```
**********************************************************
   20520 Wed Jan 30 11:23:03 2019
new/usr/src/cmd/who/who.c
10142 smatch fix for who
**********************************************************
```

```
/*
 * 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) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */


/*
 * Copyright (c) 2013 Gary Mills
 *
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 *      This program analyzes information found in /var/adm/utmpx
 *
 *      Additionally information is gathered from /etc/inittab
 *      if requested.
 *
 *
 *      Syntax:
 *
 *              who am i        Displays info on yourself
 *
 *              who -a          Displays information about All
 *                              entries in /var/adm/utmpx
 *
 *              who -b          Displays info on last boot
 *
 *              who -d          Displays info on DEAD PROCESSES
 *
 *              who -H          Displays HEADERS for output
 *
 *              who -l          Displays info on LOGIN entries
 *
 *              who -m          Same as who am i
 *
 *              who -p          Displays info on PROCESSES spawned by init
 *
```



```
 *              who -q          Displays short information on
 *                              current users who LOGGED ON
 *
 *              who -r          Displays info of current run-level
 *
 *              who -s          Displays requested info in SHORT form
 *
 *              who -t          Displays info on TIME changes
 *
 *              who -T          Displays writeability of each user
 *                              (+ writeable, - non-writeable, ? hung)
 *
 *              who -u          Displays LONG info on users
 *                              who have LOGGED ON
 */

#define         DATE_FMT        "%b %e %H:%M"

/*
 *  %b  Abbreviated month name
 *  %e  Day of month
 *  %H  hour (24-hour clock)
 *  %M  minute
 */
#include        <errno.h>
#include        <fcntl.h>
#include        <stdio.h>
#include        <string.h>
#include        <sys/types.h>
#include        <unistd.h>
#include        <stdlib.h>
#include        <sys/stat.h>
#include        <time.h>
#include        <utmpx.h>
#include        <locale.h>
#include        <pwd.h>
#include        <limits.h>

static void process(void);
static void ck_file(char *);
static void dump(void);

static struct   utmpx *utmpp;   /* pointer for getutxent()      */

/*
 * Use the full lengths from utmpx for user and line.
 */
#define NMAX    (sizeof (utmpp->ut_user))
#define LMAX    (sizeof (utmpp->ut_line))

/* Print minimum field widths. */
#define LOGIN_WIDTH     8
#define LINE_WIDTH      12

static char     comment[BUFSIZ]; /* holds inittab comment       */
static char     errmsg[BUFSIZ]; /* used in snprintf for errors  */
static int      fildes;         /* file descriptor for inittab  */
static int      Hopt = 0;       /* 1 = who -H                   */
static char     *inittab;       /* ptr to inittab contents      */
static char     *iinit;         /* index into inittab           */
static int      justme = 0;     /* 1 = who am i                 */
static struct   tm *lptr;       /* holds user login time        */
static char     *myname;        /* pointer to invoker's name    */
static char     *mytty;         /* holds device user is on      */
static char     nameval[sizeof (utmpp->ut_user) + 1]; /*  invoker's name */
static int      number = 8;     /* number of users per -q line  */
```

```
128 static int      optcnt = 0;     /* keeps count of options       */
129 static char     outbuf[BUFSIZ]; /* buffer for output            */
130 static char     *program;       /* holds name of this program   */
131 #ifdef  XPG4
132 static int      aopt = 0;       /* 1 = who -a                   */
133 static int      dopt = 0;       /* 1 = who -d                   */
134 #endif  /* XPG4 */
135 static int      qopt = 0;       /* 1 = who -q                   */
136 static int      sopt = 0;       /* 1 = who -s                   */
137 static struct   stat stbuf;     /* area for stat buffer         */
138 static struct   stat *stbufp;   /* ptr to structure             */
139 static int      terse = 1;      /* 1 = print terse msgs         */
140 static int      Topt = 0;       /* 1 = who -T                   */
141 static time_t   timnow;         /* holds current time           */
142 static int      totlusrs = 0;   /* cntr for users on system     */
143 static int      uopt = 0;       /* 1 = who -u                   */
144 static char     user[sizeof (utmpp->ut_user) + 1]; /* holds user name */
145 static int      validtype[UTMAXTYPE+1]; /* holds valid types    */
146 static int      wrap;           /* flag to indicate wrap        */
147 static char     time_buf[128];  /* holds date and time string   */
148 static char     *end;           /* used in strtol for end pointer */

150 int
151 main(int argc, char **argv)
152 {
153         int     goerr = 0;      /* non-zero indicates cmd error */
154         int     i;
155         int     optsw;          /* switch for while of getopt() */

157         (void) setlocale(LC_ALL, "");

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

164         validtype[USER_PROCESS] = 1;
165         validtype[EMPTY] = 0;
166         stbufp = &stbuf;

168         /*
169          *      Strip off path name of this command
170          */
171         for (i = strlen(argv[0]); i >= 0 && argv[0][i] != '/'; --i)
172                 ;
173         if (i >= 0)
174                 argv[0] += i+1;
175         program = argv[0];

177         /*
178          *      Buffer stdout for speed
179          */
180         setbuf(stdout, outbuf);

182         /*
183          *      Retrieve options specified on command line
184          *      XCU4 - add -m option
185          */
186         while ((optsw = getopt(argc, argv, "abdHlmn:pqrstTu")) != EOF) {
187                 optcnt++;
188                 switch (optsw) {

190                 case 'a':
191                         optcnt += 7;
192                         validtype[BOOT_TIME] = 1;
193                         validtype[DEAD_PROCESS] = 1;
```



```
194                         validtype[LOGIN_PROCESS] = 1;
195                         validtype[INIT_PROCESS] = 1;
196                         validtype[RUN_LVL] = 1;
197                         validtype[OLD_TIME] = 1;
198                         validtype[NEW_TIME] = 1;
199                         validtype[USER_PROCESS] = 1;
200 #ifdef  XPG4
201                         aopt = 1;
202 #endif  /* XPG4 */
203                         uopt = 1;
204                         Topt = 1;
205                         if (!sopt) terse = 0;
206                         break;

208                 case 'b':
209                         validtype[BOOT_TIME] = 1;
210                         if (!uopt) validtype[USER_PROCESS] = 0;
211                         break;

213                 case 'd':
214                         validtype[DEAD_PROCESS] = 1;
215                         if (!uopt) validtype[USER_PROCESS] = 0;
216 #ifdef  XPG4
217                         dopt = 1;
218 #endif  /* XPG4 */
219                         break;

221                 case 'H':
222                         optcnt--; /* Don't count Header */
223                         Hopt = 1;
224                         break;

226                 case 'l':
227                         validtype[LOGIN_PROCESS] = 1;
228                         if (!uopt) validtype[USER_PROCESS] = 0;
229                         terse = 0;
230                         break;
231                 case 'm':               /* New XCU4 option */
232                         justme = 1;
233                         break;

235                 case 'n':
236                         errno = 0;
237                         number = strtol(optarg, &end, 10);
238                         if (errno != 0 || *end != '\0') {
239                                 (void) fprintf(stderr, gettext(
240                                     "%s: Invalid numeric argument\n"),
241                                     program);
242                                 exit(1);
243                         }
244                         if (number < 1) {
245                                 (void) fprintf(stderr, gettext(
246                                     "%s: Number of users per line must "
247                                     "be at least 1\n"), program);
248                                 exit(1);
249                         }
250                         break;

252                 case 'p':
253                         validtype[INIT_PROCESS] = 1;
254                         if (!uopt) validtype[USER_PROCESS] = 0;
255                         break;

257                 case 'q':
258                         qopt = 1;
259                         break;
```

```
261                 case 'r':
262                         validtype[RUN_LVL] = 1;
263                         terse = 0;
264                         if (!uopt) validtype[USER_PROCESS] = 0;
265                         break;

267                 case 's':
268                         sopt = 1;
269                         terse = 1;
270                         break;

272                 case 't':
273                         validtype[OLD_TIME] = 1;
274                         validtype[NEW_TIME] = 1;
275                         if (!uopt) validtype[USER_PROCESS] = 0;
276                         break;

278                 case 'T':
279                         Topt = 1;
280 #ifdef  XPG4
281                         terse = 1;      /* XPG4 requires -T */
282 #else   /* XPG4 */
283                         terse = 0;
284 #endif  /* XPG4 */
285                         break;

287                 case 'u':
288                         uopt = 1;
289                         validtype[USER_PROCESS] = 1;
290                         if (!sopt) terse = 0;
291                         break;

293                 case '?':
294                         goerr++;
295                         break;
296                 default:
297                         break;
298                 }
299         }
300 #ifdef  XPG4
301         /*
302          * XCU4 changes - check for illegal sopt, Topt & aopt combination
303          */
304         if (sopt == 1) {
305                 terse = 1;
306                 if (Topt == 1 || aopt == 1)
307                         goerr++;
308         }
309 #endif  /* XPG4 */

311         if (goerr > 0) {
312 #ifdef  XPG4
313                 /*
314                  * XCU4 - slightly different usage with -s -a & -T
315                  */
316                 (void) fprintf(stderr, gettext("\nUsage:\t%s"), program);
317                 (void) fprintf(stderr,
318                     gettext(" -s [-bdHlmpqrtu] [utmpx_like_file]\n"));

320                 (void) fprintf(stderr, gettext(
321                     "\t%s [-abdHlmpqrtTu] [utmpx_like_file]\n"), program);
322 #else   /* XPG4 */
323                 (void) fprintf(stderr, gettext(
324                     "\nUsage:\t%s [-abdHlmpqrstTu] [utmpx_like_file]\n"),
325                     program);
```



```
326 #endif  /* XPG4 */
327                 (void) fprintf(stderr,
328                     gettext("\t%s -q [-n x] [utmpx_like_file]\n"), program);
329                 (void) fprintf(stderr, gettext("\t%s [am i]\n"), program);
330                 /*
331                  * XCU4 changes - be explicit with "am i" options
332                  */
333                 (void) fprintf(stderr, gettext("\t%s [am I]\n"), program);
334                 (void) fprintf(stderr, gettext(
335                     "a\tall (bdlprtu options)\n"));
336                 (void) fprintf(stderr, gettext("b\tboot time\n"));
337                 (void) fprintf(stderr, gettext("d\tdead processes\n"));
338                 (void) fprintf(stderr, gettext("H\tprint header\n"));
339                 (void) fprintf(stderr, gettext("l\tlogin processes\n"));
340                 (void) fprintf(stderr, gettext(
341                     "n #\tspecify number of users per line for -q\n"));
342                 (void) fprintf(stderr,
343                     gettext("p\tprocesses other than getty or users\n"));
344                 (void) fprintf(stderr, gettext("q\tquick %s\n"), program);
345                 (void) fprintf(stderr, gettext("r\trun level\n"));
346                 (void) fprintf(stderr, gettext(
347                 "s\tshort form of %s (no time since last output or pid)\n"),
348                     program);
349                 (void) fprintf(stderr, gettext("t\ttime changes\n"));
350                 (void) fprintf(stderr, gettext(
351                     "T\tstatus of tty (+ writable, - not writable, "
352                     "? hung)\n"));
353                 (void) fprintf(stderr, gettext("u\tuseful information\n"));
354                 (void) fprintf(stderr,
355                     gettext("m\tinformation only about current terminal\n"));
356                 (void) fprintf(stderr, gettext(
357                     "am i\tinformation about current terminal "
358                     "(same as -m)\n"));
359                 (void) fprintf(stderr, gettext(
360                     "am I\tinformation about current terminal "
361                     "(same as -m)\n"));
362                 exit(1);
363         }

365         /*
366          * XCU4: If -q option ignore all other options
367          */
368         if (qopt == 1) {
369                 Hopt = 0;
370                 sopt = 0;
371                 Topt = 0;
372                 uopt = 0;
373                 justme = 0;
374                 validtype[ACCOUNTING] = 0;
375                 validtype[BOOT_TIME] = 0;
376                 validtype[DEAD_PROCESS] = 0;
377                 validtype[LOGIN_PROCESS] = 0;
378                 validtype[INIT_PROCESS] = 0;
379                 validtype[RUN_LVL] = 0;
380                 validtype[OLD_TIME] = 0;
381                 validtype[NEW_TIME] = 0;
382                 validtype[USER_PROCESS] = 1;
383         }

385         if (argc == optind + 1) {
386                 optcnt++;
387                 ck_file(argv[optind]);
388                 (void) utmpxname(argv[optind]);
389         }

391         /*
```

```
392          *      Test for 'who am i' or 'who am I'
393          *      XCU4 - check if justme was already set by -m option
394          */
395         if (justme == 1 || (argc == 3 && strcmp(argv[1], "am") == 0 &&
396             ((argv[2][0] == 'i' || argv[2][0] == 'I') &&
397             argv[2][1] == '\0'))) {
398                 justme = 1;
399                 myname = nameval;
400                 (void) cuserid(myname);
401                 if ((mytty = ttyname(fileno(stdin))) == NULL &&
402                     (mytty = ttyname(fileno(stdout))) == NULL &&
403                     (mytty = ttyname(fileno(stderr))) == NULL) {
404                         (void) fprintf(stderr, gettext(
405                         "Must be attached to terminal for 'am I' option\n"));
406                         (void) fflush(stderr);
407                         exit(1);
408                 } else
409                         mytty += 5; /* bump past "/dev/" */
410         }

412         if (!terse) {
413                 if (Hopt)
414                         (void) printf(gettext(
415         "NAME       LINE         TIME          IDLE    PID  COMMENTS\n"));

417                 timnow = time(0);

419                 if ((fildes = open("/etc/inittab",
420                     O_NONBLOCK|O_RDONLY)) == -1) {
421                         (void) snprintf(errmsg, sizeof (errmsg),
422                             gettext("%s: Cannot open /etc/inittab"), program);
423                         perror(errmsg);
424                         exit(errno);
425                 }

427                 if (fstat(fildes, stbufp) == -1) {
428                         (void) snprintf(errmsg, sizeof (errmsg),
429                             gettext("%s: Cannot stat /etc/inittab"), program);
430                         perror(errmsg);
431                         exit(errno);
432                 }

434                 if ((inittab = malloc(stbufp->st_size + 1)) == NULL) {
435                         (void) snprintf(errmsg, sizeof (errmsg),
436                             gettext("%s: Cannot allocate %ld bytes"),
437                             program, stbufp->st_size);
438                         perror(errmsg);
439                         exit(errno);
440                 }

442                 if (read(fildes, inittab, stbufp->st_size)
443                     != stbufp->st_size) {
444                         (void) snprintf(errmsg, sizeof (errmsg),
445                             gettext("%s: Error reading /etc/inittab"),
446                             program);
447                         perror(errmsg);
448                         exit(errno);
449                 }

451                 inittab[stbufp->st_size] = '\0';
452                 iinit = inittab;
453         } else {
454                 if (Hopt) {
455 #ifdef  XPG4
456                         if (dopt) {
457                                 (void) printf(gettext(
```



```
458                         "NAME       LINE         TIME           COMMENTS\n"));
459                         } else {
460                                 (void) printf(
461                                     gettext("NAME       LINE         TIME\n"));
462                         }
463 #else   /* XPG4 */
464                         (void) printf(
465                             gettext("NAME       LINE         TIME\n"));
466 #endif  /* XPG4 */
467                 }
468         }
469         process();

471         /*
472          *      'who -q' requires EOL upon exit,
473          *      followed by total line
474          */
475         if (qopt)
476                 (void) printf(gettext("\n# users=%d\n"), totlusrs);
477         return (0);
478 }

480 static void
481 dump()
482 {
483         char    device[sizeof (utmpp->ut_line) + 1];
484         time_t hr;
485         time_t  idle;
486         time_t min;
487         char    path[sizeof (utmpp->ut_line) + 6];
488         int     pexit;
489         int     pterm;
490         int     rc;
491         char    w;      /* writeability indicator */

493         /*
494          * Get and check user name
495          */
496         if (utmpp->ut_user[0] == '\0')
497                 (void) strcpy(user, "   .");
498         else {
499                 (void) strncpy(user, utmpp->ut_user, sizeof (user));
500                 user[sizeof (user) - 1] = '\0';
501         }
502         totlusrs++;

504         /*
505          * Do print in 'who -q' format
506          */
507         if (qopt) {
508                 /*
509                  * XCU4 - Use non user macro for correct user count
510                  */
511                 if (((totlusrs - 1) % number) == 0 && totlusrs > 1)
512                         (void) printf("\n");
513                 (void) printf("%-*.*s ", LOGIN_WIDTH, NMAX, user);
514                 return;
515         }


518         pexit = (int)' ';
519         pterm = (int)' ';

521         /*
522          *      Get exit info if applicable
523          */
```

```
524         if (utmpp->ut_type == RUN_LVL || utmpp->ut_type == DEAD_PROCESS) {
525                 pterm = utmpp->ut_exit.e_termination;
526                 pexit = utmpp->ut_exit.e_exit;
527         }
528
529         /*
530          *      Massage ut_xtime field
531          */
532         lptr = localtime(&utmpp->ut_xtime);
533         (void) strftime(time_buf, sizeof (time_buf),
534             dcgettext(NULL, DATE_FMT, LC_TIME), lptr);
535
536         /*
537          *      Get and massage device
538          */
539         if (utmpp->ut_line[0] == '\0')
540                 (void) strcpy(device, "     .");
541         else {
542                 (void) strncpy(device, utmpp->ut_line,
543                     sizeof (utmpp->ut_line));
544                 device[sizeof (utmpp->ut_line)] = '\0';
545         }
546
547         /*
548          *      Get writeability if requested
549          *      XCU4 - only print + or - for user processes
550          */
551         if (Topt && (utmpp->ut_type == USER_PROCESS)) {
552                 w = '-';
553                 (void) strcpy(path, "/dev/");
554                 (void) strncpy(path + 5, utmpp->ut_line,
555                     sizeof (utmpp->ut_line));
556                 path[5 + sizeof (utmpp->ut_line)] = '\0';
557
558                 if ((rc = stat(path, stbufp)) == -1) w = '?';
559                 else if ((stbufp->st_mode & S_IWOTH) ||
560                     (stbufp->st_mode & S_IWGRP))  /* Check group & other */
561                         w = '+';
562
563         } else
564                 w = ' ';
565
566         /*
567          *      Print the TERSE portion of the output
568          */
569         (void) printf("%-*.*s %c %-12s %s", LOGIN_WIDTH, NMAX, user,
570             w, device, time_buf);
571
572         if (!terse) {
573                 /*
574                  *      Stat device for idle time
575                  *      (Don't complain if you can't)
576                  */
577                 rc = -1;
578                 if (utmpp->ut_type == USER_PROCESS) {
579                         (void) strcpy(path, "/dev/");
580                         (void) strncpy(path + 5, utmpp->ut_line,
581                             sizeof (utmpp->ut_line));
582                         path[5 + sizeof (utmpp->ut_line)] = '\0';
583                         rc = stat(path, stbufp);
584                 }
585                 if (rc != -1) {
586                         idle = timnow - stbufp->st_mtime;
587                         hr = idle/3600;
588                         min = (unsigned)(idle/60)%60;
589                         if (hr == 0 && min == 0)
```



```
590                                 (void) printf(gettext("   .  "));
591                         else {
592                                 if (hr < 24)
593                                         (void) printf(" %2d:%2.2d", (int)hr,
594                                             (int)min);
595                                 else
596                                         (void) printf(gettext("  old "));
597                         }
598                 }
599
600                 /*
601                  *      Add PID for verbose output
602                  */
603                 if (utmpp->ut_type != BOOT_TIME &&
604                     utmpp->ut_type != RUN_LVL &&
605                     utmpp->ut_type != ACCOUNTING)
606                         (void) printf("  %5ld", utmpp->ut_pid);
607
608                 /*
609                  *      Handle /etc/inittab comment
610                  */
611                 if (utmpp->ut_type == DEAD_PROCESS) {
612                         (void) printf(gettext("  id=%4.4s "),
613                             utmpp->ut_id);
614                         (void) printf(gettext("term=%-3d "), pterm);
615                         (void) printf(gettext("exit=%d  "), pexit);
616                 } else if (utmpp->ut_type != INIT_PROCESS) {
617                         /*
618                          *      Search for each entry in inittab
619                          *      string. Keep our place from
620                          *      search to search to try and
621                          *      minimize the work. Wrap once if needed
622                          *      for each entry.
623                          */
624                         wrap = 0;
625                         /*
626                          *      Look for a line beginning with
627                          *      utmpp->ut_id
628                          */
629                         while ((rc = strncmp(utmpp->ut_id, iinit,
630                             strcspn(iinit, ":"))) != 0) {
631                                 for (; *iinit != '\n'; iinit++)
632                                         ;
633                                 iinit++;
634
635                                 /*
636                                  *      Wrap once if necessary to
637                                  *      find entry in inittab
638                                  */
639                                 if (*iinit == '\0') {
640                                         if (!wrap) {
641                                                 iinit = inittab;
642                                                 wrap = 1;
643                                         }
644                                 }
645                         }
646
647                         if (*iinit != '\0') {
648                                 /*
649                                  *      We found our entry
650                                  */
651                                 for (iinit++; *iinit != '#' &&
652                                     *iinit != '\n'; iinit++)
653                                         ;
654                                 if (*iinit == '#') {
655                                         for (iinit++; *iinit == ' ' ||
```

```
656                                         *iinit == '\t'; iinit++)
657                                             ;
658                                     for (rc = 0; *iinit != '\n'; iinit++)
659                                             comment[rc++] = *iinit;
660                                     comment[rc] = '\0';
661                                 } else
662                                     (void) strcpy(comment, " ");

664                                 (void) printf("  %s", comment);
665                             } else
666                                 iinit = inittab;        /* Reset pointer */
667                     }
668                     if (utmpp->ut_type == INIT_PROCESS)
669                             (void) printf(gettext("  id=%4.4s"), utmpp->ut_id);
670             }
671 #ifdef  XPG4
672             else
673                     if (dopt && utmpp->ut_type == DEAD_PROCESS) {
674                             (void) printf(gettext("\tterm=%-3d "), pterm);
675                             (void) printf(gettext("exit=%d  "), pexit);
676                     }
677 #endif  /* XPG4 */


680             /*
681              *      Handle RUN_LVL process - If no alt. file - Only one!
682              */
683             if (utmpp->ut_type == RUN_LVL) {
684                     (void) printf("     %c  %5ld  %c", pterm, utmpp->ut_pid,
685                         pexit);
686                     if (optcnt == 1 && !validtype[USER_PROCESS]) {
687                             (void) printf("\n");
688                             exit(0);
689                     }
690             }

692             /*
693              *      Handle BOOT_TIME process -  If no alt. file - Only one!
694              */
695             if (utmpp->ut_type == BOOT_TIME) {
696                     if (optcnt == 1 && !validtype[USER_PROCESS]) {
697                             (void) printf("\n");
698                             exit(0);
699                     }
700             }

702             /*
703              *      Get remote host from utmpx structure
704              */
705             if (utmpp->ut_host[0])
701             if (utmpp && utmpp->ut_host[0])
706                     (void) printf("\t(%.*s)", sizeof (utmpp->ut_host),
707                         utmpp->ut_host);

709             /*
710              *      Now, put on the trailing EOL
711              */
712             (void) printf("\n");
713 }
```

_____***unchanged_portion_omitted_***