```
**********************************************************
   34217 Wed Jul 18 21:07:51 2012
new/usr/src/cmd/beadm/beadm.c
2447 beadm should be more descriptive about some errors
**********************************************************
______unchanged_portion_omitted_

 118 static void
 119 usage(void)
 120 {
 121         (void) fprintf(stderr, _("usage:\n"
 122             "\tbeadm subcommand cmd_options\n"
 123             "\n"
 124             "\tsubcommands:\n"
 125             "\n"
 126             "\tbeadm activate [-v] beName\n"
 126             "\tbeadm create [-a] [-d BE_desc]\n"
 126             "\tbeadm activate beName\n"
 127             "\tbeadm create [-d BE_desc]\n"
 128             "\t\t[-o property=value] ... [-p zpool] \n"
 129             "\t\t[-e nonActiveBe | beName@snapshot] [-v] beName\n"
 129             "\t\t[-e nonActiveBe | beName@snapshot] beName\n"
 130             "\tbeadm create [-d BE_desc]\n"
 131             "\t\t[-o property=value] ... [-p zpool] [-v] beName@snapshot\n"
 132             "\tbeadm destroy [-Ffsv] beName \n"
 133             "\tbeadm destroy [-Fv] beName@snapshot \n"
 134             "\tbeadm list [[-a] | [-d] [-s]] [-H] [-v] [beName]\n"
 135             "\tbeadm mount [-s ro|rw] [-v] beName [mountpoint]\n"
 136             "\tbeadm unmount [-fv] beName | mountpoint\n"
 137             "\tbeadm umount [-fv] beName | mountpoint\n"
 138             "\tbeadm rename [-v] origBeName newBeName\n"
 139             "\tbeadm rollback [-v] beName snapshot\n"
 140             "\tbeadm rollback [-v] beName@snapshot\n"));
 131             "\t\t[-o property=value] ... [-p zpool] beName@snapshot\n"
 132             "\tbeadm destroy [-Ffs] beName \n"
 133             "\tbeadm destroy [-F] beName@snapshot \n"
 134             "\tbeadm list [[-a] | [-d] [-s]] [-H] [beName]\n"
 135             "\tbeadm mount [-s ro|rw] beName [mountpoint]\n"
 136             "\tbeadm unmount [-f] beName | mountpoint\n"
 137             "\tbeadm umount [-f] beName | mountpoint\n"
 138             "\tbeadm rename origBeName newBeName\n"
 139             "\tbeadm rollback beName snapshot\n"
 140             "\tbeadm rollback beName@snapshot\n"));
 141 }
______unchanged_portion_omitted_

 662 static int
 663 be_do_activate(int argc, char **argv)
 664 {
 665         nvlist_t        *be_attrs;
 666         int             err = 1;
 667         int             c;
 668         char            *obe_name;

 670         while ((c = getopt(argc, argv, "v")) != -1) {
 671                 switch (c) {
 672                 case 'v':
 673                         libbe_print_errors(B_TRUE);
 674                         break;
 675                 default:
 676                         usage();
 677                         return (1);
 678                 }
 679         }

 681         argc -= optind;
```



```
 682         argv += optind;

 684         if (argc != 1) {
 685                 usage();
 686                 return (1);
 687         }

 689         obe_name = argv[0];

 691         if (be_nvl_alloc(&be_attrs) != 0)
 692                 return (1);

 694         if (be_nvl_add_string(be_attrs, BE_ATTR_ORIG_BE_NAME, obe_name) != 0)
 695                 goto out;

 697         err = be_activate(be_attrs);

 699         switch (err) {
 700         case BE_SUCCESS:
 701                 (void) printf(_("Activated successfully\n"));
 702                 break;
 703         case BE_ERR_BE_NOENT:
 704                 (void) fprintf(stderr, _("%s does not exist or appear "
 705                     "to be a valid BE.\nPlease check that the name of "
 706                     "the BE provided is correct.\n"), obe_name);
 707                 break;
 708         case BE_ERR_PERM:
 709         case BE_ERR_ACCESS:
 710                 (void) fprintf(stderr, _("Unable to activate %s.\n"), obe_name);
 711                 (void) fprintf(stderr, _("You have insufficient privileges to "
 712                     "execute this command.\n"));
 713                 break;
 714         case BE_ERR_ACTIVATE_CURR:
 715         default:
 716                 (void) fprintf(stderr, _("Unable to activate %s.\n"), obe_name);
 717                 (void) fprintf(stderr, "%s\n", be_err_to_str(err));
 718         }

 720 out:
 721         nvlist_free(be_attrs);
 722         return (err);
 723 }

 725 static int
 726 be_do_create(int argc, char **argv)
 727 {
 728         nvlist_t        *be_attrs;
 729         nvlist_t        *zfs_props = NULL;
 730         boolean_t       activate = B_FALSE;
 731         boolean_t       is_snap = B_FALSE;
 732         int             c;
 733         int             err = 1;
 734         char            *obe_name = NULL;
 735         char            *snap_name = NULL;
 736         char            *nbe_zpool = NULL;
 737         char            *nbe_name = NULL;
 738         char            *nbe_desc = NULL;
 739         char            *propname = NULL;
 740         char            *propval = NULL;
 741         char            *strval = NULL;

 743         while ((c = getopt(argc, argv, "ad:e:io:p:v")) != -1) {
 731         while ((c = getopt(argc, argv, "ad:e:io:p:")) != -1) {
 744                 switch (c) {
 745                 case 'a':
 746                         activate = B_TRUE;
```

```
747                     break;
748             case 'd':
749                     nbe_desc = optarg;
750                     break;
751             case 'e':
752                     obe_name = optarg;
753                     break;
754             case 'o':
755                     if (zfs_props == NULL && be_nvl_alloc(&zfs_props) != 0)
756                             return (1);

758                     propname = optarg;
759                     if ((propval = strchr(propname, '=')) == NULL) {
760                             (void) fprintf(stderr, _("missing "
761                                 "'=' for -o option\n"));
762                             goto out2;
763                     }
764                     *propval = '\0';
765                     propval++;
766                     if (nvlist_lookup_string(zfs_props, propname,
767                         &strval) == 0) {
768                             (void) fprintf(stderr, _("property '%s' "
769                                 "specified multiple times\n"), propname);
770                             goto out2;

772                     }
773                     if (be_nvl_add_string(zfs_props, propname, propval)
774                         != 0)
775                             goto out2;

777                     break;
778             case 'p':
779                     nbe_zpool = optarg;
780                     break;
781             case 'v':
782                     libbe_print_errors(B_TRUE);
783                     break;
784             default:
785                     usage();
786                     goto out2;
787             }
788     }

790     argc -= optind;
791     argv += optind;

793     if (argc != 1) {
794             usage();
795             goto out2;
796     }

798     nbe_name = argv[0];

800     if ((snap_name = strrchr(nbe_name, '@')) != NULL) {
801             if (snap_name[1] == '\0') {
802                     usage();
803                     goto out2;
804             }

806             snap_name[0] = '\0';
807             snap_name++;
808             is_snap = B_TRUE;
809     }

811     if (obe_name) {
812             if (is_snap) {
```



```
813                     usage();
814                     goto out2;
815             }

817             /*
818              * Check if obe_name is really a snapshot name.
819              * If so, split it out.
820              */
821             if ((snap_name = strrchr(obe_name, '@')) != NULL) {
822                     if (snap_name[1] == '\0') {
823                             usage();
824                             goto out2;
825                     }

827                     snap_name[0] = '\0';
828                     snap_name++;
829             }
830     } else if (is_snap) {
831             obe_name = nbe_name;
832             nbe_name = NULL;
833     }

835     if (be_nvl_alloc(&be_attrs) != 0)
836             goto out2;


839     if (zfs_props != NULL && be_nvl_add_nvlist(be_attrs,
840         BE_ATTR_ORIG_BE_NAME, zfs_props) != 0)
841             goto out;

843     if (obe_name != NULL && be_nvl_add_string(be_attrs,
844         BE_ATTR_ORIG_BE_NAME, obe_name) != 0)
845             goto out;

847     if (snap_name != NULL && be_nvl_add_string(be_attrs,
848         BE_ATTR_SNAP_NAME, snap_name) != 0)
849             goto out;

851     if (nbe_zpool != NULL && be_nvl_add_string(be_attrs,
852         BE_ATTR_NEW_BE_POOL, nbe_zpool) != 0)
853             goto out;

855     if (nbe_name != NULL && be_nvl_add_string(be_attrs,
856         BE_ATTR_NEW_BE_NAME, nbe_name) != 0)
857             goto out;

859     if (nbe_desc != NULL && be_nvl_add_string(be_attrs,
860         BE_ATTR_NEW_BE_DESC, nbe_desc) != 0)
861             goto out;

863     if (is_snap)
864             err = be_create_snapshot(be_attrs);
865     else
866             err = be_copy(be_attrs);

868     switch (err) {
869     case BE_SUCCESS:
870             if (!is_snap && !nbe_name) {
871                     /*
872                      * We requested an auto named BE; find out the
873                      * name of the BE that was created for us and
874                      * the auto snapshot created from the original BE.
875                      */
876                     if (nvlist_lookup_string(be_attrs, BE_ATTR_NEW_BE_NAME,
877                         &nbe_name) != 0) {
878                             (void) fprintf(stderr, _("failed to get %s "
```

```
879                                     "attribute\n"), BE_ATTR_NEW_BE_NAME);
880                             break;
881                     } else
882                             (void) printf(_("Auto named BE: %s\n"),
883                                 nbe_name);
884
885                     if (nvlist_lookup_string(be_attrs, BE_ATTR_SNAP_NAME,
886                         &snap_name) != 0) {
887                             (void) fprintf(stderr, _("failed to get %s "
888                                 "attribute\n"), BE_ATTR_SNAP_NAME);
889                             break;
890                     } else
891                             (void) printf(_("Auto named snapshot: %s\n"),
892                                 snap_name);
893             }
894
895             if (!is_snap && activate) {
896                     char *args[] = { "activate", "", NULL };
897                     args[1] = nbe_name;
898                     optind = 1;
899
900                     err = be_do_activate(2, args);
901                     goto out;
902             }
903
904             (void) printf(_("Created successfully\n"));
905             break;
906     case BE_ERR_BE_EXISTS:
907             (void) fprintf(stderr, _("BE %s already exists\n."
908                 "Please choose a different BE name.\n"), nbe_name);
909             break;
910     case BE_ERR_SS_EXISTS:
911             (void) fprintf(stderr, _("BE %s snapshot %s already exists.\n"
912                 "Please choose a different snapshot name.\n"), obe_name,
913                 snap_name);
914             break;
915     case BE_ERR_PERM:
916     case BE_ERR_ACCESS:
917             if (is_snap)
918                     (void) fprintf(stderr, _("Unable to create snapshot "
919                         "%s.\n"), snap_name);
920             else
921                     (void) fprintf(stderr, _("Unable to create %s.\n"),
922                         nbe_name);
923             (void) fprintf(stderr, _("You have insufficient privileges to "
924                 "execute this command.\n"));
925             break;
926     default:
927             if (is_snap)
928                     (void) fprintf(stderr, _("Unable to create snapshot "
929                         "%s.\n"), snap_name);
930             else
931                     (void) fprintf(stderr, _("Unable to create %s.\n"),
932                         nbe_name);
933             (void) fprintf(stderr, "%s\n", be_err_to_str(err));
934     }
935
936 out:
937     nvlist_free(be_attrs);
938 out2:
939     if (zfs_props != NULL)
940             nvlist_free(zfs_props);
941
942     return (err);
943 }
```



```
945 static int
946 be_do_destroy(int argc, char **argv)
947 {
948         nvlist_t        *be_attrs;
949         boolean_t       is_snap = B_FALSE;
950         boolean_t       suppress_prompt = B_FALSE;
951         int             err = 1;
952         int             c;
953         int             destroy_flags = 0;
954         char            *snap_name;
955         char            *be_name;
956
957         while ((c = getopt(argc, argv, "fFsv")) != -1) {
942         while ((c = getopt(argc, argv, "fFs")) != -1) {
958                 switch (c) {
959                 case 'f':
960                         destroy_flags |= BE_DESTROY_FLAG_FORCE_UNMOUNT;
961                         break;
962                 case 's':
963                         destroy_flags |= BE_DESTROY_FLAG_SNAPSHOTS;
964                         break;
965                 case 'v':
966                         libbe_print_errors(B_TRUE);
967                         break;
968                 case 'F':
969                         suppress_prompt = B_TRUE;
970                         break;
971                 default:
972                         usage();
973                         return (1);
974                 }
975         }
976
977         argc -= optind;
978         argv += optind;
979
980         if (argc != 1) {
981                 usage();
982                 return (1);
983         }
984
985         be_name = argv[0];
986         if (!suppress_prompt && !confirm_destroy(be_name)) {
987                 (void) printf(_("%s has not been destroyed.\n"), be_name);
988                 return (0);
989         }
990
991         if ((snap_name = strrchr(be_name, '@')) != NULL) {
992                 if (snap_name[1] == '\0') {
993                         usage();
994                         return (1);
995                 }
996
997                 is_snap = B_TRUE;
998                 *snap_name = '\0';
999                 snap_name++;
1000        }
1001
1002        if (be_nvl_alloc(&be_attrs) != 0)
1003                return (1);
1004
1005
1006        if (be_nvl_add_string(be_attrs, BE_ATTR_ORIG_BE_NAME, be_name) != 0)
1007                goto out;
1008
1009        if (is_snap) {
```

```
1010                 if (be_nvl_add_string(be_attrs, BE_ATTR_SNAP_NAME,
1011                     snap_name) != 0)
1012                         goto out;

1014                 err = be_destroy_snapshot(be_attrs);
1015         } else {
1016                 if (be_nvl_add_uint16(be_attrs, BE_ATTR_DESTROY_FLAGS,
1017                     destroy_flags) != 0)
1018                         goto out;

1020                 err = be_destroy(be_attrs);
1021         }

1023         switch (err) {
1024         case BE_SUCCESS:
1025                 (void) printf(_("Destroyed successfully\n"));
1026                 break;
1027         case BE_ERR_MOUNTED:
1028                 (void) fprintf(stderr, _("Unable to destroy %s.\n"), be_name);
1029                 (void) fprintf(stderr, _("It is currently mounted and must be "
1030                     "unmounted before it can be destroyed.\n" "Use 'beadm "
1031                     "unmount %s' to unmount the BE before destroying\nit or "
1032                     "'beadm destroy -f %s'.\n"), be_name, be_name);
1033                 break;
1034         case BE_ERR_DESTROY_CURR_BE:
1035                 (void) fprintf(stderr, _("%s is the currently active BE and "
1036                     "cannot be destroyed.\nYou must boot from another BE in "
1037                     "order to destroy %s.\n"), be_name, be_name);
1038                 break;
1039         case BE_ERR_ZONES_UNMOUNT:
1040                 (void) fprintf(stderr, _("Unable to destroy one of " "%s's "
1041                     "zone BE's.\nUse 'beadm destroy -f %s' or "
1042                     "'zfs -f destroy <dataset>'.\n"), be_name, be_name);
1043                 break;
1044         case BE_ERR_SS_NOENT:
1045                 (void) fprintf(stderr, _("%s does not exist or appear "
1046                     "to be a valid snapshot.\nPlease check that the name of "
1047                     "the snapshot provided is correct.\n"), snap_name);
1048                 break;
1049         case BE_ERR_PERM:
1050         case BE_ERR_ACCESS:
1051                 (void) fprintf(stderr, _("Unable to destroy %s.\n"), be_name);
1052                 (void) fprintf(stderr, _("You have insufficient privileges to "
1053                     "execute this command.\n"));
1054                 break;
1055         case BE_ERR_SS_EXISTS:
1056                 (void) fprintf(stderr, _("Unable to destroy %s: "
1057                     "BE has snapshots.\nUse 'beadm destroy -s %s' or "
1058                     "'zfs -r destroy <dataset>'.\n"), be_name, be_name);
1059                 break;
1060         default:
1061                 (void) fprintf(stderr, _("Unable to destroy %s.\n"), be_name);
1062                 (void) fprintf(stderr, "%s\n", be_err_to_str(err));
1063         }

1065 out:
1066         nvlist_free(be_attrs);
1067         return (err);
1068 }

1070 static int
1071 be_do_list(int argc, char **argv)
1072 {
1073         be_node_list_t  *be_nodes = NULL;
1074         boolean_t       all = B_FALSE;
1075         boolean_t       dsets = B_FALSE;
```



```
1076         boolean_t       snaps = B_FALSE;
1077         boolean_t       parsable = B_FALSE;
1078         int             err = 1;
1079         int             c = 0;
1080         char            *be_name = NULL;

1082         while ((c = getopt(argc, argv, "adsvH")) != -1) {
1064         while ((c = getopt(argc, argv, "nadsH")) != -1) {
1083                 switch (c) {
1084                 case 'a':
1085                         all = B_TRUE;
1086                         break;
1087                 case 'd':
1088                         dsets = B_TRUE;
1089                         break;
1090                 case 's':
1091                         snaps = B_TRUE;
1092                         break;
1093                 case 'v':
1094                         libbe_print_errors(B_TRUE);
1095                         break;
1096                 case 'H':
1097                         parsable = B_TRUE;
1098                         break;
1099                 default:
1100                         usage();
1101                         return (1);
1102                 }
1103         }

1105         if (all) {
1106                 if (dsets) {
1107                         (void) fprintf(stderr, _("Invalid options: -a and %s "
1108                             "are mutually exclusive.\n"), "-d");
1109                         usage();
1110                         return (1);
1111                 }
1112                 if (snaps) {
1113                         (void) fprintf(stderr, _("Invalid options: -a and %s "
1114                             "are mutually exclusive.\n"), "-s");
1115                         usage();
1116                         return (1);
1117                 }

1119                 dsets = B_TRUE;
1120                 snaps = B_TRUE;
1121         }

1123         argc -= optind;
1124         argv += optind;


1127         if (argc == 1)
1128                 be_name = argv[0];

1130         err = be_list(be_name, &be_nodes);

1132         switch (err) {
1133         case BE_SUCCESS:
1134                 print_nodes(be_name, dsets, snaps, parsable, be_nodes);
1135                 break;
1136         case BE_ERR_BE_NOENT:
1137                 if (be_name == NULL)
1138                         (void) fprintf(stderr, _("No boot environments found "
1139                             "on this system.\n"));
1140                 else {
```

```
1141                        (void) fprintf(stderr, _("%s does not exist or appear "
1142                            "to be a valid BE.\nPlease check that the name of "
1143                            "the BE provided is correct.\n"), be_name);
1144                }
1145                break;
1146        default:
1147                (void) fprintf(stderr, _("Unable to display Boot "
1148                    "Environment\n"));
1149                (void) fprintf(stderr, "%s\n", be_err_to_str(err));
1150        }

1152        if (be_nodes != NULL)
1153                be_free_list(be_nodes);
1154        return (err);
1155 }

1157 static int
1158 be_do_mount(int argc, char **argv)
1159 {
1160        nvlist_t        *be_attrs;
1161        boolean_t       shared_fs = B_FALSE;
1162        int             err = 1;
1163        int             c;
1164        int             mount_flags = 0;
1165        char            *obe_name;
1166        char            *mountpoint;
1167        char            *tmp_mp = NULL;

1169        while ((c = getopt(argc, argv, "s:v")) != -1) {
1148        while ((c = getopt(argc, argv, "s:")) != -1) {
1170                switch (c) {
1171                case 's':
1172                        shared_fs = B_TRUE;

1174                        mount_flags |= BE_MOUNT_FLAG_SHARED_FS;

1176                        if (strcmp(optarg, "rw") == 0) {
1177                                mount_flags |= BE_MOUNT_FLAG_SHARED_RW;
1178                        } else if (strcmp(optarg, "ro") != 0) {
1179                                (void) fprintf(stderr, _("The -s flag "
1180                                    "requires an argument [ rw | ro ]\n"));
1181                                usage();
1182                                return (1);
1183                        }

1185                        break;
1186                case 'v':
1187                        libbe_print_errors(B_TRUE);
1188                        break;
1189                default:
1190                        usage();
1191                        return (1);
1192                }
1193        }

1195        argc -= optind;
1196        argv += optind;

1198        if (argc < 1 || argc > 2) {
1199                usage();
1200                return (1);
1201        }

1203        obe_name = argv[0];

1205        if (argc == 2) {
```



```
1206                mountpoint = argv[1];
1207                if (mountpoint[0] != '/') {
1208                        (void) fprintf(stderr, _("Invalid mount point %s. "
1209                            "Mount point must start with a /.\n"), mountpoint);
1210                        return (1);
1211                }
1212        } else {
1213                const char *tmpdir = getenv("TMPDIR");
1214                const char *tmpname = "tmp.XXXXXX";
1215                int sz;

1217                if (tmpdir == NULL)
1218                        tmpdir = "/tmp";

1220                sz = asprintf(&tmp_mp, "%s/%s", tmpdir, tmpname);
1221                if (sz < 0) {
1222                        (void) fprintf(stderr, _("internal error: "
1223                            "out of memory\n"));
1224                        return (1);
1225                }

1227                mountpoint = mkdtemp(tmp_mp);
1228        }

1230        if (be_nvl_alloc(&be_attrs) != 0)
1231                return (1);

1233        if (be_nvl_add_string(be_attrs, BE_ATTR_ORIG_BE_NAME, obe_name) != 0)
1234                goto out;

1236        if (be_nvl_add_string(be_attrs, BE_ATTR_MOUNTPOINT, mountpoint) != 0)
1237                goto out;

1239        if (shared_fs && be_nvl_add_uint16(be_attrs, BE_ATTR_MOUNT_FLAGS,
1240            mount_flags) != 0)
1241                goto out;

1243        err = be_mount(be_attrs);

1245        switch (err) {
1246        case BE_SUCCESS:
1247                (void) printf(_("Mounted successfully on: '%s'\n"), mountpoint);
1248                break;
1249        case BE_ERR_BE_NOENT:
1226                err = 1;
1250                (void) fprintf(stderr, _("%s does not exist or appear "
1251                    "to be a valid BE.\nPlease check that the name of "
1252                    "the BE provided is correct.\n"), obe_name);
1253                break;
1254        case BE_ERR_MOUNTED:
1255                (void) fprintf(stderr, _("%s is already mounted.\n"
1256                    "Please unmount the BE before mounting it again.\n"),
1257                    obe_name);
1258                break;
1259        case BE_ERR_PERM:
1260        case BE_ERR_ACCESS:
1238                err = 1;
1261                (void) fprintf(stderr, _("Unable to mount %s.\n"), obe_name);
1262                (void) fprintf(stderr, _("You have insufficient privileges to "
1263                    "execute this command.\n"));
1264                break;
1265        default:
1244                err = 1;
1266                (void) fprintf(stderr, _("Unable to mount %s.\n"), obe_name);
1267                (void) fprintf(stderr, "%s\n", be_err_to_str(err));
1268        }
```

```
1270 out:
1271         if (tmp_mp != NULL)
1272                 free(tmp_mp);
1273         nvlist_free(be_attrs);
1274         return (err);
1275 }

1277 static int
1278 be_do_unmount(int argc, char **argv)
1279 {
1280         nvlist_t        *be_attrs;
1281         char            *obe_name;
1282         int             err = 1;
1283         int             c;
1284         int             unmount_flags = 0;

1286         while ((c = getopt(argc, argv, "fv")) != -1) {
1265         while ((c = getopt(argc, argv, "f")) != -1) {
1287                 switch (c) {
1288                 case 'f':
1289                         unmount_flags |= BE_UNMOUNT_FLAG_FORCE;
1290                         break;
1291                 case 'v':
1292                         libbe_print_errors(B_TRUE);
1293                         break;
1294                 default:
1295                         usage();
1296                         return (1);
1297                 }
1298         }

1300         argc -= optind;
1301         argv += optind;

1303         if (argc != 1) {
1304                 usage();
1305                 return (1);
1306         }

1308         obe_name = argv[0];

1310         if (be_nvl_alloc(&be_attrs) != 0)
1311                 return (1);


1314         if (be_nvl_add_string(be_attrs, BE_ATTR_ORIG_BE_NAME, obe_name) != 0)
1315                 goto out;

1317         if (be_nvl_add_uint16(be_attrs, BE_ATTR_UNMOUNT_FLAGS,
1318             unmount_flags) != 0)
1319                 goto out;

1321         err = be_unmount(be_attrs);

1323         switch (err) {
1324         case BE_SUCCESS:
1325                 (void) printf(_("Unmounted successfully\n"));
1326                 break;
1327         case BE_ERR_BE_NOENT:
1328                 (void) fprintf(stderr, _("%s does not exist or appear "
1329                     "to be a valid BE.\nPlease check that the name of "
1330                     "the BE provided is correct.\n"), obe_name);
1331                 break;
1332         case BE_ERR_UMOUNT_CURR_BE:
1333                 (void) fprintf(stderr, _("%s is the currently active BE.\n"
```



```
1334                     "It cannot be unmounted unless another BE is the "
1335                     "currently active BE.\n"), obe_name);
1336                 break;
1337         case BE_ERR_UMOUNT_SHARED:
1338                 (void) fprintf(stderr, _("%s is a shared file system and it "
1339                     "cannot be unmounted.\n"), obe_name);
1340                 break;
1341         case BE_ERR_PERM:
1342         case BE_ERR_ACCESS:
1343                 (void) fprintf(stderr, _("Unable to unmount %s.\n"), obe_name);
1344                 (void) fprintf(stderr, _("You have insufficient privileges to "
1345                     "execute this command.\n"));
1346                 break;
1347         default:
1348                 (void) fprintf(stderr, _("Unable to unmount %s.\n"), obe_name);
1349                 (void) fprintf(stderr, "%s\n", be_err_to_str(err));
1350         }

1352 out:
1353         nvlist_free(be_attrs);
1354         return (err);
1355 }

1357 static int
1358 be_do_rename(int argc, char **argv)
1359 {
1360         nvlist_t        *be_attrs;
1361         char            *obe_name;
1362         char            *nbe_name;
1363         int err = 1;
1364         int c;

1366         while ((c = getopt(argc, argv, "v")) != -1) {
1367                 switch (c) {
1368                 case 'v':
1369                         libbe_print_errors(B_TRUE);
1370                         break;
1371                 default:
1372                         usage();
1373                         return (1);
1374                 }
1375         }

1377         argc -= optind;
1378         argv += optind;

1380         if (argc != 2) {
1381                 usage();
1382                 return (1);
1383         }

1385         obe_name = argv[0];
1386         nbe_name = argv[1];

1388         if (be_nvl_alloc(&be_attrs) != 0)
1389                 return (1);

1391         if (be_nvl_add_string(be_attrs, BE_ATTR_ORIG_BE_NAME, obe_name) != 0)
1392                 goto out;

1394         if (be_nvl_add_string(be_attrs, BE_ATTR_NEW_BE_NAME, nbe_name) != 0)
1395                 goto out;

1397         err = be_rename(be_attrs);

1399         switch (err) {
```

```
1400         case BE_SUCCESS:
1401                 (void) printf(_("Renamed successfully\n"));
1402                 break;
1403         case BE_ERR_BE_NOENT:
1404                 (void) fprintf(stderr, _("%s does not exist or appear "
1405                     "to be a valid BE.\nPlease check that the name of "
1406                     "the BE provided is correct.\n"), obe_name);
1407                 break;
1408         case BE_ERR_PERM:
1409         case BE_ERR_ACCESS:
1410                 (void) fprintf(stderr, _("Rename of BE %s failed.\n"),
1411                     obe_name);
1412                 (void) fprintf(stderr, _("You have insufficient privileges to "
1413                     "execute this command.\n"));
1414                 break;
1415         default:
1416                 (void) fprintf(stderr, _("Rename of BE %s failed.\n"),
1417                     obe_name);
1418                 (void) fprintf(stderr, "%s\n", be_err_to_str(err));
1419         }

1421 out:
1422         nvlist_free(be_attrs);
1423         return (err);
1424 }

1426 static int
1427 be_do_rollback(int argc, char **argv)
1428 {
1429         nvlist_t        *be_attrs;
1430         char            *obe_name;
1431         char            *snap_name;
1432         int             err = 1;
1433         int             c;

1435         while ((c = getopt(argc, argv, "v")) != -1) {
1436                 switch (c) {
1437                 case 'v':
1438                         libbe_print_errors(B_TRUE);
1439                         break;
1440                 default:
1441                         usage();
1442                         return (1);
1443                 }
1444         }

1446         argc -= optind;
1447         argv += optind;

1449         if (argc < 1 || argc > 2) {
1450                 usage();
1451                 return (1);
1452         }

1454         obe_name = argv[0];
1455         if (argc == 2)
1456                 snap_name = argv[1];
1457         else { /* argc == 1 */
1458                 if ((snap_name = strrchr(obe_name, '@')) != NULL) {
1459                         if (snap_name[1] == '\0') {
1460                                 usage();
1461                                 return (1);
1462                         }

1464                         snap_name[0] = '\0';
1465                         snap_name++;
```



```
1466                 } else {
1467                         usage();
1468                         return (1);
1469                 }
1470         }

1472         if (be_nvl_alloc(&be_attrs) != 0)
1473                 return (1);

1475         if (be_nvl_add_string(be_attrs, BE_ATTR_ORIG_BE_NAME, obe_name) != 0)
1476                 goto out;

1478         if (be_nvl_add_string(be_attrs, BE_ATTR_SNAP_NAME, snap_name) != 0)
1479                 goto out;

1481         err = be_rollback(be_attrs);

1483         switch (err) {
1484         case BE_SUCCESS:
1485                 (void) printf(_("Rolled back successfully\n"));
1486                 break;
1487         case BE_ERR_BE_NOENT:
1488                 (void) fprintf(stderr, _("%s does not exist or appear "
1489                     "to be a valid BE.\nPlease check that the name of "
1490                     "the BE provided is correct.\n"), obe_name);
1491                 break;
1492         case BE_ERR_SS_NOENT:
1493                 (void) fprintf(stderr, _("%s does not exist or appear "
1494                     "to be a valid snapshot.\nPlease check that the name of "
1495                     "the snapshot provided is correct.\n"), snap_name);
1496                 break;
1497         case BE_ERR_PERM:
1498         case BE_ERR_ACCESS:
1499                 (void) fprintf(stderr, _("Rollback of BE %s snapshot %s "
1500                     "failed.\n"), obe_name, snap_name);
1501                 (void) fprintf(stderr, _("You have insufficient privileges to "
1502                     "execute this command.\n"));
1503                 break;
1504         default:
1505                 (void) fprintf(stderr, _("Rollback of BE %s snapshot %s "
1506                     "failed.\n"), obe_name, snap_name);
1507                 (void) fprintf(stderr, "%s\n", be_err_to_str(err));
1508         }

1510 out:
1511         nvlist_free(be_attrs);
1512         return (err);
1513 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   14405 Wed Jul 18 21:07:51 2012
new/usr/src/man/man1m/beadm.1m
2447 beadm should be more descriptive about some errors
**********************************************************
'\" te
.\" Copyright 2012 Nexenta Systems, Inc. All rights reserved.
.TH BEADM 1M "Feb 26, 2011"
.SH NAME
beadm \- utility for managing zfs boot environments
.SH SYNOPSIS
.LP
.nf
\fBbeadm\fR \fBcreate\fR [\fB-a\fR] [\fB-d\fR \fIdescription\fR]
\fBbeadm\fR create [\fB-a\fR] [\fB-d\fR \fIdescription\fR]
     [\fB-e\fR \fInon-activeBeName\fR | \fIbeName@snapshot\fR]
     [\fB-o\fR \fIproperty=value\fR] ... [\fB-p\fR \fIzpool\fR]
     [\fB-v\fR] \fIbeName\fR
     [\fB-o\fR \fIproperty=value\fR] ... [\fB-p\fR \fIzpool\FI] \fIbeName\fR
.fi

.LP
.nf
\fBbeadm\fR \fBcreate\fR [\fB-v\fR] \fIbeName@snapshot\fR
\fBbeadm\fR \fBcreate\fR \fIbeName@snapshot\fR
.fi

.LP
.nf
\fBbeadm\fR \fBdestroy\fR [\fB-fFsv\fR] \fIbeName\fR | \fIbeName@snapshot\fR
\fBbeadm\fR \fBdestroy\fR [\fB-fFs\fR] \fIbeName\fR | \fIbeName@snapshot\fR
.fi

.LP
.nf
\fBbeadm\fR \fBlist\fR [\fB-a\fR | \fB-ds\fR] [\fB-H\fR] [\fB-v\fR] [\fIbeName\f
\fBbeadm\fR \fBlist\fR [\fB-a\fR | \fB-ds\fR] [\fB-H\fR] [\fIbeName\fR]
.fi

.LP
.nf
\fBbeadm\fR \fBmount\fR [\fB-v\fR] \fIbeName\fR \fImountpoint\fR
\fBbeadm\fR \fBmount\fR \fIbeName\fR \fImountpoint\fR
.fi

.LP
.nf
\fBbeadm\fR \fBunmount\fR [\fB-fv\fR] \fIbeName\fR | \fImountpoint\fR
\fBbeadm\fR \fBunmount\fR [\fB-f\fR] \fIbeName\fR | \fImountpoint\fR
.fi

.LP
.nf
\fBbeadm\fR \fBrename\fR [\fB-v\fR] \fIbeName\fR \fInewBeName\fR
\fBbeadm\fR \fBrename\fR \fIbeName\fR \fInewBeName\fR
.fi

.LP
.nf
\fBbeadm\fR \fBactivate\fR [\fB-v\fR] \fIbeName\fR
\fBbeadm\fR \fBactivate\fR \fIbeName\fR
.fi

.LP
.nf
\fBbeadm\fR \fBrollback\fR [\fB-v\fR] \fIbeName\fR \fIsnapshot\fR
```



```
\fBbeadm\fR \fBrollback\fR \fIbeName\fR \fIsnapshot\fR
.fi

.LP
.nf
\fBbeadm\fR \fBrollback\fR [\fB-v\fR] \fIbeName@snapshot\fR
\fBbeadm\fR \fBrollback\fR \fIbeName@snapshot\fR
.fi

.SH DESCRIPTION
The \fBbeadm\fR command is the user interface for managing zfs Boot
Environments (BEs). This utility is intended to be used by System
Administrators who want to manage multiple Solaris Instances on a single
system.
.sp
The \fBbeadm\fR command supports the following operations:
.RS +4
.TP
.ie t \(bu
.el -
Create a new BE, based on the active BE.
.RE
.RS +4
.TP
.ie t \(bu
.el -
Create a new BE, based on an inactive BE.
.RE
.RS +4
.TP
.ie t \(bu
.el -
Create a snapshot of an existing BE.
.RE
.RS +4
.TP
.ie t \(bu
.el -
Create a new BE, based on an existing snapshot.
.RE
.RS +4
.TP
.ie t \(bu
.el -
Create a new BE, and copy it to a different zpool.
.RE
.RS +4
.TP
.ie t \(bu
.el -
Activate an existing, inactive BE.
.RE
.RS +4
.TP
.ie t \(bu
.el -
Mount a BE.
.RE
.RS +4
.TP
.ie t \(bu
.el -
Unmount a BE.
.RE
.RS +4
.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
163 [\fB-e\fR \fInon-activeBeName\fR | \fIbeName@snapshot\fR]
164          [\fB-o\fR
165 \fIproperty=value\fR] ... [\fB-p\fR \fIzpool\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 .RE
243 .sp
244 .ne 2
245 .na
```

```
246 \fB-v\fR
247 .ad
248 .sp .6
249 .RS 4n
250 Verbose mode. Displays verbose error messages from \fBbeadm\fR.
251 .RE
252 .RE

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

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



```
310 .ad
311 .sp .6
312 .RS 4n
313 Destroy all snapshots of the boot
314 environment.
315 .RE
316 .sp
317 .ne 2
318 .na
319 \fB-v\fR
320 .ad
321 .sp .6
322 .RS 4n
323 Verbose mode. Displays verbose error messages from \fBbeadm\fR.
324 .RE
325 .RE

327 .sp
328 .ne 2
329 .na
330 \fBbeadm\fR \fBlist\fR [\fB-a\fR | \fB-ds\fR] [\fB-H\fR] [\fB-v\fR] [\fIbeName\f
302 \fBbeadm\fR \fBlist\fR [\fB-a\fR | \fB-ds\fR] [\fB-H\fR] [\fIbeName\fR]
331 .ad
332 .sp .6
333 .RS 4n
334 Lists information about the existing boot environment named \fIbeName\fR, or
335 lists
336 information for all boot environments if \fIbeName\fR is not provided.
337 The 'Active'
338 field indicates whether the boot environment is active now,
339 represented
340 by 'N'; active on reboot, represented by 'R'; or both, represented
341 by 'NR'.
342 .sp
343 Each line in the machine parasable output has the boot environment name as the
344 first field.  The 'Space' field is displayed in bytes and the 'Created' field
345 is displayed in UTC format.  The \fB-H\fR option used with no other options
346 gives
347 the boot environment's uuid in the second field.  This field will be
348 blank if
349 the boot environment does not have a uuid.  See the EXAMPLES section.
350 .sp
351 .ne 2
352 .na
353 \fB-a\fR
354 .ad
355 .sp .6
356 .RS 4n
357 Lists all available information about the boot environment.  This includes
358 subordinate file systems and snapshots.
359 .RE
360 .sp
361 .ne 2
362 .na
363 \fB-d\fR
364 .ad
365 .sp .6
366 .RS 4n
367 Lists information about all subordinate file systems belonging to the boot
368 environment.
369 .RE
370 .sp
371 .ne 2
372 .na
373 \fB-s\fR
374 .ad
```

```
.sp .6
.RS 4n
Lists information about the snapshots of the boot environment.
.RE
.sp
.ne 2
.na
\fB-H\fR
.ad
.sp .6
.RS 4n
Do not list header information.  Each field in the list information is
separated by a semicolon.
.RE
.sp
.ne 2
.na
\fB-v\fR
.ad
.sp .6
.RS 4n
Verbose mode. Displays verbose error messages from \fBbeadm\fR.
.RE
.RE

.sp
.ne 2
.na
\fBbeadm\fR \fBmount\fR [\fB-v\fR] \fIbeName\fR \fImountpoint\fR
\fBbeadm\fR \fBmount\fR \fIbeName\fR \fImountpoint\fR
.ad
.sp .6
.RS 4n
Mounts a boot environment named beName at mountpoint.  mountpoint must be an
already existing empty directory.
.sp
.ne 2
.na
\fB-v\fR
.ad
.sp .6
.RS 4n
Verbose mode. Displays verbose error messages from \fBbeadm\fR.
.RE
.RE

.sp
.ne 2
.na
\fBbeadm\fR \fBunmount\fR [\fB-fv\fR] \fIbeName\fR | \fImountpoint\fR
\fBbeadm\fR \fBunmount\fR [\fB-f\fR] \fIbeName\fR | \fImountpoint\fR
.ad
.sp .6
.RS 4n
Unmounts the boot environment named beName. The command can also be given a path
beName mount point on the system.
.sp
.ne 2
.na
\fB-f\fR
.ad
.sp .6
.RS 4n
Forcefully unmount the boot environment even if its currently busy.
.RE
.sp
```



```
.ne 2
.na
\fB-v\fR
.ad
.sp .6
.RS 4n
Verbose mode. Displays verbose error messages from \fBbeadm\fR.
.RE
.RE

.sp
.ne 2
.na
\fBbeadm\fR \fBrename\fR [\fB-v\fR] \fIbeName\fR \fInewBeName\fR
\fBbeadm\fR \fBrename\fR \fIbeName\fR \fInewBeName\fR
.ad
.sp .6
.RS 4n
Renames the boot environment named \fIbeName\fR to \fInewBeName\fR.
.sp
.ne 2
.na
\fB-v\fR
.ad
.sp .6
.RS 4n
Verbose mode. Displays verbose error messages from \fBbeadm\fR.
.RE
.RE

.sp
.ne 2
.na
\fBbeadm\fR \fBrollback\fR [\fB-v\fR] \fIbeName\fR \fIsnapshot\fR | \fIbeName@sn
\fBbeadm\fR \fBrollback\fR \fIbeName\fR \fIsnapshot\fR | \fIbeName@snapshot\fR
.ad
.sp .6
.RS 4n
Roll back the boot environment named \fIbeName\fR to existing snapshot
of the boot environment named \fIbeName@snapshot\fR.
.sp
.ne 2
.na
\fB-v\fR
.ad
.sp .6
.RS 4n
Verbose mode. Displays verbose error messages from \fBbeadm\fR.
.RE
.RE

.sp
.ne 2
.na
\fBbeadm\fR \fBactivate\fR [\fB-v\fR] \fIbeName\fR
\fBbeadm\fR \fBactivate\fR \fIbeName\fR
.ad
.sp .6
.RS 4n
Makes beName the active BE on next reboot.
.sp
.ne 2
.na
\fB-v\fR
.ad
.sp .6
```

```
.RS 4n
Verbose mode. Displays verbose error messages from \fBbeadm\fR.
.RE
.RE

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

.SH EXAMPLES
.LP
\fBExample 1\fR: Create a new BE named BE1, by cloning the current live BE.
.sp
.in +2
.nf
\fB# beadm create BE1\fR
.fi
.in -2
.sp

.LP
\fBExample 2\fR: Create a new BE named BE2, by cloning the existing inactive
BE
named BE1.
.sp
.in +2
.nf
\fB# beadm create -e BE1 BE2\fR
.fi
.in -2
.sp

.LP
\fBExample 3\fR: Create a snapshot named now of the existing BE named BE1.
.sp
.in +2
.nf
\fB# beadm create BE1@now\fR
.fi
.in -2
.sp

.LP
\fBExample 4\fR: Create a new BE named BE3, by cloning an existing snapshot of
BE1.
.sp
.in +2
.nf
\fB# beadm create -e BE1@now BE3\fR
.fi
.in -2
.sp

.LP
\fBExample 5\fR: Create a new BE named BE4 based on the currently running BE.
Create the new BE in rpool2.
.sp
.in +2
.nf
\fB# beadm create -p rpool2 BE4\fR
.fi
.in -2
.sp
```



```
.LP
\fBExample 6\fR: Create a new BE named BE5 based on the currently running BE.
Create the new BE in rpool2, and create its datasets with compression turned
on.
.sp
.in +2
.nf
\fB# beadm create -p rpool2 -o compression=on BE5\fR
.fi
.in -2
.sp

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

.LP
\fBExample 8\fR: Activate an existing, inactive BE named BE3.
.sp
.in +2
.nf
\fB# beadm activate BE3\fR
.fi
.in -2
.sp

.LP
\fBExample 9\fR: Mount the BE named BE3 at /mnt.
.sp
.in +2
.nf
\fB# beadm mount BE3 /mnt\fR
.fi
.in -2
.sp

.LP
\fBExample 10\fR: Unmount the mounted BE named BE3.
.sp
.in +2
.nf
\fB# beadm unmount BE3\fR
.fi
.in -2
.sp

.LP
\fBExample 11\fR: Destroy the BE named BE3 without verification.
.sp
.in +2
.nf
\fB# beadm destroy -f BE3\fR
.fi
.in -2
.sp

.LP
\fBExample 12\fR: Destroy the snapshot named now of BE1.
.sp
```

```
.in +2
.nf
\fB# beadm destroy BE1@now\fR
.fi
.in -2
.sp

.LP
\fBExample 13\fR: Rename the existing, inactive BE named BE1 to BE3.
.sp
.in +2
.nf
\fB# beadm rename BE1 BE3\fR
.fi
.in -2
.sp

.LP
\fBExample 14\fR: Roll back the BE named BE1 to snapshot BE1@now.
.sp
.in +2
.nf
\fB# beadm rollback BE1 BE1@now\fR
.fi
.in -2
.sp

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

.sp
.in +2
.nf
\fB# beadm list\fR
BE  Active Mountpoint Space  Policy Created
--  ------ ---------- -----  ------ -------
BE2 -      -          72.0K  static 2008-05-21 12:26
BE3 -      -          332.0K static 2008-08-26 10:28
BE4 -      -          15.78M static 2008-09-05 18:20
BE5 NR     /          7.25G  static 2008-09-09 16:53
.fi
.in -2
.sp

.LP
\fBExample 16\fR: List all existing boot environmets and list all dataset and
snapshot information about those bootenvironments.

.sp
.in +2
.nf
\fB# beadm list -d -s\fR

BE/Dataset/Snapshot     Active Mountpoint Space   Policy Created
-------------------     ------ ---------- -----   ------ -------
BE2
   p/ROOT/BE2           -      -          36.0K   static 2008-05-21 12:26
   p/ROOT/BE2/opt       -      -          18.0K   static 2008-05-21 16:26
   p/ROOT/BE2/opt@now   -      -          0       static 2008-09-08 22:43
   p/ROOT/BE2@now       -      -          0       static 2008-09-08 22:43
BE3
   p/ROOT/BE3           -      -          192.0K  static 2008-08-26 10:28
   p/ROOT/BE3/opt       -      -          86.0K   static 2008-08-26 10:28
   p/ROOT/BE3/opt/local -      -          36.0K   static 2008-08-28 10:58
BE4
   p/ROOT/BE4           -      -          15.78M  static 2008-09-05 18:20
```



```
BE5
   p/ROOT/BE5           NR     /          6.10G   static 2008-09-09 16:53
   p/ROOT/BE5/opt       -      /opt       24.55M  static 2008-09-09 16:53
   p/ROOT/BE5/opt@bar   -      -          18.38M  static 2008-09-10 00:59
   p/ROOT/BE5/opt@foo   -      -          18.38M  static 2008-06-10 16:37
   p/ROOT/BE5@bar       -      -          139.44M static 2008-09-10 00:59
   p/ROOT/BE5@foo       -      -          912.85M static 2008-06-10 16:37
.fi
.in -2
.sp

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

.sp
.in +2
.nf
\fB# beadm list -a BE5\fR

BE/Dataset/Snapshot   Active Mountpoint Space   Policy Created
-------------------   ------ ---------- -----   ------ -------
BE5
   p/ROOT/BE5         NR     /          6.10G   static 2008-09-09 16:53
   p/ROOT/BE5/opt     -      /opt       24.55M  static 2008-09-09 16:53
   p/ROOT/BE5/opt@bar -      -          18.38M  static 2008-09-10 00:59
   p/ROOT/BE5/opt@foo -      -          18.38M  static 2008-06-10 16:37
   p/ROOT/BE5@bar     -      -          139.44M static 2008-09-10 00:59
   p/ROOT/BE5@foo     -      -          912.85M static 2008-06-10 16:37
.fi
.in -2
.sp

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

.sp
.in +2
.nf
\fB# beadm list -H\fR

BE2;;;;55296;static;1211397974
BE3;;;;339968;static;1219771706
BE4;;;;16541696;static;1220664051
BE5;215b8387-4968-627c-d2d0-f4a011414bab;NR;/;7786206208;static;1221004384
.fi
.in -2
.sp

.SH EXIT STATUS
.sp
.LP
The following exit values are returned:
.sp
.ne 2
.na
\fB0\fR
.ad
.sp .6
.RS 4n
Successful completion
.RE

.sp
.ne 2
.na
\fB>0\fR
```

```
 766 .ad
 767 .sp .6
 768 .RS 4n
 769 Failure
 770 .RE


 773 .SH FILES
 774 .sp
 775 .LP
 776 .sp
 777 .ne 2
 778 .na
 779 \fB/var/log/beadm/<beName>/create.log.<yyyymmdd_hhmmss>\fR
 780 .ad
 781 .sp .6
 782 .RS 4n
 783 Log used for capturing beadm create output
 784 .sp
 785 .nf
 786 \fIyyyymmdd_hhmmss\fR - 20071130_140558
 787 \fIyy\fR - year; 2007
 788 \fImm\fR - month; 11
 789 \fIdd\fR - day; 30
 790 \fIhh\fR - hour; 14
 791 \fImm\fR - minute; 05
 792 \fIss\fR - second; 58
 793 .fi
 794 .in -2
 795 .sp
 796 .RE
 797 .sp
 798 .LP
 799 .sp
 800 .ne 2
 801 .na
 802 \fB/etc/default/be\fR
 803 .ad
 804 .sp .6
 805 .RS 4n
 806 Contains default value for BENAME_STARTS_WITH parameter
 807 .sp
 808 .RE

 810 .SH ATTRIBUTES
 811 .sp
 812 .LP
 813 See \fBattributes\fR(5) for descriptions of the  following  attributes:
 814 .sp

 816 .sp
 817 .TS
 818 box;
 819 c | c
 820 l | l .
 821 ATTRIBUTE TYPE  ATTRIBUTE VALUE
 822 _
 823 Interface Stability     Uncommitted
 824 .TE


 827 .SH SEE ALSO
 828 .sp
 829 .LP
 830 .BR zfs (1M)
```