```
**********************************************************
  129871 Thu Aug  8 20:14:52 2013
new/usr/src/cmd/zpool/zpool_main.c
1765 assert importing pool with number at start
**********************************************************
______unchanged_portion_omitted_

1750 /*
1751  * zpool import [-d dir] [-D]
1752  *       import [-o mntopts] [-o prop=value] ... [-R root] [-D]
1753  *              [-d dir | -c cachefile] [-f] -a
1754  *       import [-o mntopts] [-o prop=value] ... [-R root] [-D]
1755  *              [-d dir | -c cachefile] [-f] [-n] [-F] <pool | id> [newpool]
1756  *
1757  *       -c     Read pool information from a cachefile instead of searching
1758  *              devices.
1759  *
1760  *       -d     Scan in a specific directory, other than /dev/dsk.  More than
1761  *              one directory can be specified using multiple '-d' options.
1762  *
1763  *       -D     Scan for previously destroyed pools or import all or only
1764  *              specified destroyed pools.
1765  *
1766  *       -R     Temporarily import the pool, with all mountpoints relative to
1767  *              the given root.  The pool will remain exported when the machine
1768  *              is rebooted.
1769  *
1770  *       -V     Import even in the presence of faulted vdevs.  This is an
1771  *              intentionally undocumented option for testing purposes, and
1772  *              treats the pool configuration as complete, leaving any bad
1773  *              vdevs in the FAULTED state. In other words, it does verbatim
1774  *              import.
1775  *
1776  *       -f     Force import, even if it appears that the pool is active.
1777  *
1778  *       -F     Attempt rewind if necessary.
1779  *
1780  *       -n     See if rewind would work, but don't actually rewind.
1781  *
1782  *       -N     Import the pool but don't mount datasets.
1783  *
1784  *       -T     Specify a starting txg to use for import. This option is
1785  *              intentionally undocumented option for testing purposes.
1786  *
1787  *       -a     Import all pools found.
1788  *
1789  *       -o     Set property=value and/or temporary mount options (without '=').
1790  *
1791  * The import command scans for pools to import, and import pools based on pool
1792  * name and GUID.  The pool can also be renamed as part of the import process.
1793  */
1794 int
1795 zpool_do_import(int argc, char **argv)
1796 {
1797         char **searchdirs = NULL;
1798         int nsearch = 0;
1799         int c;
1800         int err = 0;
1801         nvlist_t *pools = NULL;
1802         boolean_t do_all = B_FALSE;
1803         boolean_t do_destroyed = B_FALSE;
1804         char *mntopts = NULL;
1805         nvpair_t *elem;
1806         nvlist_t *config;
1807         uint64_t searchguid = 0;
1808         char *searchname = NULL;
```



```
1809         char *propval;
1810         nvlist_t *found_config;
1811         nvlist_t *policy = NULL;
1812         nvlist_t *props = NULL;
1813         boolean_t first;
1814         int flags = ZFS_IMPORT_NORMAL;
1815         uint32_t rewind_policy = ZPOOL_NO_REWIND;
1816         boolean_t dryrun = B_FALSE;
1817         boolean_t do_rewind = B_FALSE;
1818         boolean_t xtreme_rewind = B_FALSE;
1819         uint64_t pool_state, txg = -1ULL;
1820         char *cachefile = NULL;
1821         importargs_t idata = { 0 };
1822         char *endptr;

1824         /* check options */
1825         while ((c = getopt(argc, argv, ":aCc:d:DEfFmnNo:rR:T:VX")) != -1) {
1826                 switch (c) {
1827                 case 'a':
1828                         do_all = B_TRUE;
1829                         break;
1830                 case 'c':
1831                         cachefile = optarg;
1832                         break;
1833                 case 'd':
1834                         if (searchdirs == NULL) {
1835                                 searchdirs = safe_malloc(sizeof (char *));
1836                         } else {
1837                                 char **tmp = safe_malloc((nsearch + 1) *
1838                                     sizeof (char *));
1839                                 bcopy(searchdirs, tmp, nsearch *
1840                                     sizeof (char *));
1841                                 free(searchdirs);
1842                                 searchdirs = tmp;
1843                         }
1844                         searchdirs[nsearch++] = optarg;
1845                         break;
1846                 case 'D':
1847                         do_destroyed = B_TRUE;
1848                         break;
1849                 case 'f':
1850                         flags |= ZFS_IMPORT_ANY_HOST;
1851                         break;
1852                 case 'F':
1853                         do_rewind = B_TRUE;
1854                         break;
1855                 case 'm':
1856                         flags |= ZFS_IMPORT_MISSING_LOG;
1857                         break;
1858                 case 'n':
1859                         dryrun = B_TRUE;
1860                         break;
1861                 case 'N':
1862                         flags |= ZFS_IMPORT_ONLY;
1863                         break;
1864                 case 'o':
1865                         if ((propval = strchr(optarg, '=')) != NULL) {
1866                                 *propval = '\0';
1867                                 propval++;
1868                                 if (add_prop_list(optarg, propval,
1869                                     &props, B_TRUE))
1870                                         goto error;
1871                         } else {
1872                                 mntopts = optarg;
1873                         }
1874                         break;
```

```
1875                case 'R':
1876                        if (add_prop_list(zpool_prop_to_name(
1877                            ZPOOL_PROP_ALTROOT), optarg, &props, B_TRUE))
1878                                goto error;
1879                        if (nvlist_lookup_string(props,
1880                            zpool_prop_to_name(ZPOOL_PROP_CACHEFILE),
1881                            &propval) == 0)
1882                                break;
1883                        if (add_prop_list(zpool_prop_to_name(
1884                            ZPOOL_PROP_CACHEFILE), "none", &props, B_TRUE))
1885                                goto error;
1886                        break;
1887                case 'T':
1888                        errno = 0;
1889                        txg = strtoull(optarg, &endptr, 10);
1890                        if (errno != 0 || *endptr != '\0') {
1891                                (void) fprintf(stderr,
1892                                    gettext("invalid txg value\n"));
1893                                usage(B_FALSE);
1894                        }
1895                        rewind_policy = ZPOOL_DO_REWIND | ZPOOL_EXTREME_REWIND;
1896                        break;
1897                case 'V':
1898                        flags |= ZFS_IMPORT_VERBATIM;
1899                        break;
1900                case 'X':
1901                        xtreme_rewind = B_TRUE;
1902                        break;
1903                case ':':
1904                        (void) fprintf(stderr, gettext("missing argument for "
1905                            "'%c' option\n"), optopt);
1906                        usage(B_FALSE);
1907                        break;
1908                case '?':
1909                        (void) fprintf(stderr, gettext("invalid option '%c'\n"),
1910                            optopt);
1911                        usage(B_FALSE);
1912                }
1913        }

1915        argc -= optind;
1916        argv += optind;

1918        if (cachefile && nsearch != 0) {
1919                (void) fprintf(stderr, gettext("-c is incompatible with -d\n"));
1920                usage(B_FALSE);
1921        }

1923        if ((dryrun || xtreme_rewind) && !do_rewind) {
1924                (void) fprintf(stderr,
1925                    gettext("-n or -X only meaningful with -F\n"));
1926                usage(B_FALSE);
1927        }
1928        if (dryrun)
1929                rewind_policy = ZPOOL_TRY_REWIND;
1930        else if (do_rewind)
1931                rewind_policy = ZPOOL_DO_REWIND;
1932        if (xtreme_rewind)
1933                rewind_policy |= ZPOOL_EXTREME_REWIND;

1935        /* In the future, we can capture further policy and include it here */
1936        if (nvlist_alloc(&policy, NV_UNIQUE_NAME, 0) != 0 ||
1937            nvlist_add_uint64(policy, ZPOOL_REWIND_REQUEST_TXG, txg) != 0 ||
1938            nvlist_add_uint32(policy, ZPOOL_REWIND_REQUEST, rewind_policy) != 0)
1939                goto error;
```



```
1941        if (searchdirs == NULL) {
1942                searchdirs = safe_malloc(sizeof (char *));
1943                searchdirs[0] = "/dev/dsk";
1944                nsearch = 1;
1945        }

1947        /* check argument count */
1948        if (do_all) {
1949                if (argc != 0) {
1950                        (void) fprintf(stderr, gettext("too many arguments\n"));
1951                        usage(B_FALSE);
1952                }
1953        } else {
1954                if (argc > 2) {
1955                        (void) fprintf(stderr, gettext("too many arguments\n"));
1956                        usage(B_FALSE);
1957                }

1959                /*
1960                 * Check for the SYS_CONFIG privilege.  We do this explicitly
1961                 * here because otherwise any attempt to discover pools will
1962                 * silently fail.
1963                 */
1964                if (argc == 0 && !priv_ineffect(PRIV_SYS_CONFIG)) {
1965                        (void) fprintf(stderr, gettext("cannot "
1966                            "discover pools: permission denied\n"));
1967                        free(searchdirs);
1968                        nvlist_free(policy);
1969                        return (1);
1970                }
1971        }

1973        /*
1974         * Depending on the arguments given, we do one of the following:
1975         *
1976         *      <none>  Iterate through all pools and display information about
1977         *              each one.
1978         *
1979         *      -a      Iterate through all pools and try to import each one.
1980         *
1981         *      <id>    Find the pool that corresponds to the given GUID/pool
1982         *              name and import that one.
1983         *
1984         *      -D      Above options applies only to destroyed pools.
1985         */
1986        if (argc != 0) {
1987                char *endptr;

1989                errno = 0;
1990                searchguid = strtoull(argv[0], &endptr, 10);
1991                if (errno != 0 || *endptr != '\0') {
1991                if (errno != 0 || *endptr != '\0')
1992                        searchname = argv[0];
1993                        searchguid = 0;
1994                }
1995                found_config = NULL;

1997                /*
1998                 * User specified a name or guid.  Ensure it's unique.
1999                 */
2000                idata.unique = B_TRUE;
2001        }


2004        idata.path = searchdirs;
2005        idata.paths = nsearch;
```

```
2006         idata.poolname = searchname;
2007         idata.guid = searchguid;
2008         idata.cachefile = cachefile;

2010         pools = zpool_search_import(g_zfs, &idata);

2012         if (pools != NULL && idata.exists &&
2013             (argc == 1 || strcmp(argv[0], argv[1]) == 0)) {
2014                 (void) fprintf(stderr, gettext("cannot import '%s': "
2015                     "a pool with that name already exists\n"),
2016                     argv[0]);
2017                 (void) fprintf(stderr, gettext("use the form '%s "
2018                     "<pool | id> <newpool>' to give it a new name\n"),
2019                     "zpool import");
2020                 err = 1;
2021         } else if (pools == NULL && idata.exists) {
2022                 (void) fprintf(stderr, gettext("cannot import '%s': "
2023                     "a pool with that name is already created/imported,\n"),
2024                     argv[0]);
2025                 (void) fprintf(stderr, gettext("and no additional pools "
2026                     "with that name were found\n"));
2027                 err = 1;
2028         } else if (pools == NULL) {
2029                 if (argc != 0) {
2030                         (void) fprintf(stderr, gettext("cannot import '%s': "
2031                             "no such pool available\n"), argv[0]);
2032                 }
2033                 err = 1;
2034         }

2036         if (err == 1) {
2037                 free(searchdirs);
2038                 nvlist_free(policy);
2039                 return (1);
2040         }

2042         /*
2043          * At this point we have a list of import candidate configs. Even if
2044          * we were searching by pool name or guid, we still need to
2045          * post-process the list to deal with pool state and possible
2046          * duplicate names.
2047          */
2048         err = 0;
2049         elem = NULL;
2050         first = B_TRUE;
2051         while ((elem = nvlist_next_nvpair(pools, elem)) != NULL) {

2053                 verify(nvpair_value_nvlist(elem, &config) == 0);

2055                 verify(nvlist_lookup_uint64(config, ZPOOL_CONFIG_POOL_STATE,
2056                     &pool_state) == 0);
2057                 if (!do_destroyed && pool_state == POOL_STATE_DESTROYED)
2058                         continue;
2059                 if (do_destroyed && pool_state != POOL_STATE_DESTROYED)
2060                         continue;

2062                 verify(nvlist_add_nvlist(config, ZPOOL_REWIND_POLICY,
2063                     policy) == 0);

2065                 if (argc == 0) {
2066                         if (first)
2067                                 first = B_FALSE;
2068                         else if (!do_all)
2069                                 (void) printf("\n");

2071                         if (do_all) {
```



```
2072                                 err |= do_import(config, NULL, mntopts,
2073                                     props, flags);
2074                         } else {
2075                                 show_import(config);
2076                         }
2077                 } else if (searchname != NULL) {
2078                         char *name;

2080                         /*
2081                          * We are searching for a pool based on name.
2082                          */
2083                         verify(nvlist_lookup_string(config,
2084                             ZPOOL_CONFIG_POOL_NAME, &name) == 0);

2086                         if (strcmp(name, searchname) == 0) {
2087                                 if (found_config != NULL) {
2088                                         (void) fprintf(stderr, gettext(
2089                                             "cannot import '%s': more than "
2090                                             "one matching pool\n"), searchname);
2091                                         (void) fprintf(stderr, gettext(
2092                                             "import by numeric ID instead\n"));
2093                                         err = B_TRUE;
2094                                 }
2095                                 found_config = config;
2096                         }
2097                 } else {
2098                         uint64_t guid;

2100                         /*
2101                          * Search for a pool by guid.
2102                          */
2103                         verify(nvlist_lookup_uint64(config,
2104                             ZPOOL_CONFIG_POOL_GUID, &guid) == 0);

2106                         if (guid == searchguid)
2107                                 found_config = config;
2108                 }
2109         }

2111         /*
2112          * If we were searching for a specific pool, verify that we found a
2113          * pool, and then do the import.
2114          */
2115         if (argc != 0 && err == 0) {
2116                 if (found_config == NULL) {
2117                         (void) fprintf(stderr, gettext("cannot import '%s': "
2118                             "no such pool available\n"), argv[0]);
2119                         err = B_TRUE;
2120                 } else {
2121                         err |= do_import(found_config, argc == 1 ? NULL :
2122                             argv[1], mntopts, props, flags);
2123                 }
2124         }

2126         /*
2127          * If we were just looking for pools, report an error if none were
2128          * found.
2129          */
2130         if (argc == 0 && first)
2131                 (void) fprintf(stderr,
2132                     gettext("no pools available to import\n"));

2134 error:
2135         nvlist_free(props);
2136         nvlist_free(pools);
2137         nvlist_free(policy);
```

```
2138         free(searchdirs);

2140         return (err ? 1 : 0);
2141 }
```

_____***unchanged_portion_omitted_***