

new/usr/src/cmd/cron/at.c

1

```
*****
23396 Wed Aug 26 09:09:44 2015
new/usr/src/cmd/cron/at.c
5433 at(1) doesn't properly handle being invoked from a path containing spaces
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 2011 Gary Mills
24  *
25  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
26  * Use is subject to license terms.
27  */

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

32 #include <sys/resource.h>
33 #include <sys/stat.h>
34 #include <sys/types.h>

36 #include <dirent.h>
37 #include <string.h>
38 #include <stdlib.h>
39 #include <fcntl.h>
40 #include <pwd.h>
41 #include <stdio.h>
42 #include <ctype.h>
43 #include <time.h>
44 #include <signal.h>
45 #include <errno.h>
46 #include <limits.h>
47 #include <ulimit.h>
48 #include <unistd.h>
49 #include <locale.h>
50 #include <libintl.h>
51 #include <tzfile.h>
52 #include <project.h>
53 #include <paths.h>

55 #include "cron.h"

57 #define TMPFILE      "_at" /* prefix for temporary files */
58 /*
59  * Mode for creating files in ATDIR.
60  * Setuid bit on so that if an owner of a file gives that file
61  * away to someone else, the setuid bit will no longer be set.
```

new/usr/src/cmd/cron/at.c

2

```
62 * If this happens, atrun will not execute the file
63 */
64 #define ATMODE      (S_ISUID | S_IRUSR | S_IRGRP | S_IROTH)
65 #define ROOT        0 /* user-id of super-user */
66 #define MAXTRYS     100 /* max trys to create at job file */

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

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

91 static int leap(int);
92 static int atoi_for2(char *);
93 static int check_queue(char *, int);
94 static int list_jobs(int, char **, int, int);
95 static int remove_jobs(int, char **, char *);
96 static void usage(void);
97 static void catch(int);
98 static void copy(char *, FILE *, int);
99 static void atime(struct tm *, struct tm *);
100 static int not_this_project(char *);
101 static char *mkjobname(time_t);
102 static time_t parse_time(char *);
103 static time_t gtime(struct tm *);
104 static void escapestr(const char *);
105 #endif /* ! codereview */
106 void atabort(char *)__NORETURN;
107 void yyerror(void);
108 extern int yyparse(void);

110 extern void audit_at_delete(char *, char *, int);
111 extern int audit_at_create(char *, int);
112 extern int audit_cron_is_anc_name(char *);
113 extern int audit_cron_delete_anc_file(char *, char *);

115 /*
116  * Error in getdate(3G)
117  */
118 static char *errlist[] = {
119 /* 0 */      "",
120 /* 1 */      "getdate: The DATEMSK environment variable is not set",
121 /* 2 */      "getdate: Error on \"open\" of the template file",
122 /* 3 */      "getdate: Error on \"stat\" of the template file",
123 /* 4 */      "getdate: The template file is not a regular file",
124 /* 5 */      "getdate: An error is encountered while reading the template",
125 /* 6 */      "getdate: Malloc(3C) failed",
126 /* 7 */      "getdate: There is no line in the template that matches the input",
127 /* 8 */      "getdate: Invalid input specification"
```

```

128 };
129
130 int          gmtflag = 0;
131 int          mday[12] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
132 uid_t       user;
133 struct tm    *tp, at, rt;
134 static int   cshflag      = 0;
135 static int   kshflag      = 0;
136 static int   shflag       = 0;
137 static int   mflag        = 0;
138 static int   pflag        = 0;
139 static char  *shell;
140 static char  *tfname;
141 static char  pname[80];
142 static char  pname1[80];
143 static short jobtype = ATEVENT; /* set to 1 if batch job */
144 extern char  *argp;
145 extern int   per_errno;
146 static projid_t project;
147
148 int
149 main(int argc, char **argv)
150 {
151     FILE          *inputfile;
152     int           i, fd;
153     int           try = 0;
154     int           fflag = 0;
155     int           lflag = 0;
156     int           qflag = 0;
157     int           rflag = 0;
158     int           tflag = 0;
159     int           c;
160     int           tflen;
161     char          *file;
162     char          *login;
163     char          *job;
164     char          *jobfile = NULL; /* file containing job to be run */
165     char          argpbuf[LINE_MAX], timebuf[80];
166     time_t        now;
167     time_t        when = 0;
168     struct tm     *ct;
169     char          *proj;
170     struct project prj, *pprj;
171     char          mybuf[PROJECT_BUFSZ];
172     char          ipbuf[PROJECT_BUFSZ];
173
174     (void) setlocale(LC_ALL, "");
175     #if !defined(TEXT_DOMAIN) /* Should be defined by cc -D */
176     #define TEXT_DOMAIN "SYS_TEST" /* Use this only if it weren't */
177     #endif
178     (void) textdomain(TEXT_DOMAIN);
179
180     user = getuid();
181     login = getuser();
182     if (login == NULL) {
183         if (per_errno == 2)
184             atabort(BADSHELL);
185         else
186             atabort(INVALIDUSER);
187     }
188
189     if (!allowed(login, ATALLOW, ATDENY))
190         atabort(NOTALLOWED);
191
192     while ((c = getopt(argc, argv, "cklmsrp:q:t:")) != EOF)
193         switch (c) {

```

```

194         case 'c':
195             cshflag++;
196             break;
197         case 'f':
198             fflag++;
199             jobfile = optarg;
200             break;
201         case 'k':
202             kshflag++;
203             break;
204         case 'l':
205             lflag++;
206             break;
207         case 'm':
208             mflag++;
209             break;
210         case 'p':
211             proj = optarg;
212             pprj = &prj;
213             if ((pprj = getprojbyname(proj, pprj,
214                 (void *)&mybuf, sizeof(mybuf))) != NULL) {
215                 project = pprj->pj_projid;
216                 if (inproj(login, pprj->pj_name,
217                     (void *)&ipbuf, sizeof(ipbuf)))
218                     pflag++;
219             }
220             else {
221                 (void) fprintf(stderr,
222                     gettext("at: user %s is "
223                         "not a member of "
224                         "project %s (%d)\n"),
225                     login, pprj->pj_name,
226                     project);
227                 exit(2);
228             }
229             break;
230         }
231         pprj = &prj;
232         if (isdigit(proj[0]) &&
233             (pprj = getprojbyid(atoi(proj), pprj,
234                 (void *)&mybuf, sizeof(mybuf))) != NULL) {
235             project = pprj->pj_projid;
236             if (inproj(login, pprj->pj_name,
237                 (void *)&ipbuf, sizeof(ipbuf)))
238                 pflag++;
239             else {
240                 (void) fprintf(stderr,
241                     gettext("at: user %s is "
242                         "not a member of "
243                         "project %s (%d)\n"),
244                     login, pprj->pj_name,
245                     project);
246                 exit(2);
247             }
248             break;
249         }
250         (void) fprintf(stderr, gettext("at: project "
251             "%s not found.\n"), proj);
252         exit(2);
253         break;
254     case 'q':
255         qflag++;
256         if (optarg[1] != '\0')
257             atabort(BADQUEUE);
258         jobtype = *optarg - 'a';
259         if ((jobtype < 0) || (jobtype > 25))
260             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;

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);

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 }

422 static char *
423 mkjobname(t)
424 time_t t;
425 {
426     int i, fd;
427     char *name;

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 }

444 static void
445 catch(int x)
446 {
447     unlink(tfname);
448     exit(1);
449 }

452 void
453 atabort(msg)
454 char *msg;
455 {
456     fprintf(stderr, "at: %s\n", gettext(msg));

```

```

458     exit(1);
459 }

461 int
462 yywrap(void)
463 {
464     return (1);
465 }

467 void
468 yyerror(void)
469 {
470     atabort(BADTIME);
471 }

473 /*
474  * add time structures logically
475  */
476 static void
477 atime(struct tm *a, struct tm *b)
478 {
479     if ((a->tm_sec += b->tm_sec) >= 60) {
480         b->tm_min += a->tm_sec / 60;
481         a->tm_sec %= 60;
482     }
483     if ((a->tm_min += b->tm_min) >= 60) {
484         b->tm_hour += a->tm_min / 60;
485         a->tm_min %= 60;
486     }
487     if ((a->tm_hour += b->tm_hour) >= 24) {
488         b->tm_mday += a->tm_hour / 24;
489         a->tm_hour %= 24;
490     }
491     a->tm_year += b->tm_year;
492     if ((a->tm_mon += b->tm_mon) >= 12) {
493         a->tm_year += a->tm_mon / 12;
494         a->tm_mon %= 12;
495     }
496     a->tm_mday += b->tm_mday;
497     mday[1] = 28 + leap(a->tm_year);
498     while (a->tm_mday > mday[a->tm_mon]) {
499         a->tm_mday -= mday[a->tm_mon++];
500         if (a->tm_mon > 11) {
501             a->tm_mon = 0;
502             mday[1] = 28 + leap(++a->tm_year);
503         }
504     }
506 }

508 static int
509 leap(int year)
510 {
511     return (isleap(year + TM_YEAR_BASE));
512 }

514 /*
515  * return time from time structure
516  */
517 static time_t
518 gtime(tptr)
519 struct tm *tptr;
520 {
521     int i;
522     long tv;
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      */

535     if (!leap(tptr->tm_year) && at.tm_mday == 29 && at.tm_mon == 1)
536         atabort("bad date - not a leap year");

538     if ((leap(tptr->tm_year)) && tptr->tm_mon >= 2)
539         ++tv;

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 }

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     (void) putchar('\'');
558     while ((c = *str++) != '\0') {
559         if (c != '\'')
560             (void) putchar(c);
561         else
562             (void) printf("\\'"); /* ' -> \' */
563     }
564     (void) putchar('\'');
565 }

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, effusr;
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;

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     }

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     }

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");

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     }

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