```
**********************************************************
  128994 Wed Aug  8 18:32:34 2012
new/usr/src/cmd/zpool/zpool_main.c
3064 usr/src/cmd/zpool/zpool_main.c misspells "successful"
**********************************************************
______unchanged_portion_omitted_

3146 /*
3147  * zpool split [-n] [-o prop=val] ...
3148  *             [-o mntopt] ...
3149  *             [-R altroot] <pool> <newpool> [<device> ...]
3150  *
3151  *      -n      Do not split the pool, but display the resulting layout if
3152  *              it were to be split.
3153  *      -o      Set property=value, or set mount options.
3154  *      -R      Mount the split-off pool under an alternate root.
3155  *
3156  * Splits the named pool and gives it the new pool name.  Devices to be split
3157  * off may be listed, provided that no more than one device is specified
3158  * per top-level vdev mirror.  The newly split pool is left in an exported
3159  * state unless -R is specified.
3160  *
3161  * Restrictions: the top-level of the pool pool must only be made up of
3162  * mirrors; all devices in the pool must be healthy; no device may be
3163  * undergoing a resilvering operation.
3164  */
3165 int
3166 zpool_do_split(int argc, char **argv)
3167 {
3168         char *srcpool, *newpool, *propval;
3169         char *mntopts = NULL;
3170         splitflags_t flags;
3171         int c, ret = 0;
3172         zpool_handle_t *zhp;
3173         nvlist_t *config, *props = NULL;

3175         flags.dryrun = B_FALSE;
3176         flags.import = B_FALSE;

3178         /* check options */
3179         while ((c = getopt(argc, argv, ":R:no:")) != -1) {
3180                 switch (c) {
3181                 case 'R':
3182                         flags.import = B_TRUE;
3183                         if (add_prop_list(
3184                             zpool_prop_to_name(ZPOOL_PROP_ALTROOT), optarg,
3185                             &props, B_TRUE) != 0) {
3186                                 if (props)
3187                                         nvlist_free(props);
3188                                 usage(B_FALSE);
3189                         }
3190                         break;
3191                 case 'n':
3192                         flags.dryrun = B_TRUE;
3193                         break;
3194                 case 'o':
3195                         if ((propval = strchr(optarg, '=')) != NULL) {
3196                                 *propval = '\0';
3197                                 propval++;
3198                                 if (add_prop_list(optarg, propval,
3199                                     &props, B_TRUE) != 0) {
3200                                         if (props)
3201                                                 nvlist_free(props);
3202                                         usage(B_FALSE);
3203                                 }
3204                         } else {
```



```
3205                                 mntopts = optarg;
3206                         }
3207                         break;
3208                 case ':':
3209                         (void) fprintf(stderr, gettext("missing argument for "
3210                             "'%c' option\n"), optopt);
3211                         usage(B_FALSE);
3212                         break;
3213                 case '?':
3214                         (void) fprintf(stderr, gettext("invalid option '%c'\n"),
3215                             optopt);
3216                         usage(B_FALSE);
3217                         break;
3218                 }
3219         }

3221         if (!flags.import && mntopts != NULL) {
3222                 (void) fprintf(stderr, gettext("setting mntopts is only "
3223                     "valid when importing the pool\n"));
3224                 usage(B_FALSE);
3225         }

3227         argc -= optind;
3228         argv += optind;

3230         if (argc < 1) {
3231                 (void) fprintf(stderr, gettext("Missing pool name\n"));
3232                 usage(B_FALSE);
3233         }
3234         if (argc < 2) {
3235                 (void) fprintf(stderr, gettext("Missing new pool name\n"));
3236                 usage(B_FALSE);
3237         }

3239         srcpool = argv[0];
3240         newpool = argv[1];

3242         argc -= 2;
3243         argv += 2;

3245         if ((zhp = zpool_open(g_zfs, srcpool)) == NULL)
3246                 return (1);

3248         config = split_mirror_vdev(zhp, newpool, props, flags, argc, argv);
3249         if (config == NULL) {
3250                 ret = 1;
3251         } else {
3252                 if (flags.dryrun) {
3253                         (void) printf(gettext("would create '%s' with the "
3254                             "following layout:\n\n"), newpool);
3255                         print_vdev_tree(NULL, newpool, config, 0, B_FALSE);
3256                 }
3257                 nvlist_free(config);
3258         }

3260         zpool_close(zhp);

3262         if (ret != 0 || flags.dryrun || !flags.import)
3263                 return (ret);

3265         /*
3266          * The split was successful. Now we need to open the new
3267          * pool and import it.
3268          */
3269         if ((zhp = zpool_open_canfail(g_zfs, newpool)) == NULL)
3270                 return (1);
```

```
3271         if (zpool_get_state(zhp) != POOL_STATE_UNAVAIL &&
3272             zpool_enable_datasets(zhp, mntopts, 0) != 0) {
3273                 ret = 1;
3274                 (void) fprintf(stderr, gettext("Split was successful, but "
3274                 (void) fprintf(stderr, gettext("Split was succssful, but "
3275                     "the datasets could not all be mounted\n"));
3276                 (void) fprintf(stderr, gettext("Try doing '%s' with a "
3277                     "different altroot\n"), "zpool import");
3278         }
3279         zpool_close(zhp);

3281         return (ret);
3282 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   35149 Wed Aug  8 18:32:34 2012
new/usr/src/uts/common/cpr/cpr_main.c
3067 Typo in spelling "succssful"
**********************************************************
______unchanged_portion_omitted_


/*
 * The main switching point for cpr, this routine starts the ckpt
 * and state file saving routines; on resume the control is
 * returned back to here and it then calls the resume routine.
 */
int
cpr_main(int sleeptype)
{
        int rc, rc2;
        label_t saveq;
        klwp_t *tlwp = ttolwp(curthread);

        if (sleeptype == CPR_TODISK) {
                if ((rc = cpr_default_setup(1)) != 0)
                        return (rc);
                ASSERT(tlwp);
                saveq = tlwp->lwp_qsav;
        }

        if (sleeptype == CPR_TORAM) {
                rc = cpr_suspend(sleeptype);
                PMD(PMD_SX, ("cpr_suspend rets %x\n", rc))
                if (rc == 0) {
                        int i_cpr_power_down(int sleeptype);

                        /*
                         * From this point on, we should be at a high
                         * spl, interrupts disabled, and all but one
                         * cpu's paused (effectively UP/single threaded).
                         * So this is were we want to put ASSERTS()
                         * to let us know otherwise.
                         */
                        ASSERT(cpus_paused());

                        /*
                         * Now do the work of actually putting this
                         * machine to sleep!
                         */
                        rc = i_cpr_power_down(sleeptype);
                        if (rc == 0) {
                                PMD(PMD_SX, ("back from successful suspend\n"))
                                PMD(PMD_SX, ("back from succssful suspend\n"))
                        }
                        /*
                         * We do care about the return value from cpr_resume
                         * at this point, as it will tell us if one of the
                         * resume functions failed (cpr_resume_devices())
                         * However, for this to return and _not_ panic, means
                         * that we must be in one of the test functions.  So
                         * check for that and return an appropriate message.
                         */
                        rc2 = cpr_resume(sleeptype);
                        if (rc2 != 0) {
                                ASSERT(cpr_test_point > 0);
                                cmn_err(CE_NOTE,
                                    "cpr_resume returned non-zero: %d\n", rc2);
                                PMD(PMD_SX, ("cpr_resume rets %x\n", rc2))
                        }
```



```
                        ASSERT(!cpus_paused());
                } else {
                        PMD(PMD_SX, ("failed suspend, resuming\n"))
                        rc = cpr_resume(sleeptype);
                }
                return (rc);
        }
        /*
         * Remember where we are for resume after reboot
         */
        if (!setjmp(&tlwp->lwp_qsav)) {
                /*
                 * try to checkpoint the system, if failed return back
                 * to userland, otherwise power off.
                 */
                rc = cpr_suspend(sleeptype);
                if (rc || cpr_reusable_mode) {
                        /*
                         * We don't really want to go down, or
                         * something went wrong in suspend, do what we can
                         * to put the system back to an operable state then
                         * return back to userland.
                         */
                        PMD(PMD_SX, ("failed suspend, resuming\n"))
                        (void) cpr_resume(sleeptype);
                        PMD(PMD_SX, ("back from failed suspend resume\n"))
                }
        } else {
                /*
                 * This is the resumed side of longjmp, restore the previous
                 * longjmp pointer if there is one so this will be transparent
                 * to the world.
                 * This path is only for CPR_TODISK, where we reboot
                 */
                ASSERT(sleeptype == CPR_TODISK);
                tlwp->lwp_qsav = saveq;
                CPR->c_flags &= ~C_SUSPENDING;
                CPR->c_flags |= C_RESUMING;

                /*
                 * resume the system back to the original state
                 */
                rc = cpr_resume(sleeptype);
                PMD(PMD_SX, ("back from successful suspend; resume rets %x\n",
                    rc))
        }

        (void) cpr_default_setup(0);

        return (rc);
}
______unchanged_portion_omitted_
```

****
18535 Wed Aug 8 18:32:35 2012
new/usr/src/uts/intel/io/agpmaster/agpmaster.c
3067 Typo in spelling "succssful"
****
_____***unchanged_portion_omitted_***

```
 641 /*
 642  * If agp master is successfully detected, 0 is returned.
 642  * If agp master is succssfully detected, 0 is returned.
 643  * Otherwise -1 is returned.
 644  */
 645 static int
 646 detect_agp_devcice(agp_master_softc_t *master_softc,
 647     ddi_acc_handle_t acc_handle)
 648 {
 649         off_t cap;

 651         cap = agpmaster_cap_find(acc_handle);
 652         if (cap) {
 653                 master_softc->agpm_dev_type = DEVICE_IS_AGP;
 654                 master_softc->agpm_data.agpm_acaptr = cap;
 655                 return (0);
 656         } else {
 657                 return (-1);
 658         }

 660 }
```

_____***unchanged_portion_omitted_***