```
**********************************************************
   23382 Tue Aug 18 09:38:18 2015
new/usr/src/cmd/cron/at.c
5433 at(1) doesn't properly handle being invoked from a path containing spaces
**********************************************************
```

```
/*
 * 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) 2011 Gary Mills
 *
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

#include <sys/resource.h>
#include <sys/stat.h>
#include <sys/types.h>

#include <dirent.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <pwd.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include <signal.h>
#include <errno.h>
#include <limits.h>
#include <ulimit.h>
#include <unistd.h>
#include <locale.h>
#include <libintl.h>
#include <tzfile.h>
#include <project.h>
#include <paths.h>

#include "cron.h"

#define TMPFILE         "_at" /* prefix for temporary files     */
/*
 * Mode for creating files in ATDIR.
 * Setuid bit on so that if an owner of a file gives that file
 * away to someone else, the setuid bit will no longer be set.
```



```
 * If this happens, atrun will not execute the file
 */
#define ATMODE          (S_ISUID | S_IRUSR | S_IRGRP | S_IROTH)
#define ROOT            0       /* user-id of super-user */
#define MAXTRYS         100     /* max trys to create at job file */

#define BADTIME         "bad time specification"
#define BADQUEUE        "queue name must be a single character a-z"
#define NOTCQUEUE       "queue c is reserved for cron entries"
#define BADSHELL        "because your login shell isn't /usr/bin/sh,"\
                        "you can't use at"
#define WARNSHELL       "commands will be executed using %s\n"
#define CANTCD          "can't change directory to the at directory"
#define CANTCHOWN       "can't change the owner of your job to you"
#define CANTCHUID       "can't change user identifier"
#define CANTCREATE      "can't create a job for you"
#define INVALIDUSER     "you are not a valid user (no entry in /etc/passwd)"
#define NOOPENDIR       "can't open the at directory"
#define NOTALLOWED      "you are not authorized to use at.  Sorry."
#define USAGE\
        "usage: at [-c|-k|-s] [-m] [-f file] [-p project] [-q queuename] "\
            "-t time\n"\
        "       at [-c|-k|-s] [-m] [-f file] [-p project] [-q queuename] "\
            "timespec\n"\
        "       at -l [-p project] [-q queuename] [at_job_id...]\n"\
        "       at -r at_job_id ...\n"

#define FORMAT          "%a %b %e %H:%M:%S %Y"

static int leap(int);
static int atoi_for2(char *);
static int check_queue(char *, int);
static int list_jobs(int, char **, int, int);
static int remove_jobs(int, char **, char *);
static void usage(void);
static void catch(int);
static void copy(char *, FILE *, int);
static void atime(struct tm *, struct tm *);
static int not_this_project(char *);
static char *mkjobname(time_t);
static time_t parse_time(char *);
static time_t gtime(struct tm *);
static void escapestr(const char *);
#endif /* ! codereview */
void atabort(char *)__NORETURN;
void yyerror(void);
extern int yyparse(void);

extern void     audit_at_delete(char *, char *, int);
extern int      audit_at_create(char *, int);
extern int      audit_cron_is_anc_name(char *);
extern int      audit_cron_delete_anc_file(char *, char *);

/*
 * Error in getdate(3G)
 */
static char     *errlist[] = {
/* 0 */         "",
/* 1 */ "getdate: The DATEMSK environment variable is not set",
/* 2 */ "getdate: Error on \"open\" of the template file",
/* 3 */ "getdate: Error on \"stat\" of the template file",
/* 4 */ "getdate: The template file is not a regular file",
/* 5 */ "getdate: An error is encountered while reading the template",
/* 6 */ "getdate: Malloc(3C) failed",
/* 7 */ "getdate: There is no line in the template that matches the input",
/* 8 */ "getdate: Invalid input specification"
```

```
};

int             gmtflag = 0;
int             mday[12] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
uid_t           user;
struct  tm      *tp, at, rt;
static int      cshflag         = 0;
static int      kshflag         = 0;
static int      shflag          = 0;
static int      mflag           = 0;
static int      pflag           = 0;
static char     *Shell;
static char     *tfname;
static char     pname[80];
static char     pname1[80];
static short    jobtype = ATEVENT;      /* set to 1 if batch job */
extern char     *argp;
extern int      per_errno;
static projid_t project;

int
main(int argc, char **argv)
{
        FILE            *inputfile;
        int             i, fd;
        int             try = 0;
        int             fflag = 0;
        int             lflag = 0;
        int             qflag = 0;
        int             rflag = 0;
        int             tflag = 0;
        int             c;
        int             tflen;
        char            *file;
        char            *login;
        char            *job;
        char            *jobfile = NULL; /* file containing job to be run */
        char            argpbuf[LINE_MAX], timebuf[80];
        time_t          now;
        time_t          when = 0;
        struct  tm      *ct;
        char            *proj;
        struct project  prj, *pprj;
        char            mybuf[PROJECT_BUFSZ];
        char            ipbuf[PROJECT_BUFSZ];

        (void) setlocale(LC_ALL, "");
#if !defined(TEXT_DOMAIN)       /* Should be defined by cc -D */
#define TEXT_DOMAIN "SYS_TEST"  /* Use this only if it weren't */
#endif
        (void) textdomain(TEXT_DOMAIN);

        user = getuid();
        login = getuser(user);
        if (login == NULL) {
                if (per_errno == 2)
                        atabort(BADSHELL);
                else
                        atabort(INVALIDUSER);
        }

        if (!allowed(login, ATALLOW, ATDENY))
                atabort(NOTALLOWED);

        while ((c = getopt(argc, argv, "cklmsrf:p:q:t:")) != EOF)
                switch (c) {
```



```
                case 'c':
                        cshflag++;
                        break;
                case 'f':
                        fflag++;
                        jobfile = optarg;
                        break;
                case 'k':
                        kshflag++;
                        break;
                case 'l':
                        lflag++;
                        break;
                case 'm':
                        mflag++;
                        break;
                case 'p':
                        proj = optarg;
                        pprj = &prj;
                        if ((pprj = getprojbyname(proj, pprj,
                            (void *)&mybuf, sizeof (mybuf))) != NULL) {
                                project = pprj->pj_projid;
                                if (inproj(login, pprj->pj_name,
                                    (void *)&ipbuf, sizeof (ipbuf)))
                                        pflag++;
                                else {
                                        (void) fprintf(stderr,
                                            gettext("at: user %s is "
                                            "not a member of "
                                            "project %s (%d)\n"),
                                            login, pprj->pj_name,
                                            project);
                                        exit(2);
                                }
                                break;
                        }
                        pprj = &prj;
                        if (isdigit(proj[0]) &&
                            (pprj = getprojbyid(atoi(proj), pprj,
                            (void *)&mybuf, sizeof (mybuf))) != NULL) {
                                project = pprj->pj_projid;
                                if (inproj(login, pprj->pj_name,
                                    (void *)&ipbuf, sizeof (ipbuf)))
                                        pflag++;
                                else {
                                        (void) fprintf(stderr,
                                            gettext("at: user %s is "
                                            "not a member of "
                                            "project %s (%d)\n"),
                                            login, pprj->pj_name,
                                            project);
                                        exit(2);
                                }
                                break;
                        }
                        (void) fprintf(stderr, gettext("at: project "
                            "%s not found.\n"), proj);
                        exit(2);
                        break;
                case 'q':
                        qflag++;
                        if (optarg[1] != '\0')
                                atabort(BADQUEUE);
                        jobtype = *optarg - 'a';
                        if ((jobtype < 0) || (jobtype > 25))
                                atabort(BADQUEUE);
```

```
260                                 if (jobtype == 2)
261                                         atabort(NOTCQUEUE);
262                                 break;
263                         case 'r':
264                                 rflag++;
265                                 break;
266                         case 's':
267                                 shflag++;
268                                 break;
269                         case 't':
270                                 tflag++;
271                                 when = parse_time(optarg);
272                                 break;
273                         default:
274                                 usage();
275                 }

277         argc -= optind;
278         argv += optind;

280         if (lflag + rflag > 1)
281                 usage();

283         if (lflag) {
284                 if (cshflag || kshflag || shflag || mflag ||
285                     fflag || tflag || rflag)
286                         usage();
287                 return (list_jobs(argc, argv, qflag, jobtype));
288         }

290         if (rflag) {
291                 if (cshflag || kshflag || shflag || mflag ||
292                     fflag || tflag || qflag)
293                         usage();
294                 return (remove_jobs(argc, argv, login));
295         }

297         if ((argc + tflag == 0) && (jobtype != BATCHEVENT))
298                 usage();

300         if (cshflag + kshflag + shflag > 1)
301                 atabort("ambiguous shell request");

303         time(&now);

305         if (jobtype == BATCHEVENT)
306                 when = now;

308         if (when == 0) { /* figure out what time to run the job */
309                 int     argplen = sizeof (argpbuf) - 1;

311                 argpbuf[0] = '\0';
312                 argp = argpbuf;
313                 i = 0;
314                 while (i < argc) {
315                         /* guard against buffer overflow */
316                         argplen -= strlen(argv[i]) + 1;
317                         if (argplen < 0)
318                                 atabort(BADTIME);

320                         strcat(argp, argv[i]);
321                         strcat(argp, " ");
322                         i++;
323                 }
324                 if ((file = getenv("DATEMSK")) == 0 || file[0] == '\0') {
325                         tp = localtime(&now);
```



```
326                         /*
327                          * Fix for 1047182 - we have to let yyparse
328                          * check bounds on mday[] first, then fixup
329                          * the leap year case.
330                          */
331                         yyparse();

333                         mday[1] = 28 + leap(at.tm_year);

335                         if (at.tm_mday > mday[at.tm_mon])
336                                 atabort("bad date");

338                         atime(&at, &rt);
339                         when = gtime(&at);
340                         if (!gmtflag) {
341                                 when += timezone;
342                                 if (localtime(&when)->tm_isdst)
343                                         when -= (timezone-altzone);
344                         }
345                 } else {        /*   DATEMSK is set  */
346                         if ((ct = getdate(argpbuf)) == NULL)
347                                 atabort(errlist[getdate_err]);
348                         else
349                                 when = mktime(ct);
350                 }
351         }

353         if (when < now) /* time has already past */
354                 atabort("too late");

356         tflen = strlen(ATDIR) + 1 + strlen(TMPFILE) +
357             10 + 1; /* 10 for an INT_MAX pid */
358         tfname = xmalloc(tflen);
359         snprintf(tfname, tflen, "%s/%s%d", ATDIR, TMPFILE, getpid());

361         /* catch INT, HUP, TERM and QUIT signals */
362         if (signal(SIGINT, catch) == SIG_IGN)
363                 signal(SIGINT, SIG_IGN);
364         if (signal(SIGHUP, catch) == SIG_IGN)
365                 signal(SIGHUP, SIG_IGN);
366         if (signal(SIGQUIT, catch) == SIG_IGN)
367                 signal(SIGQUIT, SIG_IGN);
368         if (signal(SIGTERM, catch) == SIG_IGN)
369                 signal(SIGTERM, SIG_IGN);
370         if ((fd = open(tfname, O_CREAT|O_EXCL|O_WRONLY, ATMODE)) < 0)
371                 atabort(CANTCREATE);
372         if (chown(tfname, user, getgid()) == -1) {
373                 unlink(tfname);
374                 atabort(CANTCHOWN);
375         }
376         close(1);
377         dup(fd);
378         close(fd);
379         sprintf(pname, "%s", PROTO);
380         sprintf(pname1, "%s.%c", PROTO, 'a'+jobtype);

382         /*
383          * Open the input file with the user's permissions.
384          */
385         if (jobfile != NULL) {
386                 if ((seteuid(user) < 0) ||
387                     (inputfile = fopen(jobfile, "r")) == NULL) {
388                         unlink(tfname);
389                         fprintf(stderr, "at: %s: %s\n", jobfile, errmsg(errno));
390                         exit(1);
391                 }
```

```
392                 else
393                         seteuid(0);
394         } else
395                 inputfile = stdin;
396
397         copy(jobfile, inputfile, when);
398         while (rename(tfname, job = mkjobname(when)) == -1) {
399                 sleep(1);
400                 if (++try > MAXTRYS / 10) {
401                         unlink(tfname);
402                         atabort(CANTCREATE);
403                 }
404         }
405         unlink(tfname);
406         if (audit_at_create(job, 0))
407                 atabort(CANTCREATE);
408
409         cron_sendmsg(ADD, login, strrchr(job, '/')+1, AT);
410         if (per_errno == 2)
411                 fprintf(stderr, gettext(WARNSHELL), Shell);
412         cftime(timebuf, FORMAT, &when);
413         fprintf(stderr, gettext("job %s at %s\n"),
414             strrchr(job, '/')+1, timebuf);
415         if (when - MINUTE < HOUR)
416                 fprintf(stderr, gettext(
417                     "at: this job may not be executed at the proper time.\n"));
418         return (0);
419 }
420
421
422 static char *
423 mkjobname(t)
424 time_t t;
425 {
426         int i, fd;
427         char *name;
428
429         name = xmalloc(200);
430         for (i = 0; i < MAXTRYS; i++) {
431                 sprintf(name, "%s/%ld.%c", ATDIR, t, 'a'+jobtype);
432                 /* fix for 1099183, 1116833 - create file here, avoid race */
433                 if ((fd = open(name, O_CREAT | O_EXCL, ATMODE)) > 0) {
434                         close(fd);
435                         return (name);
436                 }
437                 t += 1;
438         }
439         atabort("queue full");
440         /* NOTREACHED */
441 }
442
443
444 static void
445 catch(int x)
446 {
447         unlink(tfname);
448         exit(1);
449 }
450
451
452 void
453 atabort(msg)
454 char *msg;
455 {
456         fprintf(stderr, "at: %s\n", gettext(msg));
```



```
457         exit(1);
458 }
459
460 int
461 yywrap(void)
462 {
463         return (1);
464 }
465
466 void
467 yyerror(void)
468 {
469         atabort(BADTIME);
470 }
471
472 /*
473  * add time structures logically
474  */
475 static void
476 atime(struct tm *a, struct tm *b)
477 {
478         if ((a->tm_sec += b->tm_sec) >= 60) {
479                 b->tm_min += a->tm_sec / 60;
480                 a->tm_sec %= 60;
481         }
482         if ((a->tm_min += b->tm_min) >= 60) {
483                 b->tm_hour += a->tm_min / 60;
484                 a->tm_min %= 60;
485         }
486         if ((a->tm_hour += b->tm_hour) >= 24) {
487                 b->tm_mday += a->tm_hour / 24;
488                 a->tm_hour %= 24;
489         }
490         a->tm_year += b->tm_year;
491         if ((a->tm_mon += b->tm_mon) >= 12) {
492                 a->tm_year += a->tm_mon / 12;
493                 a->tm_mon %= 12;
494         }
495         a->tm_mday += b->tm_mday;
496         mday[1] = 28 + leap(a->tm_year);
497         while (a->tm_mday > mday[a->tm_mon]) {
498                 a->tm_mday -= mday[a->tm_mon++];
499                 if (a->tm_mon > 11) {
500                         a->tm_mon = 0;
501                         mday[1] = 28 + leap(++a->tm_year);
502                 }
503         }
504
505 }
506
507 static int
508 leap(int year)
509 {
510         return (isleap(year + TM_YEAR_BASE));
511 }
512
513 /*
514  * return time from time structure
515  */
516 static time_t
517 gtime(tptr)
518 struct  tm *tptr;
519 {
520         int i;
521         long    tv;
522         int     dmsize[12] =
523         int     dmsize[12] =
```

```
524             {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};


527         tv = 0;
528         for (i = 1970; i != tptr->tm_year+TM_YEAR_BASE; i++)
529                 tv += (365 + isleap(i));
530                 /*
531                  * We call isleap since leap() adds
532                  * 1900 onto any value passed
533                  */
534
535         if (!leap(tptr->tm_year) && at.tm_mday == 29 && at.tm_mon == 1)
536                 atabort("bad date - not a leap year");
537
538         if ((leap(tptr->tm_year)) && tptr->tm_mon >= 2)
539                 ++tv;
540
541         for (i = 0; i < tptr->tm_mon; ++i)
542                 tv += dmsize[i];
543         tv += tptr->tm_mday - 1;
544         tv = 24 * tv + tptr->tm_hour;
545         tv = 60 * tv + tptr->tm_min;
546         tv = 60 * tv + tptr->tm_sec;
547         return (tv);
548 }
549
550 /*
551  * Escape a string to be used inside the job shell script.
552  */
553 static void
554 escapestr(const char *str)
555 {
556         char c;
557         putchar('\'');
558         while ((c = *str++) != '\0') {
559                 if (c != '\'')
560                         (void) putchar(c);
561                 else
562                         (void) printf("\'\\\'\'"); /* ' -> '\'' */
563         }
564         putchar('\'');
565 }
566
567 /*
568 #endif /* ! codereview */
569  * make job file from proto + stdin
570  */
571 static void
572 copy(char *jobfile, FILE *inputfile, int when)
573 {
574         int c;
575         FILE *pfp;
576         char *shell;
577         char    dirbuf[PATH_MAX + 1];
578         char    line[LINE_MAX];
579         char **ep;
580         mode_t um;
581         char *val;
582         extern char **environ;
583         uid_t realusr, effeusr;
584         int ttyinput;
585         int ulimit_flag = 0;
586         struct rlimit rlp;
587         struct project prj, *pprj;
588         char pbuf[PROJECT_BUFSZ];
589         char pbuf2[PROJECT_BUFSZ];
```



```
590         char *user;
591
592         /*
593          * Fix for 1099381:
594          * If the inputfile is from a tty, then turn on prompting, and
595          * put out a prompt now, instead of waiting for a lot of file
596          * activity to complete.
597          */
598         ttyinput = isatty(fileno(inputfile));
599         if (ttyinput) {
600                 fputs("at> ", stderr);
601                 fflush(stderr);
602         }
603
604         /*
605          * Fix for 1053807:
606          * Determine what shell we should use to run the job. If the user
607          * didn't explicitly request that his/her current shell be over-
608          * ridden (shflag or cshflag), then we use the current shell.
609          */
610         if (cshflag)
611                 Shell = shell = "/bin/csh";
612         else if (kshflag) {
613                 Shell = shell = "/bin/ksh";
614                 ulimit_flag = 1;
615         } else if (shflag) {
616                 Shell = shell = "/bin/sh";
617                 ulimit_flag = 1;
618         } else if (((Shell = val = getenv("SHELL")) != NULL) &&
619             (*val != '\0')) {
620                 shell = "$SHELL";
621                 if ((strstr(val, "/sh") != NULL) ||
622                     (strstr(val, "/ksh") != NULL))
623                         ulimit_flag = 1;
624         } else {
625                 /* SHELL is NULL or unset, therefore use default */
626                 Shell = shell = _PATH_BSHELL;
627                 ulimit_flag = 1;
628         }
629
630         printf(": %s job\n", jobtype ? "batch" : "at");
631         printf(": jobname: %.127s\n", (jobfile == NULL) ? "stdin" : jobfile);
632         printf(": notify by mail: %s\n", (mflag) ? "yes" : "no");
633
634         if (pflag) {
635                 (void) printf(": project: %d\n", project);
636         } else {
637                 /*
638                  * Check if current user is a member of current project.
639                  * This check is done here to avoid setproject() failure
640                  * later when the job gets executed.  If current user does
641                  * not belong to current project, user's default project
642                  * will be used instead.  This is achieved by not specifying
643                  * the project (": project: <project>\n") in the job file.
644                  */
645                 if ((user = getuser(getuid())) == NULL)
646                         atabort(INVALIDUSER);
647                 project = getprojid();
648                 pprj = getprojbyid(project, &prj, pbuf, sizeof (pbuf));
649                 if (pprj != NULL) {
650                         if (inproj(user, pprj->pj_name, pbuf2, sizeof (pbuf2)))
651                                 (void) printf(": project: %d\n", project);
652                 }
653         }
654
655         for (ep = environ; *ep; ep++) {
```

```
104                 if (strchr(*ep, '\'') != NULL)
105                         continue;
656                 if ((val = strchr(*ep, '=')) == NULL)
657                         continue;
658                 *val++ = '\0';
659                 (void) printf("export %s; %s=", *ep, *ep);
660                 escapestr(val);
661                 (void) putchar('\n');
109                 printf("export %s; %s='%s'\n", *ep, *ep, val);
662                 *--val = '=';
663         }
664         if ((pfp = fopen(pname1, "r")) == NULL &&
665             (pfp = fopen(pname, "r")) == NULL)
666                 atabort("no prototype");
667         /*
668          * Put in a line to run the proper shell using the rest of
669          * the file as input.  Note that 'exec'ing the shell will
670          * cause sh() to leave a /tmp/sh### file around. (1053807)
671          */
672         printf("%s << '...the rest of this file is shell input'\n", shell);

674         um = umask(0);
675         while ((c = getc(pfp)) != EOF) {
676                 if (c != '$')
677                         putchar(c);
678                 else switch (c =  getc(pfp)) {
679                 case EOF:
680                         goto out;
681                 case 'd':
682                         /*
683                          * Must obtain current working directory as the user
684                          */

686                         dirbuf[0] = '\0';
687                         realusr = getuid();
688                         effeusr = geteuid();
689                         /* change euid for getcwd */
690                         if (seteuid(realusr) < 0) {
691                                 atabort(CANTCHUID);
692                         }
693                         if (getcwd(dirbuf, sizeof (dirbuf)) == NULL) {
694                                 atabort(
695                                     "can't obtain current working directory");
696                         }
697                         /* change back afterwards */
698                         if (seteuid(effeusr) < 0) {
699                                 atabort(CANTCHUID);
700                         }
701                         escapestr(dirbuf);
149                         printf("%s", dirbuf);
702                         break;
703                 case 'm':
704                         printf("%o", um);
705                         break;
706                 case '<':
707                         if (ulimit_flag) {
708                                 if (getrlimit(RLIMIT_FSIZE, &rlp) == 0) {
709                                         if (rlp.rlim_cur == RLIM_INFINITY)
710                                                 printf("ulimit unlimited\n");
711                                         else
712                                                 printf("ulimit %lld\n",
713                                                     rlp.rlim_cur / 512);
714                                 }
715                         }
716                         /*
717                          * fix for 1113572 - use fputs() so that a
```



```
718                          * newline isn't appended to the one returned
719                          * with fgets(); 1099381 - prompt for input.
720                          */
721                         while (fgets(line, LINE_MAX, inputfile) != NULL) {
722                                 fputs(line, stdout);
723                                 if (ttyinput)
724                                         fputs("at> ", stderr);
725                         }
726                         if (ttyinput) /* clean up the final output */
727                                 fputs("<EOT>\n", stderr);
728                         break;
729                 case 't':
730                         printf(":%lu", when);
731                         break;
732                 default:
733                         putchar(c);
734                 }
735         }
736 out:
737         fclose(pfp);
738         fflush(NULL);
739 }
______unchanged_portion_omitted_
```