```
**********************************************************
   58084 Wed Jan 22 14:16:29 2014
new/usr/src/cmd/zlogin/zlogin.c
3091 add -n to zlogin so its more compatible with rsh command line
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2013 DEY Storage Systems, Inc.
 * Copyright (c) 2014 Gary Mills
 */

/*
 * zlogin provides three types of login which allow users in the global
 * zone to access non-global zones.
 *
 * - "interactive login" is similar to rlogin(1); for example, the user could
 *   issue 'zlogin my-zone' or 'zlogin -e ^ -l me my-zone'.   The user is
 *   granted a new pty (which is then shoved into the zone), and an I/O
 *   loop between parent and child processes takes care of the interactive
 *   session.  In this mode, login(1) (and its -c option, which means
 *   "already authenticated") is employed to take care of the initialization
 *   of the user's session.
 *
 * - "non-interactive login" is similar to su(1M); the user could issue
 *   'zlogin my-zone ls -l' and the command would be run as specified.
 *   In this mode, zlogin sets up pipes as the communication channel, and
 *   'su' is used to do the login setup work.
 *
 * - "console login" is the equivalent to accessing the tip line for a
 *   zone.  For example, the user can issue 'zlogin -C my-zone'.
 *   In this mode, zlogin contacts the zoneadmd process via unix domain
 *   socket.  If zoneadmd is not running, it starts it.  This allows the
 *   console to be available anytime the zone is installed, regardless of
 *   whether it is running.
 */

#include <sys/socket.h>
#include <sys/termios.h>
#include <sys/utsname.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/contract/process.h>
#include <sys/ctfs.h>
#include <sys/brand.h>
#include <sys/wait.h>
#include <alloca.h>
```



```
#include <assert.h>
#include <ctype.h>
#include <paths.h>
#include <door.h>
#include <errno.h>
#include <nss_dbdefs.h>
#include <poll.h>
#include <priv.h>
#include <pwd.h>
#include <unistd.h>
#include <utmpx.h>
#include <sac.h>
#include <signal.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <stropts.h>
#include <wait.h>
#include <zone.h>
#include <fcntl.h>
#include <libdevinfo.h>
#include <libintl.h>
#include <locale.h>
#include <libzonecfg.h>
#include <libcontract.h>
#include <libbrand.h>
#include <auth_list.h>
#include <auth_attr.h>
#include <secdb.h>

static int masterfd;
static struct termios save_termios;
static struct termios effective_termios;
static int save_fd;
static struct winsize winsize;
static volatile int dead;
static volatile pid_t child_pid = -1;
static int interactive = 0;
static priv_set_t *dropprivs;

static int nocmdchar = 0;
static int failsafe = 0;
static char cmdchar = '~';
static int quiet = 0;

static int pollerr = 0;

static const char *pname;
static char *username;

/*
 * When forced_login is true, the user is not prompted
 * for an authentication password in the target zone.
 */
static boolean_t forced_login = B_FALSE;

#if !defined(TEXT_DOMAIN)               /* should be defined by cc -D */
#define TEXT_DOMAIN     "SYS_TEST"      /* Use this only if it wasn't */
#endif

#define SUPATH  "/usr/bin/su"
#define FAILSAFESHELL   "/sbin/sh"
#define DEFAULTSHELL    "/sbin/sh"
#define DEF_PATH        "/usr/sbin:/usr/bin"
```

```
 129 #define CLUSTER_BRAND_NAME      "cluster"

 131 /*
 132  * The ZLOGIN_BUFSIZ is larger than PIPE_BUF so we can be sure we're clearing
 133  * out the pipe when the child is exiting.  The ZLOGIN_RDBUFSIZ must be less
 134  * than ZLOGIN_BUFSIZ (because we share the buffer in doio).  This value is
 135  * also chosen in conjunction with the HI_WATER setting to make sure we
 136  * don't fill up the pipe.  We can write FIFOHIWAT (16k) into the pipe before
 137  * blocking.  By having ZLOGIN_RDBUFSIZ set to 1k and HI_WATER set to 8k, we
 138  * know we can always write a ZLOGIN_RDBUFSIZ chunk into the pipe when there
 139  * is less than HI_WATER data already in the pipe.
 140  */
 141 #define ZLOGIN_BUFSIZ   8192
 142 #define ZLOGIN_RDBUFSIZ 1024
 143 #define HI_WATER        8192

 145 /*
 146  * See canonify() below.  CANONIFY_LEN is the maximum length that a
 147  * "canonical" sequence will expand to (backslash, three octal digits, NUL).
 148  */
 149 #define CANONIFY_LEN 5

 151 static void
 152 usage(void)
 153 {
 154         (void) fprintf(stderr, gettext("usage: %s [ -nQCES ] [ -e cmdchar ] "
 152         (void) fprintf(stderr, gettext("usage: %s [ -QCES ] [ -e cmdchar ] "
 155             "[-l user] zonename [command [args ...] ]\n"), pname);
 156         exit(2);
 157 }
_____unchanged_portion_omitted_

1726 int
1727 main(int argc, char **argv)
1728 {
1729         int arg, console = 0;
1730         zoneid_t zoneid;
1731         zone_state_t st;
1732         char *login = "root";
1733         int lflag = 0;
1734         int nflag = 0;
1735         char *zonename = NULL;
1736         char **proc_args = NULL;
1737         char **new_args, **new_env;
1738         sigset_t block_cld;
1739         char devroot[MAXPATHLEN];
1740         char *slavename, slaveshortname[MAXPATHLEN];
1741         priv_set_t *privset;
1742         int tmpl_fd;
1743         char zonebrand[MAXNAMELEN];
1744         char default_brand[MAXNAMELEN];
1745         struct stat sb;
1746         char kernzone[ZONENAME_MAX];
1747         brand_handle_t bh;
1748         char user_cmd[MAXPATHLEN];
1749         char authname[MAXAUTHS];

1751         (void) setlocale(LC_ALL, "");
1752         (void) textdomain(TEXT_DOMAIN);

1754         (void) getpname(argv[0]);
1755         username = get_username();

1757         while ((arg = getopt(argc, argv, "nECR:Se:l:Q")) != EOF) {
1754         while ((arg = getopt(argc, argv, "ECR:Se:l:Q")) != EOF) {
```



```
1758                 switch (arg) {
1759                 case 'C':
1760                         console = 1;
1761                         break;
1762                 case 'E':
1763                         nocmdchar = 1;
1764                         break;
1765                 case 'R':       /* undocumented */
1766                         if (*optarg != '/') {
1767                                 zerror(gettext("root path must be absolute."));
1768                                 exit(2);
1769                         }
1770                         if (stat(optarg, &sb) == -1 || !S_ISDIR(sb.st_mode)) {
1771                                 zerror(
1772                                     gettext("root path must be a directory."));
1773                                 exit(2);
1774                         }
1775                         zonecfg_set_root(optarg);
1776                         break;
1777                 case 'Q':
1778                         quiet = 1;
1779                         break;
1780                 case 'S':
1781                         failsafe = 1;
1782                         break;
1783                 case 'e':
1784                         set_cmdchar(optarg);
1785                         break;
1786                 case 'l':
1787                         login = optarg;
1788                         lflag = 1;
1789                         break;
1790                 case 'n':
1791                         nflag = 1;
1792                         break;
1793                 default:
1794                         usage();
1795                 }
1796         }

1798         if (console != 0) {

1800                 if (lflag != 0) {
1801                         zerror(gettext(
1802                             "-l may not be specified for console login"));
1792         if (console != 0 && lflag != 0) {
1793                 zerror(gettext("-l may not be specified for console login"));
1803                         usage();
1804                 }

1806                 if (nflag != 0) {
1807                         zerror(gettext(
1808                             "-n may not be specified for console login"));
1797         if (console != 0 && failsafe != 0) {
1798                 zerror(gettext("-S may not be specified for console login"));
1809                         usage();
1810                 }

1812                 if (failsafe != 0) {
1813                         zerror(gettext(
1814                             "-S may not be specified for console login"));
1815                         usage();
1816                 }

1818                 if (zonecfg_in_alt_root()) {
1819                         zerror(gettext(
```

```
1820                            "-R may not be specified for console login"));
1802        if (console != 0 && zonecfg_in_alt_root()) {
1803                zerror(gettext("-R may not be specified for console login"));
1821                        exit(2);
1822                }

1824        }

1826        if (failsafe != 0 && lflag != 0) {
1827                zerror(gettext("-l may not be specified for failsafe login"));
1828                usage();
1829        }

1831        if (optind == (argc - 1)) {
1832                /*
1833                 * zone name, no process name; this should be an interactive
1834                 * as long as STDIN is really a tty.
1835                 */
1836                if (nflag != 0) {
1837                        zerror(gettext(
1838                            "-n may not be specified for interactive login"));
1839                        usage();
1840                }
1841                if (isatty(STDIN_FILENO))
1842                        interactive = 1;
1843                zonename = argv[optind];
1844        } else if (optind < (argc - 1)) {
1845                if (console) {
1846                        zerror(gettext("Commands may not be specified for "
1847                            "console login."));
1848                        usage();
1849                }
1850                /* zone name and process name, and possibly some args */
1851                zonename = argv[optind];
1852                proc_args = &argv[optind + 1];
1853                interactive = 0;
1854        } else {
1855                usage();
1856        }

1858        if (getzoneid() != GLOBAL_ZONEID) {
1859                zerror(gettext("'%s' may only be used from the global zone"),
1860                    pname);
1861                return (1);
1862        }

1864        if (strcmp(zonename, GLOBAL_ZONENAME) == 0) {
1865                zerror(gettext("'%s' not applicable to the global zone"),
1866                    pname);
1867                return (1);
1868        }

1870        if (zone_get_state(zonename, &st) != Z_OK) {
1871                zerror(gettext("zone '%s' unknown"), zonename);
1872                return (1);
1873        }

1875        if (st < ZONE_STATE_INSTALLED) {
1876                zerror(gettext("cannot login to a zone which is '%s'"),
1877                    zone_state_str(st));
1878                return (1);
1879        }

1881        /*
1882         * In both console and non-console cases, we require all privs.
1883         * In the console case, because we may need to startup zoneadmd.
```



```
1884         * In the non-console case in order to do zone_enter(2), zonept()
1885         * and other tasks.
1886         */

1888        if ((privset = priv_allocset()) == NULL) {
1889                zperror(gettext("priv_allocset failed"));
1890                return (1);
1891        }

1893        if (getppriv(PRIV_EFFECTIVE, privset) != 0) {
1894                zperror(gettext("getppriv failed"));
1895                priv_freeset(privset);
1896                return (1);
1897        }

1899        if (priv_isfullset(privset) == B_FALSE) {
1900                zerror(gettext("You lack sufficient privilege to run "
1901                    "this command (all privs required)"));
1902                priv_freeset(privset);
1903                return (1);
1904        }
1905        priv_freeset(privset);

1907        /*
1908         * Check if user is authorized for requested usage of the zone
1909         */

1911        (void) snprintf(authname, MAXAUTHS, "%s%s%s",
1912            ZONE_MANAGE_AUTH, KV_OBJECT, zonename);
1913        if (chkauthattr(authname, username) == 0) {
1914                if (console) {
1915                        zerror(gettext("%s is not authorized for console "
1916                            "access to  %s zone."),
1917                            username, zonename);
1918                        return (1);
1919                } else {
1920                        (void) snprintf(authname, MAXAUTHS, "%s%s%s",
1921                            ZONE_LOGIN_AUTH, KV_OBJECT, zonename);
1922                        if (failsafe || !interactive) {
1923                                zerror(gettext("%s is not authorized for  "
1924                                    "failsafe or non-interactive login "
1925                                    "to  %s zone."), username, zonename);
1926                                return (1);
1927                        } else if (chkauthattr(authname, username) == 0) {
1928                                zerror(gettext("%s is not authorized "
1929                                    " to login to %s zone."),
1930                                    username, zonename);
1931                                return (1);
1932                        }
1933                }
1934        } else {
1935                forced_login = B_TRUE;
1936        }

1938        /*
1939         * The console is a separate case from the rest of the code; handle
1940         * it first.
1941         */
1942        if (console) {
1943                /*
1944                 * Ensure that zoneadmd for this zone is running.
1945                 */
1946                if (start_zoneadmd(zonename) == -1)
1947                        return (1);

1949                /*
```

```
1950                 * Make contact with zoneadmd.
1951                 */
1952                if (get_console_master(zonename) == -1)
1953                        return (1);

1955                if (!quiet)
1956                        (void) printf(
1957                            gettext("[Connected to zone '%s' console]\n"),
1958                            zonename);

1960                if (set_tty_rawmode(STDIN_FILENO) == -1) {
1961                        reset_tty();
1962                        zperror(gettext("failed to set stdin pty to raw mode"));
1963                        return (1);
1964                }

1966                (void) sigset(SIGWINCH, sigwinch);
1967                (void) sigwinch(0);

1969                /*
1970                 * Run the I/O loop until we get disconnected.
1971                 */
1972                doio(masterfd, -1, masterfd, -1, -1, B_FALSE);
1973                reset_tty();
1974                if (!quiet)
1975                        (void) printf(
1976                            gettext("\n[Connection to zone '%s' console "
1977                            "closed]\n"), zonename);

1979                return (0);
1980        }

1982        if (st != ZONE_STATE_RUNNING && st != ZONE_STATE_MOUNTED) {
1983                zerror(gettext("login allowed only to running zones "
1984                    "(%s is '%s')."), zonename, zone_state_str(st));
1985                return (1);
1986        }

1988        (void) strlcpy(kernzone, zonename, sizeof (kernzone));
1989        if (zonecfg_in_alt_root()) {
1990                FILE *fp = zonecfg_open_scratch("", B_FALSE);

1992                if (fp == NULL || zonecfg_find_scratch(fp, zonename,
1993                    zonecfg_get_root(), kernzone, sizeof (kernzone)) == -1) {
1994                        zerror(gettext("cannot find scratch zone %s"),
1995                            zonename);
1996                        if (fp != NULL)
1997                                zonecfg_close_scratch(fp);
1998                        return (1);
1999                }
2000                zonecfg_close_scratch(fp);
2001        }

2003        if ((zoneid = getzoneidbyname(kernzone)) == -1) {
2004                zerror(gettext("failed to get zoneid for zone '%s'"),
2005                    zonename);
2006                return (1);
2007        }

2009        /*
2010         * We need the zone root path only if we are setting up a pty.
2011         */
2012        if (zone_get_devroot(zonename, devroot, sizeof (devroot)) == -1) {
2013                zerror(gettext("could not get dev path for zone %s"),
2014                    zonename);
2015                return (1);
```



```
2016        }

2018        if (zone_get_brand(zonename, zonebrand, sizeof (zonebrand)) != Z_OK) {
2019                zerror(gettext("could not get brand for zone %s"), zonename);
2020                return (1);
2021        }
2022        /*
2023         * In the alternate root environment, the only supported
2024         * operations are mount and unmount.  In this case, just treat
2025         * the zone as native if it is cluster.  Cluster zones can be
2026         * native for the purpose of LU or upgrade, and the cluster
2027         * brand may not exist in the miniroot (such as in net install
2028         * upgrade).
2029         */
2030        if (zonecfg_default_brand(default_brand,
2031            sizeof (default_brand)) != Z_OK) {
2032                zerror(gettext("unable to determine default brand"));
2033                return (1);
2034        }
2035        if (zonecfg_in_alt_root() &&
2036            strcmp(zonebrand, CLUSTER_BRAND_NAME) == 0) {
2037                (void) strlcpy(zonebrand, default_brand, sizeof (zonebrand));
2038        }

2040        if ((bh = brand_open(zonebrand)) == NULL) {
2041                zerror(gettext("could not open brand for zone %s"), zonename);
2042                return (1);
2043        }

2045        if ((new_args = prep_args(bh, login, proc_args)) == NULL) {
2046                zperror(gettext("could not assemble new arguments"));
2047                brand_close(bh);
2048                return (1);
2049        }
2050        /*
2051         * Get the brand specific user_cmd.  This command is used to get
2052         * a passwd(4) entry for login.
2053         */
2054        if (!interactive && !failsafe) {
2055                if (zone_get_user_cmd(bh, login, user_cmd,
2056                    sizeof (user_cmd)) == NULL) {
2057                        zerror(gettext("could not get user_cmd for zone %s"),
2058                            zonename);
2059                        brand_close(bh);
2060                        return (1);
2061                }
2062        }
2063        brand_close(bh);

2065        if ((new_env = prep_env()) == NULL) {
2066                zperror(gettext("could not assemble new environment"));
2067                return (1);
2068        }

2070        if (!interactive) {
2071                if (nflag) {
2072                        int nfd;

2074                        if ((nfd = open(_PATH_DEVNULL, O_RDONLY)) < 0) {
2075                                zperror(gettext("failed to open null device"));
2076                                return (1);
2077                        }
2078                        if (nfd != STDIN_FILENO) {
2079                                if (dup2(nfd, STDIN_FILENO) < 0) {
2080                                        zperror(gettext(
2081                                            "failed to dup2 null device"));
```

```
2082                                    return (1);
2083                            }
2084                            (void) close(nfd);
2085                    }
2086                    /* /dev/null is now standard input */
2087            }
2046    if (!interactive)
2088            return (noninteractive_login(zonename, user_cmd, zoneid,
2089                new_args, new_env));
2090    }

2092    if (zonecfg_in_alt_root()) {
2093            zerror(gettext("cannot use interactive login with scratch "
2094                "zone"));
2095            return (1);
2096    }

2098    /*
2099     * Things are more complex in interactive mode; we get the
2100     * master side of the pty, then place the user's terminal into
2101     * raw mode.
2102     */
2103    if (get_master_pty() == -1) {
2104            zerror(gettext("could not setup master pty device"));
2105            return (1);
2106    }

2108    /*
2109     * Compute the "short name" of the pts.  /dev/pts/2 --> pts/2
2110     */
2111    if ((slavename = ptsname(masterfd)) == NULL) {
2112            zperror(gettext("failed to get name for pseudo-tty"));
2113            return (1);
2114    }
2115    if (strncmp(slavename, "/dev/", strlen("/dev/")) == 0)
2116            (void) strlcpy(slaveshortname, slavename + strlen("/dev/"),
2117                sizeof (slaveshortname));
2118    else
2119            (void) strlcpy(slaveshortname, slavename,
2120                sizeof (slaveshortname));

2122    if (!quiet)
2123            (void) printf(gettext("[Connected to zone '%s' %s]\n"),
2124                zonename, slaveshortname);

2126    if (set_tty_rawmode(STDIN_FILENO) == -1) {
2127            reset_tty();
2128            zperror(gettext("failed to set stdin pty to raw mode"));
2129            return (1);
2130    }

2132    if (prefork_dropprivs() != 0) {
2133            reset_tty();
2134            zperror(gettext("could not allocate privilege set"));
2135            return (1);
2136    }

2138    /*
2139     * We must mask SIGCLD until after we have coped with the fork
2140     * sufficiently to deal with it; otherwise we can race and receive the
2141     * signal before child_pid has been initialized (yes, this really
2142     * happens).
2143     */
2144    (void) sigset(SIGCLD, sigcld);
2145    (void) sigemptyset(&block_cld);
2146    (void) sigaddset(&block_cld, SIGCLD);
```



```
2147    (void) sigprocmask(SIG_BLOCK, &block_cld, NULL);

2149    /*
2150     * We activate the contract template at the last minute to
2151     * avoid intermediate functions that could be using fork(2)
2152     * internally.
2153     */
2154    if ((tmpl_fd = init_template()) == -1) {
2155            reset_tty();
2156            zperror(gettext("could not create contract"));
2157            return (1);
2158    }

2160    if ((child_pid = fork()) == -1) {
2161            (void) ct_tmpl_clear(tmpl_fd);
2162            reset_tty();
2163            zperror(gettext("could not fork"));
2164            return (1);
2165    } else if (child_pid == 0) { /* child process */
2166            int slavefd, newslave;

2168            (void) ct_tmpl_clear(tmpl_fd);
2169            (void) close(tmpl_fd);

2171            (void) sigprocmask(SIG_UNBLOCK, &block_cld, NULL);

2173            if ((slavefd = init_slave_pty(zoneid, devroot)) == -1)
2174                    return (1);

2176            /*
2177             * Close all fds except for the slave pty.
2178             */
2179            (void) fdwalk(close_func, &slavefd);

2181            /*
2182             * Temporarily dup slavefd to stderr; that way if we have
2183             * to print out that zone_enter failed, the output will
2184             * have somewhere to go.
2185             */
2186            if (slavefd != STDERR_FILENO)
2187                    (void) dup2(slavefd, STDERR_FILENO);

2189            if (zone_enter(zoneid) == -1) {
2190                    zerror(gettext("could not enter zone %s: %s"),
2191                        zonename, strerror(errno));
2192                    return (1);
2193            }

2195            if (slavefd != STDERR_FILENO)
2196                    (void) close(STDERR_FILENO);

2198            /*
2199             * We take pains to get this process into a new process
2200             * group, and subsequently a new session.  In this way,
2201             * we'll have a session which doesn't yet have a controlling
2202             * terminal.  When we open the slave, it will become the
2203             * controlling terminal; no PIDs concerning pgrps or sids
2204             * will leak inappropriately into the zone.
2205             */
2206            (void) setpgrp();

2208            /*
2209             * We need the slave pty to be referenced from the zone's
2210             * /dev in order to ensure that the devt's, etc are all
2211             * correct.  Otherwise we break ttyname and the like.
2212             */
```

```
2213                if ((newslave = open(slavename, O_RDWR)) == -1) {
2214                        (void) close(slavefd);
2215                        return (1);
2216                }
2217                (void) close(slavefd);
2218                slavefd = newslave;

2220                /*
2221                 * dup the slave to the various FDs, so that when the
2222                 * spawned process does a write/read it maps to the slave
2223                 * pty.
2224                 */
2225                (void) dup2(slavefd, STDIN_FILENO);
2226                (void) dup2(slavefd, STDOUT_FILENO);
2227                (void) dup2(slavefd, STDERR_FILENO);
2228                if (slavefd != STDIN_FILENO && slavefd != STDOUT_FILENO &&
2229                    slavefd != STDERR_FILENO) {
2230                        (void) close(slavefd);
2231                }

2233                /*
2234                 * In failsafe mode, we don't use login(1), so don't try
2235                 * setting up a utmpx entry.
2236                 */
2237                if (!failsafe)
2238                        if (setup_utmpx(slaveshortname) == -1)
2239                                return (1);

2241                /*
2242                 * The child needs to run as root to
2243                 * execute the brand's login program.
2244                 */
2245                if (setuid(0) == -1) {
2246                        zperror(gettext("insufficient privilege"));
2247                        return (1);
2248                }

2250                (void) execve(new_args[0], new_args, new_env);
2251                zperror(gettext("exec failure"));
2252                return (1);
2253        }

2255        (void) ct_tmpl_clear(tmpl_fd);
2256        (void) close(tmpl_fd);

2258        /*
2259         * The rest is only for the parent process.
2260         */
2261        (void) sigset(SIGWINCH, sigwinch);

2263        postfork_dropprivs();

2265        (void) sigprocmask(SIG_UNBLOCK, &block_cld, NULL);
2266        doio(masterfd, -1, masterfd, -1, -1, B_FALSE);

2268        reset_tty();
2269        if (!quiet)
2270                (void) fprintf(stderr,
2271                    gettext("\n[Connection to zone '%s' %s closed]\n"),
2272                    zonename, slaveshortname);

2274        if (pollerr != 0) {
2275                (void) fprintf(stderr, gettext("Error: connection closed due "
2276                    "to unexpected pollevents=0x%x.\n"), pollerr);
2277                return (1);
2278        }
```



```
2280        return (0);
2281 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    6921 Wed Jan 22 14:16:29 2014
new/usr/src/man/man1/zlogin.1
3091 add -n to zlogin so its more compatible with rsh command line
**********************************************************
'\" te
.\" Copyright (c) 2006, Sun Microsystems, Inc. All Rights Reserved
.\" The contents of this file are subject to the terms of the Common
.\" Development and Distribution License (the "License").  You may not use this
.\" file except in compliance with the License.
.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or
.\" http://www.opensolaris.org/os/licensing.  See the License for the specific
.\" language governing permissions and limitations under the License.
.\" When distributing Covered Code, include this CDDL HEADER in each file and
.\" include the License file at usr/src/OPENSOLARIS.LICENSE.  If applicable,
.\" add the following below this CDDL HEADER, with the fields enclosed by
.\" brackets "[]" replaced with your own identifying information:
.\" Portions Copyright [yyyy] [name of copyright owner]
.\" Copyright 2013 DEY Storage Systems, Inc.
.\" Copyright (c) 2014 Gary Mills
.TH ZLOGIN 1 "Jan 22, 2014"
.TH ZLOGIN 1 "Sep 8, 2013"
.SH NAME
zlogin \- enter a zone
.SH SYNOPSIS
.LP
.nf
\fBzlogin\fR [\fB-CEQ\fR] [\fB-e\fR \fIc\fR] [\fB-l\fR \fIusername\fR] \fIzonena
.fi

.LP
.nf
\fBzlogin\fR [\fB-nEQS\fR] [\fB-e\fR \fIc\fR] [\fB-l\fR \fIusername\fR] \fIzonen
\fBzlogin\fR [\fB-EQS\fR] [\fB-e\fR \fIc\fR] [\fB-l\fR \fIusername\fR] \fIzonena
     [\fIargument\fR]...
.fi

.SH DESCRIPTION
.sp
.LP
The \fBzlogin\fR utility is used by the administrator to enter an operating
system zone. Only a superuser operating in the global system zone can use this
utility.
.sp
.LP
\fBzlogin\fR operates in one of three modes:
.sp
.ne 2
.na
\fBInteractive Mode\fR
.ad
.RS 24n
If no utility argument is given and the stdin file descriptor for the
\fBzlogin\fR process is a tty device, \fBzlogin\fR operates in \fBinteractive
mode\fR. In this mode, \fBzlogin\fR creates a new pseudo terminal for use
within the login session. Programs requiring a tty device, for example,
\fBvi\fR(1), work properly in this mode. In this mode, \fBzlogin\fR invokes
\fBlogin\fR(1) to provide a suitable login session.
.RE

.sp
.ne 2
.na
\fBNon-Interactive Mode\fR
.ad
.RS 24n
```



```
If a utility is specified, \fBzlogin\fR operates in \fBnon-interactive mode\fR.
This mode can be useful for script authors since stdin, stdout, and stderr are
preserved and the exit status of \fIutility\fR is returned upon termination. In
this mode, \fBzlogin\fR invokes \fBsu\fR(1M) in order to set up the user's
environment and to provide a login environment.
.sp
The specified command is passed as a string and interpreted by a shell running
in the non-global zone. See \fBrsh\fR(1).
.RE

.sp
.ne 2
.na
\fBConsole Mode\fR
.ad
.RS 24n
If the \fB-C\fR option is specified, the user is connected to the zone console
device and \fBzlogin\fR operates in \fBconsole mode\fR. The zone console is
available once the zone is in the installed  state. Connections to the console
are persistent across reboot of the zone.
.RE

.SH OPTIONS
.sp
.LP
The following options are supported:
.sp
.ne 2
.na
\fB\fB-C\fR\fR
.ad
.RS 15n
Connects to the zone console.
.RE

.sp
.ne 2
.na
\fB\fB-e\fR \fIc\fR\fR
.ad
.RS 15n
Specifies a different escape character, \fIc\fR, for the key sequence used to
access extended functions and to disconnect from the login. The default escape
character is the tilde (\fB~\fR).
.RE

.sp
.ne 2
.na
\fB\fB-E\fR\fR
.ad
.RS 15n
Disables the ability to access extended functions or to disconnect from the
login by using the escape sequence character.
.RE

.sp
.ne 2
.na
\fB\fB-l\fR \fIusername\fR\fR
.ad
.RS 15n
Specifies a different \fIusername\fR for the zone login. If you do not use this
option, the zone username used is "root". This option is invalid if the
\fB-C\fR option is specified.
.RE
```

```
.sp
.ne 2
.na
\fB-n\fR
.ad
.RS 15n
Redirect the input of \fBzlogin\fR to \fB/dev/null\fR.
This option is useful when the command running in the local zone
and the shell which invokes \fBzlogin\fR both read from standard input.
.RE

.sp
.ne 2
.na
\fB-Q\fR
.ad
.RS 15n
Specifies quiet mode operation.  In quiet mode, extra messages indicating the
the function of \fBzlogin\fR will not be displayed, giving the possibility
to present the appearance that the command is running locally rather than
in another zone.
.RE

.sp
.ne 2
.na
\fB\fB-S\fR\fR
.ad
.RS 15n
"Safe" login mode. \fBzlogin\fR does minimal processing and does not invoke
\fBlogin\fR(1) or \fBsu\fR(1M). The \fB-S\fR option can not be used if a
username is specified through the \fB-l\fR option, and cannot be used with
console logins. This mode should only be used to recover a damaged zone when
other forms of login have become impossible.
.RE

.SS "Escape Sequences"
.sp
.LP
Lines that you type that start with the tilde character (\fB~\fR) are "escape
sequences". The escape character can be changed using the \fB-e\fR option.
.sp
.ne 2
.na
\fB\fB~.\fR\fR
.ad
.RS 6n
Disconnects from the zone. This is not the same as a logout, because the local
host breaks the connection with no warning to the zone’s end.
.RE

.SH SECURITY
.sp
.LP
Once a process has been placed in a zone other than the global zone, the
process cannot change zone again, nor can any of its children.
.SH OPERANDS
.sp
.LP
The following operands are supported:
.sp
.ne 2
.na
\fB\fIzonename\fR\fR
.ad
```



```
.RS 15n
The name of the zone to be entered.
.RE

.sp
.ne 2
.na
\fB\fIutility\fR\fR
.ad
.RS 15n
The utility to be run in the specified zone.
.RE

.sp
.ne 2
.na
\fB\fIargument...\fR\fR
.ad
.RS 15n
Arguments passed to the utility.
.RE

.SH EXIT STATUS
.sp
.LP
In interactive and non-interactive modes, the \fBzlogin\fR utility exits when
the command or shell in the non-global zone exits. In non-interactive mode, the
exit status of the remote program is returned as the exit status of
\fBzlogin\fR. In interactive mode and console login mode, the exit status is
not returned. \fBzlogin\fR returns a \fB0\fR exit status as long as no
connection-related error occurred.
.sp
.LP
In all modes, in the event that a connection to the zone cannot be established,
the connection fails unexpectedly, or the user is lacking sufficient privilege
to perform the requested operation, \fBzlogin\fR exits with status \fB1\fR.
.sp
.LP
To summarize, the following exit values are returned:
.sp
.ne 2
.na
\fB\fB0\fR\fR
.ad
.RS 7n
Successful entry.
.RE

.sp
.ne 2
.na
\fB\fB1\fR\fR
.ad
.RS 7n
Permission denied, or failure to enter the zone.
.RE

.sp
.ne 2
.na
\fBAny\fR
.ad
.RS 7n
Return code from utility, or from \fBsu\fR(1M) if operating in non-interactive
mode.
.RE
```

```
259 .SH ATTRIBUTES
260 .sp
261 .LP
262 See \fBattributes\fR(5) for descriptions of the following attributes:
263 .sp

265 .sp
266 .TS
267 box;
268 c | c
269 l | l .
270 ATTRIBUTE TYPE  ATTRIBUTE VALUE
271 _
272 Interface Stability     Evolving
273 .TE

275 .SH SEE ALSO
276 .sp
277 .LP
278 \fBlogin\fR(1), \fBrsh\fR(1), \fBvi\fR(1), \fBsu\fR(1M), \fBzoneadm\fR(1M),
279 \fBzonecfg\fR(1M), \fBattributes\fR(5), \fBzones\fR(5)
280 .SH NOTES
281 .sp
282 .LP
283 \fBzlogin\fR fails if its open files or any portion of its address space
284 corresponds to an NFS file. This includes the executable itself or the shared
285 libraries.
```