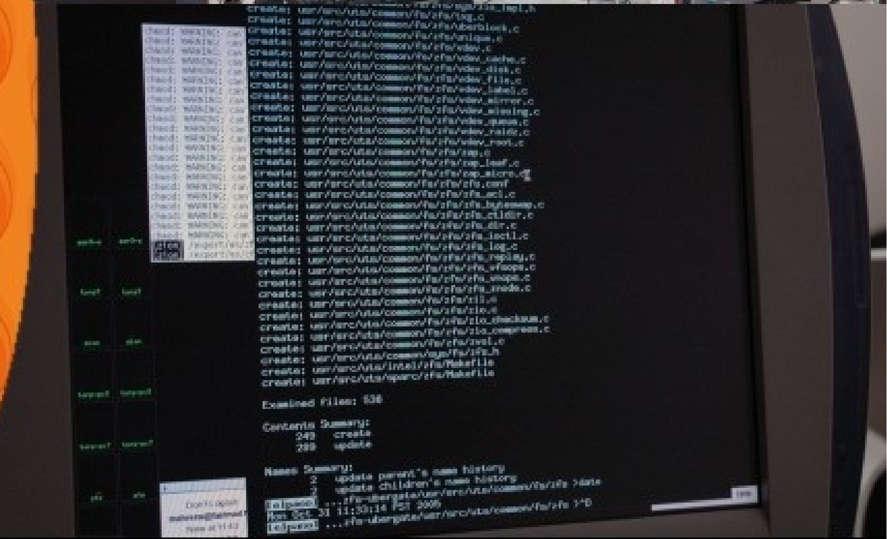
ZFS
Right Now!

Jeff Bonwick
Sun Fellow



```
create: usr/src/uts/common/fs/zfs/sys/zio_impl.h
create: usr/src/uts/common/fs/zfs/txg.c
create: usr/src/uts/common/fs/zfs/uberblock.c
create: usr/src/uts/common/fs/zfs/unique.c
create: usr/src/uts/common/fs/zfs/vdev.c
create: usr/src/uts/common/fs/zfs/vdev_cache.c
create: usr/src/uts/common/fs/zfs/vdev_disk.c
create: usr/src/uts/common/fs/zfs/vdev_file.c
create: usr/src/uts/common/fs/zfs/vdev_label.c
create: usr/src/uts/common/fs/zfs/vdev_mirror.c
create: usr/src/uts/common/fs/zfs/vdev_missing.c
create: usr/src/uts/common/fs/zfs/vdev_queue.c
create: usr/src/uts/common/fs/zfs/vdev_raidz.c
create: usr/src/uts/common/fs/zfs/vdev_root.c
create: usr/src/uts/common/fs/zfs/zap.c
create: usr/src/uts/common/fs/zfs/zap_leaf.c
create: usr/src/uts/common/fs/zfs/zap_micro.c
create: usr/src/uts/common/fs/zfs/zfs.conf
create: usr/src/uts/common/fs/zfs/zfs_acl.c
create: usr/src/uts/common/fs/zfs/zfs_byteswap.c
create: usr/src/uts/common/fs/zfs/zfs_ctldir.c
create: usr/src/uts/common/fs/zfs/zfs_dir.c
create: usr/src/uts/common/fs/zfs/zfs_ioctl.c
create: usr/src/uts/common/fs/zfs/zfs_log.c
create: usr/src/uts/common/fs/zfs/zfs_replay.c
create: usr/src/uts/common/fs/zfs/zfs_vfsops.c
create: usr/src/uts/common/fs/zfs/zfs_vnops.c
create: usr/src/uts/common/fs/zfs/zfs_znode.c
create: usr/src/uts/common/fs/zfs/zil.c
create: usr/src/uts/common/fs/zfs/zio.c
create: usr/src/uts/common/fs/zfs/zio_checksum.c
create: usr/src/uts/common/fs/zfs/zio_compress.c
create: usr/src/uts/common/fs/zfs/zvol.c
create: usr/src/uts/common/sys/fs/zfs.h
create: usr/src/uts/intel/zfs/Makefile
create: usr/src/uts/sparc/zfs/Makefile

Examined Files: 538

Contents Summary:
     249   create
     289   update

Names Summary:
       2   update parent's name history
       2   update children's name history
[illegible] ...zfs-ubergate/usr/src/uts/common/fs/zfs >date
Mon Oct 31 11:33:14 PST 2005
[illegible] ...zfs-ubergate/usr/src/uts/common/fs/zfs >^D
```

Create a Mirrored ZFS Pool, “tank”

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

That's it. You're done.

```
# df
Filesystem   size   used   avail  capacity   Mounted on
tank         233G    18K   233G        1%    /tank
```

Thank you for coming.

Goodbye.

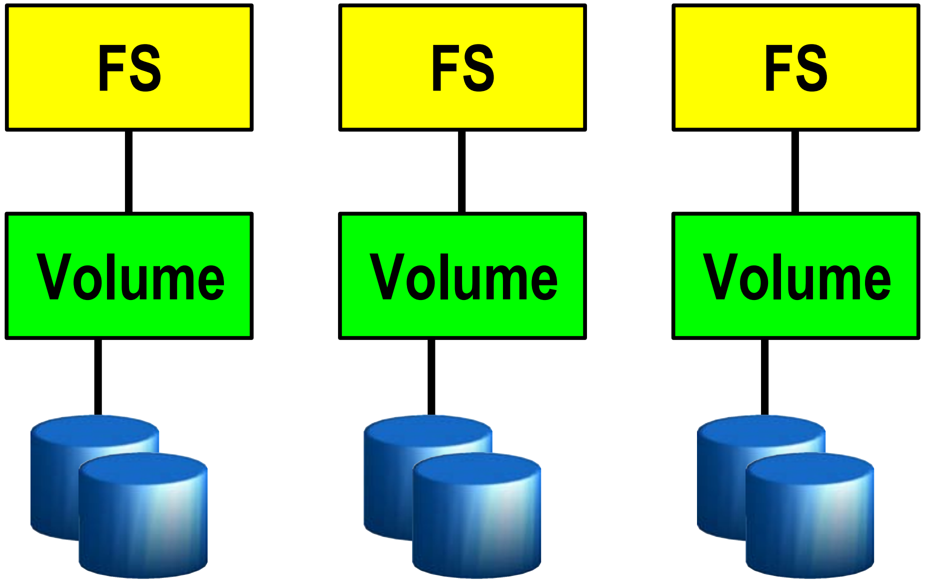
ZFS Overview

- Pooled storage
  - Completely eliminates the antique notion of volumes
  - Does for storage what VM did for memory
- Transactional objects
  - Always consistent on disk – no fsck, ever
  - Supports all object types – file, block, iSCSI, swap, ...
- Provable end-to-end data integrity
  - Detects and corrects silent data corruption
  - Historically considered “too expensive” – no longer true
- Simple administration
  - Concisely express your intent

FS/Volume Model vs. Pooled Storage

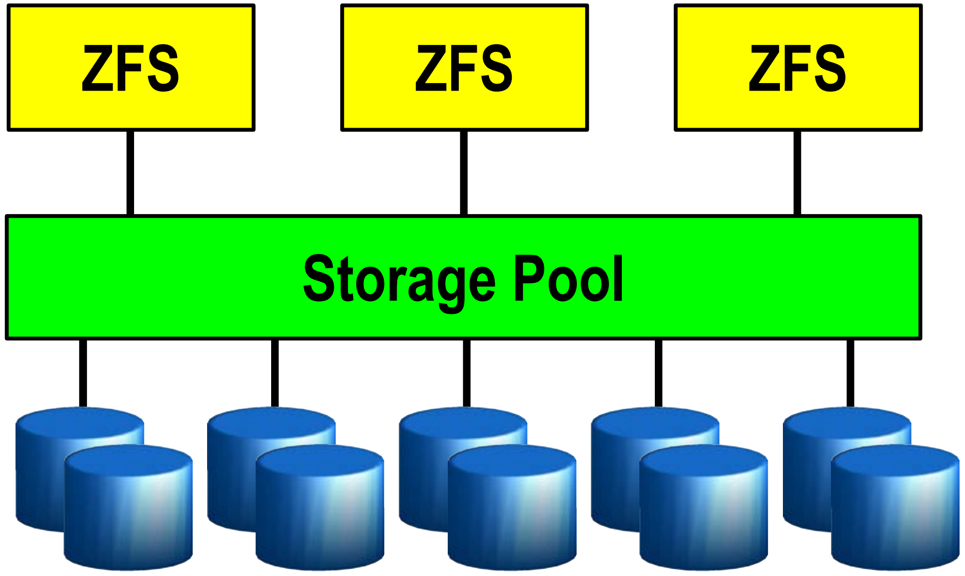
Traditional Volumes

- Abstraction: virtual disk
- Partition/volume for each FS
- Grow/shrink by hand
- Each FS has limited bandwidth
- Storage is fragmented, stranded



ZFS Pooled Storage

- Abstraction: malloc/free
- No partitions to manage
- Grow/shrink automatically
- All bandwidth always available
- All storage in the pool is shared



FS/Volume Interfaces vs. ZFS

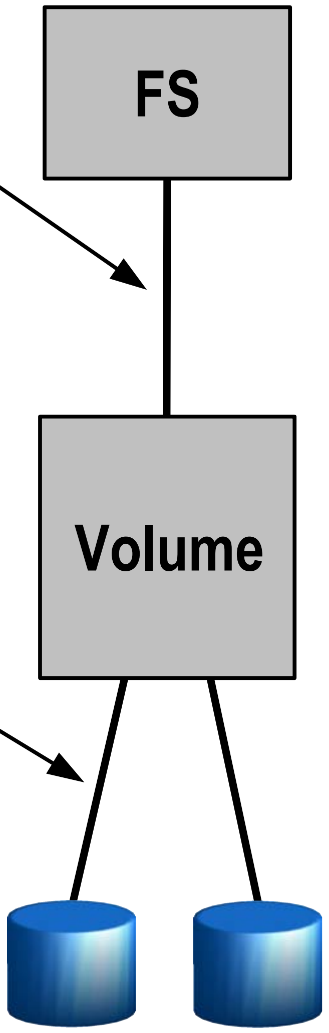
FS/Volume I/O Stack

Block Device Interface

- "Write this block, then that block, ..."
- Loss of power = loss of on-disk consistency
- Workaround: journaling, which is slow & complex

Block Device Interface

- Write each block to each disk immediately to keep mirrors in sync
- Loss of power = resync
- Synchronous and slow



ZFS I/O Stack

Object-Based Transactions

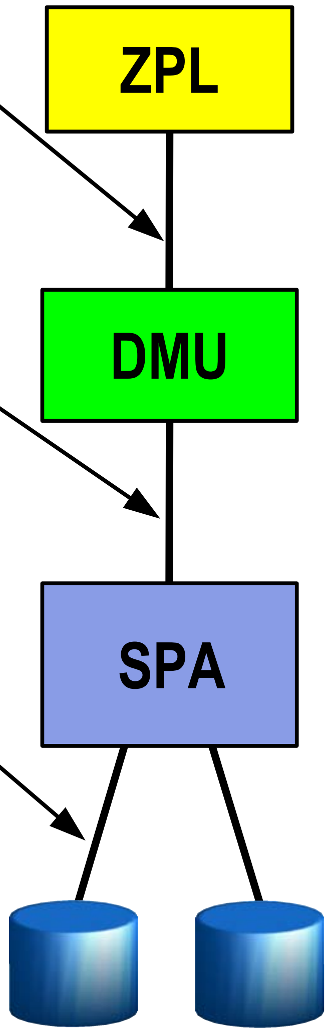
- "Make these 7 changes to these 3 objects"
- All-or-nothing

Transaction Group Commit

- Again, all-or-nothing
- Always consistent on disk
- No journal – not needed

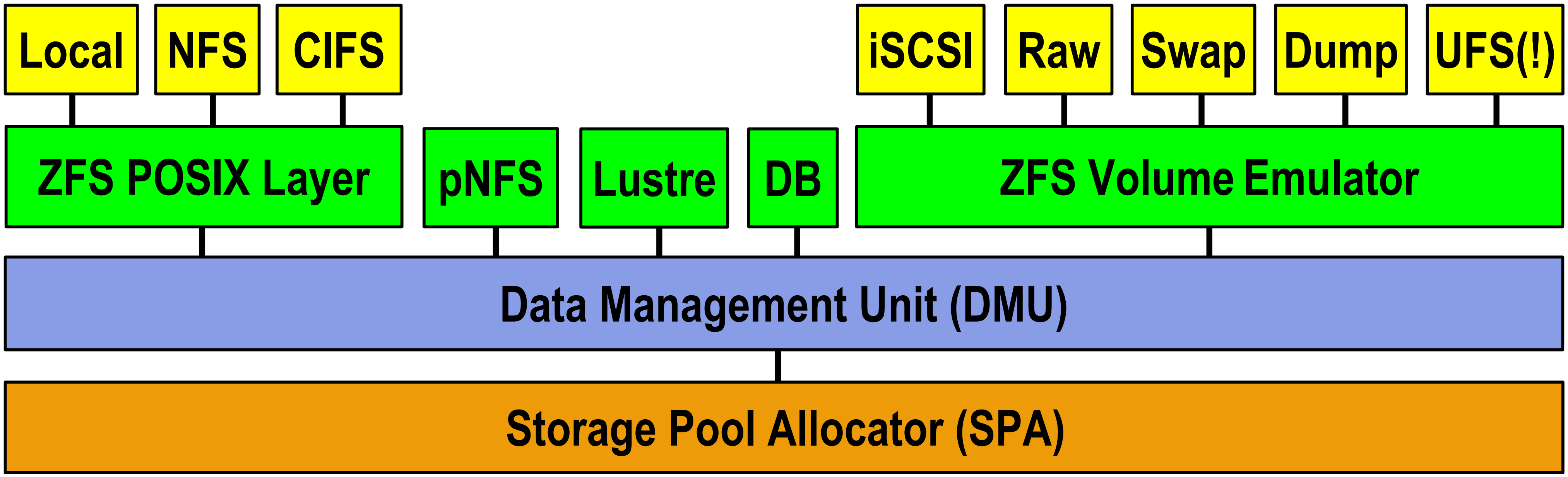
Transaction Group Batch I/O

- Schedule, aggregate, and issue I/O at will
- No resync if power lost
- Runs at platter speed



ZFS Transactional Object System

- DMU provides a general-purpose transactional object store
  - ZFS dataset = up to 2^{48} objects, each up to 2^{64} bytes
- File, block, and network datasets all build on this foundation
  - Filesystems, iSCSI targets, etc. all draw from common storage pool
  - All datasets are full-featured – snapshots, compression, encryption, etc.

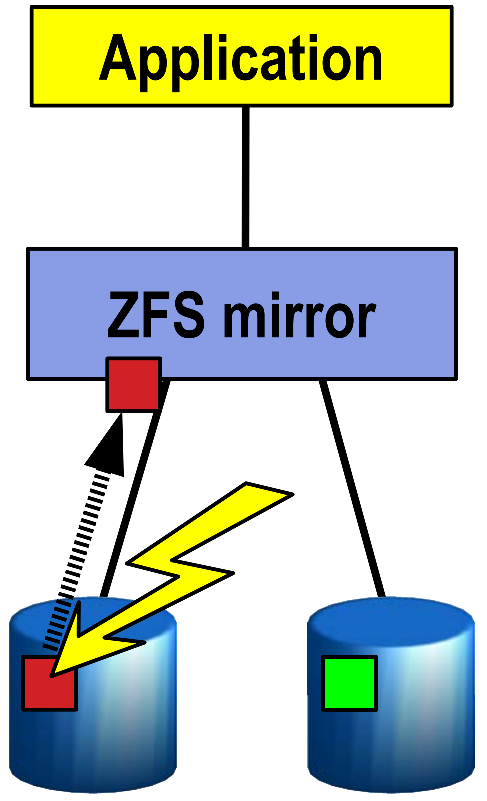


Trends in Storage Integrity

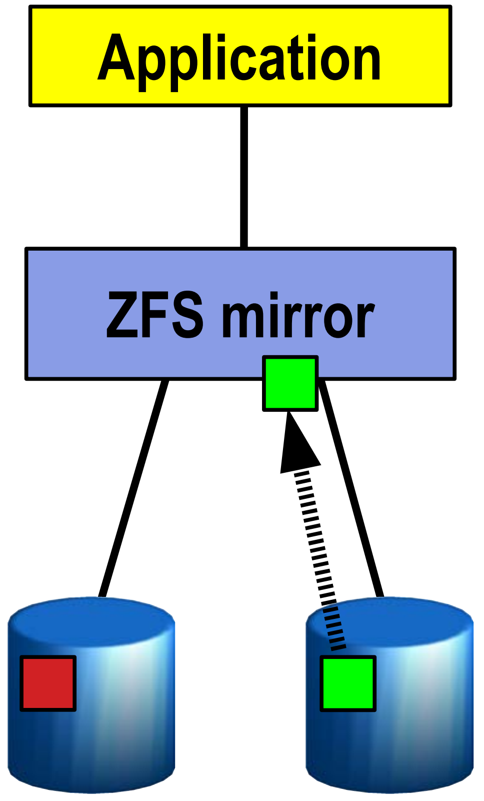
- Uncorrectable error rates have stayed roughly constant
    - 1 in 10^{14} bits (~12TB) for desktop-class drives
    - 1 in 10^{15} bits (~120TB) for enterprise-class drives
    - Bad sector every 8-20TB in practice (desktop and enterprise)
- Drive capacities doubling every 12-18 months
- Number of drives per deployment increasing
- $\longrightarrow$ Rapid increase in error rates
- Both silent and “noisy” data corruption becoming more common
- Cheap flash storage will only accelerate this trend

End-to-End Data Integrity In Action

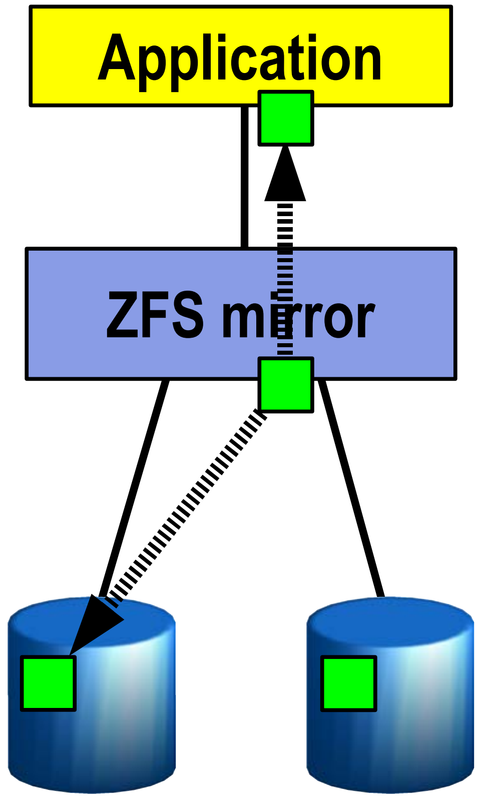
1. Application issues a read. ZFS mirror tries the first disk. Checksum detects silent data corruption, on disk or in flight.



2. ZFS tries the second disk. Checksum indicates that the block is good.



3. ZFS returns known good data to the application and repairs the damaged block.



ZFS Administration

- Pooled storage – no more volumes!
    - Up to 2^{48} datasets per pool – filesystems, iSCSI targets, swap, etc.
    - Nothing to provision!
- Filesystems become administrative control points
    - Hierarchical, with inherited properties
        - Per-dataset policy: snapshots, compression, backups, quotas, etc.
        - Who's using all the space? du(1) takes forever, but df(1M) is instant
        - Manage logically related filesystems as a group
        - Inheritance makes large-scale administration a snap
    - Policy follows the data (mounts, shares, properties, etc.)
    - Delegated administration lets users manage their own data
- Online everything

Creating Pools and Filesystems

- Create a mirrored pool named “tank”

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

- Create home directory filesystem, mounted at /export/home

```
# zfs create tank/home
# zfs set mountpoint=/export/home tank/home
```

- Create home directories for several users
  Note: automatically mounted at /export/home/{ahrens,bonwick,billm} thanks to inheritance

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

- Add more space to the pool

```
# zpool add tank mirror c4d0 c5d0
```

Setting Properties

- Automatically NFS-export all home directories

```
# zfs set sharenfs=rw tank/home
```

- Turn on compression for everything in the pool

```
# zfs set compression=on tank
```

- Limit Eric to a quota of 10g

```
# zfs set quota=10g tank/home/eschrock
```

- Guarantee Tabriz a reservation of 20g

```
# zfs set reservation=20g tank/home/tabriz
```

ZFS Snapshots

- Read-only point-in-time copy of a filesystem
    - Instantaneous creation, unlimited number
    - No additional space used – blocks copied only when they change
    - Accessible through .zfs/snapshot in root of each filesystem
        - Allows users to recover files without sysadmin intervention
- Take a snapshot of Mark's home directory

```
# zfs snapshot tank/home/marks@tuesday
```

- Roll back to a previous snapshot

```
# zfs rollback tank/home/perrin@monday
```

- Take a look at Wednesday's version of foo.c

```
$ cat ~maybee/.zfs/snapshot/wednesday/foo.c
```

ZFS Clones

- Writable copy of a snapshot
  - Instantaneous creation, unlimited number
- Ideal for storing many private copies of mostly-shared data
  - Software installations
  - Source code repositories
  - Diskless clients
  - Zones
  - Virtual machines
- Create a clone of your OpenSolaris source code

```
# zfs clone tank/solaris@monday tank/ws/lori/fix
```

ZFS Send / Receive (Backup / Restore)

- Powered by snapshots
  - Full backup: any snapshot
  - Incremental backup: any snapshot delta
  - Very fast delta generation – cost proportional to data changed
- So efficient it can drive remote replication
- Generate a full backup

```
# zfs send tank/fs@A >/backup/A
```

- Generate an incremental backup

```
# zfs send -i tank/fs@A tank/fs@B >/backup/B-A
```

- Remote replication: send incremental once per minute

```
# zfs send -i tank/fs@11:31 tank/fs@11:32 |
  ssh host zfs receive -d /tank/fs
```

ZFS Data Migration

- Host-neutral on-disk format
  - Change server from x86 to SPARC, it just works
  - Adaptive endianness: neither platform pays a tax
    - Writes always use native endianness, set bit in block pointer
    - Reads byteswap only if host endianness != block endianness
- ZFS takes care of everything
  - Forget about device paths, config files, /etc/vfstab, etc.
  - ZFS will share/unshare, mount/unmount, etc. as necessary
- Export pool from the old server

```
old# zpool export tank
```

- Physically move disks and import pool to the new server

```
new# zpool import tank
```

Forensics – Oh Yes You Did!

```
# zpool history
History for 'builds':
2007-03-06.14:37:53 zpool create builds mirror c3d0 c4d0
2007-03-06.14:37:53 zfs set sharenfs=ro,rw=cathy:zion:steam builds
2007-03-06.14:37:54 zfs create builds/fixes
2007-03-06.14:45:59 zfs create builds/pipe
2007-03-06.15:19:13 zfs destroy builds/pipe
2007-03-21.15:48:31 zfs snapshot builds/fixes@mar20
2007-03-21.15:48:47 zfs clone builds/fixes@mar20 builds/unfixes
2007-03-27.08:57:03 zfs create -V 10g builds/test
2007-03-27.08:57:22 zfs set shareiscsi=on builds/test
2007-03-27.09:06:06 zfs set volsize=20g builds/test
2007-07-29.12:48:14 zpool upgrade builds
```

Where to Learn More

- Community: http://www.opensolaris.org/os/community/zfs
- Wikipedia: http://en.wikipedia.org/wiki/ZFS
- ZFS blogs: http://blogs.sun.com/main/tags/zfs
    - ZFS internals (snapshots, RAID-Z, dynamic striping, etc.)
    - Using iSCSI, CIFS, Zones, databases, remote replication and more
    - Latest news on pNFS, Lustre, and ZFS crypto projects
- ZFS on your Mac: http://developer.apple.com/adcnews
- ZFS on FreeBSD: http://wiki.freebsd.org/ZFS
- ZFS on Linux/FUSE: http://zfs-on-fuse.blogspot.com
- ZFS as an appliance: http://www.nexenta.com