

```

*****
129807 Tue Apr 23 16:32:02 2013
new/usr/src/cmd/zpool/zpool_main.c
3745 zpool create should treat -O mountpoint and -m the same
Submitted by: Will Andrews <willa@spectrallogic.com>
Submitted by: Alan Somers <alans@spectrallogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
*****
_____unchanged_portion_omitted_____

617 /*
618 * zpool create [-fnd] [-o property=value] ...
619 * [-O file-system-property=value] ...
620 * [-R root] [-m mountpoint] <pool> <dev> ...
621 *
622 * -f Force creation, even if devices appear in use
623 * -n Do not create the pool, but display the resulting layout if it
624 * were to be created.
625 * -R Create a pool under an alternate root
626 * -m Set default mountpoint for the root dataset. By default it's
627 * '/<pool>'
628 * -o Set property=value.
629 * -d Don't automatically enable all supported pool features
630 * (individual features can be enabled with -o).
631 * -O Set fsproperty=value in the pool's root file system
632 *
633 * Creates the named pool according to the given vdev specification. The
634 * bulk of the vdev processing is done in get_vdev_spec() in zpool_vdev.c. Once
635 * we get the nvlist back from get_vdev_spec(), we either print out the contents
636 * (if '-n' was specified), or pass it to libzfs to do the creation.
637 */
638 int
639 zpool_do_create(int argc, char **argv)
640 {
641     boolean_t force = B_FALSE;
642     boolean_t dryrun = B_FALSE;
643     boolean_t enable_all_pool_feat = B_TRUE;
644     int c;
645     nvlist_t *nvroot = NULL;
646     char *poolname;
647     int ret = 1;
648     char *altroot = NULL;
649     char *mountpoint = NULL;
650     nvlist_t *fsprops = NULL;
651     nvlist_t *props = NULL;
652     char *propval;

654     /* check options */
655     while ((c = getopt(argc, argv, ":fndR:m:o:O:")) != -1) {
656         switch (c) {
657             case 'f':
658                 force = B_TRUE;
659                 break;
660             case 'n':
661                 dryrun = B_TRUE;
662                 break;
663             case 'd':
664                 enable_all_pool_feat = B_FALSE;
665                 break;
666             case 'R':
667                 altroot = optarg;
668                 if (add_prop_list(zpool_prop_to_name(
669                     ZPOOL_PROP_ALTROOT), optarg, &props, B_TRUE))
670                     goto errout;
671             if (nvlist_lookup_string(props,
672                 zpool_prop_to_name(ZPOOL_PROP_CACHEFILE),

```

```

673         &propval) == 0)
674             break;
675         if (add_prop_list(zpool_prop_to_name(
676             ZPOOL_PROP_CACHEFILE), "none", &props, B_TRUE))
677             goto errout;
678         break;
679     case 'm':
680         /* Equivalent to -O mountpoint=optarg */
681     #endif /* ! codereview */
682         mountpoint = optarg;
683         break;
684     case 'o':
685         if ((propval = strchr(optarg, '=')) == NULL) {
686             (void) fprintf(stderr, gettext("missing "
687                 "'=' for -o option\n"));
688             goto errout;
689         }
690         *propval = '\0';
691         propval++;

693         if (add_prop_list(optarg, propval, &props, B_TRUE))
694             goto errout;

696     /*
697     * If the user is creating a pool that doesn't support
698     * feature flags, don't enable any features.
699     */
700     if (zpool_name_to_prop(optarg) == ZPOOL_PROP_VERSION) {
701         char *end;
702         u_longlong_t ver;

704         ver = strtoull(propval, &end, 10);
705         if (*end == '\0' &&
706             ver < SPA_VERSION_FEATURES) {
707             enable_all_pool_feat = B_FALSE;
708         }
709     }
710     break;
711     case 'O':
712         if ((propval = strchr(optarg, '=')) == NULL) {
713             (void) fprintf(stderr, gettext("missing "
714                 "'=' for -O option\n"));
715             goto errout;
716         }
717         *propval = '\0';
718         propval++;

720     /*
721     * Mountpoints are checked and then added later.
722     * Uniquely among properties, they can be specified
723     * more than once, to avoid conflict with -m.
724     */
725     if (!strcmp(optarg,
726         zfs_prop_to_name(ZFS_PROP_MOUNTPOINT)))
727         mountpoint = propval;
728     #endif /* ! codereview */
729     if (add_prop_list(optarg, propval, &fsprops, B_FALSE))
730         goto errout;
731     break;
732     case ':':
733         (void) fprintf(stderr, gettext("missing argument for "
734             "'%c' option\n"), optopt);
735         goto badusage;
736     case '?':
737         (void) fprintf(stderr, gettext("invalid option '%c'\n"),
738             optopt);

```

```

739         goto badusage;
740     }
741 }
742
743     argc -= optind;
744     argv += optind;
745
746     /* get pool name and check number of arguments */
747     if (argc < 1) {
748         (void) fprintf(stderr, gettext("missing pool name argument\n"));
749         goto badusage;
750     }
751     if (argc < 2) {
752         (void) fprintf(stderr, gettext("missing vdev specification\n"));
753         goto badusage;
754     }
755
756     poolname = argv[0];
757
758     /*
759      * As a special case, check for use of '/' in the name, and direct the
760      * user to use 'zfs create' instead.
761      */
762     if (strchr(poolname, '/') != NULL) {
763         (void) fprintf(stderr, gettext("cannot create '%s': invalid "
764             "character '/' in pool name\n"), poolname);
765         (void) fprintf(stderr, gettext("use 'zfs create' to "
766             "create a dataset\n"));
767         goto errout;
768     }
769
770     /* pass off to get_vdev_spec for bulk processing */
771     nvroot = make_root_vdev(NULL, force, !force, B_FALSE, dryrun,
772         argc - 1, argv + 1);
773     if (nvroot == NULL)
774         goto errout;
775
776     /* make_root_vdev() allows 0 toplevel children if there are spares */
777     if (!zfs_allocatable_devs(nvroot)) {
778         (void) fprintf(stderr, gettext("invalid vdev "
779             "specification: at least one toplevel vdev must be "
780             "specified\n"));
781         goto errout;
782     }
783
784     if (altroot != NULL && altroot[0] != '/') {
785         (void) fprintf(stderr, gettext("invalid alternate root '%s': "
786             "must be an absolute path\n"), altroot);
787         goto errout;
788     }
789
790     /*
791      * Check the validity of the mountpoint and direct the user to use the
792      * '-m' mountpoint option if it looks like its in use.
793      */
794     if (mountpoint == NULL ||
795         (strcmp(mountpoint, ZFS_MOUNTPOINT_LEGACY) != 0 &&
796          strcmp(mountpoint, ZFS_MOUNTPOINT_NONE) != 0)) {
797         char buf[MAXPATHLEN];
798         DIR *dirp;
799
800         if (mountpoint && mountpoint[0] != '/') {
801             (void) fprintf(stderr, gettext("invalid mountpoint "
802                 "'%s': must be an absolute path, 'legacy', or "
803                 "'none'\n"), mountpoint);
804             goto errout;

```

```

805     }
806
807     if (mountpoint == NULL) {
808         if (altroot != NULL)
809             (void) snprintf(buf, sizeof (buf), "%s/%s",
810                 altroot, poolname);
811         else
812             (void) snprintf(buf, sizeof (buf), "%s",
813                 poolname);
814     } else {
815         if (altroot != NULL)
816             (void) snprintf(buf, sizeof (buf), "%s%s",
817                 altroot, mountpoint);
818         else
819             (void) snprintf(buf, sizeof (buf), "%s",
820                 mountpoint);
821     }
822
823     if ((dirp = opendir(buf)) == NULL && errno != ENOENT) {
824         (void) fprintf(stderr, gettext("mountpoint '%s' : "
825             "%s\n"), buf, strerror(errno));
826         (void) fprintf(stderr, gettext("use '-m' "
827             "option to provide a different default\n"));
828         goto errout;
829     } else if (dirp) {
830         int count = 0;
831
832         while (count < 3 && readdir(dirp) != NULL)
833             count++;
834         (void) closedir(dirp);
835
836         if (count > 2) {
837             (void) fprintf(stderr, gettext("mountpoint "
838                 "'%s' exists and is not empty\n"), buf);
839             (void) fprintf(stderr, gettext("use '-m' "
840                 "option to provide a "
841                 "different default\n"));
842             goto errout;
843         }
844     }
845 }
846
847     /*
848      * Now that the mountpoint's validity has been checked, ensure that
849      * the property is set appropriately prior to creating the pool.
850      */
851     if (mountpoint != NULL) {
852         ret = add_prop_list(zfs_prop_to_name(ZFS_PROP_MOUNTPOINT),
853             mountpoint, &fsprops, B_FALSE);
854         if (ret != 0)
855             goto errout;
856     }
857
858     ret = 1;
859 #endif /* ! codereview */
860     if (dryrun) {
861         /*
862          * For a dry run invocation, print out a basic message and run
863          * through all the vdevs in the list and print out in an
864          * appropriate hierarchy.
865          */
866         (void) printf(gettext("would create '%s' with the "
867             "following layout:\n\n"), poolname);
868
869         print_vdev_tree(NULL, poolname, nvroot, 0, B_FALSE);
870         if (num_logs(nvroot) > 0)

```

```

871         print_vdev_tree(NULL, "logs", nvroot, 0, B_TRUE);
873     ret = 0;
874 } else {
875     /*
876     * Hand off to libzfs.
877     */
878     if (enable_all_pool_feat) {
879         int i;
880         for (i = 0; i < SPA_FEATURES; i++) {
881             char propname[MAXPATHLEN];
882             zfeature_info_t *feat = &spa_feature_table[i];
884             (void) snprintf(propname, sizeof (propname),
885                 "feature%s", feat->fi_undef);
887             /*
888             * Skip feature if user specified it manually
889             * on the command line.
890             */
891             if (nvlist_exists(props, propname))
892                 continue;
894             ret = add_prop_list(propname,
895                 ZFS_FEATURE_ENABLED, &props, B_TRUE);
896             if (ret != 0)
897                 if (add_prop_list(propname, ZFS_FEATURE_ENABLED,
898                     &props, B_TRUE) != 0)
899                     goto errout;
900         }
901     if (zpool_create(g_zfs, poolname,
902         nvroot, props, fsprops) == 0) {
903         zfs_handle_t *pool = zfs_open(g_zfs, poolname,
904             ZFS_TYPE_FILESYSTEM);
905         if (pool != NULL) {
906             if (mountpoint != NULL)
907                 verify(zfs_prop_set(pool,
908                     zfs_prop_to_name(
909                         ZFS_PROP_MOUNTPOINT),
910                         mountpoint) == 0);
911             if (zfs_mount(pool, NULL, 0) == 0)
912                 ret = zfs_shareall(pool);
913             zfs_close(pool);
914         }
915     } else if (libzfs_errno(g_zfs) == EZFS_INVALIDNAME) {
916         (void) fprintf(stderr, gettext("pool name may have "
917             "been omitted\n"));
918     }
919 }
920 errout:
921 nvlist_free(nvroot);
922 nvlist_free(fsprops);
923 nvlist_free(props);
924 return (ret);
925 badusage:
926 nvlist_free(fsprops);
927 nvlist_free(props);
928 usage(B_FALSE);
929 return (2);
930 }

```

_____unchanged_portion_omitted_____

```

*****
102992 Tue Apr 23 16:32:02 2013
new/usr/src/lib/libzfs/common/libzfs_pool.c
3745 zpool create should treat -O mountpoint and -m the same
Submitted by: Will Andrews <willa@spectralogic.com>
Submitted by: Alan Somers <alans@spectralogic.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
*****
_____unchanged_portion_omitted_____

1070 /*
1071  * Create the named pool, using the provided vdev list. It is assumed
1072  * that the consumer has already validated the contents of the nvlist, so we
1073  * don't have to worry about error semantics.
1074  */
1075 int
1076 zpool_create(libzfs_handle_t *hdl, const char *pool, nvlist_t *nvroot,
1077             nvlist_t *props, nvlist_t *fsprops)
1078 {
1079     zfs_cmd_t zc = { 0 };
1080     nvlist_t *zc_fsprops = NULL;
1081     nvlist_t *zc_props = NULL;
1082     char msg[1024];
1083     char *altroot;
1084     int ret = -1;

1086     (void) snprintf(msg, sizeof(msg), dgettext(TEXT_DOMAIN,
1087         "cannot create '%s'", pool));

1089     if (!zpool_name_valid(hdl, B_FALSE, pool))
1090         return (zfs_error(hdl, EZFS_INVALIDNAME, msg));

1092     if (zcmd_write_conf_nvlist(hdl, &zc, nvroot) != 0)
1093         return (-1);

1095     if (props) {
1096         prop_flags_t flags = { .create = B_TRUE, .import = B_FALSE };

1098         if ((zc_props = zpool_valid_proplist(hdl, pool, props,
1099             SPA_VERSION_1, flags, msg)) == NULL) {
1100             goto create_failed;
1101         }
1102     }

1104     if (fsprops) {
1105         uint64_t zoned;
1106         char *zonestr;

1108         zoned = ((nvlist_lookup_string(fsprops,
1109             zfs_prop_to_name(ZFS_PROP_ZONED), &zonestr) == 0) &&
1110             strcmp(zonestr, "on") == 0);

1112         if ((zc_fsprops = zfs_valid_proplist(hdl,
1113             ZFS_TYPE_FILESYSTEM, fsprops, zoned, NULL, msg)) == NULL) {
1114             goto create_failed;
1115         }
1116     }
1117     if (!zc_props &&
1118         (nvlist_alloc(&zc_props, NV_UNIQUE_NAME, 0) != 0)) {
1119         goto create_failed;
1120     }
1121     if (nvlist_add_nvlist(zc_props,
1122         ZPOOL_ROOTFS_PROPS, zc_fsprops) != 0) {
1123         goto create_failed;
1124     }
}

```

```

1126     if (zc_props && zcmd_write_src_nvlist(hdl, &zc, zc_props) != 0)
1127         goto create_failed;

1129     (void) strncpy(zc.zc_name, pool, sizeof(zc.zc_name));

1131     if ((ret = zfs_ioctl(hdl, ZFS_IOC_POOL_CREATE, &zc)) != 0) {

1133         zcmd_free_nvlists(&zc);
1134         nvlist_free(zc_props);
1135         nvlist_free(zc_fsprops);

1137         switch (errno) {
1138             case EBUSY:
1139                 /*
1140                  * This can happen if the user has specified the same
1141                  * device multiple times. We can't reliably detect this
1142                  * until we try to add it and see we already have a
1143                  * label.
1144                  */
1145                 zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1146                     "one or more vdevs refer to the same device"));
1147                 return (zfs_error(hdl, EZFS_BADDEV, msg));

1149             case EOVERFLOW:
1150                 /*
1151                  * This occurs when one of the devices is below
1152                  * SPA_MINDEVSZ. Unfortunately, we can't detect which
1153                  * device was the problem device since there's no
1154                  * reliable way to determine device size from userland.
1155                  */
1156                 {
1157                     char buf[64];

1159                     zfs_nicenum(SPA_MINDEVSZ, buf, sizeof(buf));

1161                     zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1162                         "one or more devices is less than the "
1163                         "minimum size (%s)", buf));
1164                 }
1165                 return (zfs_error(hdl, EZFS_BADDEV, msg));

1167             case ENOSPC:
1168                 zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1169                     "one or more devices is out of space"));
1170                 return (zfs_error(hdl, EZFS_BADDEV, msg));

1172             case ENOTBLK:
1173                 zfs_error_aux(hdl, dgettext(TEXT_DOMAIN,
1174                     "cache device must be a disk or disk slice"));
1175                 return (zfs_error(hdl, EZFS_BADDEV, msg));

1177             default:
1178                 return (zpool_standard_error(hdl, errno, msg));
1179         }
1180     }

1182     /*
1183     * If this is an alternate root pool, then we automatically set the
1184     * mountpoint of the root dataset to be '/'.
1185     */
1186     if (nvlist_lookup_string(props, zpool_prop_to_name(ZPOOL_PROP_ALTROOT),
1187         &altroot) == 0) {
1188         zfs_handle_t *zhp;

1190         verify((zhp = zfs_open(hdl, pool, ZFS_TYPE_DATASET)) != NULL);
1191         verify(zfs_prop_set(zhp, zfs_prop_to_name(ZFS_PROP_MOUNTPOINT),

```

```
1192         "/" ) == 0);
1194         zfs_close(zhp);
1195     }
1182 create_failed:
1183     zcmd_free_nvlists(&zcmd);
1184     nvlist_free(zc_props);
1185     nvlist_free(zc_fsprops);
1186     return (ret);
1187 }
_____unchanged_portion_omitted_____
```