt
# ls –l /pool3
```




First let's check the status of the file system for size and then let's split the pool and create our instant backup copy. We will provide a name for the resulting second pool and call it 'pool4'.

```
# zfs list pool3
# zpool split pool3 pool4
# zpool status
# zfs list pool3
```

```
root@zfslab:/# zfs list pool3
NAME    USED  AVAIL  REFER  MOUNTPOINT
pool3    94K  3.91G    40K  /pool3
root@zfslab:/# zpool split pool3 pool4
root@zfslab:/# zpool status
  pool: pool3
 state: ONLINE
  scan: none requested
config:

        NAME      STATE     READ WRITE CKSUM
        pool3     ONLINE       0     0     0
          c4t2d0  ONLINE       0     0     0

errors: No known data errors

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

        NAME        STATE     READ WRITE CKSUM
        rpool       ONLINE       0     0     0
          c3t0d0s0  ONLINE       0     0     0

errors: No known data errors
root@zfslab:/# zfs list pool3
NAME     USED  AVAIL  REFER  MOUNTPOINT
pool3  95.5K  3.91G    40K  /pool3
root@zfslab:/#
```

Note that our pool now only contains a single disk but the size is still the same. And running the ls command shows that our file is still there and has not come to any harm.

```
root@zfslab:/# ls -l /pool3
total 19
-rw-r--r--   1 root     root        8850 Aug 14 17:01 psfile.txt
root@zfslab:/#
```

Our new pool doesn't show up in the list because it still needs to be imported. Let's do that now.

```
# zpool import pool4
# zpool status pool3 pool4
```

```
root@zfslab:/# zpool import pool4
root@zfslab:/# zpool status pool3 pool4
  pool: pool3
 state: ONLINE
  scan: none requested
config:

        NAME      STATE     READ WRITE CKSUM
        pool3     ONLINE       0     0     0
          c4t2d0  ONLINE       0     0     0

errors: No known data errors

  pool: pool4
 state: ONLINE
  scan: none requested
config:

        NAME      STATE     READ WRITE CKSUM
        pool4     ONLINE       0     0     0
          c4t3d0  ONLINE       0     0     0

errors: No known data errors
root@zfslab:/#
```

That confirms our split pools. Now let's verify that our file has been duplicated in the filesystem.

```
# ls -l /pool3
# ls -l /pool4
```

```
root@zfslab:/# ls -l /pool3
total 19
-rw-r--r--   1 root     root        8850 Aug 14 17:01 psfile.txt
root@zfslab:/# ls -l /pool4
total 19
-rw-r--r--   1 root     root        8850 Aug 14 17:01 psfile.txt
root@zfslab:/#
```

Now just for the heck of it, let's put the mirror back together. If this were a production system you would ensure that complete and proper backups were done before playing with splits and joins like this in a filesystem no matter how trustworthy the software may be.

First we'll need to destroy pool4 because it has the disk we want to put back into the mirror. Then we'll use the attach subcommand to bring a new disk into our nonredundant single disk pool as a mirror. With the attach command you need to list the existing device first and then the device you wish to join into the mirror.

```
# zpool destroy pool4
```

```
# zpool status pool3 pool4
# zpool attach pool3 c4t2d0 c4t3d0
# zpool status pool3
```

```
root@zfslab:/# zpool status pool3 pool4
cannot open 'pool4': no such pool
  pool: pool3
 state: ONLINE
  scan: resilvered 100K in 0h0m with 0 errors on Tue Aug 14 17:16:28 2012
config:

        NAME      STATE     READ WRITE CKSUM
        pool3     ONLINE       0     0     0
          c4t2d0  ONLINE       0     0     0

errors: No known data errors
root@zfslab:/# zpool attach pool3 c4t2d0 c4t3d0
root@zfslab:/# zpool status pool3
  pool: pool3
 state: ONLINE
  scan: resilvered 112K in 0h0m with 0 errors on Tue Aug 14 17:22:21 2012
config:

        NAME        STATE     READ WRITE CKSUM
        pool3       ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            c4t2d0  ONLINE       0     0     0
            c4t3d0  ONLINE       0     0     0

errors: No known data errors
root@zfslab:/#
```

The mirrored pool is now back to normal and the file it contained is still intact.

```
# ls –l /pool3
```

```
root@zfslab:/# ls -l /pool3
total 19
-rw-r--r--   1 root     root        8850 Aug 14 17:01 psfile.txt
root@zfslab:/#
```

## 6 Working with datasets (file systems and volumes)

Now that we understand pools, the next topic is file systems. We will use the term datasets because a zpool can provide many different types of access, not just through traditional file systems.

As we saw in the earlier exercise, a default dataset (file system) is automatically created when creating a zpool. Unlike other file system and volume managers, ZFS provides hierarchical datasets (peer, parents, children), allowing a single pool to provide many storage choices.

ZFS datasets are created, destroyed and managed using the zfs(1M) command.

## 6.1 Dataset lab setup

To begin working with datasets, let's create a simple pool, again called datapool and 4 additional datasets called bob joe fred and pat. Execute the following commands on your system…

```
# zpool create datapool mirror c4t0d0 c4t1d0
# zfs create datapool/bob
# zfs create datapool/joe
# zfs create datapool/fred
# zfs create datapool/pat
```

Use the '`zfs list -r datapool`' command to confirm your work.

```
Terminal
File  Edit  View  Terminal  Help
root@zfslab:/# zpool create datapool mirror c4t0d0 c4t1d0
root@zfslab:/# zfs create datapool/bob
root@zfslab:/# zfs create datapool/joe
root@zfslab:/# zfs create datapool/fred
root@zfslab:/# zfs create datapool/pat
root@zfslab:/#
root@zfslab:/# zfs list -r datapool
NAME            USED  AVAIL  REFER  MOUNTPOINT
datapool        240K  3.91G    35K  /datapool
datapool/bob     31K  3.91G    31K  /datapool/bob
datapool/fred    31K  3.91G    31K  /datapool/fred
datapool/joe     31K  3.91G    31K  /datapool/joe
datapool/pat     31K  3.91G    31K  /datapool/pat
```

By using `zfs list -r datapool`, we are listing all of the datasets in the pool named *datapool*. As in the earlier exercise, all of these datasets (file systems) have been automatically mounted.

If this was a traditional file system, you might think there was 19.55 GB (3.81 GB x 5) available for datapool and its 4 datasets, but the 4GB in the pool is shared across all of the datasets. To see an example of this behavior, type the following commands:

```
# mkfile 1024m /datapool/bob/bigfile
# zfs list -r datapool
```

```
root@zfslab:/# mkfile 1024m /datapool/bob/bigfile
root@zfslab:/# zfs list -r datapool
NAME             USED  AVAIL  REFER  MOUNTPOINT
datapool         982M  2.95G    35K  /datapool
datapool/bob     981M  2.95G   981M  /datapool/bob
datapool/fred     31K  2.95G    31K  /datapool/fred
datapool/joe      31K  2.95G    31K  /datapool/joe
datapool/pat      31K  2.95G    31K  /datapool/pat
```

Notice that in the USED column, datapool/bob shows 1GB in use. The other datasets show just the metadata overhead (31k), but their available space has been reduced to 2.95GB, the amount of free space available to them after the consumption of 1GB by the datapool/bob dataset.

## 6.2 Hierarchical datasets

A dataset can have children, just as a directory can have subdirectories. For datapool/fred, let's create a dataset for documents, and then underneath that, additional datasets for pictures, video and audio. Execute the following commands:

```
# zfs create datapool/fred/documents
# zfs create datapool/fred/documents/pictures
# zfs create datapool/fred/documents/video
# zfs create datapool/fred/documents/audio

# zfs list -r datapool
```

```
root@zfslab:/# zfs create datapool/fred/documents
root@zfslab:/# zfs create datapool/fred/documents/pictures
root@zfslab:/# zfs create datapool/fred/documents/video
root@zfslab:/# zfs create datapool/fred/documents/audio
root@zfslab:/#
root@zfslab:/# zfs list -r datapool
NAME                               USED  AVAIL  REFER  MOUNTPOINT
datapool                          1.00G  2.91G    35K  /datapool
datapool/bob                      1.00G  2.91G  1.00G  /datapool/bob
datapool/fred                      159K  2.91G    32K  /datapool/fred
datapool/fred/documents            127K  2.91G    34K  /datapool/fred/documents
datapool/fred/documents/audio       31K  2.91G    31K  /datapool/fred/documents/audio
datapool/fred/documents/pictures    31K  2.91G    31K  /datapool/fred/documents/pictures
datapool/fred/documents/video       31K  2.91G    31K  /datapool/fred/documents/video
datapool/joe                        31K  2.91G    31K  /datapool/joe
datapool/pat                        31K  2.91G    31K  /datapool/pat
```

## 6.3 ZFS dataset properties

ZFS datasets are flexible and can be manipulated with a myriad of properties. The next short exercise will examine some ZFS dataset properties and how to manipulate them and why.

```
root@zfslab:/# zfs get all datapool
NAME      PROPERTY              VALUE                  SOURCE
datapool  type                  filesystem             -
datapool  creation              Mon Aug 13 22:08 2012  -
datapool  used                  1.00G                  -
datapool  available             2.91G                  -
datapool  referenced            35K                    -
datapool  compressratio         1.00x                  -
datapool  mounted               yes                    -
datapool  quota                 none                   default
datapool  reservation           none                   default
datapool  recordsize            128K                   default
datapool  mountpoint            /datapool              default
datapool  sharenfs              off                    default
datapool  checksum              on                     default
datapool  compression           off                    default
datapool  atime                 on                     default
datapool  devices               on                     default
datapool  exec                  on                     default
datapool  setuid                on                     default
datapool  readonly              off                    default
datapool  zoned                 off                    default
datapool  snapdir               hidden                 default
datapool  aclmode               discard                default
datapool  aclinherit            restricted             default
datapool  canmount              on                     default
datapool  xattr                 on                     default
datapool  copies                1                      default
datapool  version               5                      -
datapool  utf8only              off                    -
datapool  normalization         none                   -
datapool  casesensitivity       mixed                  -
datapool  vscan                 off                    default
datapool  nbmand                off                    default
datapool  sharesmb              off                    default
datapool  refquota              none                   default
datapool  refreservation        none                   default
datapool  primarycache          all                    default
datapool  secondarycache        all                    default
datapool  usedbysnapshots       0                      -
datapool  usedbydataset         35K                    -
datapool  usedbychildren        1.00G                  -
datapool  usedbyrefreservation  0                      -
datapool  logbias               latency                default
datapool  dedup                 off                    default
datapool  mlslabel              none                   -
datapool  sync                  standard               default
datapool  encryption            off                    -
datapool  keysource             none                   default
datapool  keystatus             none                   -
datapool  rekeydate             -                      default
datapool  rstchown              on                     default
datapool  shadow                none                   -
```

As you can see, there are many dataset properties that can be set. For a complete explanation of each property, consult the zfs(1M) man page. We'll outline a few examples in the following exercise.

## 6.4 Quotas and reservations

ZFS dataset **quotas** are used to **limit** the amount of space consumed by a dataset and all of its children. **Reservations** are used to **guarantee** that a dataset has an allocated amount of storage that can't be consumed by other datasets in use.

To set quotas and reservations, use the zfs set command.

```
# zfs set quota=2g datapool/fred
# zfs set reservation=1.5G datapool/fred
# zfs list -r datapool
```

```
root@zfslab:/# zfs set quota=2g datapool/fred
root@zfslab:/# zfs set reservation=1.5g datapool/fred
root@zfslab:/# zfs list -r datapool
NAME                              USED  AVAIL  REFER  MOUNTPOINT
datapool                         2.50G  1.41G    35K  /datapool
datapool/bob                     1.00G  1.41G  1.00G  /datapool/bob
datapool/fred                     159K  2.00G    32K  /datapool/fred
datapool/fred/documents           127K  2.00G    34K  /datapool/fred/documents
datapool/fred/documents/audio      31K  2.00G    31K  /datapool/fred/documents/audio
datapool/fred/documents/pictures   31K  2.00G    31K  /datapool/fred/documents/pictures
datapool/fred/documents/video      31K  2.00G    31K  /datapool/fred/documents/video
datapool/joe                       31K  1.41G    31K  /datapool/joe
datapool/pat                       31K  1.41G    31K  /datapool/pat
```

The first thing to notice is that the available space for datapool/fred and all of its children is now 2GB, which was the quota we set with the command above. Also notice that the quota is inherited by all of the children.

The reservation is a bit harder to see.

Original pool size 3.91GB

In use by datapool/bob 1.0GB

Reservation by datapool/fred 1.5GB

So, datapool/joe should see 3.91GB - 1.0GB - 1.5GB = 1.4GB available.

## 6.5 Changing the mountpoint

With a traditional UNIX file systems changing a mountpoint would require a few steps, including …

- Unmounting the file system
- Making a new directory
- Editing /etc/vfstab
- Mounting the new file system

With ZFS it can be done with a single command. In the next example, let's move datapool/fred to a directory just called /fred. First let's look at the current mountpoint.

```
# zfs list -r datapool
```

```
Terminal
File  Edit  View  Terminal  Help
root@zfslab:/# zfs list -r datapool
NAME                              USED  AVAIL  REFER  MOUNTPOINT
datapool                         2.50G  1.41G    35K  /datapool
datapool/bob                     1.00G  1.41G  1.00G  /datapool/bob
datapool/fred                     159K  2.00G    32K  /datapool/fred
datapool/fred/documents           127K  2.00G    34K  /datapool/fred/documents
datapool/fred/documents/audio      31K  2.00G    31K  /datapool/fred/documents/audio
datapool/fred/documents/pictures   31K  2.00G    31K  /datapool/fred/documents/pictures
datapool/fred/documents/video      31K  2.00G    31K  /datapool/fred/documents/video
datapool/joe                       31K  1.41G    31K  /datapool/joe
datapool/pat                       31K  1.41G    31K  /datapool/pat
```

Now let's change it.

```
# zfs set mountpoint=/fred datapool/fred
```

And look at it again.

```
# zfs list -r datapool
```

```
Terminal
Window Menu w  Terminal  Help
root@zfslab:/# zfs set mountpoint=/fred datapool/fred
root@zfslab:/# zfs list -r datapool
NAME                              USED  AVAIL  REFER  MOUNTPOINT
datapool                         2.50G  1.41G    35K  /datapool
datapool/bob                     1.00G  1.41G  1.00G  /datapool/bob
datapool/fred                     159K  2.00G    32K  /fred
datapool/fred/documents           127K  2.00G    34K  /fred/documents
datapool/fred/documents/audio      31K  2.00G    31K  /fred/documents/audio
datapool/fred/documents/pictures   31K  2.00G    31K  /fred/documents/pictures
datapool/fred/documents/video      31K  2.00G    31K  /fred/documents/video
datapool/joe                       31K  1.41G    31K  /datapool/joe
datapool/pat                       31K  1.41G    31K  /datapool/pat
```

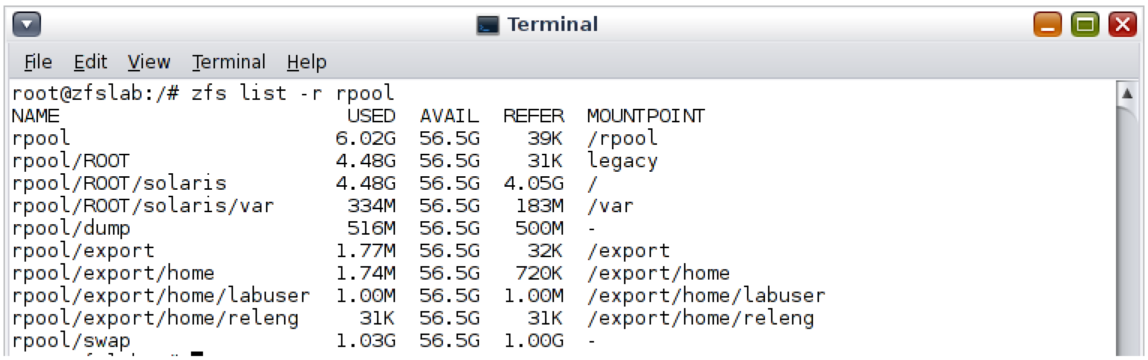
Notice that not only did the command change datapool/fred, but also all of its children, in one single command.

## 6.6 ZFS Volumes (zvols)

Let's look at another type of dataset, the zvol and what it can do.

Volumes, or zvols, provide a block level (raw and cooked) interface into the zpool. Instead of creating a file system where you place files and directories, a single object is created and then accessed as if it were a real disk device. This would be used for things like raw database files, virtual machine disk images and legacy file systems. Oracle Solaris also uses this for the swap and dump devices when installed into a zpool.

```
# zfs list -r rpool
```




In this example, rpool/dump is the dump device for Solaris and it 516MB. rpool/swap is the swap device and it is 1GB. As you can see, you can mix files and devices within the same pool.

Unlike a file system dataset, you must specifically designate the size of the device when you create it, but you can change it later if needed. It's just another dataset property. Create a volume.

```
# zfs create -V 500m datapool/vol1
```

This creates two device nodes: /dev/zvol/dsk/datapool/vol1 (cooked) and /dev/zvol/rdsk/datapool/vol1 (raw). These can be used like any other raw or cooked device. We can even put a UFS file system on it.

```
# newfs /dev/zvol/rdsk/datapool/vol1
```

```
Terminal
File Edit View Terminal Help
root@zfslab:/# zfs create -V 500m datapool/vol1
root@zfslab:/# newfs /dev/zvol/rdsk/datapool/vol1
newfs: construct a new file system /dev/zvol/rdsk/datapool/vol1: (y/n)? y
Warning: 2082 sector(s) in last cylinder unallocated
/dev/zvol/rdsk/datapool/vol1:   1023966 sectors in 167 cylinders of 48 tracks, 128 sectors
        500.0MB in 12 cyl groups (14 c/g, 42.00MB/g, 20160 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
 32, 86176, 172320, 258464, 344608, 430752, 516896, 603040, 689184, 775328,
 861472, 947616_
```

Expanding a volume is just a matter of setting the dataset property volsize to a new value. Be careful when lowering the value as this will truncate the volume and you could lose data. In this next example, let's grow our volume from 500MB to 1GB. Since there is a UFS file system on it, we'll use growfs to make the file system use the new space.

```
# zfs set volsize=1g datapool/vol1
# growfs /dev/zvol/rdsk/datapool/vol1
```

```
Terminal
File Edit View Terminal Help
root@zfslab:/# zfs set volsize=1g datapool/vol1
root@zfslab:/# growfs /dev/zvol/rdsk/datapool/vol1
Warning: 4130 sector(s) in last cylinder unallocated
/dev/zvol/rdsk/datapool/vol1:   2097118 sectors in 342 cylinders of 48 tracks, 128 sectors
        1024.0MB in 25 cyl groups (14 c/g, 42.00MB/g, 20160 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
 32, 86176, 172320, 258464, 344608, 430752, 516896, 603040, 689184, 775328,
 1292192, 1378336, 1464480, 1550624, 1636768, 1722912, 1809056, 1895200,
 1981344, 2067488
```

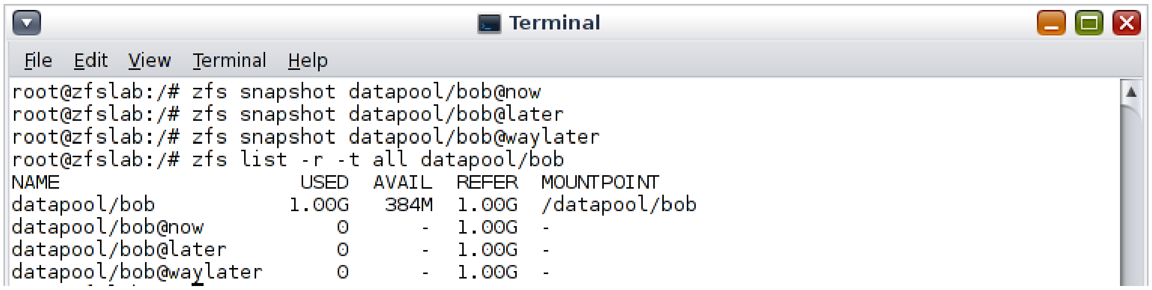
## 6.7 Snapshots and clones

ZFS provides the ability to preserve the contents of a dataset through the use of snapshots. And snapshots are easy to create and take up virtually no space when they're first created. Type the below commands to create some snapshots.

```
# zfs snapshot datapool/bob@now
```

The value after the @ denotes the name of the snapshot. Any number of snapshots can be taken.

```
# zfs snapshot datapool/bob@later
# zfs snapshot datapool/bob@waylater
# zfs list -r -t all datapool/bob
```



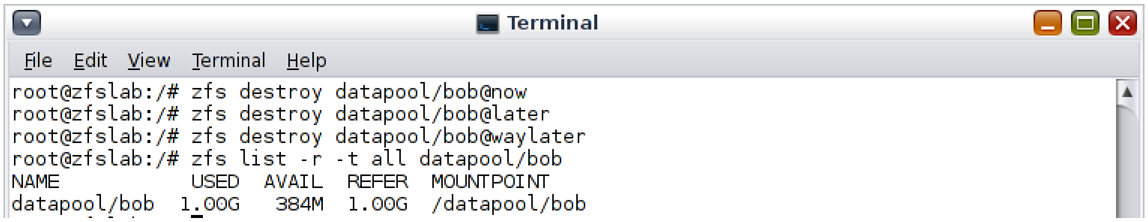

Note that the snapshots take up zero bytes.

Delete these snapshots as they won't be needed for the actual lab.

```
# zfs destroy datapool/bob@waylater
# zfs destroy datapool/bob@later
# zfs destroy datapool/bob@now
```

And verify that they're gone

```
# zfs list -r -t all datapool/bob
```




We can use our point in time snapshots to create new datasets called clones Clones are datasets, just like any other, but start off with the contents from the snapshot. Clones and snapshots make efficient use of storage. Clones only require space for the data that's different than the snapshot. That means that if 5 clones are created from a single snapshot, only 1 copy of the common data is required.

Remember that datapool/bob has a 1GB file in it? Let's take a snapshot of the datapool and then create some clones.

```
# zfs snapshot datapool/bob@original
# zfs clone datapool/bob@original datapool/newbob
# zfs clone datapool/bob@original datapool/newfred
# zfs clone datapool/bob@original datapool/newpat
# zfs clone datapool/bob@original datapool/newjoe
```

```
Terminal
File  Edit  View  Terminal  Help
root@zfslab:~# zfs snapshot datapool/bob@original
root@zfslab:~# zfs clone datapool/bob@original datapool/newbob
root@zfslab:~# zfs clone datapool/bob@original datapool/newfred
root@zfslab:~# zfs clone datapool/bob@original datapool/newpat
root@zfslab:~# zfs clone datapool/bob@original datapool/newjoe
```

Use '`zfs list -r -o space datapool`' to illustrate what's going on.

```
Terminal
File  Edit  View  Terminal  Help
root@zfslab:~# zfs list -r -o space datapool
NAME                             AVAIL   USED  USEDSNAP  USEDDS  USEDREFRESERV  USEDCHILD
datapool                          383M  3.53G         0     39K              0      3.53G
datapool/bob                      383M  1.00G         0   1.00G              0          0
datapool/fred                    1.87G   159K         0     32K              0       127K
datapool/fred/documents          1.87G   127K         0     34K              0        93K
datapool/fred/documents/audio    1.87G    31K         0     31K              0          0
datapool/fred/documents/pictures 1.87G    31K         0     31K              0          0
datapool/fred/documents/video    1.87G    31K         0     31K              0          0
datapool/joe                      383M    31K         0     31K              0          0
datapool/newbob                   383M    18K         0     18K              0          0
datapool/newfred                  383M    18K         0     18K              0          0
datapool/newjoe                   383M    18K         0     18K              0          0
datapool/newpat                   383M    18K         0     18K              0          0
datapool/pat                      383M    31K         0     31K              0          0
datapool/vol1                    1.34G  1.03G         0   62.7M           994M          0
```

We can see that there's a 1GB file in datapool/bob. Right now, that's the dataset being charged with the copy, although all of the clones can use it.

Now let's delete it in the original file system, and all of the clones, and see what happens.

```
# rm /datapool/*/bigfile
# zfs list -r -o space datapool
```

```
Terminal
File  Edit  View  Terminal  Help
root@zfslab:/datapool# rm /datapool/*/bigfile
root@zfslab:/datapool# zfs list -r -o space datapool
NAME                             AVAIL   USED  USEDSNAP  USEDDS  USEDREFRESERV  USEDCHILD
datapool                          382M  3.53G         0     39K              0      3.53G
datapool/bob                      382M  1.00G     1.00G     31K              0          0
datapool/fred                    1.87G   159K         0     32K              0       127K
datapool/fred/documents          1.87G   127K         0     34K              0        93K
datapool/fred/documents/audio    1.87G    31K         0     31K              0          0
datapool/fred/documents/pictures 1.87G    31K         0     31K              0          0
datapool/fred/documents/video    1.87G    31K         0     31K              0          0
datapool/joe                      382M    31K         0     31K              0          0
datapool/newbob                   382M    19K         0     19K              0          0
datapool/newfred                  382M    19K         0     19K              0          0
datapool/newjoe                   382M    19K         0     19K              0          0
datapool/newpat                   382M    19K         0     19K              0          0
datapool/pat                      382M    31K         0     31K              0          0
datapool/vol1                    1.34G  1.03G         0   62.7M           994M          0
```

Notice that the 1GB has not been freed (avail space is still 382M), but the USEDSNAP value for datapool/bob has gone from 0 to 1GB, indicating that the snapshot is now holding that 1GB of data. To free that space you will have to delete the snapshot. In this case you would also have to delete any clones that are derived from it.

```
# zfs destroy datapool/bob@original
# zfs destroy -R datapool/bob@original
```

```
Terminal
File Edit View Terminal Help
root@zfslab:/datapool# zfs destroy datapool/bob@original
cannot destroy 'datapool/bob@original': snapshot has dependent clones
use '-R' to destroy the following datasets:
datapool/newbob
datapool/newfred
datapool/newpat
datapool/newjoe
root@zfslab:/datapool#
root@zfslab:/datapool# zfs destroy -R datapool/bob@original
```

```
# zfs list -r -o space datapool
```

```
Terminal
File Edit View Terminal Help
root@zfslab:/datapool# zfs list -r -o space datapool
NAME                             AVAIL   USED  USEDSNAP  USEDDS  USEDREFRESERV  USEDCHILD
datapool                         1.37G  2.53G         0     34K              0      2.53G
datapool/bob                     1.37G    31K         0     31K              0          0
datapool/fred                    2.00G   159K         0     32K              0       127K
datapool/fred/documents          2.00G   127K         0     34K              0        93K
datapool/fred/documents/audio    2.00G    31K         0     31K              0          0
datapool/fred/documents/pictures 2.00G    31K         0     31K              0          0
datapool/fred/documents/video    2.00G    31K         0     31K              0          0
datapool/joe                     1.37G    31K         0     31K              0          0
datapool/pat                     1.37G    31K         0     31K              0          0
datapool/vol1                    2.34G  1.03G         0   62.7M           994M          0
```

The 1GB file that we deleted has been freed because the last snapshot holding it has been deleted.

You can also take a snapshot of a dataset and all of its children. A recursive snapshot is atomic, meaning that it is a consistent point in time picture of the contents of all of the datasets. Use -r for a recursive snapshot.

```
# zfs snapshot -r datapool/fred@now
# zfs list -r -t all datapool/fred
```

```
root@zfslab:/datapool# zfs snapshot -r datapool/fred@now
root@zfslab:/datapool# zfs list -r -t all datapool/fred
NAME                                   USED  AVAIL  REFER  MOUNTPOINT
datapool/fred                          159K  2.00G    32K  /fred
datapool/fred@now                         0      -    32K  -
datapool/fred/documents                127K  2.00G    34K  /fred/documents
datapool/fred/documents@now               0      -    34K  -
datapool/fred/documents/audio           31K  2.00G    31K  /fred/documents/audio
datapool/fred/documents/audio@now         0      -    31K  -
datapool/fred/documents/pictures        31K  2.00G    31K  /fred/documents/pictures
datapool/fred/documents/pictures@now      0      -    31K  -
datapool/fred/documents/video           31K  2.00G    31K  /fred/documents/video
datapool/fred/documents/video@now         0      -    31K  -
```

Snapshosts can also be destroyed recursively using -r.

```
# zfs destroy -r datapool/fred@now
```

```
root@zfslab:/datapool# zfs destroy -r datapool/fred@now
```

The last item we'll cover is a new command in Oracle Solaris 11, the ZFS diff command. The diff command enables a system administrator to determine the differences between different ZFS snapshots.

Let's start by creating some snapshots and adding files to a user's home directory. Assuming you have the 'labuser' in your Solaris instance let's use that home directory for our example.

```
# zfs list -r -t all rpool/export
```

Create a text file in the users home directory and take a snapshot – call the snapshot 'snap1'

```
# ps -fe >> /export/home/labuser/psfile.txt
# zfs snapshot rpool/export/home/labuser@snap1
```

```
root@zfslab:/# zfs list -r -t all rpool/export
NAME                         USED  AVAIL  REFER  MOUNTPOINT
rpool/export                6.66M  56.5G    32K  /export
rpool/export/home           6.63M  56.5G   720K  /export/home
rpool/export/home/labuser   5.92M  56.5G  5.92M  /export/home/labuser
root@zfslab:/# ps -fe >> /export/home/labuser/psfile.txt
root@zfslab:/# zfs snapshot rpool/export/home/labuser@snap1
root@zfslab:/#
```

Verify your snapshot

```
# zfs list -r -t all rpool/export
```

```
root@zfslab:/# zfs list -r -t all rpool/export
NAME                              USED  AVAIL  REFER  MOUNTPOINT
rpool/export                     6.70M  56.5G    32K  /export
rpool/export/home                6.67M  56.5G   718K  /export/home
rpool/export/home/labuser        5.97M  56.5G  5.93M  /export/home/labuser
rpool/export/home/labuser@snap1    38K      -  5.93M  -
root@zfslab:/#
```

Now let's create another file …

```
# svcs -a >> /export/home/labuser/svcsfile.txt
```

Take another snapshot, call it snap2, and confirm your snapshots again.

```
# zfs snapshot rpool/export/home/labuser@snap2
# zfs list -r -t all rpool/export
```

```
root@zfslab:/# svcs -a >> /export/home/labuser/svcsfile.txt
root@zfslab:/# zfs snapshot rpool/export/home/labuser@snap2
root@zfslab:/#
root@zfslab:/# zfs list -r -t all rpool/export
NAME                              USED  AVAIL  REFER  MOUNTPOINT
rpool/export                     6.71M  56.5G    32K  /export
rpool/export/home                6.68M  56.5G   718K  /export/home
rpool/export/home/labuser        5.98M  56.5G  5.94M  /export/home/labuser
rpool/export/home/labuser@snap1    40K      -  5.93M  -
rpool/export/home/labuser@snap2      0      -  5.94M  -
root@zfslab:/#
```

Now let's run the diff command and see what happens.

```
# zfs diff rpool/export/home/labuser@snap1 \
rpool/export/home/labuser@snap2
```

```
root@zfslab:/# zfs diff rpool/export/home/labuser@snap1 rpool/export/home/labuser@snap2
M       /export/home/labuser/
M       /export/home/labuser/.gnome2/gnome-power-manager
-       /export/home/labuser/.gnome2/gnome-power-manager/profile-50000-charging.csv
+       /export/home/labuser/.gnome2/gnome-power-manager/profile-50000-charging.csv
+       /export/home/labuser/svcsfile.txt
root@zfslab:/#
```

***Note:*** *I planned this lab to just incorporate the files we created but since we're on a laptop the profile-5000-charging.csv file kind of popped in there, you may or may not have that file in your output.*

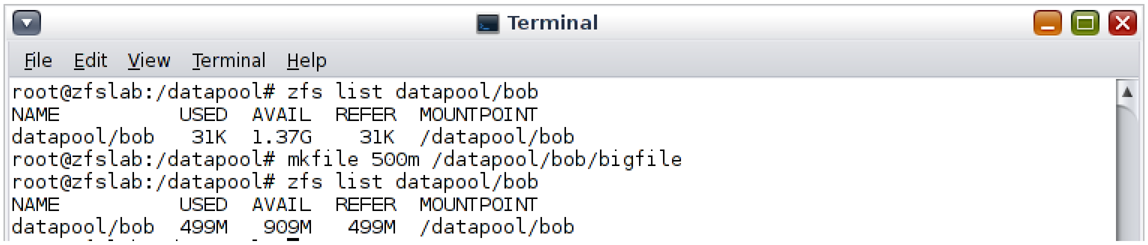
The output on the diff command indicates that the file or directory has been modified with the 'M' at the left side. The '-' indicates that the file or directory is present in the older snapshot but not in the newer one. The '+' sign indicates that the file or directory is present in the more recent snapshot but not in the older snapshot. You might also see an 'R' indicating that a file has been renamed in between snapshots.

## 6.8 Compression

Compression is a useful feature integrated with the ZFS file system. ZFS allows both compressed and noncompressed data to coexist. By turning on the compression property, all new blocks written will be compressed while the existing blocks will remain in their original state.

Let's create a 500MB file we can do some compression on. Type the following commands:

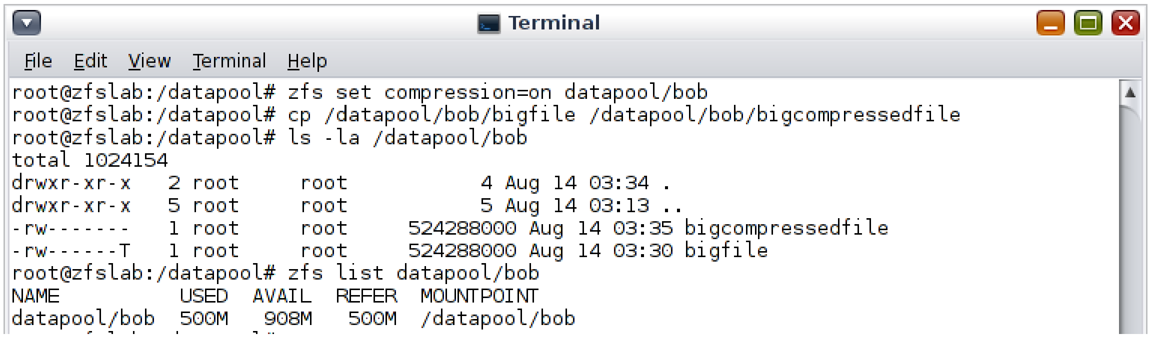
```
# zfs list datapool/bob
# mkfile 500m /datapool/bob/bigfile
# zfs list datapool/bob
```




Now let's turn on compression for datapool/bob and copy the original 500MB file. Verify that you now have 2 separate 500MB files when this is done.

Type the following commands:

```
# zfs set compression=on datapool/bob
# cp /datapool/bob/bigfile /datapool/bob/bigcompressedfile
# ls -la /datapool/bob
```




```
# zfs list datapool/bob
```

There are now 2 different 500MB files in /datapool/bob, but the ls command only says 500MB is used. It turns out that mkfile creates a file filled with zeroes. Those compress extremely well - too well, as they take up no space at all.

That concludes this lab on the ZFS File system, run this command to clean up the work we did during the course of the lab.

```
# zpool destroy –f datapool
```

## 7 Summary

In this lab you learned about the power of the ZFS File System in Oracle Solaris 11. We discussed and performed exercises to familiarize you with zpools and virtual devices (vdevs). We learned about ZFS datasets like snapshots and clones. You were also exposed to the myriad of ZFS properties and ways that ZFS can easily be updated.

The exercises were meant to provide initial exposure to these features and hopefully a basis for continued learning and eventual expertise in this powerful storage technology that's an integral part of Oracle Solaris 11.

## 8 Resources

For more information and next steps, please consult additional resources: Click the hyperlinks to access the resource.

**Oracle Solaris 11 ZFS Technology Page** – articles whitepapers, and more on ZFS

**Oracle Solaris Admin: ZFS File Systems** – documentation on ZFS

**ZFS Best Practices Guide** – excellent real world reference for all things ZFS

**ZFS Evil Tuning Guide** – great resource for detailed tuning and hard to find parameters

**Oracle Solaris 11 ZFS Administration** – Oracle University 4 day ILT training on ZFS