```
**********************************************************
   34506 Sat Nov 23 11:10:38 2013
new/usr/src/cmd/beadm/beadm.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_

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 repsonsibility 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_INVAL);

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;
385                 if ((ret = be_mount_zones(zhp, &md)) != BE_SUCCESS) {
386                         (void) _be_unmount(bt.obe_name, 0);
387                         if (gen_tmp_altroot)
388                                 free(tmp_altroot);
389                         ZFS_CLOSE(zhp);
390                         return (ret);
387                 }
388         }

390         ZFS_CLOSE(zhp);

392         /*
393          * If a NULL altroot was passed in, pass the generated altroot
394          * back to the caller in altroot.
395          */
396         if (gen_tmp_altroot) {
397                 if (ret == BE_SUCCESS || ret == BE_ERR_NO_MOUNTED_ZONE)
400         if (gen_tmp_altroot)
398                         *altroot = tmp_altroot;
399                 else
400                         free(tmp_altroot);
401         }

403         return (ret);
403         return (BE_SUCCESS);
404 }
```

*unchanged_portion_omitted_*

```
1712 /*
1713  * Function:    loopback_mount_zonepath
```



```
1714  * Description: This function loopback mounts a zonepath into the altroot
1715  *              area of the BE being mounted.
1715  *              area of the BE being mounted.  Since these are shared file
1716  *              systems, they are expected to be already mounted for the
1717  *              current BE, and this function just loopback mounts them into
1718  *              the BE mountpoint.
1716  * Parameters:
1717  *              zonepath - pointer to zone path in the current BE
1718  *              md - be_mount_data_t pointer
1719  * Returns:
1720  *              BE_SUCCESS - Success
1721  *              be_errno_t - Failure
1722  * Scope:
1723  *              Private
1724  */
1725 static int
1726 loopback_mount_zonepath(const char *zonepath, be_mount_data_t *md)
1727 {
1728         FILE            *fp = (FILE *)NULL;
1729         struct stat     st;
1730         char            *p;
1731         char            *p1;
1732         char            *parent_dir;
1733         struct extmnttab        extmtab;
1734         dev_t           dev = NODEV;
1735         char            *parentmnt;
1736         char            alt_parentmnt[MAXPATHLEN];
1737         struct mnttab   mntref;
1738         char            altzonepath[MAXPATHLEN];
1739         char            optstr[MAX_MNTOPT_STR];
1740         int             mflag = MS_OPTIONSTR;
1741         int             ret;
1742         int             err;

1744         fp = fopen(MNTTAB, "r");
1745         if (fp == NULL) {
1746                 err = errno;
1747                 be_print_err(gettext("loopback_mount_zonepath: "
1748                     "failed to open /etc/mnttab\n"));
1749                 return (errno_to_be_err(err));
1750         }

1752         /*
1753          * before attempting the loopback mount of zonepath under altroot,
1754          * we need to make sure that all intermediate file systems in the
1755          * zone path are also mounted under altroot
1756          */

1758         /* get the parent directory for zonepath */
1759         p = strrchr(zonepath, '/');
1760         if (p != NULL && p != zonepath) {
1761                 if ((parent_dir = (char *)calloc(sizeof (char),
1762                     p - zonepath + 1)) == NULL) {
1763                         ret = BE_ERR_NOMEM;
1764                         goto done;
1765                 }
1766                 (void) strlcpy(parent_dir, zonepath, p - zonepath + 1);
1767                 if (stat(parent_dir, &st) < 0) {
1768                         ret = errno_to_be_err(errno);
1769                         be_print_err(gettext("loopback_mount_zonepath: "
1770                             "failed to stat %s"),
1771                             parent_dir);
1772                         free(parent_dir);
1773                         goto done;
1774                 }
1775                 free(parent_dir);
```

```
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                 */

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                                p1 = strchr(extmtab.mnt_special, '/');
1789                                if (p1 == NULL || strncmp(p1 + 1,
1790                                    BE_CONTAINER_DS_NAME, 4) != 0 ||
1791                                    (*(p1 + 5) != '/' && *(p1 + 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        }


1827        if (!md->shared_rw) {
1828                mflag |= MS_RDONLY;
1829        }

1831        (void) snprintf(altzonepath, sizeof (altzonepath), "%s%s",
1832            md->altroot, zonepath);

1834        /* Add the "nosub" option to the mount options string */
1835        (void) strlcpy(optstr, MNTOPT_NOSUB, sizeof (optstr));

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;

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/beadm.1m
4220 beadm mount of old BE with zones fails "Read only filesystem"
4235 beadm(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"
   3 .TH BEADM 1M "Jul 25, 2013"
   4 .SH NAME
   5 beadm \- utility for managing zfs boot environments
   6 .SH SYNOPSIS
   7 .LP
   8 .nf
   9 \fBbeadm\fR \fBcreate\fR [\fB-a\fR] [\fB-d\fR \fIdescription\fR]
  10      [\fB-e\fR \fInon-activeBeName\fR | \fIbeName@snapshot\fR]
  11      [\fB-o\fR \fIproperty=value\fR] ... [\fB-p\fR \fIzpool\fR]
  12      [\fB-v\fR] \fIbeName\fR
  13 .fi

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

  20 .LP
  21 .nf
  22 \fBbeadm\fR \fBdestroy\fR [\fB-fFsv\fR] \fIbeName\fR | \fIbeName@snapshot\fR
  23 .fi

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

  30 .LP
  31 .nf
  32 \fBbeadm\fR \fBmount\fR [\fB-s\fR \fBro\fR|\fBrw\fR] [\fB-v\fR] \fIbeName\fR \fI
  32 \fBbeadm\fR \fBmount\fR [\fB-v\fR] \fIbeName\fR \fImountpoint\fR
  33 .fi

  35 .LP
  36 .nf
  37 \fBbeadm\fR \fBunmount\fR [\fB-fv\fR] \fIbeName\fR | \fImountpoint\fR
  38 .fi

  40 .LP
  41 .nf
  42 \fBbeadm\fR \fBrename\fR [\fB-v\fR] \fIbeName\fR \fInewBeName\fR
  43 .fi

  45 .LP
  46 .nf
  47 \fBbeadm\fR \fBactivate\fR [\fB-v\fR] \fIbeName\fR
  48 .fi

  50 .LP
  51 .nf
  52 \fBbeadm\fR \fBrollback\fR [\fB-v\fR] \fIbeName\fR \fIsnapshot\fR
  53 .fi

  55 .LP
  56 .nf
  57 \fBbeadm\fR \fBrollback\fR [\fB-v\fR] \fIbeName@snapshot\fR
  58 .fi
```



```
  60 .SH DESCRIPTION
  61 The \fBbeadm\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 \fBbeadm\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 zpool.
  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
```

```
.na
\fBbeadm\fR \fBcreate\fR [\fB-v\fR] \fIbeName@snapshot\fR
.ad
.sp .6
.RS 4n
Creates a snapshot of the existing BE named beName.
.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 \fBdestroy\fR [\fB-fFsv\fR] \fIbeName\fR | \fIbeName@snapshot\fR
.ad
.sp .6
.RS 4n
Destroys the boot environment named \fIbeName\fR or destroys an existing
snapshot of
the boot environment named \fIbeName@snapshot\fR.  Destroying a
boot environment
will also destroy all snapshots of that boot environment.  Use
this command
with caution.
.sp
.ne 2
.na
\fB-f\fR
.ad
.sp .6
.RS 4n
Forcefully unmount the boot environment if it is currently mounted.
.RE
.sp
.ne 2
.na
\fB-F\fR
.ad
.sp .6
.RS 4n
Force the action without prompting to verify the destruction of the boot
environment.
.RE
.sp
.ne 2
.na
\fB-s\fR
.ad
.sp .6
.RS 4n
Destroy all snapshots of the boot
environment.
.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 \fBlist\fR [\fB-a\fR | \fB-ds\fR] [\fB-H\fR] [\fB-v\fR] [\fIbeName\f
.ad
.sp .6
.RS 4n
Lists information about the existing boot environment named \fIbeName\fR, or
lists
information for all boot environments if \fIbeName\fR is not provided.
The 'Active'
field indicates whether the boot environment is active now,
represented
by 'N'; active on reboot, represented by 'R'; or both, represented
by 'NR'. In non-global zone the 'Active' field also indicates whether the
boot environment has a non-active parent BE, represented by 'x'; is active
on boot in a non-active parent BE, represented by 'b'. Activate, rollback
and snapshot operations for boot environments from non-active global parent
BE aren't supported, destroy is allowed if these boot environments aren't
active on boot.
.sp
Each line in the machine parasable output has the boot environment name as the
first field.  The 'Space' field is displayed in bytes and the 'Created' field
is displayed in UTC format.  The \fB-H\fR option used with no other options
gives
the boot environment's uuid in the second field.  This field will be
blank if
the boot environment does not have a uuid. See the EXAMPLES section.
In non-global zones, this field shows the uuid of the parent BE.
.sp
.ne 2
.na
\fB-a\fR
.ad
.sp .6
.RS 4n
Lists all available information about the boot environment.  This includes
subordinate file systems and snapshots.
.RE
.sp
.ne 2
.na
\fB-d\fR
.ad
.sp .6
.RS 4n
Lists information about all subordinate file systems belonging to the boot
environment.
.RE
.sp
.ne 2
.na
\fB-s\fR
.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-s\fR \fBro\fR|\fBrw\fR] [\fB-v\fR] \fIbeName\fR \fI
\fIbeadm\fR \fImount\fR [\fB-v\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-s\fR \fBro\fR|\fBrw\fR
.ad
.sp .6
.RS 4n
Mount the shared filesystems of the BE in read-only or read-write mode.
.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 \fBunmount\fR [\fB-fv\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
.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
.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
.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
.ad
.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)
```