

```

*****
34506 Sat Nov 23 11:10:38 2013
new/usr/src/cmd/beam/beam.c
4220 beam mount of old BE with zones fails "Read only filesystem"
4235 beam(1M) mount options undocumented in man page
*****
_____unchanged_portion_omitted_____

1164 static int
1165 be_do_mount(int argc, char **argv)
1166 {
1167     nvlist_t      *be_attrs;
1168     boolean_t     shared_fs = B_FALSE;
1169     int           err = 1;
1170     int           c;
1171     int           mount_flags = 0;
1172     char          *obe_name;
1173     char          *mountpoint;
1174     char          *tmp_mp = NULL;

1176     while ((c = getopt(argc, argv, "s:v")) != -1) {
1177         switch (c) {
1178             case 's':
1179                 shared_fs = B_TRUE;

1181                 mount_flags |= BE_MOUNT_FLAG_SHARED_FS;

1183                 if (strcmp(optarg, "rw") == 0) {
1184                     mount_flags |= BE_MOUNT_FLAG_SHARED_RW;
1185                 } else if (strcmp(optarg, "ro") != 0) {
1186                     (void) fprintf(stderr, _("The -s flag "
1187                         "requires an argument [ rw | ro ]\n"));
1188                     usage();
1189                     return (1);
1190                 }

1192                 break;
1193             case 'v':
1194                 libbe_print_errors(B_TRUE);
1195                 break;
1196             default:
1197                 usage();
1198                 return (1);
1199         }
1200     }

1202     argc -= optind;
1203     argv += optind;

1205     if (argc < 1 || argc > 2) {
1206         usage();
1207         return (1);
1208     }

1210     obe_name = argv[0];

1212     if (argc == 2) {
1213         mountpoint = argv[1];
1214         if (mountpoint[0] != '/') {
1215             (void) fprintf(stderr, _("Invalid mount point %s. "
1216                 "Mount point must start with a /\n"), mountpoint);
1217             return (1);
1218         }
1219     } else {
1220         const char *tmpdir = getenv("TMPDIR");
1221         const char *tmpname = "tmp.XXXXXX";

```

```

1222         int sz;

1224         if (tmpdir == NULL)
1225             tmpdir = "/tmp";

1227         sz = asprintf(&tmp_mp, "%s/%s", tmpdir, tmpname);
1228         if (sz < 0) {
1229             (void) fprintf(stderr, _("internal error: "
1230                 "out of memory\n"));
1231             return (1);
1232         }

1234         mountpoint = mkdtemp(tmp_mp);
1235     }

1237     if (be_nvl_alloc(&be_attrs) != 0)
1238         return (1);

1240     if (be_nvl_add_string(be_attrs, BE_ATTR_ORIG_BE_NAME, obe_name) != 0)
1241         goto out;

1243     if (be_nvl_add_string(be_attrs, BE_ATTR_MOUNTPOINT, mountpoint) != 0)
1244         goto out;

1246     if (shared_fs && be_nvl_add_uint16(be_attrs, BE_ATTR_MOUNT_FLAGS,
1247         mount_flags) != 0)
1248         goto out;

1250     err = be_mount(be_attrs);

1252     switch (err) {
1253     case BE_SUCCESS:
1254         (void) printf(_("Mounted successfully on: '%s'\n"), mountpoint);
1255         break;
1256     case BE_ERR_BE_NOENT:
1257         (void) fprintf(stderr, _("%s does not exist or appear "
1258             "to be a valid BE.\nPlease check that the name of "
1259             "the BE provided is correct.\n"), obe_name);
1260         break;
1261     case BE_ERR_MOUNTED:
1262         (void) fprintf(stderr, _("%s is already mounted.\n"
1263             "Please unmount the BE before mounting it again.\n"),
1264             obe_name);
1265         break;
1266     case BE_ERR_PERM:
1267     case BE_ERR_ACCESS:
1268         (void) fprintf(stderr, _("Unable to mount %s.\n"), obe_name);
1269         (void) fprintf(stderr, _("You have insufficient privileges to "
1270             "execute this command.\n"));
1271         break;
1272     case BE_ERR_NO_MOUNTED_ZONE:
1273         (void) fprintf(stderr, _("Mounted on '%s'.\nUnable to mount "
1274             "'one of %s's zone BE's.\n"), mountpoint, obe_name);
1275         break;
1276     default:
1277         (void) fprintf(stderr, _("Unable to mount %s.\n"), obe_name);
1278         (void) fprintf(stderr, "%s\n", be_err_to_str(err));
1279     }

1281 out:
1282     if (tmp_mp != NULL)
1283         free(tmp_mp);
1284     nvlist_free(be_attrs);
1285     return (err);
1286 }
_____unchanged_portion_omitted_____

```

```

*****
77530 Sat Nov 23 11:10:39 2013
new/usr/src/lib/libbe/common/be_mount.c
4220 beadm mount of old BE with zones fails "Read only filesystem"
4235 beadm(1M) mount options undocumented in man page
*****
unchanged_portion_omitted

227 /* ***** */
228 /* Semi-Private Functions */
229 /* ***** */

231 /*
232 * Function: _be_mount
233 * Description: Mounts a BE. If the altroot is not provided, this function
234 * will generate a temporary mountpoint to mount the BE at. It
235 * will return this temporary mountpoint to the caller via the
236 * altroot reference pointer passed in. This returned value is
237 * allocated on heap storage and is the responsibility of the
238 * caller to free.
239 * Parameters:
240 * be_name - pointer to name of BE to mount.
241 * altroot - reference pointer to altroot of where to mount BE.
242 * flags - flag indicating special handling for mounting the BE
243 * Return:
244 * BE_SUCCESS - Success
245 * be_errno_t - Failure
246 * Scope:
247 * Semi-private (library wide use only)
248 */
249 int
250 _be_mount(char *be_name, char **altroot, int flags)
251 {
252     be_transaction_data_t bt = { 0 };
253     be_mount_data_t md = { 0 };
254     zfs_handle_t *zhp;
255     char obe_root_ds[MAXPATHLEN];
256     char *mp = NULL;
257     char *tmp_altroot = NULL;
258     int ret = BE_SUCCESS, err = 0;
259     uuid_t uu = { 0 };
260     boolean_t gen_tmp_altroot = B_FALSE;

262     if (be_name == NULL || altroot == NULL)
263         return (BE_ERR_INVALID);

265     /* Set be_name as obe_name in bt structure */
266     bt.obe_name = be_name;

268     /* Find which zpool obe_name lives in */
269     if ((err = zpool_iter(g_zfs, be_find_zpool_callback, &bt)) == 0) {
270         be_print_err(gettext("be_mount: failed to "
271             "find zpool for BE (%s)\n"), bt.obe_name);
272         return (BE_ERR_BE_NOENT);
273     } else if (err < 0) {
274         be_print_err(gettext("be_mount: zpool_iter failed: %s\n"),
275             libzfs_error_description(g_zfs));
276         return (zfs_err_to_be_err(g_zfs));
277     }

279     /* Generate string for obe_name's root dataset */
280     be_make_root_ds(bt.obe_zpool, bt.obe_name, obe_root_ds,
281         sizeof (obe_root_ds));
282     bt.obe_root_ds = obe_root_ds;

284     /* Get handle to BE's root dataset */

```

```

285     if ((zhp = zfs_open(g_zfs, bt.obe_root_ds, ZFS_TYPE_FILESYSTEM)) ==
286         NULL) {
287         be_print_err(gettext("be_mount: failed to "
288             "open BE root dataset (%s): %s\n"), bt.obe_root_ds,
289             libzfs_error_description(g_zfs));
290         return (zfs_err_to_be_err(g_zfs));
291     }

293     /* Make sure BE's root dataset isn't already mounted somewhere */
294     if (zfs_is_mounted(zhp, &mp)) {
295         ZFS_CLOSE(zhp);
296         be_print_err(gettext("be_mount: %s is already mounted "
297             "at %s\n"), bt.obe_name, mp != NULL ? mp : "");
298         free(mp);
299         return (BE_ERR_MOUNTED);
300     }

302     /*
303     * Fix this BE's mountpoint if its root dataset isn't set to
304     * either 'legacy' or '/'.
305     */
306     if ((ret = fix_mountpoint(zhp)) != BE_SUCCESS) {
307         be_print_err(gettext("be_mount: mountpoint check "
308             "failed for %s\n"), bt.obe_root_ds);
309         ZFS_CLOSE(zhp);
310         return (ret);
311     }

313     /*
314     * If altroot not provided, create a temporary alternate root
315     * to mount on
316     */
317     if (*altroot == NULL) {
318         if ((ret = be_make_tmp_mountpoint(&tmp_altroot))
319             != BE_SUCCESS) {
320             be_print_err(gettext("be_mount: failed to "
321                 "make temporary mountpoint\n"));
322             ZFS_CLOSE(zhp);
323             return (ret);
324         }
325         gen_tmp_altroot = B_TRUE;
326     } else {
327         tmp_altroot = *altroot;
328     }

330     md.altroot = tmp_altroot;
331     md.shared_fs = flags & BE_MOUNT_FLAG_SHARED_FS;
332     md.shared_rw = flags & BE_MOUNT_FLAG_SHARED_RW;

334     /* Mount the BE's root file system */
335     if (getzoneid() == GLOBAL_ZONEID) {
336         if ((ret = be_mount_root(zhp, tmp_altroot)) != BE_SUCCESS) {
337             be_print_err(gettext("be_mount: failed to "
338                 "mount BE root file system\n"));
339             if (gen_tmp_altroot)
340                 free(tmp_altroot);
341             ZFS_CLOSE(zhp);
342             return (ret);
343         }
344     } else {
345         /* Legacy mount the zone root dataset */
346         if ((ret = be_mount_zone_root(zhp, &md)) != BE_SUCCESS) {
347             be_print_err(gettext("be_mount: failed to "
348                 "mount BE zone root file system\n"));
349             free(md.altroot);
350             ZFS_CLOSE(zhp);

```

```

351         return (ret);
352     }
353 }

355 /* Iterate through BE's children filesystems */
356 if ((err = zfs_iter_filesystems(zhp, be_mount_callback,
357     tmp_altroot)) != 0) {
358     be_print_err(gettext("be_mount: failed to "
359         "mount BE (%s) on %s\n"), bt.obe_name, tmp_altroot);
360     if (gen_tmp_altroot)
361         free(tmp_altroot);
362     ZFS_CLOSE(zhp);
363     return (err);
364 }

366 /*
367  * Mount shared file systems if mount flag says so.
368  */
369 if (md.shared_fs) {
370     /*
371      * Mount all ZFS file systems not under the BE's root dataset
372      */
373     (void) zpool_iter(g_zfs, zpool_shared_fs_callback, &md);

375     /* TODO: Mount all non-ZFS file systems - Not supported yet */
376 }

378 /*
379  * If we're in the global zone and the global zone has a valid uuid,
380  * mount all supported non-global zones.
381  */
382 if (getzoneid() == GLOBAL_ZONEID &&
383     !(flags & BE_MOUNT_FLAG_NO_ZONES) &&
384     be_get_uuid(bt.obe_root_ds, &uu) == BE_SUCCESS) {
385     if (be_mount_zones(zhp, &md) != BE_SUCCESS) {
386         ret = BE_ERR_NO_MOUNTED_ZONE;
387         if ((ret = be_mount_zones(zhp, &md)) != BE_SUCCESS) {
388             (void) _be_unmount(bt.obe_name, 0);
389             if (gen_tmp_altroot)
390                 free(tmp_altroot);
391             ZFS_CLOSE(zhp);
392             return (ret);
393         }
394     }

395     ZFS_CLOSE(zhp);

397     /*
398      * If a NULL altroot was passed in, pass the generated altroot
399      * back to the caller in altroot.
400      */
401     if (gen_tmp_altroot) {
402         if (ret == BE_SUCCESS || ret == BE_ERR_NO_MOUNTED_ZONE)
403             *altroot = tmp_altroot;
404         else
405             free(tmp_altroot);
406     }

407     return (ret);
408     return (BE_SUCCESS);
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

```

1714 * Description: This function loopback mounts a zonepath into the altroot
1715 * area of the BE being mounted.
1716 * area of the BE being mounted. Since these are shared file
1717 * systems, they are expected to be already mounted for the
1718 * current BE, and this function just loopback mounts them into
1719 * the BE mountpoint.
1720 * Parameters:
1721 *     zonepath - pointer to zone path in the current BE
1722 *     md - be_mount_data_t pointer
1723 * Returns:
1724 *     BE_SUCCESS - Success
1725 *     be_errno_t - Failure
1726 * Scope:
1727 *     Private
1728 */
1729 static int
1730 loopback_mount_zonepath(const char *zonepath, be_mount_data_t *md)
1731 {
1732     FILE *fp = (FILE *)NULL;
1733     struct stat st;
1734     char *p;
1735     char *pl;
1736     char *parent_dir;
1737     struct extmnttab extmnttab;
1738     dev_t dev = NODEV;
1739     char *parentmnt;
1740     char alt_parentmnt[MAXPATHLEN];
1741     struct mnttab mntref;
1742     char altzonepath[MAXPATHLEN];
1743     char optstr[MAX_MNTOPT_STR];
1744     int mflag = MS_OPTIONSTR;
1745     int ret;
1746     int err;

1747     fp = fopen(MNTTAB, "r");
1748     if (fp == NULL) {
1749         err = errno;
1750         be_print_err(gettext("loopback_mount_zonepath: "
1751             "failed to open /etc/mnttab\n"));
1752         return (errno_to_be_err(err));
1753     }

1754     /*
1755      * before attempting the loopback mount of zonepath under altroot,
1756      * we need to make sure that all intermediate file systems in the
1757      * zone path are also mounted under altroot
1758      */

1759     /* get the parent directory for zonepath */
1760     p = strrchr(zonepath, '/');
1761     if (p != NULL && p != zonepath) {
1762         if ((parent_dir = (char *)calloc(sizeof(char),
1763             p - zonepath + 1)) == NULL) {
1764             ret = BE_ERR_NOMEM;
1765             goto done;
1766         }
1767         (void) strcpy(parent_dir, zonepath, p - zonepath + 1);
1768         if (stat(parent_dir, &st) < 0) {
1769             ret = errno_to_be_err(errno);
1770             be_print_err(gettext("loopback_mount_zonepath: "
1771                 "failed to stat %s"),
1772                 parent_dir);
1773             free(parent_dir);
1774             goto done;
1775         }
1776     }
1777     free(parent_dir);
1778 }

```

```

1777     /*
1778     * After the above stat call, st.st_dev contains ID of the
1779     * device over which parent dir resides.
1780     * Now, search mnttab and find mount point of parent dir device.
1781     */
1782
1783     resetmnttab(fp);
1784     while (getextmntent(fp, &extmtab, sizeof (extmtab)) != 0) {
1785         dev = makedev(extmtab.mnt_major, extmtab.mnt_minor);
1786         if (st.st_dev == dev && strcmp(extmtab.mnt_fstype,
1787             MNTTYPE_ZFS) == 0) {
1788             pl = strchr(extmtab.mnt_special, '/');
1789             if (pl == NULL || strcmp(pl + 1,
1790                 BE_CONTAINER_DS_NAME, 4) != 0 ||
1791                 (*(pl + 5) != '/' && *(pl + 5) != '\0')) {
1792                 /*
1793                 * if parent dir is in a shared file
1794                 * system, check whether it is already
1795                 * loopback mounted under altroot or
1796                 * not. It would have been mounted
1797                 * already under altroot if it is in
1798                 * a non-shared filesystem.
1799                 */
1800                 parentmnt = strdup(extmtab.mnt_mountp);
1801                 (void) snprintf(alt_parentmnt,
1802                     sizeof (alt_parentmnt), "%s%s",
1803                     md->altroot, parentmnt);
1804                 mntref.mnt_mountp = alt_parentmnt;
1805                 mntref.mnt_special = parentmnt;
1806                 mntref.mnt_fstype = MNTTYPE_LOFS;
1807                 mntref.mnt_mntopts = NULL;
1808                 mntref.mnt_time = NULL;
1809                 resetmnttab(fp);
1810                 if (getmntany(fp, (struct mnttab *)
1811                     &extmtab, &mntref) != 0) {
1812                     ret = loopback_mount_zonepath(
1813                         parentmnt, md);
1814                     if (ret != BE_SUCCESS) {
1815                         free(parentmnt);
1816                         goto done;
1817                     }
1818                 }
1819                 free(parentmnt);
1820             }
1821             break;
1822         }
1823     }
1824 }
1825
1827 if (!md->shared_rw) {
1828     mflag |= MS_RDONLY;
1829 }
1830
1831 (void) snprintf(altzonepath, sizeof (altzonepath), "%s%s",
1832     md->altroot, zonepath);
1833
1834 /* Add the "nosub" option to the mount options string */
1835 (void) strcpy(optstr, MNTOPT_NOSUB, sizeof (optstr));
1836
1837 /* Loopback mount this dataset at the altroot */
1838 if (mount(zonepath, altzonepath, mflag, MNTTYPE_LOFS,
1839     NULL, 0, optstr, sizeof (optstr)) != 0) {
1840     err = errno;
1841     be_print_err(gettext("loopback_mount_zonepath: ")

```

```

1842         "failed to loopback mount %s at %s: %s\n"),
1843         zonepath, altzonepath, strerror(err));
1844         ret = BE_ERR_MOUNT;
1845         goto done;
1846     }
1847     ret = BE_SUCCESS;
1848
1849 done :
1850     (void) fclose(fp);
1851     return (ret);
1852 }

```

unchanged portion omitted

```

*****
15091 Sat Nov 23 11:10:39 2013
new/usr/src/man/man1m/beam.1m
4220 beam mount of old BE with zones fails "Read only filesystem"
4235 beam(1M) mount options undocumented in man page
*****
1  \" te
2  .\" Copyright 2013 Nexenta Systems, Inc. All rights reserved.
3  .TH BEADM 1M "Nov 11, 2013"
4  .TH BEADM 1M "Jul 25, 2013"
5  .SH NAME
6  beam \- utility for managing zfs boot environments
7  .SH SYNOPSIS
8  .LP
9  \fbbeam \fBcreate \fR [\fB-a \fR] [\fB-d \fR] [\fB-idescription \fR]
10 [\fB-e \fR] [\fB-inon-active-be-name \fR] | [\fB-i-be-name@snapshot \fR]
11 [\fB-o \fR] [\fB-i-property=value \fR] ... [\fB-p \fR] [\fB-izpool \fR]
12 [\fB-v \fR] [\fB-i-be-name \fR]
13 .fi

15 .LP
16 .nf
17 \fbbeam \fBcreate \fR [\fB-v \fR] [\fB-i-be-name@snapshot \fR]
18 .fi

20 .LP
21 .nf
22 \fbbeam \fBdestroy \fR [\fB-fsv \fR] [\fB-i-be-name \fR] | [\fB-i-be-name@snapshot \fR]
23 .fi

25 .LP
26 .nf
27 \fbbeam \fBlist \fR [\fB-a \fR] | [\fB-ds \fR] [\fB-H \fR] [\fB-v \fR] [\fB-i-be-name \fR]
28 .fi

30 .LP
31 .nf
32 \fbbeam \fBmount \fR [\fB-s \fR] [\fB-br \fR] | [\fB-brw \fR] [\fB-v \fR] [\fB-i-be-name \fR] [\fB-i]
32 \fbbeam \fBmount \fR [\fB-v \fR] [\fB-i-be-name \fR] [\fB-i-mountpoint \fR]
33 .fi

35 .LP
36 .nf
37 \fbbeam \fBunmount \fR [\fB-fv \fR] [\fB-i-be-name \fR] | [\fB-i-mountpoint \fR]
38 .fi

40 .LP
41 .nf
42 \fbbeam \fBrename \fR [\fB-v \fR] [\fB-i-be-name \fR] [\fB-i-new-be-name \fR]
43 .fi

45 .LP
46 .nf
47 \fbbeam \fBactivate \fR [\fB-v \fR] [\fB-i-be-name \fR]
48 .fi

50 .LP
51 .nf
52 \fbbeam \fBrollback \fR [\fB-v \fR] [\fB-i-be-name \fR] [\fB-i-snapshot \fR]
53 .fi

55 .LP
56 .nf
57 \fbbeam \fBrollback \fR [\fB-v \fR] [\fB-i-be-name@snapshot \fR]
58 .fi

```

```

60 .SH DESCRIPTION
61 The \fbbeam \fR command is the user interface for managing zfs Boot
62 Environments (BEs). This utility is intended to be used by System
63 Administrators who want to manage multiple Solaris Instances on a single
64 system.
65 .sp
66 The \fbbeam \fR command supports the following operations:
67 .RS +4
68 .TP
69 .ie t \(\bu
70 .el -
71 Create a new BE, based on the active BE.
72 .RE
73 .RS +4
74 .TP
75 .ie t \(\bu
76 .el -
77 Create a new BE, based on an inactive BE.
78 .RE
79 .RS +4
80 .TP
81 .ie t \(\bu
82 .el -
83 Create a snapshot of an existing BE.
84 .RE
85 .RS +4
86 .TP
87 .ie t \(\bu
88 .el -
89 Create a new BE, based on an existing snapshot.
90 .RE
91 .RS +4
92 .TP
93 .ie t \(\bu
94 .el -
95 Create a new BE, and copy it to a different zpools.
96 .RE
97 .RS +4
98 .TP
99 .ie t \(\bu
100 .el -
101 Activate an existing, inactive BE.
102 .RE
103 .RS +4
104 .TP
105 .ie t \(\bu
106 .el -
107 Mount a BE.
108 .RE
109 .RS +4
110 .TP
111 .ie t \(\bu
112 .el -
113 Unmount a BE.
114 .RE
115 .RS +4
116 .TP
117 .ie t \(\bu
118 .el -
119 Destroy a BE.
120 .RE
121 .RS +4
122 .TP
123 .ie t \(\bu
124 .el -

```

```

125 Destroy a snapshot of a BE.
126 .RE
127 .RS +4
128 .TP
129 .ie t \(\bu
130 .el -
131 Rename an existing, inactive BE.
132 .RE
133 .RS +4
134 .TP
135 .ie t \(\bu
136 .el -
137 Roll back a BE to an existing snapshot of a BE.
138 .RE
139 .RS +4
140 .TP
141 .ie t \(\bu
142 .el -
143 Display information about your snapshots and datasets.
144 .RE

146 .SH SUBCOMMANDS
147 The \fbbeadm\fr command has the subcommands and options listed
148 below. Also see
149 EXAMPLES below.
150 .sp
151 .ne 2
152 .na
153 \fbbeadm\fr
154 .ad
155 .sp .6
156 .RS 4n
157 Displays command usage.
158 .RE

160 .sp
161 .ne 2
162 .na
163 \fbbeadm\fr \fbcreate\fr [\fb-a\fr] [\fb-d\fr \fIdescription\fr]
164     [\fb-e\fr \fInon-activeBeName\fr | \fIbeName@snapshot\fr]
165     [\fb-o\fr \fIproperty=value\fr] ... [\fb-p\fr \fIzpool\fr]
166     [\fb-v\fr] \fIbeName\fr

168 .ad
169 .sp .6
170 .RS 4n
171 Creates a new boot environment named \fIbeName\fr. If the \fb-e\fr option is
172 not
173 provided, the new boot environment will be created as a clone of the
174 currently
175 running boot environment. If the \fb-d\fr option is provided then the
176 description is
177 also used as the title for the BE's entry in the GRUB menu for
178 x86 systems or
179 in the boot menu for SPARC systems. If the \fb-d\fr option is
180 not provided, \fIbeName\fr
181 will be used as the title.
182 .sp
183 .ne 2
184 .na
185 \fb-a\fr
186 .ad
187 .sp .6
188 .RS 4n
189 Activate the newly created BE upon creation. The default is to not activate
190 the newly created BE.

```

```

191 .RE
192 .sp
193 .ne 2
194 .na
195 \fb-d\fr \fIdescription\fr
196 .ad
197 .sp .6
198 .RS 4n
199 Create a new BE with a description associated with it.
200 .RE
201 .sp
202 .ne 2
203 .na
204 \fb-e\fr \fInon-activeBeName\fr
205 .ad
206 .sp .6
207 .RS 4n
208 Create a new BE from an existing inactive BE.
209 .RE
210 .sp
211 .ne 2
212 .na
213 \fb-e\fr \fIbeName@snapshot\fr
214 .ad
215 .sp .6
216 .RS 4n
217 Create a new BE from an existing snapshot of the BE named beName.
218 .RE
219 .sp
220 .ne 2
221 .na
222 \fb-o\fr \fIproperty=value\fr
223 .ad
224 .sp .6
225 .RS 4n
226 Create the datasets for new BE with specific ZFS properties. Multiple
227 \fb-o\fr
228 options can be specified. See \fBzfs\fr(1M) for more information on
229 the
230 \fb-o\fr option.
231 .RE
232 .sp
233 .ne 2
234 .na
235 \fb-p\fr \fIzpool\fr
236 .ad
237 .sp .6
238 .RS 4n
239 Create the new BE in the specified zpool. If this is not provided, the
240 default
241 behavior is to create the new BE in the same pool as as the origin BE.
242 This option is not supported in non-global zone.
243 .RE
244 .sp
245 .ne 2
246 .na
247 \fb-v\fr
248 .ad
249 .sp .6
250 .RS 4n
251 Verbose mode. Displays verbose error messages from \fbbeadm\fr.
252 .RE
253 .RE

255 .sp
256 .ne 2

```

```

257 .na
258 \fBbeadm\fR \fBcreate\fR [\fB-v\fR] \fIbeName@snapshot\fR
259 .ad
260 .sp .6
261 .RS 4n
262 Creates a snapshot of the existing BE named beName.
263 .sp
264 .ne 2
265 .na
266 \fB-v\fR
267 .ad
268 .sp .6
269 .RS 4n
270 Verbose mode. Displays verbose error messages from \fBbeadm\fR.
271 .RE
272 .RE

274 .sp
275 .ne 2
276 .na
277 \fBbeadm\fR \fBdestroy\fR [\fB-fFsv\fR] \fIbeName\fR | \fIbeName@snapshot\fR
278 .ad
279 .sp .6
280 .RS 4n
281 Destroys the boot environment named \fIbeName\fR or destroys an existing
282 snapshot of
283 the boot environment named \fIbeName@snapshot\fR. Destroying a
284 boot environment
285 will also destroy all snapshots of that boot environment. Use
286 this command
287 with caution.
288 .sp
289 .ne 2
290 .na
291 \fB-f\fR
292 .ad
293 .sp .6
294 .RS 4n
295 Forcefully unmount the boot environment if it is currently mounted.
296 .RE
297 .sp
298 .ne 2
299 .na
300 \fB-F\fR
301 .ad
302 .sp .6
303 .RS 4n
304 Force the action without prompting to verify the destruction of the boot
305 environment.
306 .RE
307 .sp
308 .ne 2
309 .na
310 \fB-s\fR
311 .ad
312 .sp .6
313 .RS 4n
314 Destroy all snapshots of the boot
315 environment.
316 .RE
317 .sp
318 .ne 2
319 .na
320 \fB-v\fR
321 .ad
322 .sp .6

```

```

323 .RS 4n
324 Verbose mode. Displays verbose error messages from \fBbeadm\fR.
325 .RE
326 .RE

328 .sp
329 .ne 2
330 .na
331 \fBbeadm\fR \fBlist\fR [\fB-a\fR | \fB-ds\fR] [\fB-H\fR] [\fB-v\fR] [\fIbeName\fR
332 .ad
333 .sp .6
334 .RS 4n
335 Lists information about the existing boot environment named \fIbeName\fR, or
336 lists
337 information for all boot environments if \fIbeName\fR is not provided.
338 The 'Active'
339 field indicates whether the boot environment is active now,
340 represented
341 by 'N'; active on reboot, represented by 'R'; or both, represented
342 by 'NR'. In non-global zone the 'Active' field also indicates whether the
343 boot environment has a non-active parent BE, represented by 'x'; is active
344 on boot in a non-active parent BE, represented by 'b'. Activate, rollback
345 and snapshot operations for boot environments from non-active global parent
346 BE aren't supported, destroy is allowed if these boot environments aren't
347 active on boot.
348 .sp
349 Each line in the machine parsable output has the boot environment name as the
350 first field. The 'Space' field is displayed in bytes and the 'Created' field
351 is displayed in UTC format. The \fB-H\fR option used with no other options
352 gives
353 the boot environment's uuid in the second field. This field will be
354 blank if
355 the boot environment does not have a uuid. See the EXAMPLES section.
356 In non-global zones, this field shows the uuid of the parent BE.
357 .sp
358 .ne 2
359 .na
360 \fB-a\fR
361 .ad
362 .sp .6
363 .RS 4n
364 Lists all available information about the boot environment. This includes
365 subordinate file systems and snapshots.
366 .RE
367 .sp
368 .ne 2
369 .na
370 \fB-d\fR
371 .ad
372 .sp .6
373 .RS 4n
374 Lists information about all subordinate file systems belonging to the boot
375 environment.
376 .RE
377 .sp
378 .ne 2
379 .na
380 \fB-s\fR
381 .ad
382 .sp .6
383 .RS 4n
384 Lists information about the snapshots of the boot environment.
385 .RE
386 .sp
387 .ne 2
388 .na

```

```

389 \fB-H\fR
390 .ad
391 .sp .6
392 .RS 4n
393 Do not list header information. Each field in the list information is
394 separated by a semicolon.
395 .RE
396 .sp
397 .ne 2
398 .na
399 \fB-v\fR
400 .ad
401 .sp .6
402 .RS 4n
403 Verbose mode. Displays verbose error messages from \fBbeamd\fR.
404 .RE
405 .RE

407 .sp
408 .ne 2
409 .na
410 \fBbeamd\fR \fBmount\fR [\fB-s\fR \fBro\fR|\fBrw\fR] [\fB-v\fR] \fIbeName\fR \fI
410 \fBbeamd\fR \fBmount\fR [\fB-v\fR] \fIbeName\fR \fImountpoint\fR
411 .ad
412 .sp .6
413 .RS 4n
414 Mounts a boot environment named beName at mountpoint. mountpoint must be an
415 already existing empty directory.
416 .sp
417 .ne 2
418 .na
419 \fB-s\fR \fBro\fR|\fBrw\fR
420 .ad
421 .sp .6
422 .RS 4n
423 Mount the shared filesystems of the BE in read-only or read-write mode.
424 .RE
425 .sp
426 .ne 2
427 .na
428 \fB-v\fR
429 .ad
430 .sp .6
431 .RS 4n
432 Verbose mode. Displays verbose error messages from \fBbeamd\fR.
433 .RE
434 .RE

436 .sp
437 .ne 2
438 .na
439 \fBbeamd\fR \fBunmount\fR [\fB-fv\fR] \fIbeName\fR | \fImountpoint\fR
440 .ad
441 .sp .6
442 .RS 4n
443 Unmounts the boot environment named beName. The command can also be given a path
444 beName mount point on the system.
445 .sp
446 .ne 2
447 .na
448 \fB-f\fR
449 .ad
450 .sp .6
451 .RS 4n
452 Forcefully unmount the boot environment even if its currently busy.
453 .RE

```

```

454 .sp
455 .ne 2
456 .na
457 \fB-v\fR
458 .ad
459 .sp .6
460 .RS 4n
461 Verbose mode. Displays verbose error messages from \fBbeamd\fR.
462 .RE
463 .RE

465 .sp
466 .ne 2
467 .na
468 \fBbeamd\fR \fBrename\fR [\fB-v\fR] \fIbeName\fR \fInewBeName\fR
469 .ad
470 .sp .6
471 .RS 4n
472 Renames the boot environment named \fIbeName\fR to \fInewBeName\fR.
473 .sp
474 .ne 2
475 .na
476 \fB-v\fR
477 .ad
478 .sp .6
479 .RS 4n
480 Verbose mode. Displays verbose error messages from \fBbeamd\fR.
481 .RE
482 .RE

484 .sp
485 .ne 2
486 .na
487 \fBbeamd\fR \fBrollback\fR [\fB-v\fR] \fIbeName\fR \fIsnapshot\fR | \fIbeName@sn
488 .ad
489 .sp .6
490 .RS 4n
491 Roll back the boot environment named \fIbeName\fR to existing snapshot
492 of the boot environment named \fIbeName@snapshot\fR.
493 .sp
494 .ne 2
495 .na
496 \fB-v\fR
497 .ad
498 .sp .6
499 .RS 4n
500 Verbose mode. Displays verbose error messages from \fBbeamd\fR.
501 .RE
502 .RE

504 .sp
505 .ne 2
506 .na
507 \fBbeamd\fR \fBactivate\fR [\fB-v\fR] \fIbeName\fR
508 .ad
509 .sp .6
510 .RS 4n
511 Makes beName the active BE on next reboot.
512 .sp
513 .ne 2
514 .na
515 \fB-v\fR
516 .ad
517 .sp .6
518 .RS 4n
519 Verbose mode. Displays verbose error messages from \fBbeamd\fR.

```



```

520 .RE
521 .RE

523 .SH ALTERNATE BE LOCATION
524 .LP
525 The alternate BE location outside rpool/ROOT can be configured
526 by modifying the BENAME_STARTS_WITH parameter in /etc/default/be.
527 For example: BENAME_STARTS_WITH=rootfs

529 .SH EXAMPLES
530 .LP
531 \fBExample 1\fR: Create a new BE named BE1, by cloning the current live BE.
532 .sp
533 .in +2
534 .nf
535 \fB# beadm create BE1\fR
536 .fi
537 .in -2
538 .sp

540 .LP
541 \fBExample 2\fR: Create a new BE named BE2, by cloning the existing inactive
542 BE
543 named BE1.
544 .sp
545 .in +2
546 .nf
547 \fB# beadm create -e BE1 BE2\fR
548 .fi
549 .in -2
550 .sp

552 .LP
553 \fBExample 3\fR: Create a snapshot named now of the existing BE named BE1.
554 .sp
555 .in +2
556 .nf
557 \fB# beadm create BE1@now\fR
558 .fi
559 .in -2
560 .sp

562 .LP
563 \fBExample 4\fR: Create a new BE named BE3, by cloning an existing snapshot of
564 BE1.
565 .sp
566 .in +2
567 .nf
568 \fB# beadm create -e BE1@now BE3\fR
569 .fi
570 .in -2
571 .sp

573 .LP
574 \fBExample 5\fR: Create a new BE named BE4 based on the currently running BE.
575 Create the new BE in rpool2.
576 .sp
577 .in +2
578 .nf
579 \fB# beadm create -p rpool2 BE4\fR
580 .fi
581 .in -2
582 .sp

584 .LP
585 \fBExample 6\fR: Create a new BE named BE5 based on the currently running BE.

```

```

586 Create the new BE in rpool2, and create its datasets with compression turned
587 on.
588 .sp
589 .in +2
590 .nf
591 \fB# beadm create -p rpool2 -o compression=on BE5\fR
592 .fi
593 .in -2
594 .sp

596 .LP
597 \fBExample 7\fR: Create a new BE named BE6 based on the currently running BE
598 and provide a description for it.
599 .sp
600 .in +2
601 .nf
602 \fB# beadm create -d "BE6 used as test environment" BE6\fR
603 .fi
604 .in -2
605 .sp

607 .LP
608 \fBExample 8\fR: Activate an existing, inactive BE named BE3.
609 .sp
610 .in +2
611 .nf
612 \fB# beadm activate BE3\fR
613 .fi
614 .in -2
615 .sp

617 .LP
618 \fBExample 9\fR: Mount the BE named BE3 at /mnt.
619 .sp
620 .in +2
621 .nf
622 \fB# beadm mount BE3 /mnt\fR
623 .fi
624 .in -2
625 .sp

627 .LP
628 \fBExample 10\fR: Unmount the mounted BE named BE3.
629 .sp
630 .in +2
631 .nf
632 \fB# beadm unmount BE3\fR
633 .fi
634 .in -2
635 .sp

637 .LP
638 \fBExample 11\fR: Destroy the BE named BE3 without verification.
639 .sp
640 .in +2
641 .nf
642 \fB# beadm destroy -f BE3\fR
643 .fi
644 .in -2
645 .sp

647 .LP
648 \fBExample 12\fR: Destroy the snapshot named now of BE1.
649 .sp
650 .in +2
651 .nf

```

```

652 \fB# beadm destroy BE1@now\fR
653 .fi
654 .in -2
655 .sp

657 .LP
658 \fBExample 13\fR: Rename the existing, inactive BE named BE1 to BE3.
659 .sp
660 .in +2
661 .nf
662 \fB# beadm rename BE1 BE3\fR
663 .fi
664 .in -2
665 .sp

667 .LP
668 \fBExample 14\fR: Roll back the BE named BE1 to snapshot BE1@now.
669 .sp
670 .in +2
671 .nf
672 \fB# beadm rollback BE1 BE1@now\fR
673 .fi
674 .in -2
675 .sp

677 .LP
678 \fBExample 15\fR: List all existing boot environments.

680 .sp
681 .in +2
682 .nf
683 \fB# beadm list\fR
684 BE Active Mountpoint Space Policy Created
685 --
686 BE2 - - 72.0K static 2008-05-21 12:26
687 BE3 - - 332.0K static 2008-08-26 10:28
688 BE4 - - 15.78M static 2008-09-05 18:20
689 BE5 NR / 7.25G static 2008-09-09 16:53
690 .fi
691 .in -2
692 .sp

694 .LP
695 \fBExample 16\fR: List all existing boot environments and list all dataset and
696 snapshot information about those bootenvironments.

698 .sp
699 .in +2
700 .nf
701 \fB# beadm list -d -s\fR

703 BE/Dataset/Snapshot Active Mountpoint Space Policy Created
704 -----
705 BE2
706 p/ROOT/BE2 - - 36.0K static 2008-05-21 12:26
707 p/ROOT/BE2/opt - - 18.0K static 2008-05-21 16:26
708 p/ROOT/BE2/opt@now - - 0 static 2008-09-08 22:43
709 p/ROOT/BE2@now - - 0 static 2008-09-08 22:43
710 BE3
711 p/ROOT/BE3 - - 192.0K static 2008-08-26 10:28
712 p/ROOT/BE3/opt - - 86.0K static 2008-08-26 10:28
713 p/ROOT/BE3/opt/local - - 36.0K static 2008-08-28 10:58
714 BE4
715 p/ROOT/BE4 - - 15.78M static 2008-09-05 18:20
716 BE5
717 p/ROOT/BE5 NR / 6.10G static 2008-09-09 16:53

```

```

718 p/ROOT/BE5/opt - /opt 24.55M static 2008-09-09 16:53
719 p/ROOT/BE5/opt@bar - - 18.38M static 2008-09-10 00:59
720 p/ROOT/BE5/opt@foo - - 18.38M static 2008-06-10 16:37
721 p/ROOT/BE5@bar - - 139.44M static 2008-09-10 00:59
722 p/ROOT/BE5@foo - - 912.85M static 2008-06-10 16:37
723 .fi
724 .in -2
725 .sp

727 \fBExample 17\fR: List all dataset and snapshot information about BE5

729 .sp
730 .in +2
731 .nf
732 \fB# beadm list -a BE5\fR

734 BE/Dataset/Snapshot Active Mountpoint Space Policy Created
735 -----
736 BE5
737 p/ROOT/BE5 NR / 6.10G static 2008-09-09 16:53
738 p/ROOT/BE5/opt - /opt 24.55M static 2008-09-09 16:53
739 p/ROOT/BE5/opt@bar - - 18.38M static 2008-09-10 00:59
740 p/ROOT/BE5/opt@foo - - 18.38M static 2008-06-10 16:37
741 p/ROOT/BE5@bar - - 139.44M static 2008-09-10 00:59
742 p/ROOT/BE5@foo - - 912.85M static 2008-06-10 16:37
743 .fi
744 .in -2
745 .sp

747 .LP
748 \fBExample 18\fR: List machine parsable information about all boot
749 environments.

751 .sp
752 .in +2
753 .nf
754 \fB# beadm list -H\fR

756 BE2;;;55296;static;1211397974
757 BE3;;;339968;static;1219771706
758 BE4;;;16541696;static;1220664051
759 BE5;215b8387-4968-627c-d2d0-f4a011414bab;NR;;7786206208;static;1221004384
760 .fi
761 .in -2
762 .sp

764 .SH EXIT STATUS
765 .sp
766 .LP
767 The following exit values are returned:
768 .sp
769 .ne 2
770 .na
771 \fB0\fR
772 .ad
773 .sp .6
774 .RS 4n
775 Successful completion
776 .RE

778 .sp
779 .ne 2
780 .na
781 \fB>0\fR
782 .ad
783 .sp .6

```

```
784 .RS 4n
785 Failure
786 .RE

789 .SH FILES
790 .sp
791 .LP
792 .sp
793 .ne 2
794 .na
795 \fB/var/log/beadm/<beName>/create.log.<yyyymmdd_hhmmss>\fR
796 .ad
797 .sp .6
798 .RS 4n
799 Log used for capturing beadm create output
800 .sp
801 .nf
802 \fIyyyymmdd_hhmmss\fR - 20071130_140558
803 \fIyy\fR - year; 2007
804 \fImm\fR - month; 11
805 \fIdd\fR - day; 30
806 \fIhh\fR - hour; 14
807 \fImm\fR - minute; 05
808 \fIss\fR - second; 58
809 .fi
810 .in -2
811 .sp
812 .RE
813 .sp
814 .LP
815 .sp
816 .ne 2
817 .na
818 \fB/etc/default/be\fR
819 .ad
820 .sp .6
821 .RS 4n
822 Contains default value for BENAME_STARTS_WITH parameter
823 .sp
824 .RE

826 .SH ATTRIBUTES
827 .sp
828 .LP
829 See \fBattributes\fR(5) for descriptions of the following attributes:
830 .sp

832 .sp
833 .TS
834 box;
835 c | c
836 l | l .
837 ATTRIBUTE TYPE ATTRIBUTE VALUE
838 -
839 Interface Stability Uncommitted
840 .TE

843 .SH SEE ALSO
844 .sp
845 .LP
846 .BR zfs (1M)
```