```
**********************************************************
  160449 Tue Aug  7 18:11:15 2012
new/usr/src/cmd/zfs/zfs_main.c
*** NO COMMENTS ***
**********************************************************
______unchanged_portion_omitted_

 201 #define NCOMMAND        (sizeof (command_table) / sizeof (command_table[0]))

 203 zfs_command_t *current_command;

 205 static const char *
 206 get_usage(zfs_help_t idx)
 207 {
 208         switch (idx) {
 209         case HELP_CLONE:
 210                 return (gettext("\tclone [-p] [-o property=value] ... "
 211                     "<snapshot> <filesystem|volume>\n"));
 212         case HELP_CREATE:
 213                 return (gettext("\tcreate [-p] [-o property=value] ... "
 214                     "<filesystem>\n"
 215                     "\tcreate [-ps] [-b blocksize] [-o property=value] ... "
 216                     "-V <size> <volume>\n"));
 217         case HELP_DESTROY:
 218                 return (gettext("\tdestroy [-fnpRrv] <filesystem|volume>\n"
 219                     "\tdestroy [-dnpRrv] "
 220                     "<filesystem|volume>@<snap>[%<snap>][,...]\n"));
 221         case HELP_GET:
 222                 return (gettext("\tget [-rHp] [-d max] "
 223                     "[-o \"all\" | field[,...]] [-t type[,...]] "
 224                     "[-s source[,...]]\n"
 225                     "\t    <\"all\" | property[,...]> "
 226                     "[filesystem|volume|snapshot] ...\n"));
 227         case HELP_INHERIT:
 228                 return (gettext("\tinherit [-rS] <property> "
 229                     "<filesystem|volume|snapshot> ...\n"));
 230         case HELP_UPGRADE:
 231                 return (gettext("\tupgrade [-v]\n"
 232                     "\tupgrade [-r] [-V version] <-a | filesystem ...>\n"));
 233         case HELP_LIST:
 234                 return (gettext("\tlist [-rH][-d max] "
 235                     "[-o property[,...]] [-t type[,...]] [-s property] ...\n"
 236                     "\t    [-S property] ... "
 237                     "[filesystem|volume|snapshot] ...\n"));
 238         case HELP_MOUNT:
 239                 return (gettext("\tmount\n"
 240                     "\tmount [-vO] [-o opts] <-a | filesystem>\n"));
 241         case HELP_PROMOTE:
 242                 return (gettext("\tpromote <clone-filesystem>\n"));
 243         case HELP_RECEIVE:
 244                 return (gettext("\treceive [-vnFu] <filesystem|volume|"
 245                 "snapshot>\n"
 246                 "\treceive [-vnFu] [-d | -e] <filesystem>\n"));
 247         case HELP_RENAME:
 248                 return (gettext("\trename [-f] <filesystem|volume|snapshot> "
 249                     "<filesystem|volume|snapshot>\n"
 250                     "\trename [-f] -p <filesystem|volume> <filesystem|volume>\n"
 251                     "\trename -r <snapshot> <snapshot>"));
 252         case HELP_ROLLBACK:
 253                 return (gettext("\trollback [-rRf] <snapshot>\n"));
 254         case HELP_SEND:
 255                 return (gettext("\tsend [-DnPpRrvs] [-[iI] snapshot] "
 255                 return (gettext("\tsend [-DnPpRrv] [-[iI] snapshot] "
 256                     "<snapshot>\n"));
 257         case HELP_SET:
 258                 return (gettext("\tset <property=value> "
```



```
 259                     "<filesystem|volume|snapshot> ...\n"));
 260         case HELP_SHARE:
 261                 return (gettext("\tshare <-a | filesystem>\n"));
 262         case HELP_SNAPSHOT:
 263                 return (gettext("\tsnapshot [-r] [-o property=value] ... "
 264                     "<filesystem@snapname|volume@snapname> ...\n"));
 265         case HELP_UNMOUNT:
 266                 return (gettext("\tunmount [-f] "
 267                     "<-a | filesystem|mountpoint>\n"));
 268         case HELP_UNSHARE:
 269                 return (gettext("\tunshare "
 270                     "<-a | filesystem|mountpoint>\n"));
 271         case HELP_ALLOW:
 272                 return (gettext("\tallow <filesystem|volume>\n"
 273                     "\tallow [-ldug] "
 274                     "<\"everyone\"|user|group>[,...] <perm|@setname>[,...]\n"
 275                     "\t    <filesystem|volume>\n"
 276                     "\tallow [-ld] -e <perm|@setname>[,...] "
 277                     "<filesystem|volume>\n"
 278                     "\tallow -c <perm|@setname>[,...] <filesystem|volume>\n"
 279                     "\tallow -s @setname <perm|@setname>[,...] "
 280                     "<filesystem|volume>\n"));
 281         case HELP_UNALLOW:
 282                 return (gettext("\tunallow [-rldug] "
 283                     "<\"everyone\"|user|group>[,...]\n"
 284                     "\t    [<perm|@setname>[,...]] <filesystem|volume>\n"
 285                     "\tunallow [-rld] -e [<perm|@setname>[,...]] "
 286                     "<filesystem|volume>\n"
 287                     "\tunallow [-r] -c [<perm|@setname>[,...]] "
 288                     "<filesystem|volume>\n"
 289                     "\tunallow [-r] -s @setname [<perm|@setname>[,...]] "
 290                     "<filesystem|volume>\n"));
 291         case HELP_USERSPACE:
 292                 return (gettext("\tuserspace [-hniHp] [-o field[,...]] "
 293                     "[-sS field] ... [-t type[,...]]\n"
 294                     "\t    <filesystem|snapshot>\n"));
 295         case HELP_GROUPSPACE:
 296                 return (gettext("\tgroupspace [-hniHpU] [-o field[,...]] "
 297                     "[-sS field] ... [-t type[,...]]\n"
 298                     "\t    <filesystem|snapshot>\n"));
 299         case HELP_HOLD:
 300                 return (gettext("\thold [-r] <tag> <snapshot> ...\n"));
 301         case HELP_HOLDS:
 302                 return (gettext("\tholds [-r] <snapshot> ...\n"));
 303         case HELP_RELEASE:
 304                 return (gettext("\trelease [-r] <tag> <snapshot> ...\n"));
 305         case HELP_DIFF:
 306                 return (gettext("\tdiff [-FHt] <snapshot> "
 307                     "[snapshot|filesystem]\n"));
 308         }

 310         abort();
 311         /* NOTREACHED */
 312 }
______unchanged_portion_omitted_

3539 /*
3540  * Send a backup stream to stdout.
3541  */
3542 static int
3543 zfs_do_send(int argc, char **argv)
3544 {
3545         char *fromname = NULL;
3546         char *toname = NULL;
3547         char *cp;
3548         zfs_handle_t *zhp;
```

```
3549         sendflags_t flags = { 0 };
3550         int c, err;
3551         nvlist_t *dbgnv = NULL;
3552         boolean_t extraverbose = B_FALSE;

3554         /* check options */
3555         while ((c = getopt(argc, argv, ":i:I:RDpvnPs")) != -1) {
3555         while ((c = getopt(argc, argv, ":i:I:RDpvnP")) != -1) {
3556                 switch (c) {
3557                 case 'i':
3558                         if (fromname)
3559                                 usage(B_FALSE);
3560                         fromname = optarg;
3561                         break;
3562                 case 'I':
3563                         if (fromname)
3564                                 usage(B_FALSE);
3565                         fromname = optarg;
3566                         flags.doall = B_TRUE;
3567                         break;
3568                 case 'R':
3569                         flags.replicate = B_TRUE;
3570                         break;
3571                 case 'p':
3572                         flags.props = B_TRUE;
3573                         break;
3574                 case 'P':
3575                         flags.parsable = B_TRUE;
3576                         flags.verbose = B_TRUE;
3577                         break;
3578                 case 'v':
3579                         if (flags.verbose)
3580                                 extraverbose = B_TRUE;
3581                         flags.verbose = B_TRUE;
3582                         flags.progress = B_TRUE;
3583                         break;
3584                 case 'D':
3585                         flags.dedup = B_TRUE;
3586                         break;
3587                 case 'n':
3588                         flags.dryrun = B_TRUE;
3589                         break;
3590                 case 's':
3591                         flags.sendsize = B_TRUE;
3592                         break;
3593                 case ':':
3594                         (void) fprintf(stderr, gettext("missing argument for "
3595                             "'%c' option\n"), optopt);
3596                         usage(B_FALSE);
3597                         break;
3598                 case '?':
3599                         (void) fprintf(stderr, gettext("invalid option '%c'\n"),
3600                             optopt);
3601                         usage(B_FALSE);
3602                 }
3603         }

3605         argc -= optind;
3606         argv += optind;

3608         /* check number of arguments */
3609         if (argc < 1) {
3610                 (void) fprintf(stderr, gettext("missing snapshot argument\n"));
3611                 usage(B_FALSE);
3612         }
3613         if (argc > 1) {
```



```
3614                 (void) fprintf(stderr, gettext("too many arguments\n"));
3615                 usage(B_FALSE);
3616         }

3618         if (flags.sendsize) {
3619                 (void) close(STDOUT_FILENO);
3620                 (void) open("/dev/null", O_WRONLY|O_LARGEFILE);
3621         } else if (!flags.dryrun && isatty(STDOUT_FILENO)) {
3615         if (!flags.dryrun && isatty(STDOUT_FILENO)) {
3622                 (void) fprintf(stderr,
3623                     gettext("Error: Stream can not be written to a terminal.\n"
3624                     "You must redirect standard output.\n"));
3625                 return (1);
3626         }

3628         cp = strchr(argv[0], '@');
3629         if (cp == NULL) {
3630                 (void) fprintf(stderr,
3631                     gettext("argument must be a snapshot\n"));
3632                 usage(B_FALSE);
3633         }
3634         *cp = '\0';
3635         toname = cp + 1;
3636         zhp = zfs_open(g_zfs, argv[0], ZFS_TYPE_FILESYSTEM | ZFS_TYPE_VOLUME);
3637         if (zhp == NULL)
3638                 return (1);

3640         /*
3641          * If they specified the full path to the snapshot, chop off
3642          * everything except the short name of the snapshot, but special
3643          * case if they specify the origin.
3644          */
3645         if (fromname && (cp = strchr(fromname, '@')) != NULL) {
3646                 char origin[ZFS_MAXNAMELEN];
3647                 zprop_source_t src;

3649                 (void) zfs_prop_get(zhp, ZFS_PROP_ORIGIN,
3650                     origin, sizeof (origin), &src, NULL, 0, B_FALSE);

3652                 if (strcmp(origin, fromname) == 0) {
3653                         fromname = NULL;
3654                         flags.fromorigin = B_TRUE;
3655                 } else {
3656                         *cp = '\0';
3657                         if (cp != fromname && strcmp(argv[0], fromname)) {
3658                                 (void) fprintf(stderr,
3659                                     gettext("incremental source must be "
3660                                     "in same filesystem\n"));
3661                                 usage(B_FALSE);
3662                         }
3663                         fromname = cp + 1;
3664                         if (strchr(fromname, '@') || strchr(fromname, '/')) {
3665                                 (void) fprintf(stderr,
3666                                     gettext("invalid incremental source\n"));
3667                                 usage(B_FALSE);
3668                         }
3669                 }
3670         }

3672         if (flags.replicate && fromname == NULL)
3673                 flags.doall = B_TRUE;

3675         err = zfs_send(zhp, fromname, toname, &flags, STDOUT_FILENO, NULL, 0,
3676             extraverbose ? &dbgnv : NULL);

3678         if (extraverbose && dbgnv != NULL) {
```

```
3679                 /*
3680                  * dump_nvlist prints to stdout, but that's been
3681                  * redirected to a file.  Make it print to stderr
3682                  * instead.
3683                  */
3684                 (void) dup2(STDERR_FILENO, STDOUT_FILENO);
3685                 dump_nvlist(dbgnv, 0);
3686                 nvlist_free(dbgnv);
3687         }
3688         zfs_close(zhp);

3690         return (err != 0);
3691 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   26993 Tue Aug  7 18:11:23 2012
new/usr/src/lib/libzfs/common/libzfs.h
*** NO COMMENTS ***
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
 */

#ifndef _LIBZFS_H
#define _LIBZFS_H

#include <assert.h>
#include <libnvpair.h>
#include <sys/mnttab.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/varargs.h>
#include <sys/fs/zfs.h>
#include <sys/avl.h>
#include <ucred.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Miscellaneous ZFS constants
 */
#define ZFS_MAXNAMELEN          MAXNAMELEN
#define ZPOOL_MAXNAMELEN        MAXNAMELEN
#define ZFS_MAXPROPLEN          (2 * MAXPATHLEN)
#define ZPOOL_MAXPROPLEN        MAXPATHLEN

/*
 * libzfs errors
 */
typedef enum zfs_error {
        EZFS_SUCCESS = 0,       /* no error -- success */
        EZFS_NOMEM = 2000,      /* out of memory */
        EZFS_BADPROP,           /* invalid property value */
```



```
        EZFS_PROPREADONLY,      /* cannot set readonly property */
        EZFS_PROPTYPE,          /* property does not apply to dataset type */
        EZFS_PROPNONINHERIT,    /* property is not inheritable */
        EZFS_PROPSPACE,         /* bad quota or reservation */
        EZFS_BADTYPE,           /* dataset is not of appropriate type */
        EZFS_BUSY,              /* pool or dataset is busy */
        EZFS_EXISTS,            /* pool or dataset already exists */
        EZFS_NOENT,             /* no such pool or dataset */
        EZFS_BADSTREAM,         /* bad backup stream */
        EZFS_DSREADONLY,        /* dataset is readonly */
        EZFS_VOLTOOBIG,         /* volume is too large for 32-bit system */
        EZFS_INVALIDNAME,       /* invalid dataset name */
        EZFS_BADRESTORE,        /* unable to restore to destination */
        EZFS_BADBACKUP,         /* backup failed */
        EZFS_BADTARGET,         /* bad attach/detach/replace target */
        EZFS_NODEVICE,          /* no such device in pool */
        EZFS_BADDEV,            /* invalid device to add */
        EZFS_NOREPLICAS,        /* no valid replicas */
        EZFS_RESILVERING,       /* currently resilvering */
        EZFS_BADVERSION,        /* unsupported version */
        EZFS_POOLUNAVAIL,       /* pool is currently unavailable */
        EZFS_DEVOVERFLOW,       /* too many devices in one vdev */
        EZFS_BADPATH,           /* must be an absolute path */
        EZFS_CROSSTARGET,       /* rename or clone across pool or dataset */
        EZFS_ZONED,             /* used improperly in local zone */
        EZFS_MOUNTFAILED,       /* failed to mount dataset */
        EZFS_UMOUNTFAILED,      /* failed to unmount dataset */
        EZFS_UNSHARENFSFAILED,  /* unshare(1M) failed */
        EZFS_SHARENFSFAILED,    /* share(1M) failed */
        EZFS_PERM,              /* permission denied */
        EZFS_NOSPC,             /* out of space */
        EZFS_FAULT,             /* bad address */
        EZFS_IO,                /* I/O error */
        EZFS_INTR,              /* signal received */
        EZFS_ISSPARE,           /* device is a hot spare */
        EZFS_INVALCONFIG,       /* invalid vdev configuration */
        EZFS_RECURSIVE,         /* recursive dependency */
        EZFS_NOHISTORY,         /* no history object */
        EZFS_POOLPROPS,         /* couldn't retrieve pool props */
        EZFS_POOL_NOTSUP,       /* ops not supported for this type of pool */
        EZFS_POOL_INVALARG,     /* invalid argument for this pool operation */
        EZFS_NAMETOOLONG,       /* dataset name is too long */
        EZFS_OPENFAILED,        /* open of device failed */
        EZFS_NOCAP,             /* couldn't get capacity */
        EZFS_LABELFAILED,       /* write of label failed */
        EZFS_BADWHO,            /* invalid permission who */
        EZFS_BADPERM,           /* invalid permission */
        EZFS_BADPERMSET,        /* invalid permission set name */
        EZFS_NODELEGATION,      /* delegated administration is disabled */
        EZFS_UNSHARESMBFAILED,  /* failed to unshare over smb */
        EZFS_SHARESMBFAILED,    /* failed to share over smb */
        EZFS_BADCACHE,          /* bad cache file */
        EZFS_ISL2CACHE,         /* device is for the level 2 ARC */
        EZFS_VDEVNOTSUP,        /* unsupported vdev type */
        EZFS_NOTSUP,            /* ops not supported on this dataset */
        EZFS_ACTIVE_SPARE,      /* pool has active shared spare devices */
        EZFS_UNPLAYED_LOGS,     /* log device has unplayed logs */
        EZFS_REFTAG_RELE,       /* snapshot release: tag not found */
        EZFS_REFTAG_HOLD,       /* snapshot hold: tag already exists */
        EZFS_TAGTOOLONG,        /* snapshot hold/rele: tag too long */
        EZFS_PIPEFAILED,        /* pipe create failed */
        EZFS_THREADCREATEFAILED, /* thread create failed */
        EZFS_POSTSPLIT_ONLINE,  /* onlining a disk after splitting it */
        EZFS_SCRUBBING,         /* currently scrubbing */
        EZFS_NO_SCRUB,          /* no active scrub */
        EZFS_DIFF,              /* general failure of zfs diff */
```

```
 127         EZFS_DIFFDATA,          /* bad zfs diff data */
 128         EZFS_POOLREADONLY,      /* pool is in read-only mode */
 129         EZFS_UNKNOWN
 130 } zfs_error_t;
______unchanged_portion_omitted_

 542 void libzfs_add_handle(get_all_cb_t *, zfs_handle_t *);
 543 int libzfs_dataset_cmp(const void *, const void *);

 545 /*
 546  * Functions to create and destroy datasets.
 547  */
 548 extern int zfs_create(libzfs_handle_t *, const char *, zfs_type_t,
 549     nvlist_t *);
 550 extern int zfs_create_ancestors(libzfs_handle_t *, const char *);
 551 extern int zfs_destroy(zfs_handle_t *, boolean_t);
 552 extern int zfs_destroy_snaps(zfs_handle_t *, char *, boolean_t);
 553 extern int zfs_destroy_snaps_nvl(zfs_handle_t *, nvlist_t *, boolean_t);
 554 extern int zfs_clone(zfs_handle_t *, const char *, nvlist_t *);
 555 extern int zfs_snapshot(libzfs_handle_t *, const char *, boolean_t, nvlist_t *);
 556 extern int zfs_snapshot_nvl(libzfs_handle_t *hdl, nvlist_t *snaps,
 557     nvlist_t *props);
 558 extern int zfs_rollback(zfs_handle_t *, zfs_handle_t *, boolean_t);
 559 extern int zfs_rename(zfs_handle_t *, const char *, boolean_t, boolean_t);

 561 typedef struct sendflags {
 562         /* print informational messages (ie, -v was specified) */
 563         boolean_t verbose;

 565         /* recursive send  (ie, -R) */
 566         boolean_t replicate;

 568         /* for incrementals, do all intermediate snapshots */
 569         boolean_t doall;

 571         /* if dataset is a clone, do incremental from its origin */
 572         boolean_t fromorigin;

 574         /* do deduplication */
 575         boolean_t dedup;

 577         /* send properties (ie, -p) */
 578         boolean_t props;

 580         /* do not send (no-op, ie. -n) */
 581         boolean_t dryrun;

 583         /* do not send (just calculate exact send stream size, ie. -s */
 584         boolean_t sendsize;

 586         /* parsable verbose output (ie. -P) */
 587         boolean_t parsable;

 589         /* show progress (ie. -v) */
 590         boolean_t progress;
 591 } sendflags_t;
______unchanged_portion_omitted_
```

```
**********************************************************
   87233 Tue Aug  7 18:11:29 2012
new/usr/src/lib/libzfs/common/libzfs_sendrecv.c
*** NO COMMENTS ***
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
  24  * Copyright (c) 2012 by Delphix. All rights reserved.
  25  * Copyright (c) 2012, Joyent, Inc. All rights reserved.
  26  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
  27  */

  29 #include <assert.h>
  30 #include <ctype.h>
  31 #include <errno.h>
  32 #include <libintl.h>
  33 #include <stdio.h>
  34 #include <stdlib.h>
  35 #include <strings.h>
  36 #include <unistd.h>
  37 #include <stddef.h>
  38 #include <fcntl.h>
  39 #include <sys/mount.h>
  40 #include <pthread.h>
  41 #include <umem.h>
  42 #include <time.h>

  44 #include <libzfs.h>

  46 #include "zfs_namecheck.h"
  47 #include "zfs_prop.h"
  48 #include "zfs_fletcher.h"
  49 #include "libzfs_impl.h"
  50 #include <sha2.h>
  51 #include <sys/zio_checksum.h>
  52 #include <sys/ddt.h>

  54 /* in libzfs_dataset.c */
  55 extern void zfs_setprop_error(libzfs_handle_t *, zfs_prop_t, int, char *);

  57 static int zfs_receive_impl(libzfs_handle_t *, const char *, recvflags_t *,
  58     int, const char *, nvlist_t *, avl_tree_t *, char **, int, uint64_t *);

  60 static const zio_cksum_t zero_cksum = { 0 };
```



```
  62 typedef struct dedup_arg {
  63         int     inputfd;
  64         int     outputfd;
  65         uint64_t        dedup_data_sz;
  66         boolean_t       sendsize;
  67         libzfs_handle_t  *dedup_hdl;
  68 } dedup_arg_t;
_____unchanged_portion_omitted_

 189 /*
 190  * the function used by the cksummer thread that needs to know
 191  * about the sendsize flag
 192  */
 193 static int
 194 dedup_cksum_and_write(dedup_arg_t *dda, const void *buf, uint64_t len,
 195     zio_cksum_t *zc, int outfd)
 196 {
 197         int ret = len;

 199         dda->dedup_data_sz += len;
 200         fletcher_4_incremental_native(buf, len, zc);
 201         if (!dda->sendsize)
 202                 ret = (write(outfd, buf, len));

 204         return (ret);
 205 }

 207 /*
 208  * This function is started in a separate thread when the dedup option
 209  * has been requested.  The main send thread determines the list of
 210  * snapshots to be included in the send stream and makes the ioctl calls
 211  * for each one.  But instead of having the ioctl send the output to the
 212  * the output fd specified by the caller of zfs_send()), the
 213  * ioctl is told to direct the output to a pipe, which is read by the
 214  * alternate thread running THIS function.  This function does the
 215  * dedup'ing by:
 216  *  1. building a dedup table (the DDT)
 217  *  2. doing checksums on each data block and inserting a record in the DDT
 218  *  3. looking for matching checksums, and
 219  *  4.  sending a DRR_WRITE_BYREF record instead of a write record whenever
 220  *      a duplicate block is found.
 221  * The output of this function then goes to the output fd requested
 222  * by the caller of zfs_send().
 223  */
 224 static void *
 225 cksummer(void *arg)
 226 {
 227         dedup_arg_t *dda = arg;
 228         char *buf = malloc(1<<20);
 229         dmu_replay_record_t thedrr;
 230         dmu_replay_record_t *drr = &thedrr;
 231         struct drr_begin *drrb = &thedrr.drr_u.drr_begin;
 232         struct drr_end *drre = &thedrr.drr_u.drr_end;
 233         struct drr_object *drro = &thedrr.drr_u.drr_object;
 234         struct drr_write *drrw = &thedrr.drr_u.drr_write;
 235         struct drr_spill *drrs = &thedrr.drr_u.drr_spill;
 236         FILE *ofp;
 237         int outfd;
 238         dmu_replay_record_t wbr_drr = {0};
 239         struct drr_write_byref *wbr_drrr = &wbr_drr.drr_u.drr_write_byref;
 240         dedup_table_t ddt;
 241         zio_cksum_t stream_cksum;
 242         uint64_t physmem = sysconf(_SC_PHYS_PAGES) * sysconf(_SC_PAGESIZE);
 243         uint64_t numbuckets;

 245         ddt.max_ddt_size =
```

```
246             MAX((physmem * MAX_DDT_PHYSMEM_PERCENT)/100,
247             SMALLEST_POSSIBLE_MAX_DDT_MB<<20);

249         numbuckets = ddt.max_ddt_size/(sizeof (dedup_entry_t));

251         /*
252          * numbuckets must be a power of 2.  Increase number to
253          * a power of 2 if necessary.
254          */
255         if (!ISP2(numbuckets))
256                 numbuckets = 1 << high_order_bit(numbuckets);

258         ddt.dedup_hash_array = calloc(numbuckets, sizeof (dedup_entry_t *));
259         ddt.ddecache = umem_cache_create("dde", sizeof (dedup_entry_t), 0,
260             NULL, NULL, NULL, NULL, NULL, 0);
261         ddt.cur_ddt_size = numbuckets * sizeof (dedup_entry_t *);
262         ddt.numhashbits = high_order_bit(numbuckets) - 1;
263         ddt.ddt_full = B_FALSE;

265         /* Initialize the write-by-reference block. */
266         wbr_drr.drr_type = DRR_WRITE_BYREF;
267         wbr_drr.drr_payloadlen = 0;

269         outfd = dda->outputfd;
270         ofp = fdopen(dda->inputfd, "r");
271         while (ssread(drr, sizeof (dmu_replay_record_t), ofp) != 0) {

273                 switch (drr->drr_type) {
274                 case DRR_BEGIN:
275                 {
276                         int     fflags;
277                         ZIO_SET_CHECKSUM(&stream_cksum, 0, 0, 0, 0);

279                         /* set the DEDUP feature flag for this stream */
280                         fflags = DMU_GET_FEATUREFLAGS(drrb->drr_versioninfo);
281                         fflags |= (DMU_BACKUP_FEATURE_DEDUP |
282                             DMU_BACKUP_FEATURE_DEDUPPROPS);
283                         DMU_SET_FEATUREFLAGS(drrb->drr_versioninfo, fflags);

285                         if (dedup_cksum_and_write(dda, drr,
286                             sizeof (dmu_replay_record_t),
264                         if (cksum_and_write(drr, sizeof (dmu_replay_record_t),
287                             &stream_cksum, outfd) == -1)
288                                 goto out;
289                         if (DMU_GET_STREAM_HDRTYPE(drrb->drr_versioninfo) ==
290                             DMU_COMPOUNDSTREAM && drr->drr_payloadlen != 0) {
291                                 int sz = drr->drr_payloadlen;

293                                 if (sz > 1<<20) {
294                                         free(buf);
295                                         buf = malloc(sz);
296                                 }
297                                 (void) ssread(buf, sz, ofp);
298                                 if (ferror(stdin))
299                                         perror("fread");
300                                 if (dedup_cksum_and_write(dda, buf, sz,
301                                     &stream_cksum, outfd) == -1)
278                                 if (cksum_and_write(buf, sz, &stream_cksum,
279                                     outfd) == -1)
302                                         goto out;
303                         }
304                         break;
305                 }

307                 case DRR_END:
308                 {
```



```
309                         /* use the recalculated checksum */
310                         ZIO_SET_CHECKSUM(&drre->drr_checksum,
311                             stream_cksum.zc_word[0], stream_cksum.zc_word[1],
312                             stream_cksum.zc_word[2], stream_cksum.zc_word[3]);
313                         if ((write(outfd, drr,
314                             sizeof (dmu_replay_record_t))) == -1)
315                                 goto out;
316                         dda->dedup_data_sz += sizeof (dmu_replay_record_t);
317                         break;
318                 }

320                 case DRR_OBJECT:
321                 {
322                         if (dedup_cksum_and_write(dda, drr,
323                             sizeof (dmu_replay_record_t),
299                         if (cksum_and_write(drr, sizeof (dmu_replay_record_t),
324                             &stream_cksum, outfd) == -1)
325                                 goto out;
326                         if (drro->drr_bonuslen > 0) {
327                                 (void) ssread(buf,
328                                     P2ROUNDUP((uint64_t)drro->drr_bonuslen, 8),
329                                     ofp);
330                                 if (dedup_cksum_and_write(dda, buf,
306                                 if (cksum_and_write(buf,
331                                     P2ROUNDUP((uint64_t)drro->drr_bonuslen, 8),
332                                     &stream_cksum, outfd) == -1)
333                                         goto out;
334                         }
335                         break;
336                 }

338                 case DRR_SPILL:
339                 {
340                         if (dedup_cksum_and_write(dda, drr,
341                             sizeof (dmu_replay_record_t),
316                         if (cksum_and_write(drr, sizeof (dmu_replay_record_t),
342                             &stream_cksum, outfd) == -1)
343                                 goto out;
344                         (void) ssread(buf, drrs->drr_length, ofp);
345                         if (dedup_cksum_and_write(dda, buf, drrs->drr_length,
320                         if (cksum_and_write(buf, drrs->drr_length,
346                             &stream_cksum, outfd) == -1)
347                                 goto out;
348                         break;
349                 }

351                 case DRR_FREEOBJECTS:
352                 {
353                         if (dedup_cksum_and_write(dda, drr,
354                             sizeof (dmu_replay_record_t),
328                         if (cksum_and_write(drr, sizeof (dmu_replay_record_t),
355                             &stream_cksum, outfd) == -1)
356                                 goto out;
357                         break;
358                 }

360                 case DRR_WRITE:
361                 {
362                         dataref_t       dataref;

364                         (void) ssread(buf, drrw->drr_length, ofp);

366                         /*
367                          * Use the existing checksum if it's dedup-capable,
368                          * else calculate a SHA256 checksum for it.
369                          */
```

```
371                         if (ZIO_CHECKSUM_EQUAL(drrw->drr_key.ddk_cksum,
372                             zero_cksum) ||
373                             !DRR_IS_DEDUP_CAPABLE(drrw->drr_checksumflags)) {
374                                 SHA256_CTX      ctx;
375                                 zio_cksum_t     tmpsha256;

377                                 SHA256Init(&ctx);
378                                 SHA256Update(&ctx, buf, drrw->drr_length);
379                                 SHA256Final(&tmpsha256, &ctx);
380                                 drrw->drr_key.ddk_cksum.zc_word[0] =
381                                     BE_64(tmpsha256.zc_word[0]);
382                                 drrw->drr_key.ddk_cksum.zc_word[1] =
383                                     BE_64(tmpsha256.zc_word[1]);
384                                 drrw->drr_key.ddk_cksum.zc_word[2] =
385                                     BE_64(tmpsha256.zc_word[2]);
386                                 drrw->drr_key.ddk_cksum.zc_word[3] =
387                                     BE_64(tmpsha256.zc_word[3]);
388                                 drrw->drr_checksumtype = ZIO_CHECKSUM_SHA256;
389                                 drrw->drr_checksumflags = DRR_CHECKSUM_DEDUP;
390                         }

392                         dataref.ref_guid = drrw->drr_toguid;
393                         dataref.ref_object = drrw->drr_object;
394                         dataref.ref_offset = drrw->drr_offset;

396                         if (ddt_update(dda->dedup_hdl, &ddt,
397                             &drrw->drr_key.ddk_cksum, drrw->drr_key.ddk_prop,
398                             &dataref)) {
399                                 /* block already present in stream */
400                                 wbr_drrr->drr_object = drrw->drr_object;
401                                 wbr_drrr->drr_offset = drrw->drr_offset;
402                                 wbr_drrr->drr_length = drrw->drr_length;
403                                 wbr_drrr->drr_toguid = drrw->drr_toguid;
404                                 wbr_drrr->drr_refguid = dataref.ref_guid;
405                                 wbr_drrr->drr_refobject =
406                                     dataref.ref_object;
407                                 wbr_drrr->drr_refoffset =
408                                     dataref.ref_offset;

410                                 wbr_drrr->drr_checksumtype =
411                                     drrw->drr_checksumtype;
412                                 wbr_drrr->drr_checksumflags =
413                                     drrw->drr_checksumtype;
414                                 wbr_drrr->drr_key.ddk_cksum =
415                                     drrw->drr_key.ddk_cksum;
416                                 wbr_drrr->drr_key.ddk_prop =
417                                     drrw->drr_key.ddk_prop;

419                                 if (dedup_cksum_and_write(dda, &wbr_drr,
393                                 if (cksum_and_write(&wbr_drr,
420                                     sizeof (dmu_replay_record_t), &stream_cksum,
421                                     outfd) == -1)
422                                         goto out;
423                         } else {
424                                 /* block not previously seen */
425                                 if (dedup_cksum_and_write(dda, drr,
399                                 if (cksum_and_write(drr,
426                                     sizeof (dmu_replay_record_t), &stream_cksum,
427                                     outfd) == -1)
428                                         goto out;
429                                 if (dedup_cksum_and_write(dda, buf,
403                                 if (cksum_and_write(buf,
430                                     drrw->drr_length,
431                                     &stream_cksum, outfd) == -1)
432                                         goto out;
```



```
433                                 }
434                                 break;
435                         }

437                         case DRR_FREE:
438                         {
439                                 if (dedup_cksum_and_write(dda, drr,
440                                     sizeof (dmu_replay_record_t),
413                                 if (cksum_and_write(drr, sizeof (dmu_replay_record_t),
441                                     &stream_cksum, outfd) == -1)
442                                         goto out;
443                                 break;
444                         }

446                         default:
447                                 (void) printf("INVALID record type 0x%x\n",
448                                     drr->drr_type);
449                                 /* should never happen, so assert */
450                                 assert(B_FALSE);
451                         }
452                 }
453         out:
454                 umem_cache_destroy(ddt.ddecache);
455                 free(ddt.dedup_hash_array);
456                 free(buf);
457                 (void) fclose(ofp);

459                 return (NULL);
460         }
```

*unchanged_portion_omitted_*

```
809 /*
810  * Routines specific to "zfs send"
811  */
812 typedef struct send_dump_data {
813         /* these are all just the short snapname (the part after the @) */
814         const char *fromsnap;
815         const char *tosnap;
816         char prevsnap[ZFS_MAXNAMELEN];
817         uint64_t prevsnap_obj;
818         boolean_t seenfrom, seento, replicate, doall, fromorigin;
819         boolean_t verbose, dryrun, dedup, parsable, progress;
820         boolean_t sendsize;
821         uint32_t hdr_send_sz;
822         uint64_t send_sz;
792         boolean_t verbose, dryrun, parsable, progress;
823         int outfd;
824         boolean_t err;
825         nvlist_t *fss;
826         avl_tree_t *fsavl;
827         snapfilter_cb_t *filter_cb;
828         void *filter_cb_arg;
829         nvlist_t *debugnv;
830         char holdtag[ZFS_MAXNAMELEN];
831         int cleanup_fd;
832         uint64_t size;
833 } send_dump_data_t;
```

*unchanged_portion_omitted_*

```
895 /*
896  * Dumps a backup of the given snapshot (incremental from fromsnap if it's not
897  * NULL) to the file descriptor specified by outfd.
898  */
899 static int
900 dump_ioctl(zfs_handle_t *zhp, const char *fromsnap, uint64_t fromsnap_obj,
901     boolean_t fromorigin, int outfd, nvlist_t *debugnv,
```

```
902     boolean_t sendsize, uint64_t *sendcounter)
871     boolean_t fromorigin, int outfd, nvlist_t *debugnv)
903 {
904         zfs_cmd_t zc = { 0 };
905         libzfs_handle_t *hdl = zhp->zfs_hdl;
906         nvlist_t *thisdbg;

908         assert(zhp->zfs_type == ZFS_TYPE_SNAPSHOT);
909         assert(fromsnap_obj == 0 || !fromorigin);

911         (void) strlcpy(zc.zc_name, zhp->zfs_name, sizeof (zc.zc_name));
912         zc.zc_cookie = outfd;
913         zc.zc_obj = fromorigin;
914         zc.zc_sendobj = zfs_prop_get_int(zhp, ZFS_PROP_OBJSETID);
915         zc.zc_fromobj = fromsnap_obj;
916         zc.zc_sendsize = sendsize;
917         zc.zc_sendcounter = 0;

919         VERIFY(0 == nvlist_alloc(&thisdbg, NV_UNIQUE_NAME, 0));
920         if (fromsnap && fromsnap[0] != '\0') {
921                 VERIFY(0 == nvlist_add_string(thisdbg,
922                     "fromsnap", fromsnap));
923         }

925         if (zfs_ioctl(zhp->zfs_hdl, ZFS_IOC_SEND, &zc) != 0) {
926                 char errbuf[1024];
927                 (void) snprintf(errbuf, sizeof (errbuf), dgettext(TEXT_DOMAIN,
928                     "warning: cannot send '%s'"), zhp->zfs_name);

930                 VERIFY(0 == nvlist_add_uint64(thisdbg, "error", errno));
931                 if (debugnv) {
932                         VERIFY(0 == nvlist_add_nvlist(debugnv,
933                             zhp->zfs_name, thisdbg));
934                 }
935                 nvlist_free(thisdbg);

937                 switch (errno) {
938                 case EXDEV:
939                         zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
940                             "not an earlier snapshot from the same fs"));
941                         return (zfs_error(hdl, EZFS_CROSSTARGET, errbuf));

943                 case ENOENT:
944                         if (zfs_dataset_exists(hdl, zc.zc_name,
945                             ZFS_TYPE_SNAPSHOT)) {
946                                 zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
947                                     "incremental source (@%s) does not exist"),
948                                     zc.zc_value);
949                         }
950                         return (zfs_error(hdl, EZFS_NOENT, errbuf));

952                 case EDQUOT:
953                 case EFBIG:
954                 case EIO:
955                 case ENOLINK:
956                 case ENOSPC:
957                 case ENOSTR:
958                 case ENXIO:
959                 case EPIPE:
960                 case ERANGE:
961                 case EFAULT:
962                 case EROFS:
963                         zfs_error_aux(hdl, strerror(errno));
964                         return (zfs_error(hdl, EZFS_BADBACKUP, errbuf));

966                 default:
```



```
967                         return (zfs_standard_error(hdl, errno, errbuf));
968                 }
969         }

971         *sendcounter = (uint64_t)zc.zc_sendcounter;
972         if (debugnv)
973                 VERIFY(0 == nvlist_add_nvlist(debugnv, zhp->zfs_name, thisdbg));
974         nvlist_free(thisdbg);

976         return (0);
977 }
______unchanged_portion_omitted_

1065 static int
1066 dump_snapshot(zfs_handle_t *zhp, void *arg)
1067 {
1068         send_dump_data_t *sdd = arg;
1069         progress_arg_t pa = { 0 };
1070         pthread_t tid;

1072         char *thissnap;
1073         int err;
1074         boolean_t isfromsnap, istosnap, fromorigin;
1075         boolean_t exclude = B_FALSE;

1077         thissnap = strchr(zhp->zfs_name, '@') + 1;
1078         isfromsnap = (sdd->fromsnap != NULL &&
1079             strcmp(sdd->fromsnap, thissnap) == 0);

1081         if (!sdd->seenfrom && isfromsnap) {
1082                 err = hold_for_send(zhp, sdd);
1083                 if (err == 0) {
1084                         sdd->seenfrom = B_TRUE;
1085                         (void) strcpy(sdd->prevsnap, thissnap);
1086                         sdd->prevsnap_obj = zfs_prop_get_int(zhp,
1087                             ZFS_PROP_OBJSETID);
1088                 } else if (err == ENOENT) {
1089                         err = 0;
1090                 }
1091                 zfs_close(zhp);
1092                 return (err);
1093         }

1095         if (sdd->seento || !sdd->seenfrom) {
1096                 zfs_close(zhp);
1097                 return (0);
1098         }

1100         istosnap = (strcmp(sdd->tosnap, thissnap) == 0);
1101         if (istosnap)
1102                 sdd->seento = B_TRUE;

1104         if (!sdd->doall && !isfromsnap && !istosnap) {
1105                 if (sdd->replicate) {
1106                         char *snapname;
1107                         nvlist_t *snapprops;
1108                         /*
1109                          * Filter out all intermediate snapshots except origin
1110                          * snapshots needed to replicate clones.
1111                          */
1112                         nvlist_t *nvfs = fsavl_find(sdd->fsavl,
1113                             zhp->zfs_dmustats.dds_guid, &snapname);

1115                         VERIFY(0 == nvlist_lookup_nvlist(nvfs,
1116                             "snapprops", &snapprops));
1117                         VERIFY(0 == nvlist_lookup_nvlist(snapprops,
```

```
                            thissnap, &snapprops));
                        exclude = !nvlist_exists(snapprops, "is_clone_origin");
                } else {
                        exclude = B_TRUE;
                }
        }

        /*
         * If a filter function exists, call it to determine whether
         * this snapshot will be sent.
         */
        if (exclude || (sdd->filter_cb != NULL &&
            sdd->filter_cb(zhp, sdd->filter_cb_arg) == B_FALSE)) {
                /*
                 * This snapshot is filtered out.  Don't send it, and don't
                 * set prevsnap_obj, so it will be as if this snapshot didn't
                 * exist, and the next accepted snapshot will be sent as
                 * an incremental from the last accepted one, or as the
                 * first (and full) snapshot in the case of a replication,
                 * non-incremental send.
                 */
                zfs_close(zhp);
                return (0);
        }

        err = hold_for_send(zhp, sdd);
        if (err) {
                if (err == ENOENT)
                        err = 0;
                zfs_close(zhp);
                return (err);
        }

        fromorigin = sdd->prevsnap[0] == '\0' &&
            (sdd->fromorigin || sdd->replicate);

        /* print out to-from and approximate size in verbose mode */
        if (sdd->verbose) {
                /* print preamble */
                uint64_t size;
                err = estimate_ioctl(zhp, sdd->prevsnap_obj,
                    fromorigin, &size);

                if (sdd->parsable) {
                        if (sdd->prevsnap[0] != '\0') {
                                (void) fprintf(stderr, "incremental\t%s\t%s",
                                    sdd->prevsnap, zhp->zfs_name);
                        } else {
                                (void) fprintf(stderr, "full\t%s",
                                    zhp->zfs_name);
                        }
                } else {
                        (void) fprintf(stderr, dgettext(TEXT_DOMAIN,
                            "send from @%s to %s"),
                            sdd->prevsnap, zhp->zfs_name);
                }

                if (sdd->sendsize) {
                        /*
                         * we are going to print out the exact stream size info,
                         * so skip the estimate
                         */
                        (void) fprintf(stderr, "\n");
                } else {
                        /*
                         * provide stream size estimate otherwise
```



```
                         */
                        uint64_t size;
                        err = estimate_ioctl(zhp, sdd->prevsnap_obj,
                            fromorigin, &size);

                        if (err == 0) {
                                if (sdd->parsable) {
                                        (void) fprintf(stderr, "\t%llu\n",
                                            (longlong_t)size);
                                } else {
                                        char buf[16];
                                        zfs_nicenum(size, buf, sizeof (buf));
                                        (void) fprintf(stderr,
                                            dgettext(TEXT_DOMAIN,
                                            " estimated size is %s\n"),
                                            buf);
                                        (void) fprintf(stderr, dgettext(TEXT_DOMAIN,
                                            " estimated size is %s\n"), buf);
                                }
                                sdd->size += size;
                        } else {
                                /* could not estimate */
                                (void) fprintf(stderr, "\n");
                        }
                }
        }

        if (!sdd->dryrun) {
                uint64_t sendcounter = 0;
                boolean_t track_progress = (sdd->progress && !sdd->sendsize);
                boolean_t sendsize = B_FALSE;
                /*
                 * If progress reporting is requested, spawn a new thread to
                 * poll ZFS_IOC_SEND_PROGRESS at a regular interval.
                 */
                if (track_progress) {
                if (sdd->progress) {
                        pa.pa_zhp = zhp;
                        pa.pa_fd = sdd->outfd;
                        pa.pa_parsable = sdd->parsable;

                        if (err = pthread_create(&tid, NULL,
                            send_progress_thread, &pa)) {
                                zfs_close(zhp);
                                return (err);
                        }
                }


                /*
                 * We need to reset the sendsize flag being sent to
                 * kernel if sdd->dedup is set. With dedup, the file
                 * descriptor sent to kernel is one end of the pipe,
                 * and we would want the data back in the pipe for
                 * cksummer() to calculate the exact size of the dedup-ed
                 * stream. So reset the sendsize flag such that
                 * kernel writes to the pipe.
                 */

                sendsize = sdd->dedup ? B_FALSE : sdd->sendsize;

                err = dump_ioctl(zhp, sdd->prevsnap, sdd->prevsnap_obj,
                    fromorigin, sdd->outfd, sdd->debugnv,
                    sendsize, &sendcounter);
                    fromorigin, sdd->outfd, sdd->debugnv);
```

```
1242                 sdd->send_sz += sendcounter;

1244                 if (track_progress) {
1174                 if (sdd->progress) {
1245                         (void) pthread_cancel(tid);
1246                         (void) pthread_join(tid, NULL);
1247                 }
1248         }

1250         (void) strcpy(sdd->prevsnap, thissnap);
1251         sdd->prevsnap_obj = zfs_prop_get_int(zhp, ZFS_PROP_OBJSETID);
1252         zfs_close(zhp);
1253         return (err);
1254 }
______unchanged_portion_omitted_

1426 /*
1427  * Generate a send stream for the dataset identified by the argument zhp.
1428  *
1429  * The content of the send stream is the snapshot identified by
1430  * 'tosnap'.  Incremental streams are requested in two ways:
1431  *     - from the snapshot identified by "fromsnap" (if non-null) or
1432  *     - from the origin of the dataset identified by zhp, which must
1433  *       be a clone.  In this case, "fromsnap" is null and "fromorigin"
1434  *       is TRUE.
1435  *
1436  * The send stream is recursive (i.e. dumps a hierarchy of snapshots) and
1437  * uses a special header (with a hdrtype field of DMU_COMPOUNDSTREAM)
1438  * if "replicate" is set.  If "doall" is set, dump all the intermediate
1439  * snapshots. The DMU_COMPOUNDSTREAM header is used in the "doall"
1440  * case too. If "props" is set, send properties.
1441  */
1442 int
1443 zfs_send(zfs_handle_t *zhp, const char *fromsnap, const char *tosnap,
1444     sendflags_t *flags, int outfd, snapfilter_cb_t filter_func,
1445     void *cb_arg, nvlist_t **debugnvp)
1446 {
1447         char errbuf[1024];
1448         send_dump_data_t sdd = { 0 };
1449         int err = 0;
1450         nvlist_t *fss = NULL;
1451         avl_tree_t *fsavl = NULL;
1452         static uint64_t holdseq;
1453         int spa_version;
1454         pthread_t tid;
1455         int pipefd[2];
1456         dedup_arg_t dda = { 0 };
1457         int featureflags = 0;

1459         (void) snprintf(errbuf, sizeof (errbuf), dgettext(TEXT_DOMAIN,
1460             "cannot send '%s'"), zhp->zfs_name);

1462         if (fromsnap && fromsnap[0] == '\0') {
1463                 zfs_error_aux(zhp->zfs_hdl, dgettext(TEXT_DOMAIN,
1464                     "zero-length incremental source"));
1465                 return (zfs_error(zhp->zfs_hdl, EZFS_NOENT, errbuf));
1466         }

1468         if (zhp->zfs_type == ZFS_TYPE_FILESYSTEM) {
1469                 uint64_t version;
1470                 version = zfs_prop_get_int(zhp, ZFS_PROP_VERSION);
1471                 if (version >= ZPL_VERSION_SA) {
1472                         featureflags |= DMU_BACKUP_FEATURE_SA_SPILL;
1473                 }
1474         }
```



```
1476         if (flags->dedup && !flags->dryrun) {
1477                 featureflags |= (DMU_BACKUP_FEATURE_DEDUP |
1478                     DMU_BACKUP_FEATURE_DEDUPPROPS);
1479                 if (err = pipe(pipefd)) {
1480                         zfs_error_aux(zhp->zfs_hdl, strerror(errno));
1481                         return (zfs_error(zhp->zfs_hdl, EZFS_PIPEFAILED,
1482                             errbuf));
1483                 }
1484                 dda.outputfd = outfd;
1485                 dda.inputfd = pipefd[1];
1486                 dda.dedup_hdl = zhp->zfs_hdl;
1487                 dda.sendsize = flags->sendsize;
1488                 if (err = pthread_create(&tid, NULL, cksummer, &dda)) {
1489                         (void) close(pipefd[0]);
1490                         (void) close(pipefd[1]);
1491                         zfs_error_aux(zhp->zfs_hdl, strerror(errno));
1492                         return (zfs_error(zhp->zfs_hdl,
1493                             EZFS_THREADCREATEFAILED, errbuf));
1494                 }
1495         }

1497         if (flags->replicate || flags->doall || flags->props) {
1498                 dmu_replay_record_t drr = { 0 };
1499                 char *packbuf = NULL;
1500                 size_t buflen = 0;
1501                 zio_cksum_t zc = { 0 };

1503                 if (flags->replicate || flags->props) {
1504                         nvlist_t *hdrnv;

1506                         VERIFY(0 == nvlist_alloc(&hdrnv, NV_UNIQUE_NAME, 0));
1507                         if (fromsnap) {
1508                                 VERIFY(0 == nvlist_add_string(hdrnv,
1509                                     "fromsnap", fromsnap));
1510                         }
1511                         VERIFY(0 == nvlist_add_string(hdrnv, "tosnap", tosnap));
1512                         if (!flags->replicate) {
1513                                 VERIFY(0 == nvlist_add_boolean(hdrnv,
1514                                     "not_recursive"));
1515                         }

1517                         err = gather_nvlist(zhp->zfs_hdl, zhp->zfs_name,
1518                             fromsnap, tosnap, flags->replicate, &fss, &fsavl);
1519                         if (err)
1520                                 goto err_out;
1521                         VERIFY(0 == nvlist_add_nvlist(hdrnv, "fss", fss));
1522                         err = nvlist_pack(hdrnv, &packbuf, &buflen,
1523                             NV_ENCODE_XDR, 0);
1524                         if (debugnvp)
1525                                 *debugnvp = hdrnv;
1526                         else
1527                                 nvlist_free(hdrnv);
1528                         if (err) {
1529                                 fsavl_destroy(fsavl);
1530                                 nvlist_free(fss);
1531                                 goto stderr_out;
1532                         }
1533                 }

1535                 if (!flags->dryrun) {
1536                         /* write first begin record */
1537                         drr.drr_type = DRR_BEGIN;
1538                         drr.drr_u.drr_begin.drr_magic = DMU_BACKUP_MAGIC;
1539                         DMU_SET_STREAM_HDRTYPE(drr.drr_u.drr_begin.
1540                             drr_versioninfo, DMU_COMPOUNDSTREAM);
1541                         DMU_SET_FEATUREFLAGS(drr.drr_u.drr_begin.
```

```
1542                            drr_versioninfo, featureflags);
1543                    (void) snprintf(drr.drr_u.drr_begin.drr_toname,
1544                        sizeof (drr.drr_u.drr_begin.drr_toname),
1545                        "%s@%s", zhp->zfs_name, tosnap);
1546                    drr.drr_payloadlen = buflen;
1547                    err = cksum_and_write(&drr, sizeof (drr), &zc, outfd);
1548                    sdd.hdr_send_sz += sizeof (drr);

1550                    /* write header nvlist */
1551                    if (err != -1 && packbuf != NULL) {
1552                            err = cksum_and_write(packbuf, buflen, &zc,
1553                                outfd);
1554                            sdd.hdr_send_sz += buflen;
1555                    }
1556                    free(packbuf);
1557                    if (err == -1) {
1558                            fsavl_destroy(fsavl);
1559                            nvlist_free(fss);
1560                            err = errno;
1561                            goto stderr_out;
1562                    }

1564                    /* write end record */
1565                    bzero(&drr, sizeof (drr));
1566                    drr.drr_type = DRR_END;
1567                    drr.drr_u.drr_end.drr_checksum = zc;
1568                    err = write(outfd, &drr, sizeof (drr));
1569                    sdd.hdr_send_sz += sizeof (drr);
1570                    if (err == -1) {
1571                            fsavl_destroy(fsavl);
1572                            nvlist_free(fss);
1573                            err = errno;
1574                            goto stderr_out;
1575                    }

1577                    err = 0;
1578            }
1579    }

1581    /* dump each stream */
1582    sdd.fromsnap = fromsnap;
1583    sdd.tosnap = tosnap;
1584    if (flags->dedup)
1585            sdd.outfd = pipefd[0];
1586    else
1587            sdd.outfd = outfd;
1588    sdd.replicate = flags->replicate;
1589    sdd.doall = flags->doall;
1590    sdd.fromorigin = flags->fromorigin;
1591    sdd.fss = fss;
1592    sdd.fsavl = fsavl;
1593    sdd.verbose = flags->verbose;
1594    sdd.dedup = flags->dedup;
1595    sdd.sendsize = flags->sendsize;
1596    sdd.parsable = flags->parsable;
1597    sdd.progress = flags->progress;
1598    sdd.dryrun = flags->dryrun;
1599    sdd.filter_cb = filter_func;
1600    sdd.filter_cb_arg = cb_arg;
1601    if (debugnvp)
1602            sdd.debugnv = *debugnvp;

1604    /*
1605     * Some flags require that we place user holds on the datasets that are
1606     * being sent so they don't get destroyed during the send. We can skip
1607     * this step if the pool is imported read-only since the datasets cannot
```



```
1608     * be destroyed.
1609     */
1610    if (!flags->dryrun && !zpool_get_prop_int(zfs_get_pool_handle(zhp),
1611        ZPOOL_PROP_READONLY, NULL) &&
1612        zfs_spa_version(zhp, &spa_version) == 0 &&
1613        spa_version >= SPA_VERSION_USERREFS &&
1614        (flags->doall || flags->replicate)) {
1615            ++holdseq;
1616            (void) snprintf(sdd.holdtag, sizeof (sdd.holdtag),
1617                ".send-%d-%llu", getpid(), (u_longlong_t)holdseq);
1618            sdd.cleanup_fd = open(ZFS_DEV, O_RDWR|O_EXCL);
1619            if (sdd.cleanup_fd < 0) {
1620                    err = errno;
1621                    goto stderr_out;
1622            }
1623    } else {
1624            sdd.cleanup_fd = -1;
1625    }
1626    if (flags->verbose && !flags->sendsize) {
1550    if (flags->verbose) {
1627            /*
1628             * Do a verbose no-op dry run to get all the verbose output
1629             * before generating any data.  Then do a non-verbose real
1630             * run to generate the streams.
1631             */
1632            sdd.dryrun = B_TRUE;
1633            err = dump_filesystems(zhp, &sdd);
1634            sdd.dryrun = flags->dryrun;
1635            sdd.verbose = B_FALSE;
1636            if (flags->parsable) {
1637                    (void) fprintf(stderr, "size\t%llu\n",
1638                        (longlong_t)sdd.size);
1639            } else {
1640                    char buf[16];
1641                    zfs_nicenum(sdd.size, buf, sizeof (buf));
1642                    (void) fprintf(stderr, dgettext(TEXT_DOMAIN,
1643                        "total estimated size is %s\n"), buf);
1644            }
1645    }
1646    err = dump_filesystems(zhp, &sdd);
1647    fsavl_destroy(fsavl);
1648    nvlist_free(fss);

1650    if (flags->dedup) {
1651            (void) close(pipefd[0]);
1652            (void) pthread_join(tid, NULL);
1653            sdd.send_sz = dda.dedup_data_sz;
1654    }

1656    if (sdd.cleanup_fd != -1) {
1657            VERIFY(0 == close(sdd.cleanup_fd));
1658            sdd.cleanup_fd = -1;
1659    }

1661    if (!flags->dryrun && (flags->replicate || flags->doall ||
1662        flags->props)) {
1663            /*
1664             * write final end record.  NB: want to do this even if
1665             * there was some error, because it might not be totally
1666             * failed.
1667             */
1668            dmu_replay_record_t drr = { 0 };
1669            drr.drr_type = DRR_END;
1670            if (write(outfd, &drr, sizeof (drr)) == -1) {
1671                    return (zfs_standard_error(zhp->zfs_hdl,
1672                        errno, errbuf));
```

```
1673                 }
1674                 sdd.hdr_send_sz += sizeof (drr);
1675         }

1677         if (flags->sendsize) {
1678                 if (flags->verbose) {
1679                         fprintf(stderr, "Send stream header size (bytes): "
1680                             "%u\n", sdd.hdr_send_sz);
1681                         fprintf(stderr, "Send stream data size (bytes):  "
1682                             "%llu\n", sdd.send_sz);
1683                         fprintf(stderr, "Total send stream size (bytes):  "
1684                             "%llu\n", sdd.send_sz + (uint64_t)sdd.hdr_send_sz);
1685                 } else {
1686                         fprintf(stderr, "Total send stream size (bytes):  "
1687                             "%llu\n", sdd.send_sz + (uint64_t)sdd.hdr_send_sz);
1688                 }
1689         }

1691         return (err || sdd.err);

1693 stderr_out:
1694         err = zfs_standard_error(zhp->zfs_hdl, err, errbuf);
1695 err_out:
1696         if (sdd.cleanup_fd != -1)
1697                 VERIFY(0 == close(sdd.cleanup_fd));
1698         if (flags->dedup) {
1699                 (void) pthread_cancel(tid);
1700                 (void) pthread_join(tid, NULL);
1701                 (void) close(pipefd[0]);
1702         }
1703         return (err);
1704 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
  106149 Tue Aug  7 18:11:43 2012
new/usr/src/man/man1m/zfs.1m
*** NO COMMENTS ***
**********************************************************
   1 '\" te
   2 .\" Copyright (c) 2009 Sun Microsystems, Inc. All Rights Reserved.
   3 .\" Copyright (c) 2012 by Delphix. All rights reserved.
   4 .\" Copyright 2012 Nexenta Systems, Inc. All Rights Reserved.
   4 .\" Copyright (c) 2012 Nexenta Systems, Inc. All Rights Reserved.
   5 .\" Copyright (c) 2012, Joyent, Inc. All rights reserved.
   6 .\" The contents of this file are subject to the terms of the Common Development
   7 .\"  See the License for the specific language governing permissions and limitat
   8 .\" the fields enclosed by brackets "[]" replaced with your own identifying info
   9 .\" Copyright 2011 Joshua M. Clulow <josh@sysmgr.org>
  10 .TH ZFS 1M "28 Jul 2011"
  11 .SH NAME
  12 zfs \- configures ZFS file systems
  13 .SH SYNOPSIS
  14 .LP
  15 .nf
  16 \fBzfs\fR [\fB-?\fR]
  17 .fi

  19 .LP
  20 .nf
  21 \fBzfs\fR \fBcreate\fR [\fB-p\fR] [\fB-o\fR \fIproperty\fR=\fIvalue\fR] ... \fIf
  22 .fi

  24 .LP
  25 .nf
  26 \fBzfs\fR \fBcreate\fR [\fB-ps\fR] [\fB-b\fR \fIblocksize\fR] [\fB-o\fR \fIprope
  27 .fi

  29 .LP
  30 .nf
  31 \fBzfs\fR \fBdestroy\fR [\fB-fnpRrv\fR] \fIfilesystem\fR|\fIvolume\fR
  32 .fi

  34 .LP
  35 .nf
  36 \fBzfs\fR \fBdestroy\fR [\fB-dnpRrv\fR] \fIfilesystem\fR|\fIvolume\fR@\fIsnap\fR
  37 .fi

  39 .LP
  40 .nf
  41 \fBzfs\fR \fBsnapshot\fR [\fB-r\fR] [\fB-o\fR \fIproperty\fR=\fIvalue\fR]...
  42      \fIfilesystem@snapname\fR|\fIvolume@snapname\fR...
  43 .fi

  45 .LP
  46 .nf
  47 \fBzfs\fR \fBrollback\fR [\fB-rRf\fR] \fIsnapshot\fR
  48 .fi

  50 .LP
  51 .nf
  52 \fBzfs\fR \fBclone\fR [\fB-p\fR] [\fB-o\fR \fIproperty\fR=\fIvalue\fR] ... \fIsn
  53 .fi

  55 .LP
  56 .nf
  57 \fBzfs\fR \fBpromote\fR \fIclone-filesystem\fR
  58 .fi

  60 .LP
```



```
  61 .nf
  62 \fBzfs\fR \fBrename\fR [\fB-f\fR] \fIfilesystem\fR|\fIvolume\fR|\fIsnapshot\fR
  63      \fIfilesystem\fR|\fIvolume\fR|\fIsnapshot\fR
  64 .fi

  66 .LP
  67 .nf
  68 \fBzfs\fR \fBrename\fR [\fB-fp\fR] \fIfilesystem\fR|\fIvolume\fR \fIfilesystem\f
  69 .fi

  71 .LP
  72 .nf
  73 \fBzfs\fR \fBrename\fR \fB-r\fR \fIsnapshot\fR \fIsnapshot\fR
  74 .fi

  76 .LP
  77 .nf
  78 \fBzfs\fR \fBlist\fR [\fB-r\fR|\fB-d\fR \fIdepth\fR][\fB-H\fR][\fB-o\fR \fIprope
  79      [\fB-s\fR \fIproperty\fR] ... [\fB-S\fR \fIproperty\fR] ... [\fIfilesystem\
  80 .fi

  82 .LP
  83 .nf
  84 \fBzfs\fR \fBset\fR \fIproperty\fR=\fIvalue\fR \fIfilesystem\fR|\fIvolume\fR|\fI
  85 .fi

  87 .LP
  88 .nf
  89 \fBzfs\fR \fBget\fR [\fB-r\fR|\fB-d\fR \fIdepth\fR][\fB-Hp\fR][\fB-o\fR \fIfield
  90      [\fB-s\fR \fIsource\fR[,...]] "\fIall\fR" | \fIproperty\fR[,...] \fIfilesyst
  91 .fi

  93 .LP
  94 .nf
  95 \fBzfs\fR \fBinherit\fR [\fB-r\fR] \fIproperty\fR \fIfilesystem\fR|\fIvolume|sna
  96 .fi

  98 .LP
  99 .nf
 100 \fBzfs\fR \fBupgrade\fR [\fB-v\fR]
 101 .fi

 103 .LP
 104 .nf
 105 \fBzfs\fR \fBupgrade\fR [\fB-r\fR] [\fB-V\fR \fIversion\fR] \fB-a\fR | \fIfilesy
 106 .fi

 108 .LP
 109 .nf
 110 \fBzfs\fR \fBuserspace\fR [\fB-niHp\fR] [\fB-o\fR \fIfield\fR[,...]] [\fB-sS\fR
 111      [\fB-t\fR \fItype\fR[,...]] \fIfilesystem\fR|\fIsnapshot\fR
 112 .fi

 114 .LP
 115 .nf
 116 \fBzfs\fR \fBgroupspace\fR [\fB-niHp\fR] [\fB-o\fR \fIfield\fR[,...]] [\fB-sS\fR
 117      [\fB-t\fR \fItype\fR[,...]] \fIfilesystem\fR|\fIsnapshot\fR
 118 .fi

 120 .LP
 121 .nf
 122 \fBzfs\fR \fBmount\fR
 123 .fi

 125 .LP
 126 .nf
```

```
\fBzfs\fR \fBmount\fR [\fB-vO\fR] [\fB-o \fIoptions\fR\fR] \fB-a\fR | \fIfilesys
.fi

.LP
.nf
\fBzfs\fR \fBunmount\fR [\fB-f\fR] \fB-a\fR | \fIfilesystem\fR|\fImountpoint\fR
.fi

.LP
.nf
\fBzfs\fR \fBshare\fR \fB-a\fR | \fIfilesystem\fR
.fi

.LP
.nf
\fBzfs\fR \fBunshare\fR \fB-a\fR \fIfilesystem\fR|\fImountpoint\fR
.fi

.LP
.nf
\fBzfs\fR \fBsend\fR [\fB-DnPpRrvs\fR] [\fB-\fR[\fBiI\fR] \fIsnapshot\fR] \fIsna
\fIzfs\fR \fIsend\fR [\fB-DnPpRrv\fR] [\fB-\fR[\fBiI\fR] \fIsnapshot\fR] \fIsnap
.fi

.LP
.nf
\fBzfs\fR \fBreceive\fR [\fB-vnFu\fR] \fIfilesystem\fR|\fIvolume\fR|\fIsnapshot\
.fi

.LP
.nf
\fBzfs\fR \fBreceive\fR [\fB-vnFu\fR] [\fB-d\fR|\fB-e\fR] \fIfilesystem\fR
.fi

.LP
.nf
\fBzfs\fR \fBallow\fR \fIfilesystem\fR|\fIvolume\fR
.fi

.LP
.nf
\fBzfs\fR \fBallow\fR [\fB-ldug\fR] "\fIeveryone\fR"|\fIuser\fR|\fIgroup\fR[,...
     \fIfilesystem\fR|\fIvolume\fR
.fi

.LP
.nf
\fBzfs\fR \fBallow\fR [\fB-ld\fR] \fB-e\fR \fIperm\fR|@\fIsetname\fR[,...] \fIfi
.fi

.LP
.nf
\fBzfs\fR \fBallow\fR \fB-c\fR \fIperm\fR|@\fIsetname\fR[,...] \fIfilesystem\fR|
.fi

.LP
.nf
\fBzfs\fR \fBallow\fR \fB-s\fR @\fIsetname\fR \fIperm\fR|@\fIsetname\fR[,...] \f
.fi

.LP
.nf
\fBzfs\fR \fBunallow\fR [\fB-rldug\fR] "\fIeveryone\fR"|\fIuser\fR|\fIgroup\fR[,
     \fIfilesystem\fR|\fIvolume\fR
.fi
```



```
.LP
.nf
\fBzfs\fR \fBunallow\fR [\fB-rld\fR] \fB-e\fR [\fIperm\fR|@\fIsetname\fR[,... ]]
.fi

.LP
.nf
\fBzfs\fR \fBunallow\fR [\fB-r\fR] \fB-c\fR [\fIperm\fR|@\fIsetname\fR[ ... ]] \
.fi

.LP
.nf
\fBzfs\fR \fBunallow\fR [\fB-r\fR] \fB-s\fR @\fIsetname\fR [\fIperm\fR|@\fIsetna
.fi

.LP
.nf
\fBzfs\fR \fBhold\fR [\fB-r\fR] \fItag\fR \fIsnapshot\fR...
.fi

.LP
.nf
\fBzfs\fR \fBholds\fR [\fB-r\fR] \fIsnapshot\fR...
.fi

.LP
.nf
\fBzfs\fR \fBrelease\fR [\fB-r\fR] \fItag\fR \fIsnapshot\fR...
.fi

.SH DESCRIPTION
.sp
.LP
The \fBzfs\fR command configures \fBZFS\fR datasets within a \fBZFS\fR storage
pool, as described in \fBzpool\fR(1M). A dataset is identified by a unique path
within the \fBZFS\fR namespace. For example:
.sp
.in +2
.nf
pool/{filesystem,volume,snapshot}
.fi
.in -2
.sp

.sp
.LP
where the maximum length of a dataset name is \fBMAXNAMELEN\fR (256 bytes).
.sp
.LP
A dataset can be one of the following:
.sp
.ne 2
.na
\fB\fIfile system\fR\fR
.ad
.sp .6
.RS 4n
A \fBZFS\fR dataset of type \fBfilesystem\fR can be mounted within the standard
system namespace and behaves like other file systems. While \fBZFS\fR file
systems are designed to be \fBPOSIX\fR compliant, known issues exist that
prevent compliance in some cases. Applications that depend on standards
conformance might fail due to nonstandard behavior when checking file system
free space.
.RE

.sp
```

```
.ne 2
.na
\fB\fIvolume\fR\fR
.ad
.sp .6
.RS 4n
A logical volume exported as a raw or block device. This type of dataset should
only be used under special circumstances. File systems are typically used in
most environments.
.RE

.sp
.ne 2
.na
\fB\fIsnapshot\fR\fR
.ad
.sp .6
.RS 4n
A read-only version of a file system or volume at a given point in time. It is
specified as \fIfilesystem@name\fR or \fIvolume@name\fR.
.RE

.SS "ZFS File System Hierarchy"
.sp
.LP
A \fBZFS\fR storage pool is a logical collection of devices that provide space
for datasets. A storage pool is also the root of the \fBZFS\fR file system
hierarchy.
.sp
.LP
The root of the pool can be accessed as a file system, such as mounting and
unmounting, taking snapshots, and setting properties. The physical storage
characteristics, however, are managed by the \fBzpool\fR(1M) command.
.sp
.LP
See \fBzpool\fR(1M) for more information on creating and administering pools.
.SS "Snapshots"
.sp
.LP
A snapshot is a read-only copy of a file system or volume. Snapshots can be
created extremely quickly, and initially consume no additional space within the
pool. As data within the active dataset changes, the snapshot consumes more
data than would otherwise be shared with the active dataset.
.sp
.LP
Snapshots can have arbitrary names. Snapshots of volumes can be cloned or
rolled back, but cannot be accessed independently.
.sp
.LP
File system snapshots can be accessed under the \fB\&.zfs/snapshot\fR directory
in the root of the file system. Snapshots are automatically mounted on demand
and may be unmounted at regular intervals. The visibility of the \fB\&.zfs\fR
directory can be controlled by the \fBsnapdir\fR property.
.SS "Clones"
.sp
.LP
A clone is a writable volume or file system whose initial contents are the same
as another dataset. As with snapshots, creating a clone is nearly
instantaneous, and initially consumes no additional space.
.sp
.LP
Clones can only be created from a snapshot. When a snapshot is cloned, it
creates an implicit dependency between the parent and child. Even though the
clone is created somewhere else in the dataset hierarchy, the original snapshot
cannot be destroyed as long as a clone exists. The \fBorigin\fR property
exposes this dependency, and the \fBdestroy\fR command lists any such
```



```
dependencies, if they exist.
.sp
.LP
The clone parent-child dependency relationship can be reversed by using the
\fBpromote\fR subcommand. This causes the "origin" file system to become a
clone of the specified file system, which makes it possible to destroy the file
system that the clone was created from.
.SS "Mount Points"
.sp
.LP
Creating a \fBZFS\fR file system is a simple operation, so the number of file
systems per system is likely to be numerous. To cope with this, \fBZFS\fR
automatically manages mounting and unmounting file systems without the need to
edit the \fB/etc/vfstab\fR file. All automatically managed file systems are
mounted by \fBZFS\fR at boot time.
.sp
.LP
By default, file systems are mounted under \fB/\fIpath\fR\fR, where \fIpath\fR
is the name of the file system in the \fBZFS\fR namespace. Directories are
created and destroyed as needed.
.sp
.LP
A file system can also have a mount point set in the \fBmountpoint\fR property.
This directory is created as needed, and \fBZFS\fR automatically mounts the
file system when the \fBzfs mount -a\fR command is invoked (without editing
\fB/etc/vfstab\fR). The \fBmountpoint\fR property can be inherited, so if
\fBpool/home\fR has a mount point of \fB/export/stuff\fR, then
\fBpool/home/user\fR automatically inherits a mount point of
\fB/export/stuff/user\fR.
.sp
.LP
A file system \fBmountpoint\fR property of \fBnone\fR prevents the file system
from being mounted.
.sp
.LP
If needed, \fBZFS\fR file systems can also be managed with traditional tools
(\fBmount\fR, \fBumount\fR, \fB/etc/vfstab\fR). If a file system's mount point
is set to \fBlegacy\fR, \fBZFS\fR makes no attempt to manage the file system,
and the administrator is responsible for mounting and unmounting the file
system.
.SS "Zones"
.sp
.LP
A \fBZFS\fR file system can be added to a non-global zone by using the
\fBzonecfg\fR \fBadd fs\fR subcommand. A \fBZFS\fR file system that is added to
a non-global zone must have its \fBmountpoint\fR property set to \fBlegacy\fR.
.sp
.LP
The physical properties of an added file system are controlled by the global
administrator. However, the zone administrator can create, modify, or destroy
files within the added file system, depending on how the file system is
mounted.
.sp
.LP
A dataset can also be delegated to a non-global zone by using the \fBzonecfg\fR
\fBadd dataset\fR subcommand. You cannot delegate a dataset to one zone and the
children of the same dataset to another zone. The zone administrator can change
properties of the dataset or any of its children. However, the \fBquota\fR
property is controlled by the global administrator.
.sp
.LP
A \fBZFS\fR volume can be added as a device to a non-global zone by using the
\fBzonecfg\fR \fBadd device\fR subcommand. However, its physical properties can
be modified only by the global administrator.
.sp
.LP
```

```
390 For more information about \fBzonecfg\fR syntax, see \fBzonecfg\fR(1M).
391 .sp
392 .LP
393 After a dataset is delegated to a non-global zone, the \fBzoned\fR property is
394 automatically set. A zoned file system cannot be mounted in the global zone,
395 since the zone administrator might have to set the mount point to an
396 unacceptable value.
397 .sp
398 .LP
399 The global administrator can forcibly clear the \fBzoned\fR property, though
400 this should be done with extreme care. The global administrator should verify
401 that all the mount points are acceptable before clearing the property.
402 .SS "Native Properties"
403 .sp
404 .LP
405 Properties are divided into two types, native properties and user-defined (or
406 "user") properties. Native properties either export internal statistics or
407 control \fBZFS\fR behavior. In addition, native properties are either editable
408 or read-only. User properties have no effect on \fBZFS\fR behavior, but you can
409 use them to annotate datasets in a way that is meaningful in your environment.
410 For more information about user properties, see the "User Properties" section,
411 below.
412 .sp
413 .LP
414 Every dataset has a set of properties that export statistics about the dataset
415 as well as control various behaviors. Properties are inherited from the parent
416 unless overridden by the child. Some properties apply only to certain types of
417 datasets (file systems, volumes, or snapshots).
418 .sp
419 .LP
420 The values of numeric properties can be specified using human-readable suffixes
421 (for example, \fBk\fR, \fBKB\fR, \fBM\fR, \fBGb\fR, and so forth, up to \fBZ\fR
422 for zettabyte). The following are all valid (and equal) specifications:
423 .sp
424 .in +2
425 .nf
426 1536M, 1.5g, 1.50GB
427 .fi
428 .in -2
429 .sp

431 .sp
432 .LP
433 The values of non-numeric properties are case sensitive and must be lowercase,
434 except for \fBmountpoint\fR, \fBsharenfs\fR, and \fBsharesmb\fR.
435 .sp
436 .LP
437 The following native properties consist of read-only statistics about the
438 dataset. These properties can be neither set, nor inherited. Native properties
439 apply to all dataset types unless otherwise noted.
440 .sp
441 .ne 2
442 .na
443 \fB\fBavailable\fR\fR
444 .ad
445 .sp .6
446 .RS 4n
447 The amount of space available to the dataset and all its children, assuming
448 that there is no other activity in the pool. Because space is shared within a
449 pool, availability can be limited by any number of factors, including physical
450 pool size, quotas, reservations, or other datasets within the pool.
451 .sp
452 This property can also be referred to by its shortened column name,
453 \fBavail\fR.
454 .RE
```



```
456 .sp
457 .ne 2
458 .na
459 \fB\fBcompressratio\fR\fR
460 .ad
461 .sp .6
462 .RS 4n
463 For non-snapshots, the compression ratio achieved for the \fBused\fR
464 space of this dataset, expressed as a multiplier.  The \fBused\fR
465 property includes descendant datasets, and, for clones, does not include
466 the space shared with the origin snapshot.  For snapshots, the
467 \fBcompressratio\fR is the same as the \fBrefcompressratio\fR property.
468 Compression can be turned on by running: \fBzfs set compression=on
469 \fIdataset\fR\fR. The default value is \fBoff\fR.
470 .RE

472 .sp
473 .ne 2
474 .na
475 \fB\fBcreation\fR\fR
476 .ad
477 .sp .6
478 .RS 4n
479 The time this dataset was created.
480 .RE

482 .sp
483 .ne 2
484 .na
485 \fB\fBclones\fR\fR
486 .ad
487 .sp .6
488 .RS 4n
489 For snapshots, this property is a comma-separated list of filesystems or
490 volumes which are clones of this snapshot.  The clones' \fBorigin\fR property
491 is this snapshot.  If the \fBclones\fR property is not empty, then this
492 snapshot can not be destroyed (even with the \fB-r\fR or \fB-f\fR options).
493 .RE

495 .sp
496 .ne 2
497 .na
498 \fB\fBdefer_destroy\fR\fR
499 .ad
500 .sp .6
501 .RS 4n
502 This property is \fBon\fR if the snapshot has been marked for deferred destroy
503 by using the \fBzfs destroy\fR \fB-d\fR command. Otherwise, the property is
504 \fBoff\fR.
505 .RE

507 .sp
508 .ne 2
509 .na
510 \fB\fBmounted\fR\fR
511 .ad
512 .sp .6
513 .RS 4n
514 For file systems, indicates whether the file system is currently mounted. This
515 property can be either \fByes\fR or \fBno\fR.
516 .RE

518 .sp
519 .ne 2
520 .na
521 \fB\fBorigin\fR\fR
```

.ad
.sp .6
.RS 4n
For cloned file systems or volumes, the snapshot from which the clone was created. See also the \fBclones\fR property.
.RE

.sp
.ne 2
.na
\fB\fBreferenced\fR\fR
.ad
.sp .6
.RS 4n
The amount of data that is accessible by this dataset, which may or may not be shared with other datasets in the pool. When a snapshot or clone is created, it initially references the same amount of space as the file system or snapshot it was created from, since its contents are identical.
.sp
This property can also be referred to by its shortened column name, \fBrefer\fR.
.RE

.sp
.ne 2
.na
\fB\fBrefcompressratio\fR\fR
.ad
.sp .6
.RS 4n
The compression ratio achieved for the \fBreferenced\fR space of this dataset, expressed as a multiplier. See also the \fBcompressratio\fR property.
.RE

.sp
.ne 2
.na
\fB\fBtype\fR\fR
.ad
.sp .6
.RS 4n
The type of dataset: \fBfilesystem\fR, \fBvolume\fR, or \fBsnapshot\fR.
.RE

.sp
.ne 2
.na
\fB\fBused\fR\fR
.ad
.sp .6
.RS 4n
The amount of space consumed by this dataset and all its descendents. This is the value that is checked against this dataset’s quota and reservation. The space used does not include this dataset’s reservation, but does take into account the reservations of any descendent datasets. The amount of space that a dataset consumes from its parent, as well as the amount of space that are freed if this dataset is recursively destroyed, is the greater of its space used and its reservation.
.sp
When snapshots (see the "Snapshots" section) are created, their space is initially shared between the snapshot and the file system, and possibly with previous snapshots. As the file system changes, space that was previously shared becomes unique to the snapshot, and counted in the snapshot’s space used. Additionally, deleting snapshots can increase the amount of space unique to (and used by) other snapshots.

.sp
The amount of space used, available, or referenced does not take into account pending changes. Pending changes are generally accounted for within a few seconds. Committing a change to a disk using \fBfsync\fR(3c) or \fBO_SYNC\fR does not necessarily guarantee that the space usage information is updated immediately.
.RE

.sp
.ne 2
.na
\fB\fBusedby*\fR\fR
.ad
.sp .6
.RS 4n
The \fBusedby*\fR properties decompose the \fBused\fR properties into the various reasons that space is used. Specifically, \fBused\fR = \fBusedbychildren\fR + \fBusedbydataset\fR + \fBusedbyrefreservation\fR +, \fBusedbysnapshots\fR. These properties are only available for datasets created on \fBzpool\fR "version 13" pools.
.RE

.sp
.ne 2
.na
\fB\fBusedbychildren\fR\fR
.ad
.sp .6
.RS 4n
The amount of space used by children of this dataset, which would be freed if all the dataset’s children were destroyed.
.RE

.sp
.ne 2
.na
\fB\fBusedbydataset\fR\fR
.ad
.sp .6
.RS 4n
The amount of space used by this dataset itself, which would be freed if the dataset were destroyed (after first removing any \fBrefreservation\fR and destroying any necessary snapshots or descendents).
.RE

.sp
.ne 2
.na
\fB\fBusedbyrefreservation\fR\fR
.ad
.sp .6
.RS 4n
The amount of space used by a \fBrefreservation\fR set on this dataset, which would be freed if the \fBrefreservation\fR was removed.
.RE

.sp
.ne 2
.na
\fB\fBusedbysnapshots\fR\fR
.ad
.sp .6
.RS 4n
The amount of space consumed by snapshots of this dataset. In particular, it is the amount of space that would be freed if all of this dataset’s snapshots were destroyed. Note that this is not simply the sum of the snapshots’ \fBused\fR

```
654 properties because space can be shared by multiple snapshots.
655 .RE

657 .sp
658 .ne 2
659 .na
660 \fB\fBuserused@\fR\fIuser\fR\fR
661 .ad
662 .sp .6
663 .RS 4n
664 The amount of space consumed by the specified user in this dataset. Space is
665 charged to the owner of each file, as displayed by \fBls\fR \fB-l\fR. The
666 amount of space charged is displayed by \fBdu\fR and \fBls\fR \fB-s\fR. See the
667 \fBzfs userspace\fR subcommand for more information.
668 .sp
669 Unprivileged users can access only their own space usage. The root user, or a
670 user who has been granted the \fBuserused\fR privilege with \fBzfs allow\fR,
671 can access everyone's usage.
672 .sp
673 The \fBuserused@\fR... properties are not displayed by \fBzfs get all\fR. The
674 user's name must be appended after the \fB@\fR symbol, using one of the
675 following forms:
676 .RS +4
677 .TP
678 .ie t \(bu
679 .el o
680 \fIPOSIX name\fR (for example, \fBjoe\fR)
681 .RE
682 .RS +4
683 .TP
684 .ie t \(bu
685 .el o
686 \fIPOSIX numeric ID\fR (for example, \fB789\fR)
687 .RE
688 .RS +4
689 .TP
690 .ie t \(bu
691 .el o
692 \fISID name\fR (for example, \fBjoe.smith@mydomain\fR)
693 .RE
694 .RS +4
695 .TP
696 .ie t \(bu
697 .el o
698 \fISID numeric ID\fR (for example, \fBS-1-123-456-789\fR)
699 .RE
700 .RE

702 .sp
703 .ne 2
704 .na
705 \fB\fBuserrefs\fR\fR
706 .ad
707 .sp .6
708 .RS 4n
709 This property is set to the number of user holds on this snapshot. User holds
710 are set by using the \fBzfs hold\fR command.
711 .RE

713 .sp
714 .ne 2
715 .na
716 \fB\fBgroupused@\fR\fIgroup\fR\fR
717 .ad
718 .sp .6
719 .RS 4n
```



```
720 The amount of space consumed by the specified group in this dataset. Space is
721 charged to the group of each file, as displayed by \fBls\fR \fB-l\fR. See the
722 \fBuserused@\fR\fIuser\fR property for more information.
723 .sp
724 Unprivileged users can only access their own groups' space usage. The root
725 user, or a user who has been granted the \fBgroupused\fR privilege with \fBzfs
726 allow\fR, can access all groups' usage.
727 .RE

729 .sp
730 .ne 2
731 .na
732 \fB\fBvolblocksize\fR=\fIblocksize\fR\fR
733 .ad
734 .sp .6
735 .RS 4n
736 For volumes, specifies the block size of the volume. The \fBblocksize\fR cannot
737 be changed once the volume has been written, so it should be set at volume
738 creation time. The default \fBblocksize\fR for volumes is 8 Kbytes. Any power
739 of 2 from 512 bytes to 128 Kbytes is valid.
740 .sp
741 This property can also be referred to by its shortened column name,
742 \fBvolblock\fR.
743 .RE

745 .sp
746 .ne 2
747 .na
748 \fB\fBwritten\fR\fR
749 .ad
750 .sp .6
751 .RS 4n
752 The amount of \fBreferenced\fR space written to this dataset since the
753 previous snapshot.
754 .RE

756 .sp
757 .ne 2
758 .na
759 \fB\fBwritten@\fR\fIsnapshot\fR\fR
760 .ad
761 .sp .6
762 .RS 4n
763 The amount of \fBreferenced\fR space written to this dataset since the
764 specified snapshot.  This is the space that is referenced by this dataset
765 but was not referenced by the specified snapshot.
766 .sp
767 The \fIsnapshot\fR may be specified as a short snapshot name (just the part
768 after the \fB@\fR), in which case it will be interpreted as a snapshot in
769 the same filesystem as this dataset.
770 The \fIsnapshot\fR be a full snapshot name (\fIfilesystem\fR@\fIsnapshot\fR),
771 which for clones may be a snapshot in the origin's filesystem (or the origin
772 of the origin's filesystem, etc).
773 .RE

775 .sp
776 .LP
777 The following native properties can be used to change the behavior of a
778 \fBZFS\fR dataset.
779 .sp
780 .ne 2
781 .na
782 \fB\fBaclinherit\fR=\fBdiscard\fR | \fBnoallow\fR | \fBrestricted\fR |
783 \fBpassthrough\fR | \fBpassthrough-x\fR\fR
784 .ad
785 .sp .6
```

```
786 .RS 4n
787 Controls how \fBACL\fR entries are inherited when files and directories are
788 created. A file system with an \fBaclinherit\fR property of \fBdiscard\fR does
789 not inherit any \fBACL\fR entries. A file system with an \fBaclinherit\fR
790 property value of \fBnoallow\fR only inherits inheritable \fBACL\fR entries
791 that specify "deny" permissions. The property value \fBrestricted\fR (the
792 default) removes the \fBwrite_acl\fR and \fBwrite_owner\fR permissions when the
793 \fBACL\fR entry is inherited. A file system with an \fBaclinherit\fR property
794 value of \fBpassthrough\fR inherits all inheritable \fBACL\fR entries without
795 any modifications made to the \fBACL\fR entries when they are inherited. A file
796 system with an \fBaclinherit\fR property value of \fBpassthrough-x\fR has the
797 same meaning as \fBpassthrough\fR, except that the \fBowner@\fR, \fBgroup@\fR,
798 and \fBeveryone@\fR \fBACE\fRs inherit the execute permission only if the file
799 creation mode also requests the execute bit.
800 .sp
801 When the property value is set to \fBpassthrough\fR, files are created with a
802 mode determined by the inheritable \fBACE\fRs. If no inheritable \fBACE\fRs
803 exist that affect the mode, then the mode is set in accordance to the requested
804 mode from the application.
805 .RE

807 .sp
808 .ne 2
809 .na
810 \fB\fBaclmode\fR=\fBdiscard\fR | \fBgroupmask\fR | \fBpassthrough\fR\fR
811 .ad
812 .sp .6
813 .RS 4n
814 Controls how an \fBACL\fR is modified during \fBchmod\fR(2). A file system with
815 an \fBaclmode\fR property of \fBdiscard\fR (the default) deletes all \fBACL\fR
816 entries that do not represent the mode of the file. An \fBaclmode\fR property
817 of \fBgroupmask\fR reduces permissions granted in all \fBALLOW\fR entries found
818 in the \fBACL\fR such that they are no greater than the group permissions
819 specified by \fBchmod\fR.  A file system with an \fBaclmode\fR property of
820 \fBpassthrough\fR indicates that no changes are made to the \fBACL\fR other
821 than creating or updating the necessary \fBACL\fR entries to
822 represent the new mode of the file or directory.
823 .RE

825 .sp
826 .ne 2
827 .na
828 \fB\fBatime\fR=\fBon\fR | \fBoff\fR\fR
829 .ad
830 .sp .6
831 .RS 4n
832 Controls whether the access time for files is updated when they are read.
833 Turning this property off avoids producing write traffic when reading files and
834 can result in significant performance gains, though it might confuse mailers
835 and other similar utilities. The default value is \fBon\fR.
836 .RE

838 .sp
839 .ne 2
840 .na
841 \fB\fBcanmount\fR=\fBon\fR | \fBoff\fR | \fBnoauto\fR\fR
842 .ad
843 .sp .6
844 .RS 4n
845 If this property is set to \fBoff\fR, the file system cannot be mounted, and is
846 ignored by \fBzfs mount -a\fR. Setting this property to \fBoff\fR is similar to
847 setting the \fBmountpoint\fR property to \fBnone\fR, except that the dataset
848 still has a normal \fBmountpoint\fR property, which can be inherited. Setting
849 this property to \fBoff\fR allows datasets to be used solely as a mechanism to
850 inherit properties. One example of setting \fBcanmount=\fR\fBoff\fR is to have
851 two datasets with the same \fBmountpoint\fR, so that the children of both
```



```
852 datasets appear in the same directory, but might have different inherited
853 characteristics.
854 .sp
855 When the \fBnoauto\fR option is set, a dataset can only be mounted and
856 unmounted explicitly. The dataset is not mounted automatically when the dataset
857 is created or imported, nor is it mounted by the \fBzfs mount -a\fR command or
858 unmounted by the \fBzfs unmount -a\fR command.
859 .sp
860 This property is not inherited.
861 .RE

863 .sp
864 .ne 2
865 .na
866 \fB\fBchecksum\fR=\fBon\fR | \fBoff\fR | \fBfletcher2,\fR| \fBfletcher4\fR |
867 \fBsha256\fR\fR
868 .ad
869 .sp .6
870 .RS 4n
871 Controls the checksum used to verify data integrity. The default value is
872 \fBon\fR, which automatically selects an appropriate algorithm (currently,
873 \fBfletcher2\fR, but this may change in future releases). The value \fBoff\fR
874 disables integrity checking on user data. Disabling checksums is \fBNOT\fR a
875 recommended practice.
876 .sp
877 Changing this property affects only newly-written data.
878 .RE

880 .sp
881 .ne 2
882 .na
883 \fB\fBcompression\fR=\fBon\fR | \fBoff\fR | \fBlzjb\fR | \fBgzip\fR |
884 \fBgzip-\fR\fIN\fR | \fBzle\fR\fR
885 .ad
886 .sp .6
887 .RS 4n
888 Controls the compression algorithm used for this dataset. The \fBlzjb\fR
889 compression algorithm is optimized for performance while providing decent data
890 compression. Setting compression to \fBon\fR uses the \fBlzjb\fR compression
891 algorithm. The \fBgzip\fR compression algorithm uses the same compression as
892 the \fBgzip\fR(1) command. You can specify the \fBgzip\fR level by using the
893 value \fBgzip-\fR\fIN\fR where \fIN\fR is an integer from 1 (fastest) to 9
894 (best compression ratio). Currently, \fBgzip\fR is equivalent to \fBgzip-6\fR
895 (which is also the default for \fBgzip\fR(1)). The \fBzle\fR compression
896 algorithm compresses runs of zeros.
897 .sp
898 This property can also be referred to by its shortened column name
899 \fBcompress\fR. Changing this property affects only newly-written data.
900 .RE

902 .sp
903 .ne 2
904 .na
905 \fB\fBcopies\fR=\fB1\fR | \fB2\fR | \fB3\fR\fR
906 .ad
907 .sp .6
908 .RS 4n
909 Controls the number of copies of data stored for this dataset. These copies are
910 in addition to any redundancy provided by the pool, for example, mirroring or
911 RAID-Z. The copies are stored on different disks, if possible. The space used
912 by multiple copies is charged to the associated file and dataset, changing the
913 \fBused\fR property and counting against quotas and reservations.
914 .sp
915 Changing this property only affects newly-written data. Therefore, set this
916 property at file system creation time by using the \fB-o\fR
917 \fBcopies=\fR\fIN\fR option.
```

```
 918 .RE

 920 .sp
 921 .ne 2
 922 .na
 923 \fB\fBdevices\fR=\fBon\fR | \fBoff\fR\fR
 924 .ad
 925 .sp .6
 926 .RS 4n
 927 Controls whether device nodes can be opened on this file system. The default
 928 value is \fBon\fR.
 929 .RE

 931 .sp
 932 .ne 2
 933 .na
 934 \fB\fBexec\fR=\fBon\fR | \fBoff\fR\fR
 935 .ad
 936 .sp .6
 937 .RS 4n
 938 Controls whether processes can be executed from within this file system. The
 939 default value is \fBon\fR.
 940 .RE

 942 .sp
 943 .ne 2
 944 .na
 945 \fB\fBmountpoint\fR=\fIpath\fR | \fBnone\fR | \fBlegacy\fR\fR
 946 .ad
 947 .sp .6
 948 .RS 4n
 949 Controls the mount point used for this file system. See the "Mount Points"
 950 section for more information on how this property is used.
 951 .sp
 952 When the \fBmountpoint\fR property is changed for a file system, the file
 953 system and any children that inherit the mount point are unmounted. If the new
 954 value is \fBlegacy\fR, then they remain unmounted. Otherwise, they are
 955 automatically remounted in the new location if the property was previously
 956 \fBlegacy\fR or \fBnone\fR, or if they were mounted before the property was
 957 changed. In addition, any shared file systems are unshared and shared in the
 958 new location.
 959 .RE

 961 .sp
 962 .ne 2
 963 .na
 964 \fB\fBnbmand\fR=\fBon\fR | \fBoff\fR\fR
 965 .ad
 966 .sp .6
 967 .RS 4n
 968 Controls whether the file system should be mounted with \fBnbmand\fR (Non
 969 Blocking mandatory locks). This is used for \fBCIFS\fR clients. Changes to this
 970 property only take effect when the file system is umounted and remounted. See
 971 \fBmount\fR(1M) for more information on \fBnbmand\fR mounts.
 972 .RE

 974 .sp
 975 .ne 2
 976 .na
 977 \fB\fBprimarycache\fR=\fBall\fR | \fBnone\fR | \fBmetadata\fR\fR
 978 .ad
 979 .sp .6
 980 .RS 4n
 981 Controls what is cached in the primary cache (ARC). If this property is set to
 982 \fBall\fR, then both user data and metadata is cached. If this property is set
 983 to \fBnone\fR, then neither user data nor metadata is cached. If this property
```



```
 984 is set to \fBmetadata\fR, then only metadata is cached. The default value is
 985 \fBall\fR.
 986 .RE

 988 .sp
 989 .ne 2
 990 .na
 991 \fB\fBquota\fR=\fIsize\fR | \fBnone\fR\fR
 992 .ad
 993 .sp .6
 994 .RS 4n
 995 Limits the amount of space a dataset and its descendents can consume. This
 996 property enforces a hard limit on the amount of space used. This includes all
 997 space consumed by descendents, including file systems and snapshots. Setting a
 998 quota on a descendent of a dataset that already has a quota does not override
 999 the ancestor's quota, but rather imposes an additional limit.
1000 .sp
1001 Quotas cannot be set on volumes, as the \fBvolsize\fR property acts as an
1002 implicit quota.
1003 .RE

1005 .sp
1006 .ne 2
1007 .na
1008 \fB\fBuserquota@\fR\fIuser\fR=\fIsize\fR | \fBnone\fR\fR
1009 .ad
1010 .sp .6
1011 .RS 4n
1012 Limits the amount of space consumed by the specified user. User space
1013 consumption is identified by the \fBuserspace@\fR\fIuser\fR property.
1014 .sp
1015 Enforcement of user quotas may be delayed by several seconds. This delay means
1016 that a user might exceed their quota before the system notices that they are
1017 over quota and begins to refuse additional writes with the \fBEDQUOT\fR error
1018 message . See the \fBzfs userspace\fR subcommand for more information.
1019 .sp
1020 Unprivileged users can only access their own groups' space usage. The root
1021 user, or a user who has been granted the \fBuserquota\fR privilege with \fBzfs
1022 allow\fR, can get and set everyone's quota.
1023 .sp
1024 This property is not available on volumes, on file systems before version 4, or
1025 on pools before version 15. The \fBuserquota@\fR... properties are not
1026 displayed by \fBzfs get all\fR. The user's name must be appended after the
1027 \fB@\fR symbol, using one of the following forms:
1028 .RS +4
1029 .TP
1030 .ie t \(bu
1031 .el o
1032 \fIPOSIX name\fR (for example, \fBjoe\fR)
1033 .RE
1034 .RS +4
1035 .TP
1036 .ie t \(bu
1037 .el o
1038 \fIPOSIX numeric ID\fR (for example, \fB789\fR)
1039 .RE
1040 .RS +4
1041 .TP
1042 .ie t \(bu
1043 .el o
1044 \fISID name\fR (for example, \fBjoe.smith@mydomain\fR)
1045 .RE
1046 .RS +4
1047 .TP
1048 .ie t \(bu
1049 .el o
```

\fISID numeric ID\fR (for example, \fBS-1-123-456-789\fR)
.RE
.RE

.sp
.ne 2
.na
\fB\fBgroupquota@\fR\fIgroup\fR=\fIsize\fR | \fBnone\fR\fR
.ad
.sp .6
.RS 4n
Limits the amount of space consumed by the specified group. Group space consumption is identified by the \fBuserquota@\fR\fIuser\fR property.
.sp
Unprivileged users can access only their own groups' space usage. The root user, or a user who has been granted the \fBgroupquota\fR privilege with \fBzfs allow\fR, can get and set all groups' quotas.
.RE

.sp
.ne 2
.na
\fB\fBreadonly\fR=\fBon\fR | \fBoff\fR\fR
.ad
.sp .6
.RS 4n
Controls whether this dataset can be modified. The default value is \fBoff\fR.
.sp
This property can also be referred to by its shortened column name, \fBrdonly\fR.
.RE

.sp
.ne 2
.na
\fB\fBrecordsize\fR=\fIsize\fR\fR
.ad
.sp .6
.RS 4n
Specifies a suggested block size for files in the file system. This property is designed solely for use with database workloads that access files in fixed-size records. \fBZFS\fR automatically tunes block sizes according to internal algorithms optimized for typical access patterns.
.sp
For databases that create very large files but access them in small random chunks, these algorithms may be suboptimal. Specifying a \fBrecordsize\fR greater than or equal to the record size of the database can result in significant performance gains. Use of this property for general purpose file systems is strongly discouraged, and may adversely affect performance.
.sp
The size specified must be a power of two greater than or equal to 512 and less than or equal to 128 Kbytes.
.sp
Changing the file system's \fBrecordsize\fR affects only files created afterward; existing files are unaffected.
.sp
This property can also be referred to by its shortened column name, \fBrecsize\fR.
.RE

.sp
.ne 2
.na
\fB\fBrefquota\fR=\fIsize\fR | \fBnone\fR\fR
.ad
.sp .6

.RS 4n
Limits the amount of space a dataset can consume. This property enforces a hard limit on the amount of space used. This hard limit does not include space used by descendents, including file systems and snapshots.
.RE

.sp
.ne 2
.na
\fB\fBrefreservation\fR=\fIsize\fR | \fBnone\fR\fR
.ad
.sp .6
.RS 4n
The minimum amount of space guaranteed to a dataset, not including its descendents. When the amount of space used is below this value, the dataset is treated as if it were taking up the amount of space specified by \fBrefreservation\fR. The \fBrefreservation\fR reservation is accounted for in the parent datasets' space used, and counts against the parent datasets' quotas and reservations.
.sp
If \fBrefreservation\fR is set, a snapshot is only allowed if there is enough free pool space outside of this reservation to accommodate the current number of "referenced" bytes in the dataset.
.sp
This property can also be referred to by its shortened column name, \fBrefreserv\fR.
.RE

.sp
.ne 2
.na
\fB\fBreservation\fR=\fIsize\fR | \fBnone\fR\fR
.ad
.sp .6
.RS 4n
The minimum amount of space guaranteed to a dataset and its descendents. When the amount of space used is below this value, the dataset is treated as if it were taking up the amount of space specified by its reservation. Reservations are accounted for in the parent datasets' space used, and count against the parent datasets' quotas and reservations.
.sp
This property can also be referred to by its shortened column name, \fBreserv\fR.
.RE

.sp
.ne 2
.na
\fB\fBsecondarycache\fR=\fBall\fR | \fBnone\fR | \fBmetadata\fR\fR
.ad
.sp .6
.RS 4n
Controls what is cached in the secondary cache (L2ARC). If this property is set to \fBall\fR, then both user data and metadata is cached. If this property is set to \fBnone\fR, then neither user data nor metadata is cached. If this property is set to \fBmetadata\fR, then only metadata is cached. The default value is \fBall\fR.
.RE

.sp
.ne 2
.na
\fB\fBsetuid\fR=\fBon\fR | \fBoff\fR\fR
.ad
.sp .6
.RS 4n

Controls whether the set-\fBUID\fR bit is respected for the file system. The
default value is \fBon\fR.
.RE

.sp
.ne 2
.na
\fB\fBshareiscsi\fR=\fBon\fR | \fBoff\fR\fR
.ad
.sp .6
.RS 4n
Like the \fBsharenfs\fR property, \fBshareiscsi\fR indicates whether a
\fBZFS\fR volume is exported as an \fBiSCSI\fR target. The acceptable values
for this property are \fBon\fR, \fBoff\fR, and \fBtype=disk\fR. The default
value is \fBoff\fR. In the future, other target types might be supported. For
example, \fBtape\fR.
.sp
You might want to set \fBshareiscsi=on\fR for a file system so that all
\fBZFS\fR volumes within the file system are shared by default. However,
setting this property on a file system has no direct effect.
.RE

.sp
.ne 2
.na
\fB\fBsharesmb\fR=\fBon\fR | \fBoff\fR | \fIopts\fR\fR
.ad
.sp .6
.RS 4n
Controls whether the file system is shared by using the Solaris \fBCIFS\fR
service, and what options are to be used. A file system with the \fBsharesmb\fR
property set to \fBoff\fR is managed through traditional tools such as
\fBsharemgr\fR(1M). Otherwise, the file system is automatically shared and
unshared with the \fBzfs share\fR and \fBzfs unshare\fR commands. If the
property is set to \fBon\fR, the \fBsharemgr\fR(1M) command is invoked with no
options. Otherwise, the \fBsharemgr\fR(1M) command is invoked with options
equivalent to the contents of this property.
.sp
Because \fBSMB\fR shares requires a resource name, a unique resource name is
constructed from the dataset name. The constructed name is a copy of the
dataset name except that the characters in the dataset name, which would be
illegal in the resource name, are replaced with underscore (\fB_\fR)
characters. A pseudo property "name" is also supported that allows you to
replace the data set name with a specified name. The specified name is then
used to replace the prefix dataset in the case of inheritance. For example, if
the dataset \fBdata/home/john\fR is set to \fBname=john\fR, then
\fBdata/home/john\fR has a resource name of \fBjohn\fR. If a child dataset of
\fBdata/home/john/backups\fR, it has a resource name of \fBjohn_backups\fR.
.sp
When SMB shares are created, the SMB share name appears as an entry in the
\fB\&.zfs/shares\fR directory. You can use the \fBls\fR or \fBchmod\fR command
to display the share-level ACLs on the entries in this directory.
.sp
When the \fBsharesmb\fR property is changed for a dataset, the dataset and any
children inheriting the property are re-shared with the new options, only if
the property was previously set to \fBoff\fR, or if they were shared before the
property was changed. If the new property is set to \fBoff\fR, the file systems
are unshared.
.RE

.sp
.ne 2
.na
\fB\fBsharenfs\fR=\fBon\fR | \fBoff\fR | \fIopts\fR\fR
.ad
.sp .6

.RS 4n
Controls whether the file system is shared via \fBNFS\fR, and what options are
used. A file system with a \fBsharenfs\fR property of \fBoff\fR is managed
through traditional tools such as \fBshare\fR(1M), \fBunshare\fR(1M), and
\fBdfstab\fR(4). Otherwise, the file system is automatically shared and
unshared with the \fBzfs share\fR and \fBzfs unshare\fR commands. If the
property is set to \fBon\fR, the \fBshare\fR(1M) command is invoked with no
options. Otherwise, the \fBshare\fR(1M) command is invoked with options
equivalent to the contents of this property.
.sp
When the \fBsharenfs\fR property is changed for a dataset, the dataset and any
children inheriting the property are re-shared with the new options, only if
the property was previously \fBoff\fR, or if they were shared before the
property was changed. If the new property is \fBoff\fR, the file systems are
unshared.
.RE

.sp
.ne 2
.na
\fB\fBlogbias\fR = \fBlatency\fR | \fBthroughput\fR\fR
.ad
.sp .6
.RS 4n
Provide a hint to ZFS about handling of synchronous requests in this dataset.
If \fBlogbias\fR is set to \fBlatency\fR (the default), ZFS will use pool log
devices (if configured) to handle the requests at low latency. If \fBlogbias\fR
is set to \fBthroughput\fR, ZFS will not use configured pool log devices. ZFS
will instead optimize synchronous operations for global pool throughput and
efficient use of resources.
.RE

.sp
.ne 2
.na
\fB\fBsnapdir\fR=\fBhidden\fR | \fBvisible\fR\fR
.ad
.sp .6
.RS 4n
Controls whether the \fB\&.zfs\fR directory is hidden or visible in the root of
the file system as discussed in the "Snapshots" section. The default value is
\fBhidden\fR.
.RE

.sp
.ne 2
.na
\fB\fBsync\fR=\fBdefault\fR | \fBalways\fR | \fBdisabled\fR\fR
.ad
.sp .6
.RS 4n
Controls the behavior of synchronous requests (e.g. fsync, O_DSYNC).
\fBdefault\fR is the POSIX specified behavior of ensuring all synchronous
requests are written to stable storage and all devices are flushed to ensure
data is not cached by device controllers (this is the default). \fBalways\fR
causes every file system transaction to be written and flushed before its
system call returns. This has a large performance penalty. \fBdisabled\fR
disables synchronous requests. File system transactions are only committed to
stable storage periodically. This option will give the highest performance.
However, it is very dangerous as ZFS would be ignoring the synchronous
transaction demands of applications such as databases or NFS. Administrators
should only use this option when the risks are understood.
.RE

.sp
.ne 2

```
.na
\fB\fBversion\fR=\fB1\fR | \fB2\fR | \fBcurrent\fR\fR
.ad
.sp .6
.RS 4n
The on-disk version of this file system, which is independent of the pool
version. This property can only be set to later supported versions. See the
\fBzfs upgrade\fR command.
.RE

.sp
.ne 2
.na
\fB\fBvolsize\fR=\fIsize\fR\fR
.ad
.sp .6
.RS 4n
For volumes, specifies the logical size of the volume. By default, creating a
volume establishes a reservation of equal size. For storage pools with a
version number of 9 or higher, a \fBrefreservation\fR is set instead. Any
changes to \fBvolsize\fR are reflected in an equivalent change to the
reservation (or \fBrefreservation\fR). The \fBvolsize\fR can only be set to a
multiple of \fBvolblocksize\fR, and cannot be zero.
.sp
The reservation is kept equal to the volume's logical size to prevent
unexpected behavior for consumers. Without the reservation, the volume could
run out of space, resulting in undefined behavior or data corruption, depending
on how the volume is used. These effects can also occur when the volume size is
changed while it is in use (particularly when shrinking the size). Extreme care
should be used when adjusting the volume size.
.sp
Though not recommended, a "sparse volume" (also known as "thin provisioning")
can be created by specifying the \fB-s\fR option to the \fBzfs create -V\fR
command, or by changing the reservation after the volume has been created. A
"sparse volume" is a volume where the reservation is less then the volume size.
Consequently, writes to a sparse volume can fail with \fBENOSPC\fR when the
pool is low on space. For a sparse volume, changes to \fBvolsize\fR are not
reflected in the reservation.
.RE

.sp
.ne 2
.na
\fB\fBvscan\fR=\fBon\fR | \fBoff\fR\fR
.ad
.sp .6
.RS 4n
Controls whether regular files should be scanned for viruses when a file is
opened and closed. In addition to enabling this property, the virus scan
service must also be enabled for virus scanning to occur. The default value is
\fBoff\fR.
.RE

.sp
.ne 2
.na
\fB\fBxattr\fR=\fBon\fR | \fBoff\fR\fR
.ad
.sp .6
.RS 4n
Controls whether extended attributes are enabled for this file system. The
default value is \fBon\fR.
.RE

.sp
.ne 2
```



```
.na
\fB\fBzoned\fR=\fBon\fR | \fBoff\fR\fR
.ad
.sp .6
.RS 4n
Controls whether the dataset is managed from a non-global zone. See the "Zones"
section for more information. The default value is \fBoff\fR.
.RE

.sp
.LP
The following three properties cannot be changed after the file system is
created, and therefore, should be set when the file system is created. If the
properties are not set with the \fBzfs create\fR or \fBzpool create\fR
commands, these properties are inherited from the parent dataset. If the parent
dataset lacks these properties due to having been created prior to these
features being supported, the new file system will have the default values for
these properties.
.sp
.ne 2
.na
\fB\fBcasesensitivity\fR=\fBsensitive\fR | \fBinsensitive\fR | \fBmixed\fR\fR
.ad
.sp .6
.RS 4n
Indicates whether the file name matching algorithm used by the file system
should be case-sensitive, case-insensitive, or allow a combination of both
styles of matching. The default value for the \fBcasesensitivity\fR property is
\fBsensitive\fR. Traditionally, UNIX and POSIX file systems have case-sensitive
file names.
.sp
The \fBmixed\fR value for the \fBcasesensitivity\fR property indicates that the
file system can support requests for both case-sensitive and case-insensitive
matching behavior. Currently, case-insensitive matching behavior on a file
system that supports mixed behavior is limited to the Solaris CIFS server
product. For more information about the \fBmixed\fR value behavior, see the
\fISolaris ZFS Administration Guide\fR.
.RE

.sp
.ne 2
.na
\fB\fBnormalization\fR = \fBnone\fR | \fBformC\fR | \fBformD\fR | \fBformKC\fR
| \fBformKD\fR\fR
.ad
.sp .6
.RS 4n
Indicates whether the file system should perform a \fBunicode\fR normalization
of file names whenever two file names are compared, and which normalization
algorithm should be used. File names are always stored unmodified, names are
normalized as part of any comparison process. If this property is set to a
legal value other than \fBnone\fR, and the \fButf8only\fR property was left
unspecified, the \fButf8only\fR property is automatically set to \fBon\fR. The
default value of the \fBnormalization\fR property is \fBnone\fR. This property
cannot be changed after the file system is created.
.RE

.sp
.ne 2
.na
\fB\fButf8only\fR=\fBon\fR | \fBoff\fR\fR
.ad
.sp .6
.RS 4n
Indicates whether the file system should reject file names that include
characters that are not present in the \fBUTF-8\fR character code set. If this
```

property is explicitly set to \fBoff\fR, the normalization property must either not be explicitly set or be set to \fBnone\fR. The default value for the \fButf8only\fR property is \fBoff\fR. This property cannot be changed after the file system is created.
.RE

.sp
.LP
The \fBcasesensitivity\fR, \fBnormalization\fR, and \fButf8only\fR properties are also new permissions that can be assigned to non-privileged users by using the \fBZFS\fR delegated administration feature.
.SS "Temporary Mount Point Properties"
.sp
.LP
When a file system is mounted, either through \fBmount\fR(1M) for legacy mounts or the \fBzfs mount\fR command for normal file systems, its mount options are set according to its properties. The correlation between properties and mount options is as follows:
.sp
.in +2
.nf
    PROPERTY   MOUNT OPTION
     devices   devices/nodevices
     exec   exec/noexec
     readonly   ro/rw
     setuid   setuid/nosetuid
     xattr   xattr/noxattr
.fi
.in -2
.sp

.sp
.LP
In addition, these options can be set on a per-mount basis using the \fB-o\fR option, without affecting the property that is stored on disk. The values specified on the command line override the values stored in the dataset. The \fB-nosuid\fR option is an alias for \fBnodevices,nosetuid\fR. These properties are reported as "temporary" by the \fBzfs get\fR command. If the properties are changed while the dataset is mounted, the new setting overrides any temporary settings.
.SS "User Properties"
.sp
.LP
In addition to the standard native properties, \fBZFS\fR supports arbitrary user properties. User properties have no effect on \fBZFS\fR behavior, but applications or administrators can use them to annotate datasets (file systems, volumes, and snapshots).
.sp
.LP
User property names must contain a colon (\fB:\fR) character to distinguish them from native properties. They may contain lowercase letters, numbers, and the following punctuation characters: colon (\fB:\fR), dash (\fB-\fR), period (\fB\&.\fR), and underscore (\fB_\fR). The expected convention is that the property name is divided into two portions such as \fImodule\fR\fB:\fR\fIproperty\fR, but this namespace is not enforced by \fBZFS\fR. User property names can be at most 256 characters, and cannot begin with a dash (\fB-\fR).
.sp
.LP
When making programmatic use of user properties, it is strongly suggested to use a reversed \fBDNS\fR domain name for the \fImodule\fR component of property names to reduce the chance that two independently-developed packages use the same property name for different purposes. Property names beginning with \fBcom.sun\fR. are reserved for use by Sun Microsystems.
.sp
.LP

The values of user properties are arbitrary strings, are always inherited, and are never validated. All of the commands that operate on properties (\fBzfs list\fR, \fBzfs get\fR, \fBzfs set\fR, and so forth) can be used to manipulate both native properties and user properties. Use the \fBzfs inherit\fR command to clear a user property . If the property is not defined in any parent dataset, it is removed entirely. Property values are limited to 1024 characters.
.SS "ZFS Volumes as Swap or Dump Devices"
.sp
.LP
During an initial installation a swap device and dump device are created on \fBZFS\fR volumes in the \fBZFS\fR root pool. By default, the swap area size is based on 1/2 the size of physical memory up to 2 Gbytes. The size of the dump device depends on the kernel’s requirements at installation time. Separate \fBZFS\fR volumes must be used for the swap area and dump devices. Do not swap to a file on a \fBZFS\fR file system. A \fBZFS\fR swap file configuration is not supported.
.sp
.LP
If you need to change your swap area or dump device after the system is installed or upgraded, use the \fBswap\fR(1M) and \fBdumpadm\fR(1M) commands. If you need to change the size of your swap area or dump device, see the \fISolaris ZFS Administration Guide\fR.
.SH SUBCOMMANDS
.sp
.LP
All subcommands that modify state are logged persistently to the pool in their original form.
.sp
.ne 2
.na
\fB\fBzfs ?\fR\fR
.ad
.sp .6
.RS 4n
Displays a help message.
.RE

.sp
.ne 2
.na
\fB\fBzfs create\fR [\fB-p\fR] [\fB-o\fR \fIproperty\fR=\fIvalue\fR] ...
\fIfilesystem\fR\fR
.ad
.sp .6
.RS 4n
Creates a new \fBZFS\fR file system. The file system is automatically mounted according to the \fBmountpoint\fR property inherited from the parent.
.sp
.ne 2
.na
\fB\fB-p\fR\fR
.ad
.sp .6
.RS 4n
Creates all the non-existing parent datasets. Datasets created in this manner are automatically mounted according to the \fBmountpoint\fR property inherited from their parent. Any property specified on the command line using the \fB-o\fR option is ignored. If the target filesystem already exists, the operation completes successfully.
.RE

.sp
.ne 2
.na
\fB\fB-o\fR \fIproperty\fR=\fIvalue\fR\fR

```
.ad
.sp .6
.RS 4n
Sets the specified property as if the command \fBzfs set\fR
\fIproperty\fR=\fIvalue\fR was invoked at the same time the dataset was
created. Any editable \fBZFS\fR property can also be set at creation time.
Multiple \fB-o\fR options can be specified. An error results if the same
property is specified in multiple \fB-o\fR options.
.RE

.RE

.sp
.ne 2
.na
\fB\fBzfs create\fR [\fB-ps\fR] [\fB-b\fR \fIblocksize\fR] [\fB-o\fR
\fIproperty\fR=\fIvalue\fR] ... \fB-V\fR \fIsize\fR \fIvolume\fR\fR
.ad
.sp .6
.RS 4n
Creates a volume of the given size. The volume is exported as a block device in
\fB/dev/zvol/{dsk,rdsk}/\fR\fIpath\fR, where \fIpath\fR is the name of the
volume in the \fBZFS\fR namespace. The size represents the logical size as
exported by the device. By default, a reservation of equal size is created.
.sp
\fIsize\fR is automatically rounded up to the nearest 128 Kbytes to ensure that
the volume has an integral number of blocks regardless of \fIblocksize\fR.
.sp
.ne 2
.na
\fB\fB-p\fR\fR
.ad
.sp .6
.RS 4n
Creates all the non-existing parent datasets. Datasets created in this manner
are automatically mounted according to the \fBmountpoint\fR property inherited
from their parent. Any property specified on the command line using the
\fB-o\fR option is ignored. If the target filesystem already exists, the
operation completes successfully.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR\fR
.ad
.sp .6
.RS 4n
Creates a sparse volume with no reservation. See \fBvolsize\fR in the Native
Properties section for more information about sparse volumes.
.RE

.sp
.ne 2
.na
\fB\fB-o\fR \fIproperty\fR=\fIvalue\fR\fR
.ad
.sp .6
.RS 4n
Sets the specified property as if the \fBzfs set\fR \fIproperty\fR=\fIvalue\fR
command was invoked at the same time the dataset was created. Any editable
\fBZFS\fR property can also be set at creation time. Multiple \fB-o\fR options
can be specified. An error results if the same property is specified in
multiple \fB-o\fR options.
.RE
```



```
.sp
.ne 2
.na
\fB\fB-b\fR \fIblocksize\fR\fR
.ad
.sp .6
.RS 4n
Equivalent to \fB-o\fR \fBvolblocksize\fR=\fIblocksize\fR. If this option is
specified in conjunction with \fB-o\fR \fBvolblocksize\fR, the resulting
behavior is undefined.
.RE

.RE

.sp
.ne 2
.na
\fBzfs destroy\fR [\fB-fnpRrv\fR] \fIfilesystem\fR|\fIvolume\fR
.ad
.sp .6
.RS 4n
Destroys the given dataset. By default, the command unshares any file systems
that are currently shared, unmounts any file systems that are currently
mounted, and refuses to destroy a dataset that has active dependents (children
or clones).
.sp
.ne 2
.na
\fB\fB-r\fR\fR
.ad
.sp .6
.RS 4n
Recursively destroy all children.
.RE

.sp
.ne 2
.na
\fB\fB-R\fR\fR
.ad
.sp .6
.RS 4n
Recursively destroy all dependents, including cloned file systems outside the
target hierarchy.
.RE

.sp
.ne 2
.na
\fB\fB-f\fR\fR
.ad
.sp .6
.RS 4n
Force an unmount of any file systems using the \fBunmount -f\fR command. This
option has no effect on non-file systems or unmounted file systems.
.RE

.sp
.ne 2
.na
\fB\fB-n\fR\fR
.ad
.sp .6
.RS 4n
Do a dry-run ("No-op") deletion.  No data will be deleted.  This is
useful in conjunction with the \fB-v\fR or \fB-p\fR flags to determine what
```

data would be deleted.
.RE

.sp
.ne 2
.na
\fB\fB-p\fR\fR
.ad
.sp .6
.RS 4n
Print machine-parsable verbose information about the deleted data.
.RE

.sp
.ne 2
.na
\fB\fB-v\fR\fR
.ad
.sp .6
.RS 4n
Print verbose information about the deleted data.
.RE
.sp
Extreme care should be taken when applying either the \fB-r\fR or the \fB-R\fR
options, as they can destroy large portions of a pool and cause unexpected
behavior for mounted file systems in use.
.RE

.sp
.ne 2
.na
\fBzfs destroy\fR [\fB-dnpRrv\fR] \fIfilesystem\fR|\fIvolume\fR@\fIsnap\fR[%\fIs
.ad
.sp .6
.RS 4n
The given snapshots are destroyed immediately if and only if the \fBzfs
destroy\fR command without the \fB-d\fR option would have destroyed it. Such
immediate destruction would occur, for example, if the snapshot had no clones
and the user-initiated reference count were zero.
.sp
If a snapshot does not qualify for immediate destruction, it is marked for
deferred deletion. In this state, it exists as a usable, visible snapshot until
both of the preconditions listed above are met, at which point it is destroyed.
.sp
An inclusive range of snapshots may be specified by separating the
first and last snapshots with a percent sign.
The first and/or last snapshots may be left blank, in which case the
filesystem’s oldest or newest snapshot will be implied.
.sp
Multiple snapshots
(or ranges of snapshots) of the same filesystem or volume may be specified
in a comma-separated list of snapshots.
Only the snapshot’s short name (the
part after the \fB@\fR) should be specified when using a range or
comma-separated list to identify multiple snapshots.
.sp
.ne 2
.na
\fB\fB-d\fR\fR
.ad
.sp .6
.RS 4n
Defer snapshot deletion.
.RE

.sp

.ne 2
.na
\fB\fB-r\fR\fR
.ad
.sp .6
.RS 4n
Destroy (or mark for deferred deletion) all snapshots with this name in
descendent file systems.
.RE

.sp
.ne 2
.na
\fB\fB-R\fR\fR
.ad
.sp .6
.RS 4n
Recursively destroy all dependents.
.RE

.sp
.ne 2
.na
\fB\fB-n\fR\fR
.ad
.sp .6
.RS 4n
Do a dry-run ("No-op") deletion. No data will be deleted. This is
useful in conjunction with the \fB-v\fR or \fB-p\fR flags to determine what
data would be deleted.
.RE

.sp
.ne 2
.na
\fB\fB-p\fR\fR
.ad
.sp .6
.RS 4n
Print machine-parsable verbose information about the deleted data.
.RE

.sp
.ne 2
.na
\fB\fB-v\fR\fR
.ad
.sp .6
.RS 4n
Print verbose information about the deleted data.
.RE

.sp
Extreme care should be taken when applying either the \fB-r\fR or the \fB-f\fR
options, as they can destroy large portions of a pool and cause unexpected
behavior for mounted file systems in use.
.RE

.RE

.sp
.ne 2
.na
\fB\fBzfs snapshot\fR [\fB-r\fR] [\fB-o\fR \fIproperty\fR=\fIvalue\fR] ...
\fIfilesystem@snapname\fR|\fIvolume@snapname\fR\fR...
.ad

```
1842 .sp .6
1843 .RS 4n
1844 Creates snapshots with the given names. All previous modifications by
1845 successful system calls to the file system are part of the snapshots.
1846 Snapshots are taken atomically, so that all snapshots correspond to the same
1847 moment in time. See the "Snapshots" section for details.
1848 .sp
1849 .ne 2
1850 .na
1851 \fB\fB-r\fR\fR
1852 .ad
1853 .sp .6
1854 .RS 4n
1855 Recursively create snapshots of all descendent datasets
1856 .RE

1858 .sp
1859 .ne 2
1860 .na
1861 \fB\fB-o\fR \fIproperty\fR=\fIvalue\fR\fR
1862 .ad
1863 .sp .6
1864 .RS 4n
1865 Sets the specified property; see \fBzfs create\fR for details.
1866 .RE

1868 .RE

1870 .sp
1871 .ne 2
1872 .na
1873 \fB\fBzfs rollback\fR [\fB-rRf\fR] \fIsnapshot\fR\fR
1874 .ad
1875 .sp .6
1876 .RS 4n
1877 Roll back the given dataset to a previous snapshot. When a dataset is rolled
1878 back, all data that has changed since the snapshot is discarded, and the
1879 dataset reverts to the state at the time of the snapshot. By default, the
1880 command refuses to roll back to a snapshot other than the most recent one. In
1881 order to do so, all intermediate snapshots must be destroyed by specifying the
1882 \fB-r\fR option.
1883 .sp
1884 The \fB-rR\fR options do not recursively destroy the child snapshots of a
1885 recursive snapshot. Only the top-level recursive snapshot is destroyed by
1886 either of these options. To completely roll back a recursive snapshot, you must
1887 rollback the individual child snapshots.
1888 .sp
1889 .ne 2
1890 .na
1891 \fB\fB-r\fR\fR
1892 .ad
1893 .sp .6
1894 .RS 4n
1895 Recursively destroy any snapshots more recent than the one specified.
1896 .RE

1898 .sp
1899 .ne 2
1900 .na
1901 \fB\fB-R\fR\fR
1902 .ad
1903 .sp .6
1904 .RS 4n
1905 Recursively destroy any more recent snapshots, as well as any clones of those
1906 snapshots.
1907 .RE
```



```
1909 .sp
1910 .ne 2
1911 .na
1912 \fB\fB-f\fR\fR
1913 .ad
1914 .sp .6
1915 .RS 4n
1916 Used with the \fB-R\fR option to force an unmount of any clone file systems
1917 that are to be destroyed.
1918 .RE

1920 .RE

1922 .sp
1923 .ne 2
1924 .na
1925 \fB\fBzfs clone\fR [\fB-p\fR] [\fB-o\fR \fIproperty\fR=\fIvalue\fR] ...
1926 \fIsnapshot\fR \fIfilesystem\fR|\fIvolume\fR\fR
1927 .ad
1928 .sp .6
1929 .RS 4n
1930 Creates a clone of the given snapshot. See the "Clones" section for details.
1931 The target dataset can be located anywhere in the \fBZFS\fR hierarchy, and is
1932 created as the same type as the original.
1933 .sp
1934 .ne 2
1935 .na
1936 \fB\fB-p\fR\fR
1937 .ad
1938 .sp .6
1939 .RS 4n
1940 Creates all the non-existing parent datasets. Datasets created in this manner
1941 are automatically mounted according to the \fBmountpoint\fR property inherited
1942 from their parent. If the target filesystem or volume already exists, the
1943 operation completes successfully.
1944 .RE

1946 .sp
1947 .ne 2
1948 .na
1949 \fB\fB-o\fR \fIproperty\fR=\fIvalue\fR\fR
1950 .ad
1951 .sp .6
1952 .RS 4n
1953 Sets the specified property; see \fBzfs create\fR for details.
1954 .RE

1956 .RE

1958 .sp
1959 .ne 2
1960 .na
1961 \fB\fBzfs promote\fR \fIclone-filesystem\fR\fR
1962 .ad
1963 .sp .6
1964 .RS 4n
1965 Promotes a clone file system to no longer be dependent on its "origin"
1966 snapshot. This makes it possible to destroy the file system that the clone was
1967 created from. The clone parent-child dependency relationship is reversed, so
1968 that the origin file system becomes a clone of the specified file system.
1969 .sp
1970 The snapshot that was cloned, and any snapshots previous to this snapshot, are
1971 now owned by the promoted clone. The space they use moves from the origin file
1972 system to the promoted clone, so enough space must be available to accommodate
1973 these snapshots. No new space is consumed by this operation, but the space
```

accounting is adjusted. The promoted clone must not have any conflicting
snapshot names of its own. The \fBrename\fR subcommand can be used to rename
any conflicting snapshots.
.RE

.sp
.ne 2
.na
\fB\fBzfs rename\fR [\fB-f\fR] \fIfilesystem\fR|\fIvolume\fR|\fIsnapshot\fR\fR
.ad
.br
.na
\fB\fIfilesystem\fR|\fIvolume\fR|\fIsnapshot\fR\fR
.ad
.br
.na
\fB\fBzfs rename\fR [\fB-fp\fR] \fIfilesystem\fR|\fIvolume\fR
\fIfilesystem\fR|\fIvolume\fR\fR
.ad
.sp .6
.RS 4n
Renames the given dataset. The new target can be located anywhere in the
\fBZFS\fR hierarchy, with the exception of snapshots. Snapshots can only be
renamed within the parent file system or volume. When renaming a snapshot, the
parent file system of the snapshot does not need to be specified as part of the
second argument. Renamed file systems can inherit new mount points, in which
case they are unmounted and remounted at the new mount point.
.sp
.ne 2
.na
\fB\fB-p\fR\fR
.ad
.sp .6
.RS 4n
Creates all the nonexistent parent datasets. Datasets created in this manner
are automatically mounted according to the \fBmountpoint\fR property inherited
from their parent.
.RE

.sp
.ne 2
.na
\fB\fB-f\fR\fR
.ad
.sp .6
.RS 4n
Force unmount any filesystems that need to be unmounted in the process.
.RE

.RE

.sp
.ne 2
.na
\fB\fBzfs rename\fR \fB-r\fR \fIsnapshot\fR \fIsnapshot\fR\fR
.ad
.sp .6
.RS 4n
Recursively rename the snapshots of all descendent datasets. Snapshots are the
only dataset that can be renamed recursively.
.RE

.sp
.ne 2
.na
\fB\fBzfs\fR \fBlist\fR [\fB-r\fR|\fB-d\fR \fIdepth\fR] [\fB-H\fR] [\fB-o\fR

\fIproperty\fR[,\fI\&...\fR]] [\fB-t\fR \fItype\fR[,\fI\&...\fR]] [\fB-s\fR
\fIproperty\fR] ... [\fB-S\fR \fIproperty\fR] ...
[\fIfilesystem\fR|\fIvolume\fR|\fIsnapshot\fR] ...\fR
.ad
.sp .6
.RS 4n
Lists the property information for the given datasets in tabular form. If
specified, you can list property information by the absolute pathname or the
relative pathname. By default, all file systems and volumes are displayed.
Snapshots are displayed if the \fBlistsnaps\fR property is \fBon\fR (the
default is \fBoff\fR) . The following fields are displayed,
\fBname,used,available,referenced,mountpoint\fR.
.sp
.ne 2
.na
\fB\fB-H\fR\fR
.ad
.sp .6
.RS 4n
Used for scripting mode. Do not print headers and separate fields by a single
tab instead of arbitrary white space.
.RE

.sp
.ne 2
.na
\fB\fB-r\fR\fR
.ad
.sp .6
.RS 4n
Recursively display any children of the dataset on the command line.
.RE

.sp
.ne 2
.na
\fB\fB-d\fR \fIdepth\fR\fR
.ad
.sp .6
.RS 4n
Recursively display any children of the dataset, limiting the recursion to
\fIdepth\fR. A depth of \fB1\fR will display only the dataset and its direct
children.
.RE

.sp
.ne 2
.na
\fB\fB-o\fR \fIproperty\fR\fR
.ad
.sp .6
.RS 4n
A comma-separated list of properties to display. The property must be:
.RS +4
.TP
.ie t \(bu
.el o
One of the properties described in the "Native Properties" section
.RE
.RS +4
.TP
.ie t \(bu
.el o
A user property
.RE
.RS +4

```
.TP
.ie t \(bu
.el o
The value \fBname\fR to display the dataset name
.RE
.RS +4
.TP
.ie t \(bu
.el o
The value \fBspace\fR to display space usage properties on file systems and
volumes. This is a shortcut for specifying \fB-o
name,avail,used,usedsnap,usedds,usedrefreserv,usedchild\fR \fB-t
filesystem,volume\fR syntax.
.RE
.RE

.sp
.ne 2
.na
\fB\fB-s\fR \fIproperty\fR\fR
.ad
.sp .6
.RS 4n
A property for sorting the output by column in ascending order based on the
value of the property. The property must be one of the properties described in
the "Properties" section, or the special value \fBname\fR to sort by the
dataset name. Multiple properties can be specified at one time using multiple
\fB-s\fR property options. Multiple \fB-s\fR options are evaluated from left to
right in decreasing order of importance.
.sp
The following is a list of sorting criteria:
.RS +4
.TP
.ie t \(bu
.el o
Numeric types sort in numeric order.
.RE
.RS +4
.TP
.ie t \(bu
.el o
String types sort in alphabetical order.
.RE
.RS +4
.TP
.ie t \(bu
.el o
Types inappropriate for a row sort that row to the literal bottom, regardless
of the specified ordering.
.RE
.RS +4
.TP
.ie t \(bu
.el o
If no sorting options are specified the existing behavior of \fBzfs list\fR is
preserved.
.RE
.RE

.sp
.ne 2
.na
\fB\fB-S\fR \fIproperty\fR\fR
.ad
.sp .6
.RS 4n
```



```
Same as the \fB-s\fR option, but sorts by property in descending order.
.RE

.sp
.ne 2
.na
\fB\fB-t\fR \fItype\fR\fR
.ad
.sp .6
.RS 4n
A comma-separated list of types to display, where \fItype\fR is one of
\fBfilesystem\fR, \fBsnapshot\fR , \fBvolume\fR, or \fBall\fR. For example,
specifying \fB-t snapshot\fR displays only snapshots.
.RE

.RE

.sp
.ne 2
.na
\fB\fBzfs set\fR \fIproperty\fR=\fIvalue\fR
\fIfilesystem\fR|\fIvolume\fR|\fIsnapshot\fR ...\fR
.ad
.sp .6
.RS 4n
Sets the property to the given value for each dataset. Only some properties can
be edited. See the "Properties" section for more information on what properties
can be set and acceptable values. Numeric values can be specified as exact
values, or in a human-readable form with a suffix of \fBB\fR, \fBK\fR, \fBM\fR,
\fBG\fR, \fBT\fR, \fBP\fR, \fBE\fR, \fBZ\fR (for bytes, kilobytes, megabytes,
gigabytes, terabytes, petabytes, exabytes, or zettabytes, respectively). User
properties can be set on snapshots. For more information, see the "User
Properties" section.
.RE

.sp
.ne 2
.na
\fB\fBzfs get\fR [\fB-r\fR|\fB-d\fR \fIdepth\fR] [\fB-Hp\fR] [\fB-o\fR
\fIfield\fR[,...] [\fB-t\fR \fItype\fR[,...]] [\fB-s\fR \fIsource\fR[,...] "\fIa
\fIproperty\fR[,...] \fIfilesystem\fR|\fIvolume\fR|\fIsnapshot\fR ...\fR
.ad
.sp .6
.RS 4n
Displays properties for the given datasets. If no datasets are specified, then
the command displays properties for all datasets on the system. For each
property, the following columns are displayed:
.sp
.in +2
.nf
    name      Dataset name
     property  Property name
     value     Property value
     source    Property source. Can either be local, default,
               temporary, inherited, or none (-).
.fi
.in -2
.sp

All columns are displayed by default, though this can be controlled by using
the \fB-o\fR option. This command takes a comma-separated list of properties as
described in the "Native Properties" and "User Properties" sections.
.sp
The special value \fBall\fR can be used to display all properties that apply to
the given dataset's type (filesystem, volume, or snapshot).
.sp
```

```
.ne 2
.na
\fB\fB-r\fR\fR
.ad
.sp .6
.RS 4n
Recursively display properties for any children.
.RE

.sp
.ne 2
.na
\fB\fB-d\fR \fIdepth\fR\fR
.ad
.sp .6
.RS 4n
Recursively display any children of the dataset, limiting the recursion to
\fIdepth\fR. A depth of \fB1\fR will display only the dataset and its direct
children.
.RE

.sp
.ne 2
.na
\fB\fB-H\fR\fR
.ad
.sp .6
.RS 4n
Display output in a form more easily parsed by scripts. Any headers are
omitted, and fields are explicitly separated by a single tab instead of an
arbitrary amount of space.
.RE

.sp
.ne 2
.na
\fB\fB-o\fR \fIfield\fR\fR
.ad
.sp .6
.RS 4n
A comma-separated list of columns to display. \fBname,property,value,source\fR
is the default value.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR \fIsource\fR\fR
.ad
.sp .6
.RS 4n
A comma-separated list of sources to display. Those properties coming from a
source other than those in this list are ignored. Each source must be one of
the following: \fBlocal,default,inherited,temporary,none\fR. The default value
is all sources.
.RE

.sp
.ne 2
.na
\fB\fB-p\fR\fR
.ad
.sp .6
.RS 4n
Display numbers in parseable (exact) values.
.RE
```



```
.RE

.sp
.ne 2
.na
\fB\fBzfs inherit\fR [\fB-r\fR] \fIproperty\fR
\fIfilesystem\fR|\fIvolume\fR|\fIsnapshot\fR ...\fR
.ad
.sp .6
.RS 4n
Clears the specified property, causing it to be inherited from an ancestor. If
no ancestor has the property set, then the default value is used. See the
"Properties" section for a listing of default values, and details on which
properties can be inherited.
.sp
.ne 2
.na
\fB\fB-r\fR\fR
.ad
.sp .6
.RS 4n
Recursively inherit the given property for all children.
.RE

.RE

.sp
.ne 2
.na
\fB\fBzfs upgrade\fR [\fB-v\fR]\fR
.ad
.sp .6
.RS 4n
Displays a list of file systems that are not the most recent version.
.RE

.sp
.ne 2
.na
\fB\fBzfs upgrade\fR [\fB-r\fR] [\fB-V\fR \fIversion\fR] [\fB-a\fR |
\fIfilesystem\fR]\fR
.ad
.sp .6
.RS 4n
Upgrades file systems to a new on-disk version. Once this is done, the file
systems will no longer be accessible on systems running older versions of the
software. \fBzfs send\fR streams generated from new snapshots of these file
systems cannot be accessed on systems running older versions of the software.
.sp
In general, the file system version is independent of the pool version. See
\fBzpool\fR(1M) for information on the \fBzpool upgrade\fR command.
.sp
In some cases, the file system version and the pool version are interrelated
and the pool version must be upgraded before the file system version can be
upgraded.
.sp
.ne 2
.na
\fB\fB-a\fR\fR
.ad
.sp .6
.RS 4n
Upgrade all file systems on all imported pools.
.RE
```

```
.sp
.ne 2
.na
\fB\fIfilesystem\fR\fR
.ad
.sp .6
.RS 4n
Upgrade the specified file system.
.RE

.sp
.ne 2
.na
\fB\fB-r\fR\fR
.ad
.sp .6
.RS 4n
Upgrade the specified file system and all descendent file systems
.RE

.sp
.ne 2
.na
\fB\fB-V\fR \fIversion\fR\fR
.ad
.sp .6
.RS 4n
Upgrade to the specified \fIversion\fR. If the \fB-V\fR flag is not specified,
this command upgrades to the most recent version. This option can only be used
to increase the version number, and only up to the most recent version
supported by this software.
.RE

.RE

.sp
.ne 2
.na
\fB\fBzfs userspace\fR [\fB-niHp\fR] [\fB-o\fR \fIfield\fR[,...]] [\fB-sS\fR
\fIfield\fR]... [\fB-t\fR \fItype\fR [,...]] \fIfilesystem\fR |
\fIsnapshot\fR\fR
.ad
.sp .6
.RS 4n
Displays space consumed by, and quotas on, each user in the specified
filesystem or snapshot. This corresponds to the \fBuserused@\fR\fIuser\fR and
\fBuserquota@\fR\fIuser\fR properties.
.sp
.ne 2
.na
\fB\fB-n\fR\fR
.ad
.sp .6
.RS 4n
Print numeric ID instead of user/group name.
.RE

.sp
.ne 2
.na
\fB\fB-H\fR\fR
.ad
.sp .6
.RS 4n
Do not print headers, use tab-delimited output.
.RE
```



```
.sp
.ne 2
.na
\fB\fB-p\fR\fR
.ad
.sp .6
.RS 4n
Use exact (parseable) numeric output.
.RE

.sp
.ne 2
.na
\fB\fB-o\fR \fIfield\fR[,...]\fR
.ad
.sp .6
.RS 4n
Display only the specified fields from the following set,
\fBtype,name,used,quota\fR.The default is to display all fields.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR \fIfield\fR\fR
.ad
.sp .6
.RS 4n
Sort output by this field. The \fIs\fR and \fIS\fR flags may be specified
multiple times to sort first by one field, then by another. The default is
\fB-s type\fR \fB-s name\fR.
.RE

.sp
.ne 2
.na
\fB\fB-S\fR \fIfield\fR\fR
.ad
.sp .6
.RS 4n
Sort by this field in reverse order. See \fB-s\fR.
.RE

.sp
.ne 2
.na
\fB\fB-t\fR \fItype\fR[,...]\fR
.ad
.sp .6
.RS 4n
Print only the specified types from the following set,
\fBall,posixuser,smbuser,posixgroup,smbgroup\fR.
.sp
The default is \fB-t posixuser,smbuser\fR
.sp
The default can be changed to include group types.
.RE

.sp
.ne 2
.na
\fB\fB-i\fR\fR
.ad
.sp .6
.RS 4n
```

Translate SID to POSIX ID. The POSIX ID may be ephemeral if no mapping exists. Normal POSIX interfaces (for example, \fBstat\fR(2), \fBls\fR \fB-l\fR) perform this translation, so the \fB-i\fR option allows the output from \fBzfs userspace\fR to be compared directly with those utilities. However, \fB-i\fR may lead to confusion if some files were created by an SMB user before a SMB-to-POSIX name mapping was established. In such a case, some files are owned by the SMB entity and some by the POSIX entity. However, the \fB-i\fR option will report that the POSIX entity has the total usage and quota for both.
.RE

.RE

.sp
.ne 2
.na
\fB\fBzfs groupspace\fR [\fB-niHp\fR] [\fB-o\fR \fIfield\fR[,...]] [\fB-sS\fR \fIfield\fR]... [\fB-t\fR \fItype\fR [,...]] \fIfilesystem\fR |
\fIsnapshot\fR\fR
.ad
.sp .6
.RS 4n
Displays space consumed by, and quotas on, each group in the specified filesystem or snapshot. This subcommand is identical to \fBzfs userspace\fR, except that the default types to display are \fB-t posixgroup,smbgroup\fR.
.sp
.in +2
.nf
-
.fi
.in -2
.sp

.RE

.sp
.ne 2
.na
\fB\fBzfs mount\fR\fR
.ad
.sp .6
.RS 4n
Displays all \fBZFS\fR file systems currently mounted.
.RE

.sp
.ne 2
.na
\fB\fBzfs mount\fR [\fB-vO\fR] [\fB-o\fR \fIoptions\fR] \fB-a\fR |
\fIfilesystem\fR\fR
.ad
.sp .6
.RS 4n
Mounts \fBZFS\fR file systems. Invoked automatically as part of the boot process.
.sp
.ne 2
.na
\fB\fB-o\fR \fIoptions\fR\fR
.ad
.sp .6
.RS 4n
An optional, comma-separated list of mount options to use temporarily for the duration of the mount. See the "Temporary Mount Point Properties" section for details.
.RE

.sp
.ne 2
.na
\fB\fB-O\fR\fR
.ad
.sp .6
.RS 4n
Perform an overlay mount. See \fBmount\fR(1M) for more information.
.RE

.sp
.ne 2
.na
\fB\fB-v\fR\fR
.ad
.sp .6
.RS 4n
Report mount progress.
.RE

.sp
.ne 2
.na
\fB\fB-a\fR\fR
.ad
.sp .6
.RS 4n
Mount all available \fBZFS\fR file systems. Invoked automatically as part of the boot process.
.RE

.sp
.ne 2
.na
\fB\fIfilesystem\fR\fR
.ad
.sp .6
.RS 4n
Mount the specified filesystem.
.RE

.RE

.sp
.ne 2
.na
\fB\fBzfs unmount\fR [\fB-f\fR] \fB-a\fR | \fIfilesystem\fR|\fImountpoint\fR\fR
.ad
.sp .6
.RS 4n
Unmounts currently mounted \fBZFS\fR file systems. Invoked automatically as part of the shutdown process.
.sp
.ne 2
.na
\fB\fB-f\fR\fR
.ad
.sp .6
.RS 4n
Forcefully unmount the file system, even if it is currently in use.
.RE

.sp
.ne 2
.na
\fB\fB-a\fR\fR

```
2634 .ad
2635 .sp .6
2636 .RS 4n
2637 Unmount all available \fBZFS\fR file systems. Invoked automatically as part of
2638 the boot process.
2639 .RE

2641 .sp
2642 .ne 2
2643 .na
2644 \fB\fIfilesystem\fR|\fImountpoint\fR\fR
2645 .ad
2646 .sp .6
2647 .RS 4n
2648 Unmount the specified filesystem. The command can also be given a path to a
2649 \fBZFS\fR file system mount point on the system.
2650 .RE

2652 .RE

2654 .sp
2655 .ne 2
2656 .na
2657 \fB\fBzfs share\fR \fB-a\fR | \fIfilesystem\fR\fR
2658 .ad
2659 .sp .6
2660 .RS 4n
2661 Shares available \fBZFS\fR file systems.
2662 .sp
2663 .ne 2
2664 .na
2665 \fB\fB-a\fR\fR
2666 .ad
2667 .sp .6
2668 .RS 4n
2669 Share all available \fBZFS\fR file systems. Invoked automatically as part of
2670 the boot process.
2671 .RE

2673 .sp
2674 .ne 2
2675 .na
2676 \fB\fIfilesystem\fR\fR
2677 .ad
2678 .sp .6
2679 .RS 4n
2680 Share the specified filesystem according to the \fBsharenfs\fR and
2681 \fBsharesmb\fR properties. File systems are shared when the \fBsharenfs\fR or
2682 \fBsharesmb\fR property is set.
2683 .RE

2685 .RE

2687 .sp
2688 .ne 2
2689 .na
2690 \fB\fBzfs unshare\fR \fB-a\fR | \fIfilesystem\fR|\fImountpoint\fR\fR
2691 .ad
2692 .sp .6
2693 .RS 4n
2694 Unshares currently shared \fBZFS\fR file systems. This is invoked automatically
2695 as part of the shutdown process.
2696 .sp
2697 .ne 2
2698 .na
2699 \fB\fB-a\fR\fR
```



```
2700 .ad
2701 .sp .6
2702 .RS 4n
2703 Unshare all available \fBZFS\fR file systems. Invoked automatically as part of
2704 the boot process.
2705 .RE

2707 .sp
2708 .ne 2
2709 .na
2710 \fB\fIfilesystem\fR|\fImountpoint\fR\fR
2711 .ad
2712 .sp .6
2713 .RS 4n
2714 Unshare the specified filesystem. The command can also be given a path to a
2715 \fBZFS\fR file system shared on the system.
2716 .RE

2718 .RE

2720 .sp
2721 .ne 2
2722 .na
2723 \fBzfs send\fR [\fB-DnPpRrvs\fR] [\fB-\fR[\fBiI\fR] \fIsnapshot\fR] \fIsnapshot\
2723 \fBzfs send\fR [\fB-DnPpRrv\fR] [\fB-\fR[\fBiI\fR] \fIsnapshot\fR] \fIsnapshot\f
2724 .ad
2725 .sp .6
2726 .RS 4n
2727 Creates a stream representation of the second \fIsnapshot\fR, which is written
2728 to standard output. The output can be redirected to a file or to a different
2729 system (for example, using \fBssh\fR(1). By default, a full stream is
2730 generated.
2731 .sp
2732 .ne 2
2733 .na
2734 \fB\fB-i\fR \fIsnapshot\fR\fR
2735 .ad
2736 .sp .6
2737 .RS 4n
2738 Generate an incremental stream from the first \fIsnapshot\fR to the second
2739 \fIsnapshot\fR. The incremental source (the first \fIsnapshot\fR) can be
2740 specified as the last component of the snapshot name (for example, the part
2741 after the \fB@\fR), and it is assumed to be from the same file system as the
2742 second \fIsnapshot\fR.
2743 .sp
2744 If the destination is a clone, the source may be the origin snapshot, which
2745 must be fully specified (for example, \fBpool/fs@origin\fR, not just
2746 \fB@origin\fR).
2747 .RE

2749 .sp
2750 .ne 2
2751 .na
2752 \fB\fB-I\fR \fIsnapshot\fR\fR
2753 .ad
2754 .sp .6
2755 .RS 4n
2756 Generate a stream package that sends all intermediary snapshots from the first
2757 snapshot to the second snapshot. For example, \fB-I @a fs@d\fR is similar to
2758 \fB-i @a fs@b; -i @b fs@c; -i @c fs@d\fR. The incremental source snapshot may
2759 be specified as with the \fB-i\fR option.
2760 .RE

2762 .sp
2763 .ne 2
2764 .na
```

\fB\fB-R\fR\fR
.ad
.sp .6
.RS 4n
Generate a replication stream package, which will replicate the specified filesystem, and all descendent file systems, up to the named snapshot. When received, all properties, snapshots, descendent file systems, and clones are preserved.
.sp
If the \fB-i\fR or \fB-I\fR flags are used in conjunction with the \fB-R\fR flag, an incremental replication stream is generated. The current values of properties, and current snapshot and file system names are set when the stream is received. If the \fB-F\fR flag is specified when this stream is received, snapshots and file systems that do not exist on the sending side are destroyed.
.RE

.sp
.ne 2
.na
\fB\fB-D\fR\fR
.ad
.sp .6
.RS 4n
Generate a deduplicated stream. Blocks which would have been sent multiple times in the send stream will only be sent once. The receiving system must also support this feature to recieve a deduplicated stream. This flag can be used regardless of the dataset’s \fBdedup\fR property, but performance will be much better if the filesystem uses a dedup-capable checksum (eg. \fBsha256\fR).
.RE

.sp
.ne 2
.na
\fB\fB-r\fR\fR
.ad
.sp .6
.RS 4n
Recursively send all descendant snapshots. This is similar to the \fB-R\fR flag, but information about deleted and renamed datasets is not included, and property information is only included if the \fB-p\fR flag is specified.
.RE

.sp
.ne 2
.na
\fB\fB-p\fR\fR
.ad
.sp .6
.RS 4n
Include the dataset’s properties in the stream. This flag is implicit when \fB-R\fR is specified. The receiving system must also support this feature.
.RE

.sp
.ne 2
.na
\fB\fB-n\fR\fR
.ad
.sp .6
.RS 4n
Do a dry-run ("No-op") send. Do not generate any actual send data. This is useful in conjunction with the \fB-v\fR or \fB-P\fR flags to determine what data will be sent.
.RE

.sp
.ne 2
.na
\fB\fB-s\fR\fR
.ad
.sp .6
.RS 4n
Calculate send stream size. Do not generate any actual send data. This is useful when one needs to know stream size in order to store the stream externall With \fB-v\fR specified, provides info on stream header and stream data portion sizes, in addition to the total stream size.
.RE

.sp
.ne 2
.na
\fB\fB-P\fR\fR
.ad
.sp .6
.RS 4n
Print machine-parsable verbose information about the stream package generated.
.RE

.sp
.ne 2
.na
\fB\fB-v\fR\fR
.ad
.sp .6
.RS 4n
Print verbose information about the stream package generated. This information includes a per-second report of how much data has been sent.
.RE

The format of the stream is committed. You will be able to receive your streams on future versions of \fBZFS\fR.
.RE

.sp
.ne 2
.na
\fB\fBzfs receive\fR [\fB-vnFu\fR]
\fIfilesystem\fR|\fIvolume\fR|\fIsnapshot\fR\fR
.ad
.br
.na
\fB\fBzfs receive\fR [\fB-vnFu\fR] [\fB-d\fR|\fB-e\fR] \fIfilesystem\fR\fR
.ad
.sp .6
.RS 4n
Creates a snapshot whose contents are as specified in the stream provided on standard input. If a full stream is received, then a new file system is created as well. Streams are created using the \fBzfs send\fR subcommand, which by default creates a full stream. \fBzfs recv\fR can be used as an alias for \fBzfs receive\fR.
.sp
If an incremental stream is received, then the destination file system must already exist, and its most recent snapshot must match the incremental stream’s source. For \fBzvols\fR, the destination device link is destroyed and recreated, which means the \fBzvol\fR cannot be accessed during the \fBreceive\fR operation.
.sp
When a snapshot replication package stream that is generated by using the \fBzfs send\fR \fB-R\fR command is received, any snapshots that do not exist on the sending location are destroyed by using the \fBzfs destroy\fR \fB-d\fR command.

```
2897 .sp
2898 The name of the snapshot (and file system, if a full stream is received) that
2899 this subcommand creates depends on the argument type and the use of the
2900 \fB-d\fR or \fB-e\fR options.
2901 .sp
2902 If the argument is a snapshot name, the specified \fIsnapshot\fR is created. If
2903 the argument is a file system or volume name, a snapshot with the same name as
2904 the sent snapshot is created within the specified \fIfilesystem\fR or
2905 \fIvolume\fR.  If neither of the \fB-d\fR or \fB-e\fR options are specified,
2906 the provided target snapshot name is used exactly as provided.
2907 .sp
2908 The \fB-d\fR and \fB-e\fR options cause the file system name of the target
2909 snapshot to be determined by appending a portion of the sent snapshot's name to
2910 the specified target \fIfilesystem\fR. If the \fB-d\fR option is specified, all
2911 but the first element of the sent snapshot's file system path (usually the
2912 pool name) is used and any required intermediate file systems within the
2913 specified one are created.  If the \fB-e\fR option is specified, then only the
2914 last element of the sent snapshot's file system name (i.e. the name of the
2915 source file system itself) is used as the target file system name.
2916 .sp
2917 .ne 2
2918 .na
2919 \fB\fB-d\fR\fR
2920 .ad
2921 .sp .6
2922 .RS 4n
2923 Discard the first element of the sent snapshot's file system name, using
2924 the remaining elements to determine the name of the target file system for
2925 the new snapshot as described in the paragraph above.
2926 .RE

2928 .sp
2929 .ne 2
2930 .na
2931 \fB\fB-e\fR\fR
2932 .ad
2933 .sp .6
2934 .RS 4n
2935 Discard all but the last element of the sent snapshot's file system name,
2936 using that element to determine the name of the target file system for
2937 the new snapshot as described in the paragraph above.
2938 .RE

2940 .sp
2941 .ne 2
2942 .na
2943 \fB\fB-u\fR\fR
2944 .ad
2945 .sp .6
2946 .RS 4n
2947 File system that is associated with the received stream is not mounted.
2948 .RE

2950 .sp
2951 .ne 2
2952 .na
2953 \fB\fB-v\fR\fR
2954 .ad
2955 .sp .6
2956 .RS 4n
2957 Print verbose information about the stream and the time required to perform the
2958 receive operation.
2959 .RE

2961 .sp
2962 .ne 2
```



```
2963 .na
2964 \fB\fB-n\fR\fR
2965 .ad
2966 .sp .6
2967 .RS 4n
2968 Do not actually receive the stream. This can be useful in conjunction with the
2969 \fB-v\fR option to verify the name the receive operation would use.
2970 .RE

2972 .sp
2973 .ne 2
2974 .na
2975 \fB\fB-F\fR\fR
2976 .ad
2977 .sp .6
2978 .RS 4n
2979 Force a rollback of the file system to the most recent snapshot before
2980 performing the receive operation. If receiving an incremental replication
2981 stream (for example, one generated by \fBzfs send -R -[iI]\fR), destroy
2982 snapshots and file systems that do not exist on the sending side.
2983 .RE

2985 .RE

2987 .sp
2988 .ne 2
2989 .na
2990 \fB\fBzfs allow\fR \fIfilesystem\fR | \fIvolume\fR\fR
2991 .ad
2992 .sp .6
2993 .RS 4n
2994 Displays permissions that have been delegated on the specified filesystem or
2995 volume. See the other forms of \fBzfs allow\fR for more information.
2996 .RE

2998 .sp
2999 .ne 2
3000 .na
3001 \fB\fBzfs allow\fR [\fB-ldug\fR] "\fIeveryone\fR"|\fIuser\fR|\fIgroup\fR[,...]
3002 \fIperm\fR|@\fIsetname\fR[,...] \fIfilesystem\fR| \fIvolume\fR\fR
3003 .ad
3004 .br
3005 .na
3006 \fB\fBzfs allow\fR [\fB-ld\fR] \fB-e\fR \fIperm\fR|@\fIsetname\fR[,...]
3007 \fIfilesystem\fR | \fIvolume\fR\fR
3008 .ad
3009 .sp .6
3010 .RS 4n
3011 Delegates \fBZFS\fR administration permission for the file systems to
3012 non-privileged users.
3013 .sp
3014 .ne 2
3015 .na
3016 \fB[\fB-ug\fR] "\fIeveryone\fR"|\fIuser\fR|\fIgroup\fR[,...]\fR
3017 .ad
3018 .sp .6
3019 .RS 4n
3020 Specifies to whom the permissions are delegated. Multiple entities can be
3021 specified as a comma-separated list. If neither of the \fB-ug\fR options are
3022 specified, then the argument is interpreted preferentially as the keyword
3023 "everyone", then as a user name, and lastly as a group name. To specify a user
3024 or group named "everyone", use the \fB-u\fR or \fB-g\fR options. To specify a
3025 group with the same name as a user, use the \fB-g\fR options.
3026 .RE

3028 .sp
```

```
3029 .ne 2
3030 .na
3031 \fB[\fB-e\fR] \fIperm\fR|@\fIsetname\fR[,...]\fR
3032 .ad
3033 .sp .6
3034 .RS 4n
3035 Specifies that the permissions be delegated to "everyone." Multiple permissions
3036 may be specified as a comma-separated list. Permission names are the same as
3037 \fBZFS\fR subcommand and property names. See the property list below. Property
3038 set names, which begin with an at sign (\fB@\fR) , may be specified. See the
3039 \fB-s\fR form below for details.
3040 .RE

3042 .sp
3043 .ne 2
3044 .na
3045 \fB[\fB-ld\fR] \fIfilesystem\fR|\fIvolume\fR\fR
3046 .ad
3047 .sp .6
3048 .RS 4n
3049 Specifies where the permissions are delegated. If neither of the \fB-ld\fR
3050 options are specified, or both are, then the permissions are allowed for the
3051 file system or volume, and all of its descendents. If only the \fB-l\fR option
3052 is used, then is allowed "locally" only for the specified file system. If only
3053 the \fB-d\fR option is used, then is allowed only for the descendent file
3054 systems.
3055 .RE

3057 .RE

3059 .sp
3060 .LP
3061 Permissions are generally the ability to use a \fBZFS\fR subcommand or change a
3062 \fBZFS\fR property. The following permissions are available:
3063 .sp
3064 .in +2
3065 .nf
3066 NAME            TYPE           NOTES
3067 allow           subcommand     Must also have the permission that is being
3068                                allowed
3069 clone           subcommand     Must also have the 'create' ability and 'mount'
3070                                ability in the origin file system
3071 create          subcommand     Must also have the 'mount' ability
3072 destroy         subcommand     Must also have the 'mount' ability
3073 mount           subcommand     Allows mount/umount of ZFS datasets
3074 promote         subcommand     Must also have the 'mount'
3075                                and 'promote' ability in the origin file system
3076 receive         subcommand     Must also have the 'mount' and 'create' ability
3077 rename          subcommand     Must also have the 'mount' and 'create'
3078                                ability in the new parent
3079 rollback        subcommand     Must also have the 'mount' ability
3080 send            subcommand
3081 share           subcommand     Allows sharing file systems over NFS or SMB
3082                                protocols
3083 snapshot        subcommand     Must also have the 'mount' ability
3084 groupquota      other          Allows accessing any groupquota@... property
3085 groupused       other          Allows reading any groupused@... property
3086 userprop        other          Allows changing any user property
3087 userquota       other          Allows accessing any userquota@... property
3088 userused        other          Allows reading any userused@... property

3090 aclinherit      property
3091 aclmode         property
3092 atime           property
3093 canmount        property
3094 casesensitivity  property
```



```
3095 checksum        property
3096 compression     property
3097 copies          property
3098 devices         property
3099 exec            property
3100 mountpoint      property
3101 nbmand          property
3102 normalization   property
3103 primarycache    property
3104 quota           property
3105 readonly        property
3106 recordsize      property
3107 refquota        property
3108 refreservation  property
3109 reservation     property
3110 secondarycache  property
3111 setuid          property
3112 shareiscsi      property
3113 sharenfs        property
3114 sharesmb        property
3115 snapdir         property
3116 utf8only        property
3117 version         property
3118 volblocksize    property
3119 volsize         property
3120 vscan           property
3121 xattr           property
3122 zoned           property
3123 .fi
3124 .in -2
3125 .sp

3127 .sp
3128 .ne 2
3129 .na
3130 \fB\fBzfs allow\fR \fB-c\fR \fIperm\fR|@\fIsetname\fR[,...]
3131 \fIfilesystem\fR|\fIvolume\fR\fR
3132 .ad
3133 .sp .6
3134 .RS 4n
3135 Sets "create time" permissions. These permissions are granted (locally) to the
3136 creator of any newly-created descendent file system.
3137 .RE

3139 .sp
3140 .ne 2
3141 .na
3142 \fB\fBzfs allow\fR \fB-s\fR @\fIsetname\fR \fIperm\fR|@\fIsetname\fR[,...]
3143 \fIfilesystem\fR|\fIvolume\fR\fR
3144 .ad
3145 .sp .6
3146 .RS 4n
3147 Defines or adds permissions to a permission set. The set can be used by other
3148 \fBzfs allow\fR commands for the specified file system and its descendents.
3149 Sets are evaluated dynamically, so changes to a set are immediately reflected.
3150 Permission sets follow the same naming restrictions as ZFS file systems, but
3151 the name must begin with an "at sign" (\fB@\fR), and can be no more than 64
3152 characters long.
3153 .RE

3155 .sp
3156 .ne 2
3157 .na
3158 \fB\fBzfs unallow\fR [\fB-rldug\fR]
3159 "\fIeveryone\fR"|\fIuser\fR|\fIgroup\fR[,...]
3160 [\fIperm\fR|@\fIsetname\fR[, ...]] \fIfilesystem\fR|\fIvolume\fR\fR
```

```
3161 .ad
3162 .br
3163 .na
3164 \fB\fBzfs unallow\fR [\fB-rld\fR] \fB-e\fR [\fIperm\fR|@\fIsetname\fR [,...]]
3165 \fIfilesystem\fR|\fIvolume\fR\fR
3166 .ad
3167 .br
3168 .na
3169 \fB\fBzfs unallow\fR [\fB-r\fR] \fB-c\fR [\fIperm\fR|@\fIsetname\fR[,...]]\fR
3170 .ad
3171 .br
3172 .na
3173 \fB\fIfilesystem\fR|\fIvolume\fR\fR
3174 .ad
3175 .sp .6
3176 .RS 4n
3177 Removes permissions that were granted with the \fBzfs allow\fR command. No
3178 permissions are explicitly denied, so other permissions granted are still in
3179 effect. For example, if the permission is granted by an ancestor. If no
3180 permissions are specified, then all permissions for the specified \fIuser\fR,
3181 \fIgroup\fR, or \fIeveryone\fR are removed. Specifying "everyone" (or using the
3182 \fB-e\fR option) only removes the permissions that were granted to "everyone",
3183 not all permissions for every user and group. See the \fBzfs allow\fR command
3184 for a description of the \fB-ldugec\fR options.
3185 .sp
3186 .ne 2
3187 .na
3188 \fB\fB-r\fR\fR
3189 .ad
3190 .sp .6
3191 .RS 4n
3192 Recursively remove the permissions from this file system and all descendents.
3193 .RE

3195 .RE

3197 .sp
3198 .ne 2
3199 .na
3200 \fB\fBzfs unallow\fR [\fB-r\fR] \fB-s\fR @\fIsetname\fR
3201 [\fIperm\fR|@\fIsetname\fR[,...]]\fR
3202 .ad
3203 .br
3204 .na
3205 \fB\fIfilesystem\fR|\fIvolume\fR\fR
3206 .ad
3207 .sp .6
3208 .RS 4n
3209 Removes permissions from a permission set. If no permissions are specified,
3210 then all permissions are removed, thus removing the set entirely.
3211 .RE

3213 .sp
3214 .ne 2
3215 .na
3216 \fB\fBzfs hold\fR [\fB-r\fR] \fItag\fR \fIsnapshot\fR...\fR
3217 .ad
3218 .sp .6
3219 .RS 4n
3220 Adds a single reference, named with the \fItag\fR argument, to the specified
3221 snapshot or snapshots. Each snapshot has its own tag namespace, and tags must
3222 be unique within that space.
3223 .sp
3224 If a hold exists on a snapshot, attempts to destroy that snapshot by using the
3225 \fBzfs destroy\fR command return \fBEBUSY\fR.
3226 .sp
```



```
3227 .ne 2
3228 .na
3229 \fB\fB-r\fR\fR
3230 .ad
3231 .sp .6
3232 .RS 4n
3233 Specifies that a hold with the given tag is applied recursively to the
3234 snapshots of all descendent file systems.
3235 .RE

3237 .RE

3239 .sp
3240 .ne 2
3241 .na
3242 \fB\fBzfs holds\fR [\fB-r\fR] \fIsnapshot\fR...\fR
3243 .ad
3244 .sp .6
3245 .RS 4n
3246 Lists all existing user references for the given snapshot or snapshots.
3247 .sp
3248 .ne 2
3249 .na
3250 \fB\fB-r\fR\fR
3251 .ad
3252 .sp .6
3253 .RS 4n
3254 Lists the holds that are set on the named descendent snapshots, in addition to
3255 listing the holds on the named snapshot.
3256 .RE

3258 .RE

3260 .sp
3261 .ne 2
3262 .na
3263 \fB\fBzfs release\fR [\fB-r\fR] \fItag\fR \fIsnapshot\fR...\fR
3264 .ad
3265 .sp .6
3266 .RS 4n
3267 Removes a single reference, named with the \fItag\fR argument, from the
3268 specified snapshot or snapshots. The tag must already exist for each snapshot.
3269 .sp
3270 If a hold exists on a snapshot, attempts to destroy that snapshot by using the
3271 \fBzfs destroy\fR command return \fBEBUSY\fR.
3272 .sp
3273 .ne 2
3274 .na
3275 \fB\fB-r\fR\fR
3276 .ad
3277 .sp .6
3278 .RS 4n
3279 Recursively releases a hold with the given tag on the snapshots of all
3280 descendent file systems.
3281 .RE

3283 .RE

3285 .SH EXAMPLES
3286 .LP
3287 \fBExample 1 \fRCreating a ZFS File System Hierarchy
3288 .sp
3289 .LP
3290 The following commands create a file system named \fBpool/home\fR and a file
3291 system named \fBpool/home/bob\fR. The mount point \fB/export/home\fR is set for
3292 the parent file system, and is automatically inherited by the child file
```

```
3293 system.

3295 .sp
3296 .in +2
3297 .nf
3298 # \fBzfs create pool/home\fR
3299 # \fBzfs set mountpoint=/export/home pool/home\fR
3300 # \fBzfs create pool/home/bob\fR
3301 .fi
3302 .in -2
3303 .sp

3305 .LP
3306 \fBExample 2 \fRCreating a ZFS Snapshot
3307 .sp
3308 .LP
3309 The following command creates a snapshot named \fByesterday\fR. This snapshot
3310 is mounted on demand in the \fB\&.zfs/snapshot\fR directory at the root of the
3311 \fBpool/home/bob\fR file system.

3313 .sp
3314 .in +2
3315 .nf
3316 # \fBzfs snapshot pool/home/bob@yesterday\fR
3317 .fi
3318 .in -2
3319 .sp

3321 .LP
3322 \fBExample 3 \fRCreating and Destroying Multiple Snapshots
3323 .sp
3324 .LP
3325 The following command creates snapshots named \fByesterday\fR of
3326 \fBpool/home\fR and all of its descendent file systems. Each snapshot is
3327 mounted on demand in the \fB\&.zfs/snapshot\fR directory at the root of its
3328 file system. The second command destroys the newly created snapshots.

3330 .sp
3331 .in +2
3332 .nf
3333 # \fBzfs snapshot -r pool/home@yesterday\fR
3334 # \fBzfs destroy -r pool/home@yesterday\fR
3335 .fi
3336 .in -2
3337 .sp

3339 .LP
3340 \fBExample 4 \fRDisabling and Enabling File System Compression
3341 .sp
3342 .LP
3343 The following command disables the \fBcompression\fR property for all file
3344 systems under \fBpool/home\fR. The next command explicitly enables
3345 \fBcompression\fR for \fBpool/home/anne\fR.

3347 .sp
3348 .in +2
3349 .nf
3350 # \fBzfs set compression=off pool/home\fR
3351 # \fBzfs set compression=on pool/home/anne\fR
3352 .fi
3353 .in -2
3354 .sp

3356 .LP
3357 \fBExample 5 \fRListing ZFS Datasets
3358 .sp
```



```
3359 .LP
3360 The following command lists all active file systems and volumes in the system.
3361 Snapshots are displayed if the \fBlistsnaps\fR property is \fBon\fR. The
3362 default is \fBoff\fR. See \fBzpool\fR(1M) for more information on pool
3363 properties.

3365 .sp
3366 .in +2
3367 .nf
3368 # \fBzfs list\fR
3369    NAME                     USED  AVAIL  REFER  MOUNTPOINT
3370    pool                     450K   457G    18K  /pool
3371    pool/home                315K   457G    21K  /export/home
3372    pool/home/anne            18K   457G    18K  /export/home/anne
3373    pool/home/bob            276K   457G   276K  /export/home/bob
3374 .fi
3375 .in -2
3376 .sp

3378 .LP
3379 \fBExample 6 \fRSetting a Quota on a ZFS File System
3380 .sp
3381 .LP
3382 The following command sets a quota of 50 Gbytes for \fBpool/home/bob\fR.

3384 .sp
3385 .in +2
3386 .nf
3387 # \fBzfs set quota=50G pool/home/bob\fR
3388 .fi
3389 .in -2
3390 .sp

3392 .LP
3393 \fBExample 7 \fRListing ZFS Properties
3394 .sp
3395 .LP
3396 The following command lists all properties for \fBpool/home/bob\fR.

3398 .sp
3399 .in +2
3400 .nf
3401 # \fBzfs get all pool/home/bob\fR
3402 NAME           PROPERTY              VALUE                  SOURCE
3403 pool/home/bob  type                  filesystem             -
3404 pool/home/bob  creation              Tue Jul 21 15:53 2009  -
3405 pool/home/bob  used                  21K                    -
3406 pool/home/bob  available             20.0G                  -
3407 pool/home/bob  referenced            21K                    -
3408 pool/home/bob  compressratio         1.00x                  -
3409 pool/home/bob  mounted               yes                    -
3410 pool/home/bob  quota                 20G                    local
3411 pool/home/bob  reservation           none                   default
3412 pool/home/bob  recordsize            128K                   default
3413 pool/home/bob  mountpoint            /pool/home/bob         default
3414 pool/home/bob  sharenfs              off                    default
3415 pool/home/bob  checksum              on                     default
3416 pool/home/bob  compression           on                     local
3417 pool/home/bob  atime                 on                     default
3418 pool/home/bob  devices               on                     default
3419 pool/home/bob  exec                  on                     default
3420 pool/home/bob  setuid                on                     default
3421 pool/home/bob  readonly              off                    default
3422 pool/home/bob  zoned                 off                    default
3423 pool/home/bob  snapdir               hidden                 default
3424 pool/home/bob  aclmode               discard                default
```

```
pool/home/bob  aclinherit            restricted            default
pool/home/bob  canmount              on                    default
pool/home/bob  shareiscsi            off                   default
pool/home/bob  xattr                 on                    default
pool/home/bob  copies                1                     default
pool/home/bob  version               4                     -
pool/home/bob  utf8only              off                   -
pool/home/bob  normalization         none                  -
pool/home/bob  casesensitivity       sensitive             -
pool/home/bob  vscan                 off                   default
pool/home/bob  nbmand                off                   default
pool/home/bob  sharesmb              off                   default
pool/home/bob  refquota              none                  default
pool/home/bob  refreservation        none                  default
pool/home/bob  primarycache          all                   default
pool/home/bob  secondarycache        all                   default
pool/home/bob  usedbysnapshots       0                     -
pool/home/bob  usedbydataset         21K                   -
pool/home/bob  usedbychildren        0                     -
pool/home/bob  usedbyrefreservation  0                     -
.fi
.in -2
.sp

.sp
.LP
The following command gets a single property value.

.sp
.in +2
.nf
# \fBzfs get -H -o value compression pool/home/bob\fR
on
.fi
.in -2
.sp

.sp
.LP
The following command lists all properties with local settings for
\fBpool/home/bob\fR.

.sp
.in +2
.nf
# \fBzfs get -r -s local -o name,property,value all pool/home/bob\fR
NAME           PROPERTY              VALUE
pool/home/bob  quota                 20G
pool/home/bob  compression           on
.fi
.in -2
.sp

.LP
\fBExample 8 \fRRolling Back a ZFS File System
.sp
.LP
The following command reverts the contents of \fBpool/home/anne\fR to the
snapshot named \fByesterday\fR, deleting all intermediate snapshots.

.sp
.in +2
.nf
# \fBzfs rollback -r pool/home/anne@yesterday\fR
.fi
.in -2
```



```
.sp

.LP
\fBExample 9 \fRCreating a ZFS Clone
.sp
.LP
The following command creates a writable file system whose initial contents are
the same as \fBpool/home/bob@yesterday\fR.

.sp
.in +2
.nf
# \fBzfs clone pool/home/bob@yesterday pool/clone\fR
.fi
.in -2
.sp

.LP
\fBExample 10 \fRPromoting a ZFS Clone
.sp
.LP
The following commands illustrate how to test out changes to a file system, and
then replace the original file system with the changed one, using clones, clone
promotion, and renaming:

.sp
.in +2
.nf
# \fBzfs create pool/project/production\fR
  populate /pool/project/production with data
# \fBzfs snapshot pool/project/production@today\fR
# \fBzfs clone pool/project/production@today pool/project/beta\fR
make changes to /pool/project/beta and test them
# \fBzfs promote pool/project/beta\fR
# \fBzfs rename pool/project/production pool/project/legacy\fR
# \fBzfs rename pool/project/beta pool/project/production\fR
once the legacy version is no longer needed, it can be destroyed
# \fBzfs destroy pool/project/legacy\fR
.fi
.in -2
.sp

.LP
\fBExample 11 \fRInheriting ZFS Properties
.sp
.LP
The following command causes \fBpool/home/bob\fR and \fBpool/home/anne\fR to
inherit the \fBchecksum\fR property from their parent.

.sp
.in +2
.nf
# \fBzfs inherit checksum pool/home/bob pool/home/anne\fR
.fi
.in -2
.sp

.LP
\fBExample 12 \fRRemotely Replicating ZFS Data
.sp
.LP
The following commands send a full stream and then an incremental stream to a
remote machine, restoring them into \fBpoolB/received/fs@a\fRand
\fBpoolB/received/fs@b\fR, respectively. \fBpoolB\fR must contain the file
system \fBpoolB/received\fR, and must not initially contain
\fBpoolB/received/fs\fR.
```

```
.sp
.in +2
.nf
# \fBzfs send pool/fs@a | \e\fR
   \fBssh host zfs receive poolB/received/fs@a\fR
# \fBzfs send -i a pool/fs@b | ssh host \e\fR
   \fBzfs receive poolB/received/fs\fR
.fi
.in -2
.sp

.LP
\fBExample 13 \fRUsing the \fBzfs receive\fR \fB-d\fR Option
.sp
.LP
The following command sends a full stream of \fBpoolA/fsA/fsB@snap\fR to a
remote machine, receiving it into \fBpoolB/received/fsA/fsB@snap\fR. The
\fBfsA/fsB@snap\fR portion of the received snapshot's name is determined from
the name of the sent snapshot. \fBpoolB\fR must contain the file system
\fBpoolB/received\fR. If \fBpoolB/received/fsA\fR does not exist, it is created
as an empty file system.

.sp
.in +2
.nf
# \fBzfs send poolA/fsA/fsB@snap | \e
   ssh host zfs receive -d poolB/received\fR
.fi
.in -2
.sp

.LP
\fBExample 14 \fRSetting User Properties
.sp
.LP
The following example sets the user-defined \fBcom.example:department\fR
property for a dataset.

.sp
.in +2
.nf
# \fBzfs set com.example:department=12345 tank/accounting\fR
.fi
.in -2
.sp

.LP
\fBExample 15 \fRCreating a ZFS Volume as an iSCSI Target Device
.sp
.LP
The following example shows how to create a \fBZFS\fR volume as an \fBiSCSI\fR
target.

.sp
.in +2
.nf
# \fBzfs create -V 2g pool/volumes/vol1\fR
# \fBzfs set shareiscsi=on pool/volumes/vol1\fR
# \fBiscsitadm list target\fR
Target: pool/volumes/vol1
 iSCSI Name:
 iqn.1986-03.com.sun:02:7b4b02a6-3277-eb1b-e686-a24762c52a8c
 Connections: 0
.fi
.in -2
```



```
.sp

.sp
.LP
After the \fBiSCSI\fR target is created, set up the \fBiSCSI\fR initiator. For
more information about the Solaris \fBiSCSI\fR initiator, see
\fBiscsitadm\fR(1M).
.LP
\fBExample 16 \fRPerforming a Rolling Snapshot
.sp
.LP
The following example shows how to maintain a history of snapshots with a
consistent naming scheme. To keep a week's worth of snapshots, the user
destroys the oldest snapshot, renames the remaining snapshots, and then creates
a new snapshot, as follows:

.sp
.in +2
.nf
# \fBzfs destroy -r pool/users@7daysago\fR
# \fBzfs rename -r pool/users@6daysago @7daysago\fR
# \fBzfs rename -r pool/users@5daysago @6daysago\fR
# \fBzfs rename -r pool/users@yesterday @5daysago\fR
# \fBzfs rename -r pool/users@yesterday @4daysago\fR
# \fBzfs rename -r pool/users@yesterday @3daysago\fR
# \fBzfs rename -r pool/users@yesterday @2daysago\fR
# \fBzfs rename -r pool/users@today @yesterday\fR
# \fBzfs snapshot -r pool/users@today\fR
.fi
.in -2
.sp

.LP
\fBExample 17 \fRSetting \fBsharenfs\fR Property Options on a ZFS File System
.sp
.LP
The following commands show how to set \fBsharenfs\fR property options to
enable \fBrw\fR access for a set of \fBIP\fR addresses and to enable root
access for system \fBneo\fR on the \fBtank/home\fR file system.

.sp
.in +2
.nf
# \fB# zfs set sharenfs='rw=@123.123.0.0/16,root=neo' tank/home\fR
.fi
.in -2
.sp

.sp
.LP
If you are using \fBDNS\fR for host name resolution, specify the fully
qualified hostname.

.LP
\fBExample 18 \fRDelegating ZFS Administration Permissions on a ZFS Dataset
.sp
.LP
The following example shows how to set permissions so that user \fBcindys\fR
can create, destroy, mount, and take snapshots on \fBtank/cindys\fR. The
permissions on \fBtank/cindys\fR are also displayed.

.sp
.in +2
.nf
# \fBzfs allow cindys create,destroy,mount,snapshot tank/cindys\fR
# \fBzfs allow tank/cindys\fR
```

```
3689 ----------------------------------------------------------
3690 Local+Descendent permissions on (tank/cindys)
3691          user cindys create,destroy,mount,snapshot
3692 ----------------------------------------------------------
3693 .fi
3694 .in -2
3695 .sp

3697 .sp
3698 .LP
3699 Because the \fBtank/cindys\fR mount point permission is set to 755 by default,
3700 user \fBcindys\fR will be unable to mount file systems under \fBtank/cindys\fR.
3701 Set an \fBACL\fR similar to the following syntax to provide mount point access:
3702 .sp
3703 .in +2
3704 .nf
3705 # \fBchmod A+user:cindys:add_subdirectory:allow /tank/cindys\fR
3706 .fi
3707 .in -2
3708 .sp

3710 .LP
3711 \fBExample 19 \fRDelegating Create Time Permissions on a ZFS Dataset
3712 .sp
3713 .LP
3714 The following example shows how to grant anyone in the group \fBstaff\fR to
3715 create file systems in \fBtank/users\fR. This syntax also allows staff members
3716 to destroy their own file systems, but not destroy anyone else's file system.
3717 The permissions on \fBtank/users\fR are also displayed.

3719 .sp
3720 .in +2
3721 .nf
3722 # \fB# zfs allow staff create,mount tank/users\fR
3723 # \fBzfs allow -c destroy tank/users\fR
3724 # \fBzfs allow tank/users\fR
3725 ----------------------------------------------------------
3726 Create time permissions on (tank/users)
3727          create,destroy
3728 Local+Descendent permissions on (tank/users)
3729          group staff create,mount
3730 ----------------------------------------------------------
3731 .fi
3732 .in -2
3733 .sp

3735 .LP
3736 \fBExample 20 \fRDefining and Granting a Permission Set on a ZFS Dataset
3737 .sp
3738 .LP
3739 The following example shows how to define and grant a permission set on the
3740 \fBtank/users\fR file system. The permissions on \fBtank/users\fR are also
3741 displayed.

3743 .sp
3744 .in +2
3745 .nf
3746 # \fBzfs allow -s @pset create,destroy,snapshot,mount tank/users\fR
3747 # \fBzfs allow staff @pset tank/users\fR
3748 # \fBzfs allow tank/users\fR
3749 ----------------------------------------------------------
3750 Permission sets on (tank/users)
3751        @pset create,destroy,mount,snapshot
3752 Create time permissions on (tank/users)
3753        create,destroy
3754 Local+Descendent permissions on (tank/users)
```



```
3755        group staff @pset,create,mount
3756 ----------------------------------------------------------
3757 .fi
3758 .in -2
3759 .sp

3761 .LP
3762 \fBExample 21 \fRDelegating Property Permissions on a ZFS Dataset
3763 .sp
3764 .LP
3765 The following example shows to grant the ability to set quotas and reservations
3766 on the \fBusers/home\fR file system. The permissions on \fBusers/home\fR are
3767 also displayed.

3769 .sp
3770 .in +2
3771 .nf
3772 # \fBzfs allow cindys quota,reservation users/home\fR
3773 # \fBzfs allow users/home\fR
3774 ----------------------------------------------------------
3775 Local+Descendent permissions on (users/home)
3776        user cindys quota,reservation
3777 ----------------------------------------------------------
3778 cindys% \fBzfs set quota=10G users/home/marks\fR
3779 cindys% \fBzfs get quota users/home/marks\fR
3780 NAME              PROPERTY  VALUE              SOURCE
3781 users/home/marks  quota     10G                local
3782 .fi
3783 .in -2
3784 .sp

3786 .LP
3787 \fBExample 22 \fRRemoving ZFS Delegated Permissions on a ZFS Dataset
3788 .sp
3789 .LP
3790 The following example shows how to remove the snapshot permission from the
3791 \fBstaff\fR group on the \fBtank/users\fR file system. The permissions on
3792 \fBtank/users\fR are also displayed.

3794 .sp
3795 .in +2
3796 .nf
3797 # \fBzfs unallow staff snapshot tank/users\fR
3798 # \fBzfs allow tank/users\fR
3799 ----------------------------------------------------------
3800 Permission sets on (tank/users)
3801        @pset create,destroy,mount,snapshot
3802 Create time permissions on (tank/users)
3803        create,destroy
3804 Local+Descendent permissions on (tank/users)
3805        group staff @pset,create,mount
3806 ----------------------------------------------------------
3807 .fi
3808 .in -2
3809 .sp

3811 .SH EXIT STATUS
3812 .sp
3813 .LP
3814 The following exit values are returned:
3815 .sp
3816 .ne 2
3817 .na
3818 \fB\fB0\fR\fR
3819 .ad
3820 .sp .6
```

```
.RS 4n
Successful completion.
.RE

.sp
.ne 2
.na
\fB\fB1\fR\fR
.ad
.sp .6
.RS 4n
An error occurred.
.RE

.sp
.ne 2
.na
\fB\fB2\fR\fR
.ad
.sp .6
.RS 4n
Invalid command line options were specified.
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
Interface Stability     Committed
.TE

.SH SEE ALSO
.sp
.LP
\fBssh\fR(1), \fBiscsitadm\fR(1M), \fBmount\fR(1M), \fBshare\fR(1M),
\fBsharemgr\fR(1M), \fBunshare\fR(1M), \fBzonecfg\fR(1M), \fBzpool\fR(1M),
\fBchmod\fR(2), \fBstat\fR(2), \fBwrite\fR(2), \fBfsync\fR(3C),
\fBdfstab\fR(4), \fBattributes\fR(5)
.sp
.LP
See the \fBgzip\fR(1) man page, which is not part of the SunOS man page
collection.
.sp
.LP
For information about using the \fBZFS\fR web-based management tool and other
\fBZFS\fR features, see the \fISolaris ZFS Administration Guide\fR.
```

```
**********************************************************
   45840 Tue Aug  7 18:11:52 2012
new/usr/src/uts/common/fs/zfs/dmu_send.c
*** NO COMMENTS ***
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
 */

#include <sys/dmu.h>
#include <sys/dmu_impl.h>
#include <sys/dmu_tx.h>
#include <sys/dbuf.h>
#include <sys/dnode.h>
#include <sys/zfs_context.h>
#include <sys/dmu_objset.h>
#include <sys/dmu_traverse.h>
#include <sys/dsl_dataset.h>
#include <sys/dsl_dir.h>
#include <sys/dsl_prop.h>
#include <sys/dsl_pool.h>
#include <sys/dsl_synctask.h>
#include <sys/zfs_ioctl.h>
#include <sys/zap.h>
#include <sys/zio_checksum.h>
#include <sys/zfs_znode.h>
#include <zfs_fletcher.h>
#include <sys/avl.h>
#include <sys/ddt.h>
#include <sys/zfs_onexit.h>

/* Set this tunable to TRUE to replace corrupt data with 0x2f5baddb10c */
int zfs_send_corrupt_data = B_FALSE;

static char *dmu_recv_tag = "dmu_recv_tag";

static int
dump_bytes(dmu_sendarg_t *dsp, void *buf, int len)
{
        dsl_dataset_t *ds = dsp->dsa_os->os_dsl_dataset;
        ssize_t resid; /* have to get resid to get detailed errno */
        ASSERT3U(len % 8, ==, 0);
```



```
        dsp->dsa_err = 0;
        if (!dsp->sendsize) {
                fletcher_4_incremental_native(buf, len, &dsp->dsa_zc);
                dsp->dsa_err = vn_rdwr(UIO_WRITE, dsp->dsa_vp,
                    (caddr_t)buf, len,
                    0, UIO_SYSSPACE, FAPPEND, RLIM64_INFINITY,
                    CRED(), &resid);
        }
        0, UIO_SYSSPACE, FAPPEND, RLIM64_INFINITY, CRED(), &resid);

        mutex_enter(&ds->ds_sendstream_lock);
        *dsp->dsa_off += len;
        mutex_exit(&ds->ds_sendstream_lock);

        return (dsp->dsa_err);
}
_____unchanged_portion_omitted_

#define BP_SPAN(dnp, level) \
        (((uint64_t)dnp->dn_datablkszsec) << (SPA_MINBLOCKSHIFT + \
        (level) * (dnp->dn_indblkshift - SPA_BLKPTRSHIFT)))

/* ARGSUSED */
static int
backup_cb(spa_t *spa, zilog_t *zilog, const blkptr_t *bp, arc_buf_t *pbuf,
    const zbookmark_t *zb, const dnode_phys_t *dnp, void *arg)
{
        dmu_sendarg_t *dsp = arg;
        dmu_object_type_t type = bp ? BP_GET_TYPE(bp) : DMU_OT_NONE;
        int err = 0;

        if (issig(JUSTLOOKING) && issig(FORREAL))
                return (EINTR);

        if (zb->zb_object != DMU_META_DNODE_OBJECT &&
            DMU_OBJECT_IS_SPECIAL(zb->zb_object)) {
                return (0);
        } else if (bp == NULL && zb->zb_object == DMU_META_DNODE_OBJECT) {
                uint64_t span = BP_SPAN(dnp, zb->zb_level);
                uint64_t dnobj = (zb->zb_blkid * span) >> DNODE_SHIFT;
                err = dump_freeobjects(dsp, dnobj, span >> DNODE_SHIFT);
        } else if (bp == NULL) {
                uint64_t span = BP_SPAN(dnp, zb->zb_level);
                err = dump_free(dsp, zb->zb_object, zb->zb_blkid * span, span);
        } else if (zb->zb_level > 0 || type == DMU_OT_OBJSET) {
                return (0);
        } else if (type == DMU_OT_DNODE) {
                dnode_phys_t *blk;
                int i;
                int blksz = BP_GET_LSIZE(bp);
                uint32_t aflags = ARC_WAIT;
                arc_buf_t *abuf;

                if (dsl_read(NULL, spa, bp, pbuf,
                    arc_getbuf_func, &abuf, ZIO_PRIORITY_ASYNC_READ,
                    ZIO_FLAG_CANFAIL, &aflags, zb) != 0)
                        return (EIO);

                blk = abuf->b_data;
                for (i = 0; i < blksz >> DNODE_SHIFT; i++) {
                        uint64_t dnobj = (zb->zb_blkid <<
                            (DNODE_BLOCK_SHIFT - DNODE_SHIFT)) + i;
                        err = dump_dnode(dsp, dnobj, blk+i);
                        if (err)
                                break;
```

```
348                 }
349                 (void) arc_buf_remove_ref(abuf, &abuf);
350         } else if (type == DMU_OT_SA) {
351                 uint32_t aflags = ARC_WAIT;
352                 arc_buf_t *abuf;
353                 int blksz = BP_GET_LSIZE(bp);

355                 if (arc_read_nolock(NULL, spa, bp,
356                     arc_getbuf_func, &abuf, ZIO_PRIORITY_ASYNC_READ,
357                     ZIO_FLAG_CANFAIL, &aflags, zb) != 0)
358                         return (EIO);

360                 err = dump_spill(dsp, zb->zb_object, blksz, abuf->b_data);
361                 (void) arc_buf_remove_ref(abuf, &abuf);
362         } else { /* it's a level-0 block of a regular object */
363                 uint32_t aflags = ARC_WAIT;
364                 arc_buf_t *abuf = NULL;
365                 void *buf = NULL;
361                 arc_buf_t *abuf;
366                 int blksz = BP_GET_LSIZE(bp);

368                 if (!dsp->sendsize) {
369                         if (dsl_read(NULL, spa, bp, pbuf,
370                             arc_getbuf_func, &abuf, ZIO_PRIORITY_ASYNC_READ,
371                             ZIO_FLAG_CANFAIL, &aflags, zb) != 0) {
372                                 if (zfs_send_corrupt_data) {
373                                 /* Send a block filled with 0x"zfs badd bloc" */
374                                         abuf = arc_buf_alloc(spa, blksz, &abuf,
375                                             ARC_BUFC_DATA);
376                                         uint64_t *ptr;
377                                         for (ptr = abuf->b_data;
378                                             (char *)ptr <
379                                             (char *)abuf->b_data + blksz;
373                                         (char *)ptr < (char *)abuf->b_data + blksz;
380                                             ptr++)
381                                                 *ptr = 0x2f5baddb10c;
382                                 } else {
383                                         return (EIO);
384                                 }
385                         }
386                         buf = abuf->b_data;
387                 }

389                 err = dump_data(dsp, type, zb->zb_object, zb->zb_blkid * blksz,
390                     blksz, bp, buf);
391                 if (!dsp->sendsize) {
382                     blksz, bp, abuf->b_data);
392                         (void) arc_buf_remove_ref(abuf, &abuf);
393                 }
394         }

396         ASSERT(err == 0 || err == EINTR);
397         return (err);
398 }
```

*_____unchanged_portion_omitted_*

```
440 int
441 dmu_send(objset_t *tosnap, objset_t *fromsnap, int outfd, vnode_t *vp,
442     offset_t *off, boolean_t sendsize)
431     offset_t *off)
443 {
444         dsl_dataset_t *ds = tosnap->os_dsl_dataset;
445         dsl_dataset_t *fromds = fromsnap ? fromsnap->os_dsl_dataset : NULL;
446         dmu_replay_record_t *drr;
447         dmu_sendarg_t *dsp;
```



```
448         int err;
449         uint64_t fromtxg = 0;

451         /* tosnap must be a snapshot */
452         if (ds->ds_phys->ds_next_snap_obj == 0)
453                 return (EINVAL);

455         /*
456          * fromsnap must be an earlier snapshot from the same fs as tosnap,
457          * or the origin's fs.
458          */
459         if (fromds != NULL && !is_before(ds, fromds))
460                 return (EXDEV);

462         drr = kmem_zalloc(sizeof (dmu_replay_record_t), KM_SLEEP);
463         drr->drr_type = DRR_BEGIN;
464         drr->drr_u.drr_begin.drr_magic = DMU_BACKUP_MAGIC;
465         DMU_SET_STREAM_HDRTYPE(drr->drr_u.drr_begin.drr_versioninfo,
466             DMU_SUBSTREAM);

468 #ifdef _KERNEL
469         if (dmu_objset_type(tosnap) == DMU_OST_ZFS) {
470                 uint64_t version;
471                 if (zfs_get_zplprop(tosnap, ZFS_PROP_VERSION, &version) != 0) {
472                         kmem_free(drr, sizeof (dmu_replay_record_t));
473                         return (EINVAL);
474                 }
475                 if (version == ZPL_VERSION_SA) {
476                         DMU_SET_FEATUREFLAGS(
477                             drr->drr_u.drr_begin.drr_versioninfo,
478                             DMU_BACKUP_FEATURE_SA_SPILL);
479                 }
480         }
481 #endif

483         drr->drr_u.drr_begin.drr_creation_time =
484             ds->ds_phys->ds_creation_time;
485         drr->drr_u.drr_begin.drr_type = tosnap->os_phys->os_type;
486         if (fromds != NULL && ds->ds_dir != fromds->ds_dir)
487                 drr->drr_u.drr_begin.drr_flags |= DRR_FLAG_CLONE;
488         drr->drr_u.drr_begin.drr_toguid = ds->ds_phys->ds_guid;
489         if (ds->ds_phys->ds_flags & DS_FLAG_CI_DATASET)
490                 drr->drr_u.drr_begin.drr_flags |= DRR_FLAG_CI_DATA;

492         if (fromds)
493                 drr->drr_u.drr_begin.drr_fromguid = fromds->ds_phys->ds_guid;
494         dsl_dataset_name(ds, drr->drr_u.drr_begin.drr_toname);

496         if (fromds)
497                 fromtxg = fromds->ds_phys->ds_creation_txg;

499         dsp = kmem_zalloc(sizeof (dmu_sendarg_t), KM_SLEEP);

501         dsp->dsa_drr = drr;
502         dsp->dsa_vp = vp;
503         dsp->dsa_outfd = outfd;
504         dsp->dsa_proc = curproc;
505         dsp->dsa_os = tosnap;
506         dsp->dsa_off = off;
507         dsp->dsa_toguid = ds->ds_phys->ds_guid;
508         ZIO_SET_CHECKSUM(&dsp->dsa_zc, 0, 0, 0, 0);
509         dsp->dsa_pending_op = PENDING_NONE;
510         dsp->sendsize = sendsize;

512         mutex_enter(&ds->ds_sendstream_lock);
513         list_insert_head(&ds->ds_sendstreams, dsp);
```

```
514         mutex_exit(&ds->ds_sendstream_lock);

516         if (dump_bytes(dsp, drr, sizeof (dmu_replay_record_t)) != 0) {
517                 err = dsp->dsa_err;
518                 goto out;
519         }

521         if (dsp->sendsize) {
522                 err = traverse_dataset(ds, fromtxg,
523                     TRAVERSE_PRE | TRAVERSE_PREFETCH_METADATA,
509         err = traverse_dataset(ds, fromtxg, TRAVERSE_PRE | TRAVERSE_PREFETCH,
524                     backup_cb, dsp);
525         } else {
526                 err = traverse_dataset(ds,
527                     fromtxg, TRAVERSE_PRE | TRAVERSE_PREFETCH,
528                     backup_cb, dsp);
529         }

531         if (dsp->dsa_pending_op != PENDING_NONE)
532                 if (dump_bytes(dsp, drr, sizeof (dmu_replay_record_t)) != 0)
533                         err = EINTR;

535         if (err) {
536                 if (err == EINTR && dsp->dsa_err)
537                         err = dsp->dsa_err;
538                 goto out;
539         }

541         bzero(drr, sizeof (dmu_replay_record_t));
542         drr->drr_type = DRR_END;
543         drr->drr_u.drr_end.drr_checksum = dsp->dsa_zc;
544         drr->drr_u.drr_end.drr_toguid = dsp->dsa_toguid;

546         if (dump_bytes(dsp, drr, sizeof (dmu_replay_record_t)) != 0) {
547                 err = dsp->dsa_err;
548                 goto out;
549         }

551 out:
552         mutex_enter(&ds->ds_sendstream_lock);
553         list_remove(&ds->ds_sendstreams, dsp);
554         mutex_exit(&ds->ds_sendstream_lock);

556         kmem_free(drr, sizeof (dmu_replay_record_t));
557         kmem_free(dsp, sizeof (dmu_sendarg_t));

559         return (err);
560 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   29425 Tue Aug  7 18:11:54 2012
new/usr/src/uts/common/fs/zfs/sys/dmu.h
*** NO COMMENTS ***
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
 */

/* Portions Copyright 2010 Robert Milkowski */

#ifndef _SYS_DMU_H
#define _SYS_DMU_H

/*
 * This file describes the interface that the DMU provides for its
 * consumers.
 *
 * The DMU also interacts with the SPA.  That interface is described in
 * dmu_spa.h.
 */

#include <sys/inttypes.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/cred.h>
#include <sys/time.h>
#include <sys/fs/zfs.h>

#ifdef __cplusplus
extern "C" {
#endif

struct uio;
struct xuio;
struct page;
struct vnode;
struct spa;
struct zilog;
struct zio;
struct blkptr;
```



```
struct zap_cursor;
struct dsl_dataset;
struct dsl_pool;
struct dnode;
struct drr_begin;
struct drr_end;
struct zbookmark;
struct spa;
struct nvlist;
struct arc_buf;
struct zio_prop;
struct sa_handle;

typedef struct objset objset_t;
typedef struct dmu_tx dmu_tx_t;
typedef struct dsl_dir dsl_dir_t;

typedef enum dmu_object_byteswap {
        DMU_BSWAP_UINT8,
        DMU_BSWAP_UINT16,
        DMU_BSWAP_UINT32,
        DMU_BSWAP_UINT64,
        DMU_BSWAP_ZAP,
        DMU_BSWAP_DNODE,
        DMU_BSWAP_OBJSET,
        DMU_BSWAP_ZNODE,
        DMU_BSWAP_OLDACL,
        DMU_BSWAP_ACL,
        /*
         * Allocating a new byteswap type number makes the on-disk format
         * incompatible with any other format that uses the same number.
         *
         * Data can usually be structured to work with one of the
         * DMU_BSWAP_UINT* or DMU_BSWAP_ZAP types.
         */
        DMU_BSWAP_NUMFUNCS
} dmu_object_byteswap_t;
_____unchanged_portion_omitted_

typedef void dmu_sync_cb_t(zgd_t *arg, int error);
int dmu_sync(struct zio *zio, uint64_t txg, dmu_sync_cb_t *done, zgd_t *zgd);

/*
 * Find the next hole or data block in file starting at *off
 * Return found offset in *off. Return ESRCH for end of file.
 */
int dmu_offset_next(objset_t *os, uint64_t object, boolean_t hole,
    uint64_t *off);

/*
 * Initial setup and final teardown.
 */
extern void dmu_init(void);
extern void dmu_fini(void);

typedef void (*dmu_traverse_cb_t)(objset_t *os, void *arg, struct blkptr *bp,
    uint64_t object, uint64_t offset, int len);
void dmu_traverse_objset(objset_t *os, uint64_t txg_start,
    dmu_traverse_cb_t cb, void *arg);

int dmu_send(objset_t *tosnap, objset_t *fromsnap,
    int outfd, struct vnode *vp, offset_t *off, boolean_t sendsize);
    int outfd, struct vnode *vp, offset_t *off);
int dmu_send_estimate(objset_t *tosnap, objset_t *fromsnap, uint64_t *sizep);

typedef struct dmu_recv_cookie {
```

```
790         /*
791          * This structure is opaque!
792          *
793          * If logical and real are different, we are recving the stream
794          * into the "real" temporary clone, and then switching it with
795          * the "logical" target.
796          */
797         struct dsl_dataset *drc_logical_ds;
798         struct dsl_dataset *drc_real_ds;
799         struct drr_begin *drc_drrb;
800         char *drc_tosnap;
801         char *drc_top_ds;
802         boolean_t drc_newfs;
803         boolean_t drc_force;
804         struct avl_tree *drc_guid_to_ds_map;
805 } dmu_recv_cookie_t;
```

unchanged_portion_omitted_

```
**********************************************************
    8300 Tue Aug  7 18:12:00 2012
new/usr/src/uts/common/fs/zfs/sys/dmu_impl.h
*** NO COMMENTS ***
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */

#ifndef _SYS_DMU_IMPL_H
#define _SYS_DMU_IMPL_H

#include <sys/txg_impl.h>
#include <sys/zio.h>
#include <sys/dnode.h>
#include <sys/zfs_context.h>
#include <sys/zfs_ioctl.h>

#ifdef __cplusplus
extern "C" {
#endif

/*
 * This is the locking strategy for the DMU.  Numbers in parenthesis are
 * cases that use that lock order, referenced below:
 *
 * ARC is self-contained
 * bplist is self-contained
 * refcount is self-contained
 * txg is self-contained (hopefully!)
 * zst_lock
 * zf_rwlock
 *
 * XXX try to improve evicting path?
 *
 * dp_config_rwlock > os_obj_lock > dn_struct_rwlock >
 *      dn_dbufs_mtx > hash_mutexes > db_mtx > dd_lock > leafs
 *
 * dp_config_rwlock
 *    must be held before: everything
 *    protects dd namespace changes
 *    protects property changes globally
 *    held from:
```



```
 *      dsl_dir_open/r:
 *      dsl_dir_create_sync/w:
 *      dsl_dir_sync_destroy/w:
 *      dsl_dir_rename_sync/w:
 *      dsl_prop_changed_notify/r:
 *
 * os_obj_lock
 *   must be held before:
 *      everything except dp_config_rwlock
 *   protects os_obj_next
 *   held from:
 *      dmu_object_alloc: dn_dbufs_mtx, db_mtx, hash_mutexes, dn_struct_rwlock
 *
 * dn_struct_rwlock
 *   must be held before:
 *      everything except dp_config_rwlock and os_obj_lock
 *   protects structure of dnode (eg. nlevels)
 *      db_blkptr can change when syncing out change to nlevels
 *      dn_maxblkid
 *      dn_nlevels
 *      dn_*blksz*
 *      phys nlevels, maxblkid, physical blkptr_t's (?)
 *   held from:
 *      callers of dbuf_read_impl, dbuf_hold[_impl], dbuf_prefetch
 *      dmu_object_info_from_dnode: dn_dirty_mtx (dn_datablksz)
 *      dmu_tx_count_free:
 *      dbuf_read_impl: db_mtx, dmu_zfetch()
 *      dmu_zfetch: zf_rwlock/r, zst_lock, dbuf_prefetch()
 *      dbuf_new_size: db_mtx
 *      dbuf_dirty: db_mtx
 *      dbuf_findbp: (callers, phys? - the real need)
 *      dbuf_create: dn_dbufs_mtx, hash_mutexes, db_mtx (phys?)
 *      dbuf_prefetch: dn_dirty_mtx, hash_mutexes, db_mtx, dn_dbufs_mtx
 *      dbuf_hold_impl: hash_mutexes, db_mtx, dn_dbufs_mtx, dbuf_findbp()
 *      dnode_sync/w (increase_indirection): db_mtx (phys)
 *      dnode_set_blksz/w: dn_dbufs_mtx (dn_*blksz*)
 *      dnode_new_blkid/w: (dn_maxblkid)
 *      dnode_free_range/w: dn_dirty_mtx (dn_maxblkid)
 *      dnode_next_offset: (phys)
 *
 * dn_dbufs_mtx
 *    must be held before:
 *      db_mtx, hash_mutexes
 *    protects:
 *      dn_dbufs
 *      dn_evicted
 *    held from:
 *      dmu_evict_user: db_mtx (dn_dbufs)
 *      dbuf_free_range: db_mtx (dn_dbufs)
 *      dbuf_remove_ref: db_mtx, callees:
 *              dbuf_hash_remove: hash_mutexes, db_mtx
 *      dbuf_create: hash_mutexes, db_mtx (dn_dbufs)
 *      dnode_set_blksz: (dn_dbufs)
 *
 * hash_mutexes (global)
 *   must be held before:
 *      db_mtx
 *   protects dbuf_hash_table (global) and db_hash_next
 *   held from:
 *      dbuf_find: db_mtx
 *      dbuf_hash_insert: db_mtx
 *      dbuf_hash_remove: db_mtx
 *
 * db_mtx (meta-leaf)
 *   must be held before:
 *      dn_mtx, dn_dirty_mtx, dd_lock (leaf mutexes)
```

```
 128  *   protects:
 129  *      db_state
 130  *      db_holds
 131  *      db_buf
 132  *      db_changed
 133  *      db_data_pending
 134  *      db_dirtied
 135  *      db_link
 136  *      db_dirty_node (??)
 137  *      db_dirtycnt
 138  *      db_d.*
 139  *      db.*
 140  *   held from:
 141  *      dbuf_dirty: dn_mtx, dn_dirty_mtx
 142  *      dbuf_dirty->dsl_dir_willuse_space: dd_lock
 143  *      dbuf_dirty->dbuf_new_block->dsl_dataset_block_freeable: dd_lock
 144  *      dbuf_undirty: dn_dirty_mtx (db_d)
 145  *      dbuf_write_done: dn_dirty_mtx (db_state)
 146  *      dbuf_*
 147  *      dmu_buf_update_user: none (db_d)
 148  *      dmu_evict_user: none (db_d) (maybe can eliminate)
 149  *      dbuf_find: none (db_holds)
 150  *      dbuf_hash_insert: none (db_holds)
 151  *      dmu_buf_read_array_impl: none (db_state, db_changed)
 152  *      dmu_sync: none (db_dirty_node, db_d)
 153  *      dnode_reallocate: none (db)
 154  *
 155  * dn_mtx (leaf)
 156  *   protects:
 157  *      dn_dirty_dbufs
 158  *      dn_ranges
 159  *      phys accounting
 160  *      dn_allocated_txg
 161  *      dn_free_txg
 162  *      dn_assigned_txg
 163  *      dd_assigned_tx
 164  *      dn_notxholds
 165  *      dn_dirtyctx
 166  *      dn_dirtyctx_firstset
 167  *      (dn_phys copy fields?)
 168  *      (dn_phys contents?)
 169  *   held from:
 170  *      dnode_*
 171  *      dbuf_dirty: none
 172  *      dbuf_sync: none (phys accounting)
 173  *      dbuf_undirty: none (dn_ranges, dn_dirty_dbufs)
 174  *      dbuf_write_done: none (phys accounting)
 175  *      dmu_object_info_from_dnode: none (accounting)
 176  *      dmu_tx_commit: none
 177  *      dmu_tx_hold_object_impl: none
 178  *      dmu_tx_try_assign: dn_notxholds(cv)
 179  *      dmu_tx_unassign: none
 180  *
 181  * dd_lock
 182  *    must be held before:
 183  *      ds_lock
 184  *      ancestors' dd_lock
 185  *    protects:
 186  *      dd_prop_cbs
 187  *      dd_sync_*
 188  *      dd_used_bytes
 189  *      dd_tempreserved
 190  *      dd_space_towrite
 191  *      dd_myname
 192  *      dd_phys accounting?
 193  *    held from:
```



```
 194  *      dsl_dir_*
 195  *      dsl_prop_changed_notify: none (dd_prop_cbs)
 196  *      dsl_prop_register: none (dd_prop_cbs)
 197  *      dsl_prop_unregister: none (dd_prop_cbs)
 198  *      dsl_dataset_block_freeable: none (dd_sync_*)
 199  *
 200  * os_lock (leaf)
 201  *   protects:
 202  *      os_dirty_dnodes
 203  *      os_free_dnodes
 204  *      os_dnodes
 205  *      os_downgraded_dbufs
 206  *      dn_dirtyblksz
 207  *      dn_dirty_link
 208  *   held from:
 209  *      dnode_create: none (os_dnodes)
 210  *      dnode_destroy: none (os_dnodes)
 211  *      dnode_setdirty: none (dn_dirtyblksz, os_*_dnodes)
 212  *      dnode_free: none (dn_dirtyblksz, os_*_dnodes)
 213  *
 214  * ds_lock
 215  *    protects:
 216  *      ds_objset
 217  *      ds_open_refcount
 218  *      ds_snapname
 219  *      ds_phys accounting
 220  *      ds_phys userrefs zapobj
 221  *      ds_reserved
 222  *    held from:
 223  *      dsl_dataset_*
 224  *
 225  * dr_mtx (leaf)
 226  *    protects:
 227  *      dr_children
 228  *    held from:
 229  *      dbuf_dirty
 230  *      dbuf_undirty
 231  *      dbuf_sync_indirect
 232  *      dnode_new_blkid
 233  */

 235 struct objset;
 236 struct dmu_pool;

 238 typedef struct dmu_xuio {
 239         int next;
 240         int cnt;
 241         struct arc_buf **bufs;
 242         iovec_t *iovp;
 243 } dmu_xuio_t;
_____unchanged_portion_omitted_

 282 typedef struct dmu_sendarg {
 283         list_node_t dsa_link;
 284         dmu_replay_record_t *dsa_drr;
 285         vnode_t *dsa_vp;
 286         int dsa_outfd;
 287         struct proc *dsa_proc;
 288         offset_t *dsa_off;
 289         objset_t *dsa_os;
 290         zio_cksum_t dsa_zc;
 291         uint64_t dsa_toguid;
 292         int dsa_err;
 293         dmu_pendop_t dsa_pending_op;
 294         boolean_t sendsize;
 295 } dmu_sendarg_t;
_____unchanged_portion_omitted_
```

```
**********************************************************
    9938 Tue Aug  7 18:12:03 2012
new/usr/src/uts/common/fs/zfs/sys/zfs_ioctl.h
*** NO COMMENTS ***
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright 2012, Nexenta Systems Inc. All rights reserved.
 */

#ifndef _SYS_ZFS_IOCTL_H
#define _SYS_ZFS_IOCTL_H

#include <sys/cred.h>
#include <sys/dmu.h>
#include <sys/zio.h>
#include <sys/dsl_deleg.h>
#include <sys/spa.h>
#include <sys/zfs_stat.h>

#ifdef _KERNEL
#include <sys/nvpair.h>
#endif  /* _KERNEL */

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * The structures in this file are passed between userland and the
 * kernel.  Userland may be running a 32-bit process, while the kernel
 * is 64-bit.  Therefore, these structures need to compile the same in
 * 32-bit and 64-bit.  This means not using type "long", and adding
 * explicit padding so that the 32-bit structure will not be packed more
 * tightly than the 64-bit structure (which requires 64-bit alignment).
 */

/*
 * Property values for snapdir
 */
#define ZFS_SNAPDIR_HIDDEN              0
#define ZFS_SNAPDIR_VISIBLE             1

/*
 * Field manipulation macros for the drr_versioninfo field of the
```



```
 * send stream header.
 */

/*
 * Header types for zfs send streams.
 */
typedef enum drr_headertype {
        DMU_SUBSTREAM = 0x1,
        DMU_COMPOUNDSTREAM = 0x2
} drr_headertype_t;
```

*unchanged_portion_omitted_*

```
typedef struct zfs_cmd {
        char            zc_name[MAXPATHLEN];    /* name of pool or dataset */
        uint64_t        zc_nvlist_src;          /* really (char *) */
        uint64_t        zc_nvlist_src_size;
        uint64_t        zc_nvlist_dst;          /* really (char *) */
        uint64_t        zc_nvlist_dst_size;
        boolean_t       zc_nvlist_dst_filled;   /* put an nvlist in dst? */
        int             zc_pad2;

        /*
         * The following members are for legacy ioctls which haven't been
         * converted to the new method.
         */
        uint64_t        zc_history;             /* really (char *) */
        char            zc_value[MAXPATHLEN * 2];
        char            zc_string[MAXNAMELEN];
        char            zc_top_ds[MAXPATHLEN];
        uint64_t        zc_guid;
        uint64_t        zc_nvlist_conf;         /* really (char *) */
        uint64_t        zc_nvlist_conf_size;
        uint64_t        zc_cookie;
        uint64_t        zc_objset_type;
        uint64_t        zc_perm_action;
        uint64_t        zc_history_len;
        uint64_t        zc_history_offset;
        uint64_t        zc_obj;
        uint64_t        zc_iflags;              /* internal to zfs(7fs) */
        zfs_share_t     zc_share;
        dmu_objset_stats_t zc_objset_stats;
        struct drr_begin zc_begin_record;
        zinject_record_t zc_inject_record;
        boolean_t       zc_defer_destroy;
        boolean_t       zc_temphold;
        uint64_t        zc_action_handle;
        int             zc_cleanup_fd;
        uint8_t         zc_pad[4];              /* alignment */
        uint64_t        zc_sendobj;
        uint64_t        zc_fromobj;
        uint64_t        zc_createtxg;
        zfs_stat_t      zc_stat;
        uint64_t        zc_sendcounter;
        boolean_t       zc_sendsize;
} zfs_cmd_t;
```

*unchanged_portion_omitted_*

```
**********************************************************
  143749 Tue Aug  7 18:12:11 2012
new/usr/src/uts/common/fs/zfs/zfs_ioctl.c
*** NO COMMENTS ***
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
 * Portions Copyright 2011 Martin Matuska
 * Copyright 2012 Nexenta Systems, Inc.  All rights reserved.
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 2012, Joyent, Inc. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 */

/*
 * ZFS ioctls.
 *
 * This file handles the ioctls to /dev/zfs, used for configuring ZFS storage
 * pools and filesystems, e.g. with /sbin/zfs and /sbin/zpool.
 *
 * There are two ways that we handle ioctls: the legacy way where almost
 * all of the logic is in the ioctl callback, and the new way where most
 * of the marshalling is handled in the common entry point, zfsdev_ioctl().
 *
 * Non-legacy ioctls should be registered by calling
 * zfs_ioctl_register() from zfs_ioctl_init().  The ioctl is invoked
 * from userland by lzc_ioctl().
 *
 * The registration arguments are as follows:
 *
 * const char *name
 *   The name of the ioctl.  This is used for history logging.  If the
 *   ioctl returns successfully (the callback returns 0), and allow_log
 *   is true, then a history log entry will be recorded with the input &
 *   output nvlists.  The log entry can be printed with "zpool history -i".
 *
 * zfs_ioc_t ioc
 *   The ioctl request number, which userland will pass to ioctl(2).
 *   The ioctl numbers can change from release to release, because
 *   the caller (libzfs) must be matched to the kernel.
 *
 * zfs_secpolicy_func_t *secpolicy
 *   This function will be called before the zfs_ioc_func_t, to
 *   determine if this operation is permitted.  It should return EPERM
 *   on failure, and 0 on success.  Checks include determining if the
```



```
 *   dataset is visible in this zone, and if the user has either all
 *   zfs privileges in the zone (SYS_MOUNT), or has been granted permission
 *   to do this operation on this dataset with "zfs allow".
 *
 * zfs_ioc_namecheck_t namecheck
 *   This specifies what to expect in the zfs_cmd_t:zc_name -- a pool
 *   name, a dataset name, or nothing.  If the name is not well-formed,
 *   the ioctl will fail and the callback will not be called.
 *   Therefore, the callback can assume that the name is well-formed
 *   (e.g. is null-terminated, doesn't have more than one '@' character,
 *   doesn't have invalid characters).
 *
 * zfs_ioc_poolcheck_t pool_check
 *   This specifies requirements on the pool state.  If the pool does
 *   not meet them (is suspended or is readonly), the ioctl will fail
 *   and the callback will not be called.  If any checks are specified
 *   (i.e. it is not POOL_CHECK_NONE), namecheck must not be NO_NAME.
 *   Multiple checks can be or-ed together (e.g. POOL_CHECK_SUSPENDED |
 *   POOL_CHECK_READONLY).
 *
 * boolean_t smush_outnvlist
 *   If smush_outnvlist is true, then the output is presumed to be a
 *   list of errors, and it will be "smushed" down to fit into the
 *   caller's buffer, by removing some entries and replacing them with a
 *   single "N_MORE_ERRORS" entry indicating how many were removed.  See
 *   nvlist_smush() for details.  If smush_outnvlist is false, and the
 *   outnvlist does not fit into the userland-provided buffer, then the
 *   ioctl will fail with ENOMEM.
 *
 * zfs_ioc_func_t *func
 *   The callback function that will perform the operation.
 *
 *   The callback should return 0 on success, or an error number on
 *   failure.  If the function fails, the userland ioctl will return -1,
 *   and errno will be set to the callback's return value.  The callback
 *   will be called with the following arguments:
 *
 *   const char *name
 *     The name of the pool or dataset to operate on, from
 *     zfs_cmd_t:zc_name.  The 'namecheck' argument specifies the
 *     expected type (pool, dataset, or none).
 *
 *   nvlist_t *innvl
 *     The input nvlist, deserialized from zfs_cmd_t:zc_nvlist_src.  Or
 *     NULL if no input nvlist was provided.  Changes to this nvlist are
 *     ignored.  If the input nvlist could not be deserialized, the
 *     ioctl will fail and the callback will not be called.
 *
 *   nvlist_t *outnvl
 *     The output nvlist, initially empty.  The callback can fill it in,
 *     and it will be returned to userland by serializing it into
 *     zfs_cmd_t:zc_nvlist_dst.  If it is non-empty, and serialization
 *     fails (e.g. because the caller didn't supply a large enough
 *     buffer), then the overall ioctl will fail.  See the
 *     'smush_nvlist' argument above for additional behaviors.
 *
 *     There are two typical uses of the output nvlist:
 *       - To return state, e.g. property values.  In this case,
 *         smush_outnvlist should be false.  If the buffer was not large
 *         enough, the caller will reallocate a larger buffer and try
 *         the ioctl again.
 *
 *       - To return multiple errors from an ioctl which makes on-disk
 *         changes.  In this case, smush_outnvlist should be true.
 *         Ioctls which make on-disk modifications should generally not
 *         use the outnvl if they succeed, because the caller can not
```

```
 127  *         distinguish between the operation failing, and
 128  *         deserialization failing.
 129  */

 131 #include <sys/types.h>
 132 #include <sys/param.h>
 133 #include <sys/errno.h>
 134 #include <sys/uio.h>
 135 #include <sys/buf.h>
 136 #include <sys/modctl.h>
 137 #include <sys/open.h>
 138 #include <sys/file.h>
 139 #include <sys/kmem.h>
 140 #include <sys/conf.h>
 141 #include <sys/cmn_err.h>
 142 #include <sys/stat.h>
 143 #include <sys/zfs_ioctl.h>
 144 #include <sys/zfs_vfsops.h>
 145 #include <sys/zfs_znode.h>
 146 #include <sys/zap.h>
 147 #include <sys/spa.h>
 148 #include <sys/spa_impl.h>
 149 #include <sys/vdev.h>
 150 #include <sys/priv_impl.h>
 151 #include <sys/dmu.h>
 152 #include <sys/dsl_dir.h>
 153 #include <sys/dsl_dataset.h>
 154 #include <sys/dsl_prop.h>
 155 #include <sys/dsl_deleg.h>
 156 #include <sys/dmu_objset.h>
 157 #include <sys/dmu_impl.h>
 158 #include <sys/ddi.h>
 159 #include <sys/sunddi.h>
 160 #include <sys/sunldi.h>
 161 #include <sys/policy.h>
 162 #include <sys/zone.h>
 163 #include <sys/nvpair.h>
 164 #include <sys/pathname.h>
 165 #include <sys/mount.h>
 166 #include <sys/sdt.h>
 167 #include <sys/fs/zfs.h>
 168 #include <sys/zfs_ctldir.h>
 169 #include <sys/zfs_dir.h>
 170 #include <sys/zfs_onexit.h>
 171 #include <sys/zvol.h>
 172 #include <sys/dsl_scan.h>
 173 #include <sharefs/share.h>
 174 #include <sys/dmu_objset.h>

 176 #include "zfs_namecheck.h"
 177 #include "zfs_prop.h"
 178 #include "zfs_deleg.h"
 179 #include "zfs_comutil.h"

 181 extern struct modlfs zfs_modlfs;

 183 extern void zfs_init(void);
 184 extern void zfs_fini(void);

 186 ldi_ident_t zfs_li = NULL;
 187 dev_info_t *zfs_dip;

 189 uint_t zfs_fsyncer_key;
 190 extern uint_t rrw_tsd_key;
 191 static uint_t zfs_allow_log_key;
```



```
 193 typedef int zfs_ioc_legacy_func_t(zfs_cmd_t *);
 194 typedef int zfs_ioc_func_t(const char *, nvlist_t *, nvlist_t *);
 195 typedef int zfs_secpolicy_func_t(zfs_cmd_t *, nvlist_t *, cred_t *);

 197 typedef enum {
 198         NO_NAME,
 199         POOL_NAME,
 200         DATASET_NAME
 201 } zfs_ioc_namecheck_t;
_____unchanged_portion_omitted_

3128 /*
3129  * innvl: {
3130  *     "type" -> dmu_objset_type_t (int32)
3131  *     (optional) "props" -> { prop -> value }
3132  * }
3133  *
3134  * outnvl: propname -> error code (int32)
3135  */
3136 static int
3137 zfs_ioc_create(const char *fsname, nvlist_t *innvl, nvlist_t *outnvl)
3138 {
3139         int error = 0;
3140         zfs_creat_t zct = { 0 };
3141         nvlist_t *nvprops = NULL;
3142         void (*cbfunc)(objset_t *os, void *arg, cred_t *cr, dmu_tx_t *tx);
3143         int32_t type32;
3144         dmu_objset_type_t type;
3145         boolean_t is_insensitive = B_FALSE;
3146         char parent[MAXNAMELEN];

3148         if (nvlist_lookup_int32(innvl, "type", &type32) != 0)
3149                 return (EINVAL);
3150         type = type32;
3151         (void) nvlist_lookup_nvlist(innvl, "props", &nvprops);

3153         switch (type) {
3154         case DMU_OST_ZFS:
3155                 cbfunc = zfs_create_cb;
3156                 break;

3158         case DMU_OST_ZVOL:
3159                 cbfunc = zvol_create_cb;
3160                 break;

3162         default:
3163                 cbfunc = NULL;
3164                 break;
3165         }
3166         if (strchr(fsname, '@') ||
3167             strchr(fsname, '%'))
3168                 return (EINVAL);

3170         zct.zct_props = nvprops;

3172         if (cbfunc == NULL)
3173                 return (EINVAL);

3175         if (zfs_get_parent(fsname, parent, MAXNAMELEN) == 0 &&
3175         if (zfs_get_parent(name, parent, MAXNAMELEN) == 0 &&
3176             zfs_is_wormed(parent)) {
3177                 return (EPERM);
3178         }

3180         if (type == DMU_OST_ZVOL) {
3181                 uint64_t volsize, volblocksize;
```

```
3183                 if (nvprops == NULL)
3184                         return (EINVAL);
3185                 if (nvlist_lookup_uint64(nvprops,
3186                     zfs_prop_to_name(ZFS_PROP_VOLSIZE), &volsize) != 0)
3187                         return (EINVAL);

3189                 if ((error = nvlist_lookup_uint64(nvprops,
3190                     zfs_prop_to_name(ZFS_PROP_VOLBLOCKSIZE),
3191                     &volblocksize)) != 0 && error != ENOENT)
3192                         return (EINVAL);

3194                 if (error != 0)
3195                         volblocksize = zfs_prop_default_numeric(
3196                             ZFS_PROP_VOLBLOCKSIZE);

3198                 if ((error = zvol_check_volblocksize(
3199                     volblocksize)) != 0 ||
3200                     (error = zvol_check_volsize(volsize,
3201                     volblocksize)) != 0)
3202                         return (error);
3203         } else if (type == DMU_OST_ZFS) {
3204                 int error;

3206                 /*
3207                  * We have to have normalization and
3208                  * case-folding flags correct when we do the
3209                  * file system creation, so go figure them out
3210                  * now.
3211                  */
3212                 VERIFY(nvlist_alloc(&zct.zct_zplprops,
3213                     NV_UNIQUE_NAME, KM_SLEEP) == 0);
3214                 error = zfs_fill_zplprops(fsname, nvprops,
3215                     zct.zct_zplprops, &is_insensitive);
3216                 if (error != 0) {
3217                         nvlist_free(zct.zct_zplprops);
3218                         return (error);
3219                 }
3220         }

3222         error = dmu_objset_create(fsname, type,
3223             is_insensitive ? DS_FLAG_CI_DATASET : 0, cbfunc, &zct);
3224         nvlist_free(zct.zct_zplprops);

3226         /*
3227          * It would be nice to do this atomically.
3228          */
3229         if (error == 0) {
3230                 error = zfs_set_prop_nvlist(fsname, ZPROP_SRC_LOCAL,
3231                     nvprops, outnvl);
3232                 if (error != 0)
3233                         (void) dmu_objset_destroy(fsname, B_FALSE);
3234         }
3235         return (error);
3236 }
```

*unchanged_portion_omitted_*

```
4101 /*
4102  * inputs:
4103  * zc_name      name of snapshot to send
4104  * zc_cookie    file descriptor to send stream to
4105  * zc_obj       fromorigin flag (mutually exclusive with zc_fromobj)
4106  * zc_sendobj   objsetid of snapshot to send
4107  * zc_fromobj   objsetid of incremental fromsnap (may be zero)
4108  * zc_guid      if set, estimate size of stream only.  zc_cookie is ignored.
4109  *              output size in zc_objset_type.
```



```
4110  *
4111  * outputs: none
4112  */
4113 static int
4114 zfs_ioc_send(zfs_cmd_t *zc)
4115 {
4116         objset_t *fromsnap = NULL;
4117         objset_t *tosnap;
4118         int error;
4119         offset_t off;
4120         dsl_dataset_t *ds;
4121         dsl_dataset_t *dsfrom = NULL;
4122         spa_t *spa;
4123         dsl_pool_t *dp;
4124         boolean_t estimate = (zc->zc_guid != 0);

4126         error = spa_open(zc->zc_name, &spa, FTAG);
4127         if (error)
4128                 return (error);

4130         dp = spa_get_dsl(spa);
4131         rw_enter(&dp->dp_config_rwlock, RW_READER);
4132         error = dsl_dataset_hold_obj(dp, zc->zc_sendobj, FTAG, &ds);
4133         rw_exit(&dp->dp_config_rwlock);
4134         spa_close(spa, FTAG);
4135         if (error)
4136                 return (error);

4138         error = dmu_objset_from_ds(ds, &tosnap);
4139         if (error) {
4140                 dsl_dataset_rele(ds, FTAG);
4141                 return (error);
4142         }

4144         if (zc->zc_fromobj != 0) {
4145                 rw_enter(&dp->dp_config_rwlock, RW_READER);
4146                 error = dsl_dataset_hold_obj(dp, zc->zc_fromobj, FTAG, &dsfrom);
4147                 rw_exit(&dp->dp_config_rwlock);
4148                 if (error) {
4149                         dsl_dataset_rele(ds, FTAG);
4150                         return (error);
4151                 }
4152                 error = dmu_objset_from_ds(dsfrom, &fromsnap);
4153                 if (error) {
4154                         dsl_dataset_rele(dsfrom, FTAG);
4155                         dsl_dataset_rele(ds, FTAG);
4156                         return (error);
4157                 }
4158         }

4160         if (zc->zc_obj) {
4161                 dsl_pool_t *dp = ds->ds_dir->dd_pool;

4163                 if (fromsnap != NULL) {
4164                         dsl_dataset_rele(dsfrom, FTAG);
4165                         dsl_dataset_rele(ds, FTAG);
4166                         return (EINVAL);
4167                 }

4169                 if (dsl_dir_is_clone(ds->ds_dir)) {
4170                         rw_enter(&dp->dp_config_rwlock, RW_READER);
4171                         error = dsl_dataset_hold_obj(dp,
4172                             ds->ds_dir->dd_phys->dd_origin_obj, FTAG, &dsfrom);
4173                         rw_exit(&dp->dp_config_rwlock);
4174                         if (error) {
4175                                 dsl_dataset_rele(ds, FTAG);
```

```
4176                                    return (error);
4177                            }
4178                            error = dmu_objset_from_ds(dsfrom, &fromsnap);
4179                            if (error) {
4180                                    dsl_dataset_rele(dsfrom, FTAG);
4181                                    dsl_dataset_rele(ds, FTAG);
4182                                    return (error);
4183                            }
4184                    }
4185            }

4187            if (estimate) {
4188                    error = dmu_send_estimate(tosnap, fromsnap,
4189                        &zc->zc_objset_type);
4190            } else {
4191                    offset_t off_starting;
4192                    file_t *fp = getf(zc->zc_cookie);
4193                    if (fp == NULL) {
4194                            dsl_dataset_rele(ds, FTAG);
4195                            if (dsfrom)
4196                                    dsl_dataset_rele(dsfrom, FTAG);
4197                            return (EBADF);
4198                    }

4200                    off = fp->f_offset;
4201                    off_starting = off;
4202                    error = dmu_send(tosnap, fromsnap, zc->zc_cookie, fp->f_vnode,
4203                        &off, zc->zc_sendsize);
4200                    error = dmu_send(tosnap, fromsnap,
4201                        zc->zc_cookie, fp->f_vnode, &off);

4205                    zc->zc_sendcounter = off - off_starting;
4206                    if (VOP_SEEK(fp->f_vnode, fp->f_offset, &off, NULL) == 0)
4207                            fp->f_offset = off;
4208                    releasef(zc->zc_cookie);
4209            }
4210            if (dsfrom)
4211                    dsl_dataset_rele(dsfrom, FTAG);
4212            dsl_dataset_rele(ds, FTAG);
4213            return (error);
4214 }
```

*unchanged_portion_omitted_*

```
5133 /*
5134  * innvl: {
5135  *     "fd" -> file descriptor to write stream to (int32)
5136  *     (optional) "fromsnap" -> full snap name to send an incremental from
5137  * }
5138  *
5139  * outnvl is unused
5140  */
5141 /* ARGSUSED */
5142 static int
5143 zfs_ioc_send_new(const char *snapname, nvlist_t *innvl, nvlist_t *outnvl)
5144 {
5145            objset_t *fromsnap = NULL;
5146            objset_t *tosnap;
5147            int error;
5148            offset_t off;
5149            char *fromname;
5150            int fd;

5152            error = nvlist_lookup_int32(innvl, "fd", &fd);
5153            if (error != 0)
5154                    return (EINVAL);
```



```
5156            error = dmu_objset_hold(snapname, FTAG, &tosnap);
5157            if (error)
5158                    return (error);

5160            error = nvlist_lookup_string(innvl, "fromsnap", &fromname);
5161            if (error == 0) {
5162                    error = dmu_objset_hold(fromname, FTAG, &fromsnap);
5163                    if (error) {
5164                            dmu_objset_rele(tosnap, FTAG);
5165                            return (error);
5166                    }
5167            }

5169            file_t *fp = getf(fd);
5170            if (fp == NULL) {
5171                    dmu_objset_rele(tosnap, FTAG);
5172                    if (fromsnap != NULL)
5173                            dmu_objset_rele(fromsnap, FTAG);
5174                    return (EBADF);
5175            }

5177            off = fp->f_offset;
5178            error = dmu_send(tosnap, fromsnap, fd, fp->f_vnode, &off, B_FALSE);
5175            error = dmu_send(tosnap, fromsnap, fd, fp->f_vnode, &off);

5180            if (VOP_SEEK(fp->f_vnode, fp->f_offset, &off, NULL) == 0)
5181                    fp->f_offset = off;
5182            releasef(fd);
5183            if (fromsnap != NULL)
5184                    dmu_objset_rele(fromsnap, FTAG);
5185            dmu_objset_rele(tosnap, FTAG);
5186            return (error);
5187 }
```

*unchanged_portion_omitted_*