```
**********************************************************
   13281 Mon May 12 22:34:33 2014
new/usr/src/cmd/printf/printf.c
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
/*
 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
 * Copyright 2010 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 1989, 1993
 *      The Regents of the University of California.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include <sys/types.h>

#include <err.h>
#include <errno.h>
#include <inttypes.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <alloca.h>
#include <ctype.h>
#include <locale.h>
#include <note.h>

#define warnx1(a, b, c)         warnx(a)
#define warnx2(a, b, c)         warnx(a, b)
#define warnx3(a, b, c)         warnx(a, b, c)

#define PTRDIFF(x, y)   ((uintptr_t)(x) - (uintptr_t)(y))

#define _(x)    gettext(x)

#define PF(f, func) do {                                                \
        char *b = NULL;                                                 \
        int dollar = 0;                                                 \
        if (*f == '$')  {                                               \
```



```
                dollar++;                                               \
                *f = '%';                                               \
        }                                                               \
        if (havewidth)                                                  \
                if (haveprec)                                           \
                        (void) asprintf(&b, f, fieldwidth, precision, func); \
                else                                                    \
                        (void) asprintf(&b, f, fieldwidth, func);       \
        else if (haveprec)                                              \
                (void) asprintf(&b, f, precision, func);                \
        else                                                            \
                (void) asprintf(&b, f, func);                           \
        if (b) {                                                        \
                (void) fputs(b, stdout);                                \
                free(b);                                                \
        }                                                               \
        if (dollar)                                                     \
                *f = '$';                                               \
_NOTE(CONSTCOND) } while (0)

static int       asciicode(void);
static char     *doformat(char *, int *);
static int       escape(char *, int, size_t *);
static int       getchr(void);
static int       getfloating(long double *, int);
static int       getint(int *);
static int       getnum(intmax_t *, uintmax_t *, int);
static const char
                *getstr(void);
static char     *mknum(char *, char);
static void      usage(void);

static const char digits[] = "0123456789";

static int  myargc;
static char **myargv;
static char **gargv;
static char **maxargv;

int
main(int argc, char *argv[])
{
        size_t len;
        int end, rval;
        int chopped, end, rval;
        char *format, *fmt, *start;

        (void) setlocale(LC_ALL, "");

        argv++;
        argc--;

        /*
         * POSIX says: Standard utilities that do not accept options,
         * but that do accept operands, shall recognize "--" as a
         * first argument to be discarded.
         */
        if (argc && strcmp(argv[0], "--") == 0) {
                argc--;
                argv++;
        }

        if (argc < 1) {
                usage();
                return (1);
        }
```

```
118         /*
119          * Basic algorithm is to scan the format string for conversion
120          * specifications -- once one is found, find out if the field
121          * width or precision is a '*'; if it is, gather up value.  Note,
122          * format strings are reused as necessary to use up the provided
123          * arguments, arguments of zero/null string are provided to use
124          * up the format string.
125          */
126         fmt = format = *argv;
127         (void) escape(fmt, 1, &len);    /* backslash interpretation */
128         chopped = escape(fmt, 1, &len);         /* backslash interpretation */
128         rval = end = 0;
129         gargv = ++argv;

131         for (;;) {
132                 maxargv = gargv;
133                 char **maxargv = gargv;

134                 myargv = gargv;
135                 for (myargc = 0; gargv[myargc]; myargc++)
136                         /* nop */;
137                 start = fmt;
138                 while (fmt < format + len) {
139                         if (fmt[0] == '%') {
140                                 (void) fwrite(start, 1, PTRDIFF(fmt, start),
141                                     stdout);
142                                 if (fmt[1] == '%') {
143                                         /* %% prints a % */
144                                         (void) putchar('%');
145                                         fmt += 2;
146                                 } else {
147                                         fmt = doformat(fmt, &rval);
148                                         if (fmt == NULL)
149                                                 return (1);
150                                         end = 0;
151                                 }
152                                 start = fmt;
153                         } else
154                                 fmt++;
155                         if (gargv > maxargv)
156                                 maxargv = gargv;
157                 }
158                 gargv = maxargv;

160                 if (end == 1) {
161                         warnx1(_("missing format character"), NULL, NULL);
162                         return (1);
163                 }
164                 (void) fwrite(start, 1, PTRDIFF(fmt, start), stdout);
165                 if (!*gargv)
166                 if (chopped || !*gargv)
166                         return (rval);
167                 /* Restart at the beginning of the format string. */
168                 fmt = format;
169                 end = 1;
170         }
171         /* NOTREACHED */
172 }


175 static char *
176 doformat(char *fmt, int *rval)
177 doformat(char *start, int *rval)
177 {
178         static const char skip1[] = "#'-+ 0";
```



```
180         static const char skip2[] = "0123456789";
181         char *fmt;
179         int fieldwidth, haveprec, havewidth, mod_ldbl, precision;
180         char convch, nextch;
181         char *start;
182         char **fargv;
183         char *dptr;
184         int l;

186         start = alloca(strlen(fmt) + 1);
185         fmt = start + 1;

188         dptr = start;
189         *dptr++ = '%';
190         *dptr = 0;

192         fmt++;

194         /* look for "n$" field index specifier */
195         l = strspn(fmt, digits);
196         if ((l > 0) && (fmt[l] == '$')) {
197                 int idx = atoi(fmt);
188         fmt += strspn(fmt, skip2);
189         if ((*fmt == '$') && (fmt != (start + 1))) {
190                 int idx = atoi(start + 1);
198                 if (idx <= myargc) {
199                         gargv = &myargv[idx - 1];
200                 } else {
201                         gargv = &myargv[myargc];
202                 }
203                 if (gargv > maxargv) {
204                         maxargv = gargv;
205                 }
206                 fmt += l + 1;

208                 /* save format argument */
209                 fargv = gargv;
196                 start = fmt;
197                 fmt++;
210         } else {
211                 fargv = NULL;
199                 fmt = start + 1;
212         }

214         /* skip to field width */
215         while (*fmt && strchr(skip1, *fmt) != NULL) {
216                 *dptr++ = *fmt++;
217                 *dptr = 0;
218         }


203         fmt += strspn(fmt, skip1);
221         if (*fmt == '*') {

223                 fmt++;
224                 l = strspn(fmt, digits);
225                 if ((l > 0) && (fmt[l] == '$')) {
226                         int idx = atoi(fmt);
227                         if (fargv == NULL) {
228                                 warnx1(_("incomplete use of n$"), NULL, NULL);
229                                 return (NULL);
230                         }
231                         if (idx <= myargc) {
232                                 gargv = &myargv[idx - 1];
233                         } else {
234                                 gargv = &myargv[myargc];
```

```
235                         }
236                         fmt += l + 1;
237                 } else if (fargv != NULL) {
238                         warnx1(_("incomplete use of n$"), NULL, NULL);
239                         return (NULL);
240                 }

242                 if (getint(&fieldwidth))
243                         return (NULL);
244                 if (gargv > maxargv) {
245                         maxargv = gargv;
246                 }
247                 havewidth = 1;

249                 *dptr++ = '*';
250                 *dptr = 0;
208                 ++fmt;
251         } else {
252                 havewidth = 0;

254                 /* skip to possible '.', get following precision */
255                 while (isdigit(*fmt)) {
256                         *dptr++ = *fmt++;
257                         *dptr = 0;
213                 fmt += strspn(fmt, skip2);
258                 }
259         }

261         if (*fmt == '.') {
262                 /* precision present? */
263                 fmt++;
264                 *dptr++ = '.';

217                 ++fmt;
266                 if (*fmt == '*') {

268                         fmt++;
269                         l = strspn(fmt, digits);
270                         if ((l > 0) && (fmt[l] == '$')) {
271                                 int idx = atoi(fmt);
272                                 if (fargv == NULL) {
273                                         warnx1(_("incomplete use of n$"),
274                                             NULL, NULL);
275                                         return (NULL);
276                                 }
277                                 if (idx <= myargc) {
278                                         gargv = &myargv[idx - 1];
279                                 } else {
280                                         gargv = &myargv[myargc];
281                                 }
282                                 fmt += l + 1;
283                         } else if (fargv != NULL) {
284                                 warnx1(_("incomplete use of n$"), NULL, NULL);
285                                 return (NULL);
286                         }

288                         if (getint(&precision))
289                                 return (NULL);
290                         if (gargv > maxargv) {
291                                 maxargv = gargv;
292                         }
293                         haveprec = 1;
294                         *dptr++ = '*';
295                         *dptr = 0;
222                         ++fmt;
296                 } else {
```



```
297                         haveprec = 0;

299                         /* skip to conversion char */
300                         while (isdigit(*fmt)) {
301                                 *dptr++ = *fmt++;
302                                 *dptr = 0;
227                         fmt += strspn(fmt, skip2);
303                         }
304                 }
305         } else
306                 haveprec = 0;
307         if (!*fmt) {
308                 warnx1(_("missing format character"), NULL, NULL);
309                 return (NULL);
310         }
311         *dptr++ = *fmt;
312         *dptr = 0;

314         /*
315          * Look for a length modifier.  POSIX doesn't have these, so
316          * we only support them for floating-point conversions, which
317          * are extensions.  This is useful because the L modifier can
318          * be used to gain extra range and precision, while omitting
319          * it is more likely to produce consistent results on different
320          * architectures.  This is not so important for integers
321          * because overflow is the only bad thing that can happen to
322          * them, but consider the command  printf %a 1.1
323          */
324         if (*fmt == 'L') {
325                 mod_ldbl = 1;
326                 fmt++;
327                 if (!strchr("aAeEfFgG", *fmt)) {
328                         warnx2(_("bad modifier L for %%%c"), *fmt, NULL);
329                         return (NULL);
330                 }
331         } else {
332                 mod_ldbl = 0;
333         }

335         /* save the current arg offset, and set to the format arg */
336         if (fargv != NULL) {
337                 gargv = fargv;
338         }

340         convch = *fmt;
341         nextch = *++fmt;

343         *fmt = '\0';
344         switch (convch) {
345         case 'b': {
346                 size_t len;
347                 char *p;
348                 int getout;

350                 p = strdup(getstr());
351                 if (p == NULL) {
352                         warnx2("%s", strerror(ENOMEM), NULL);
353                         return (NULL);
354                 }
355                 getout = escape(p, 0, &len);
356                 (void) fputs(p, stdout);
272                 *(fmt - 1) = 's';
273                 PF(start, p);
274                 *(fmt - 1) = 'b';
357                 free(p);
```

```
                if (getout)
                        exit(*rval);
                        return (fmt);
                break;
        }
        case 'c': {
                char p;

                p = getchr();
                PF(start, p);
                break;
        }
        case 's': {
                const char *p;

                p = getstr();
                PF(start, p);
                break;
        }
        case 'd': case 'i': case 'o': case 'u': case 'x': case 'X': {
                char *f;
                intmax_t val;
                uintmax_t uval;
                int signedconv;

                signedconv = (convch == 'd' || convch == 'i');
                if ((f = mknum(start, convch)) == NULL)
                        return (NULL);
                if (getnum(&val, &uval, signedconv))
                        *rval = 1;
                if (signedconv)
                        PF(f, val);
                else
                        PF(f, uval);
                break;
        }
        case 'e': case 'E':
        case 'f': case 'F':
        case 'g': case 'G':
        case 'a': case 'A': {
                long double p;

                if (getfloating(&p, mod_ldbl))
                        *rval = 1;
                if (mod_ldbl)
                        PF(start, p);
                else
                        PF(start, (double)p);
                break;
        }
        default:
                warnx2(_("illegal format character %c"), convch, NULL);
                return (NULL);
        }
        *fmt = nextch;

        /* return the gargv to the next element */
        return (fmt);
}
```

*unchanged_portion_omitted_*

```
static int
escape(char *fmt, int percent, size_t *len)
{
        char *save, *store, c;
        int value;
```



```
        for (save = store = fmt; ((c = *fmt) != 0); ++fmt, ++store) {
                if (c != '\\') {
                        *store = c;
                        continue;
                }
                switch (*++fmt) {
                case '\0':              /* EOS, user error */
                        *store = '\\';
                        *++store = '\0';
                        *len = PTRDIFF(store, save);
                        return (0);
                case '\\':              /* backslash */
                case '\'':              /* single quote */
                        *store = *fmt;
                        break;
                case 'a':               /* bell/alert */
                        *store = '\a';
                        break;
                case 'b':               /* backspace */
                        *store = '\b';
                        break;
                case 'c':
                        if (!percent) {
                                *store = '\0';
                                *len = PTRDIFF(store, save);
                                return (1);
                        }
                        *store = 'c';
                        break;
                case 'f':               /* form-feed */
                        *store = '\f';
                        break;
                case 'n':               /* newline */
                        *store = '\n';
                        break;
                case 'r':               /* carriage-return */
                        *store = '\r';
                        break;
                case 't':               /* horizontal tab */
                        *store = '\t';
                        break;
                case 'v':               /* vertical tab */
                        *store = '\v';
                        break;
                                        /* octal constant */
                case '0': case '1': case '2': case '3':
                case '4': case '5': case '6': case '7':
                        c = (!percent && *fmt == '0') ? 4 : 3;
                        for (value = 0;
                            c-- && *fmt >= '0' && *fmt <= '7'; ++fmt) {
                                value <<= 3;
                                value += *fmt - '0';
                        }
                        --fmt;
                        if (percent && value == '%') {
                                *store++ = '%';
                                *store = '%';
                        } else
                                *store = (char)value;
                        break;
                default:
                        *store = *fmt;
                        break;
                }
        }
```

```
        *store = '\0';
        *len = PTRDIFF(store, save);
        return (0);
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   22739 Mon May 12 22:34:33 2014
new/usr/src/man/man1/printf.1
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
'\" te
.\" Copyright 2014 Garrett D'Amore <garrett@damore.org>
.\" Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved
.\" Copyright 1992, X/Open Company Limited All Rights Reserved
.\" Portions Copyright (c) 1982-2007 AT&T Knowledge Ventures
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for
.\" permission to reproduce portions of its copyrighted documentation.
.\" Original documentation from The Open Group can be obtained online at
.\" http://www.opengroup.org/bookstore/.
.\" The Institute of Electrical and Electronics Engineers and The Open Group,
.\" have given us permission to reprint portions of their documentation. In the
.\" following statement, the phrase "this text" refers to portions of the
.\" system documentation. Portions of this text are reprinted and reproduced in
.\" electronic form in the Sun OS Reference Manual, from IEEE Std 1003.1, 2004
.\" Edition, Standard for Information Technology -- Portable Operating System
.\" Interface (POSIX), The Open Group Base Specifications Issue 6, Copyright
.\" (C) 2001-2004 by the Institute of Electrical and Electronics Engineers,
.\" Inc and The Open Group. In the event of any discrepancy between these
.\" versions and the original IEEE and The Open Group Standard, the original
.\" IEEE and The Open Group Standard is the referee document. The original
.\" Standard can be obtained online at
.\" http://www.opengroup.org/unix/online.html.
.\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
.\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
.\" are reprinted and reproduced in electronic form in the Sun OS Reference Manu
.\" and Electronics Engineers, Inc and The Open Group. In the event of any discr
.\" This notice shall appear on any product containing this material.
.\" 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]
.TH PRINTF 1 "May 11, 2014"
.\" The contents of this file are subject to the terms of the Common Development
.\"  See the License for the specific language governing permissions and limitat
.\" the fields enclosed by brackets "[]" replaced with your own identifying info
.TH PRINTF 1 "Aug 11, 2009"
.SH NAME
printf \- write formatted output
.SH SYNOPSIS
.SS "/usr/bin/printf"
.LP
.nf
\fBprintf\fR \fIformat\fR [\fIargument\fR]...
.fi

.SS "ksh93"
.LP
.nf
\fBprintf\fR \fIformat\fR [\fIstring\fR...]
.fi
```



```
.SH DESCRIPTION
.SS "/usr/bin/printf"
.sp
.LP
The \fBprintf\fR utility writes each string operand to standard output using
\fIformat\fR to control the output format.
.SH OPERANDS
.SS "/usr/bin/printf"
.sp
.LP
The following operands are supported by \fB/usr/bin/printf\fR:
.sp
.ne 2
.na
\fB\fIformat\fR\fR
.ad
.RS 12n
A string describing the format to use to write the remaining operands. The
\fIformat\fR operand is used as the \fIformat\fR string described on the
\fBformats\fR(5) manual page, with the following exceptions:
.RS +4
.TP
.ie t \(bu
.el o
A \fBSPACE\fR character in the format string, in any context other than a flag
of a conversion specification, is treated as an ordinary character that is
copied to the output.
.RE
.RS +4
.TP
.ie t \(bu
.el o
A character in the format string is treated as a character, not as a
\fBSPACE\fR character.
.RE
.RS +4
.TP
.ie t \(bu
.el o
In addition to the escape sequences described on the \fBformats\fR(5) manual
page (\fB\e\e\fR, \fB\ea\fR, \fB\eb\fR, \fB\ef\fR, \fB\en\fR, \fB\er\fR,
\fB\et\fR, \fB\ev\fR), \fB\e\fR\fIddd\fR, where \fIddd\fR is a one-, two- or
three-digit octal number, is written as a byte with the numeric value specified
by the octal number.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The program does not precede or follow output from the \fBd\fR or \fBu\fR
conversion specifications with blank characters not specified by the
\fIformat\fR operand.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The program does not precede output from the \fBo\fR conversion specification
with zeros not specified by the \fIformat\fR operand.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The argument used for the conversion character (or width or precision
parameters, see below) may be taken from the \fIn\fRnth argument instead
```

of the next unused argument, by specifying \fIn\fR\fB$\fR immediately following the \fB%\fR character, or the \fB*\fR character (for width or precision arguments).
If \fIn\fR\fB$\fR appears in any conversions in the format string, then it must be used for all conversions, including any variable width or precision specifiers.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The special character \fB*\fR may be used instead of a string of decimal digits to indicate a minimum field width or a precision. In this case the next available argument is used (or the \fIn\fRth if the form \fIn\fR\fB$\fR is used), treating its value as a decimal string.
.RE
.RS +4
.TP
.ie t \(bu
.el o
An additional conversion character, \fBb\fR, is supported as follows. The argument is taken to be a string that can contain backslash-escape sequences. The following backslash-escape sequences are supported:
.RS +4
.TP
.ie t \(bu
.el o
the escape sequences listed on the \fBformats\fR(5) manual page (\fB\e\e\fR, \fB\ea\fR, \fB\eb\fR, \fB\ef\fR, \fB\en\fR, \fB\er\fR, \fB\et\fR, \fB\ev\fR), which are converted to the characters they represent
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fB\e0\fR\fIddd\fR, where \fIddd\fR is a zero-, one-, two- or three-digit octal number that is converted to a byte with the numeric value specified by the octal number
.RE
.RS +4
.TP
.ie t \(bu
.el o
\fB\ec\fR, which is written and causes \fBprintf\fR to ignore any remaining characters in the string operand containing it, any remaining string operands and any additional characters in the \fIformat\fR operand.
.RE
.RE
The interpretation of a backslash followed by any other sequence of characters is unspecified.
.sp
Bytes from the converted string are written until the end of the string or the number of bytes indicated by the precision specification is reached. If the precision is omitted, it is taken to be infinite, so all bytes up to the end of the converted string are written. For each specification that consumes an argument, the next argument operand is evaluated and converted to the appropriate type for the conversion as specified below. The \fIformat\fR operand is reused as often as necessary to satisfy the argument operands. Any extra \fBc\fR or \fBs\fR conversion specifications are evaluated as if a null string argument were supplied; other extra conversion specifications are
evaluated as if a zero argument were supplied.
.sp
When there are more argument operands than format specifiers, and the format includes \fIn\fR\fB$\fR position indicators, then the format is reprocessed from the beginning as above, but with the argument list starting from the next argument after the highest \fIn\fRth argument previously

encountered.
.sp
If the \fIformat\fR operand
evaluated as if a zero argument were supplied. If the \fIformat\fR operand
contains no conversion specifications and \fIargument\fR operands are present, the results are unspecified. If a character sequence in the \fIformat\fR operand begins with a \fB%\fR character, but does not form a valid conversion specification, the behavior is unspecified.
.RE

.sp
.ne 2
.na
\fB\fIargument\fR\fR
.ad
.RS 12n
The strings to be written to standard output, under the control of \fBformat\fR. The \fIargument\fR operands are treated as strings if the corresponding conversion character is \fBb\fR, \fBc\fR or \fBs\fR. Otherwise, it is evaluated as a C constant, as described by the ISO C standard, with the following extensions:
.RS +4
.TP
.ie t \(bu
.el o
A leading plus or minus sign is allowed.
.RE
.RS +4
.TP
.ie t \(bu
.el o
If the leading character is a single- or double-quote, the value is the numeric value in the underlying codeset of the character following the single- or double-quote.
.RE
If an argument operand cannot be completely converted into an internal value appropriate to the corresponding conversion specification, a diagnostic message is written to standard error and the utility does not exit with a zero exit status, but continues processing any remaining operands and writes the value accumulated at the time the error was detected to standard output.
.RE

.SS "ksh93"
.sp
.LP
The \fIformat\fR operands support the full range of ANSI C/C99/XPG6 formatting specifiers as well as additional specifiers:
.sp
.ne 2
.na
\fB\fB%b\fR\fR
.ad
.RS 6n
Each character in the string operand is processed specially, as follows:
.sp
.ne 2
.na
\fB\fB\ea\fR\fR
.ad
.RS 8n
Alert character.
.RE

.sp
.ne 2
.na

```
\fB\fB\eb\fR\fR
.ad
.RS 8n
Backspace character.
.RE

.sp
.ne 2
.na
\fB\fB\ec\fR\fR
.ad
.RS 8n
Terminate output without appending NEWLINE. The remaining string operands are
ignored.
.RE

.sp
.ne 2
.na
\fB\fB\eE\fR\fR
.ad
.RS 8n
Escape character (\fBASCII\fR octal \fB033\fR).
.RE

.sp
.ne 2
.na
\fB\fB\ef\fR\fR
.ad
.RS 8n
FORM FEED character.
.RE

.sp
.ne 2
.na
\fB\fB\en\fR\fR
.ad
.RS 8n
NEWLINE character.
.RE

.sp
.ne 2
.na
\fB\fB\et\fR\fR
.ad
.RS 8n
TAB character.
.RE

.sp
.ne 2
.na
\fB\fB\ev\fR\fR
.ad
.RS 8n
Vertical tab character.
.RE

.sp
.ne 2
.na
\fB\fB\e\e\fR\fR
.ad
```



```
.RS 8n
Backslash character.
.RE

.sp
.ne 2
.na
\fB\fB\e0\fR\fIx\fR\fR
.ad
.RS 8n
The 8-bit character whose \fBASCII\fR code is the \fB1\fR-, \fB2\fR-, or
\fB3\fR-digit octal number \fIx\fR.
.RE

.RE

.sp
.ne 2
.na
\fB\fB%B\fR\fR
.ad
.RS 6n
Treat the argument as a variable name and output the value without converting
it to a string. This is most useful for variables of type \fB-b\fR.
.RE

.sp
.ne 2
.na
\fB\fB%H\fR\fR
.ad
.RS 6n
Output string with characters \fB<\fR, \fB&\fR, \fB>\fR, \fB"\fR, and
non-printable characters, properly escaped for use in HTML and XML documents.
.RE

.sp
.ne 2
.na
\fB\fB%P\fR\fR
.ad
.RS 6n
Treat \fIstring\fR as an extended regular expression and convert it to a shell
pattern.
.RE

.sp
.ne 2
.na
\fB\fB%q\fR\fR
.ad
.RS 6n
Output \fIstring\fR quoted in a manner that it can be read in by the shell to
get back the same string. However, empty strings resulting from missing string
operands are not quoted.
.RE

.sp
.ne 2
.na
\fB\fB%R\fR\fR
.ad
.RS 6n
Treat \fIstring\fR as an shell pattern expression and convert it to an extended
regular expression.
.RE
```

```
.sp
.ne 2
.na
\fB\fB%T\fR\fR
.ad
.RS 6n
Treat \fIstring\fR as a date/time string and format it. The \fBT\fR can be
preceded by (\fIdformat\fR), where \fIdformat\fR is a date format as defined by
the \fBdate\fR(1) command.
.RE

.sp
.ne 2
.na
\fB\fB%Z\fR\fR
.ad
.RS 6n
Output a byte whose value is \fB0\fR.
.RE

.sp
.LP
When performing conversions of \fIstring\fR to satisfy a numeric format
specifier, if the first character of \fIstring\fR is \fB"or'\fR, the value is
the numeric value in the underlying code set of the character following the
\fB"or'\fR. Otherwise, \fIstring\fR is treated like a shell arithmetic
expression and evaluated.
.sp
.LP
If a \fIstring\fR operand cannot be completely converted into a value
appropriate for that format specifier, an error occurs, but remaining
\fIstring\fR operands continue to be processed.
.sp
.LP
In addition to the format specifier extensions, the following extensions of
ANSI C/C99/XPG6 are permitted in format specifiers:
.RS +4
.TP
.ie t \(bu
.el o
The escape sequences \fB\eE\fR and \fB\ee\fR expand to the escape character
which is octal 033 in ASCII.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The escape sequence \fB\ecx\fR expands to CTRL-x.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The escape sequence \fB\eC[.\fR\fIname\fR\fB\&.]\fR expands to the collating
element \fIname\fR.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The escape sequence \fB\ex{hex}\fR expands to the character corresponding to
the hexadecimal value \fBhex\fR.
The escape sequence \fB\ex{hex}\fRexpands to the character corresponding to the
hexadecimal value \fBhex\fR.
.RE
```



```
.RS +4
.TP
.ie t \(bu
.el o
The format modifier flag = can be used to center a field to a specified width.
When the output is a terminal, the character width is used rather than the
number of bytes.
.RE
.RS +4
.TP
.ie t \(bu
.el o
Each of the integral format specifiers can have a third modifier after width
and precision that specifies the base of the conversion from 2 to 64. In this
case, the \fB#\fR modifier causes \fIbase\fR\fB#\fR to be prepended to the
value.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The \fB#\fR modifier can be used with the \fBd\fR specifier when no base is
specified to cause the output to be written in units of 1000 with a suffix of
one of \fBk M G T P E\fR.
.RE
.RS +4
.TP
.ie t \(bu
.el o
The \fB#\fR modifier can be used with the \fBi\fR specifier to cause the output
to be written in units of \fB1024\fR with a suffix of one of \fBKi Mi Gi Ti Pi
Ei\fR.
.RE
.sp
.LP
If there are more \fIstring\fR operands than format specifiers, the format
string is reprocessed from the beginning. If there are fewer \fIstring\fR
operands than format specifiers, then \fIstring\fR specifiers are treated as if
empty strings were supplied, numeric conversions are treated as if \fB0\fR was
supplied, and time conversions are treated as if \fBnow\fR was supplied.
.sp
.LP
When there are more argument operands than format specifiers, and the
format includes \fIn\fR\fB$\fR position indicators, then the format is
reprocessed from the beginning as above, but with the argument list starting
from the next argument after the highest \fIn\fRth argument previously
encountered.
.sp
.LP
\fB/usr/bin/printf\fR is equivalent to \fBksh93\fR's \fBprintf\fR built-in and
\fBprint -f\fR, which allows additional options to be specified.
.SH USAGE
.SS "/usr/bin/printf"
.sp
.LP
The \fBprintf\fR utility, like the \fBprintf\fR(3C) function on which it is
based, makes no special provision for dealing with multi-byte characters when
using the \fB%c\fR conversion specification. Applications should be extremely
cautious using either of these features when there are multi-byte characters in
the character set.
.sp
.LP
Field widths and precisions cannot be specified as \fB*\fR.
.sp
.LP
The \fB%b\fR conversion specification is not part of the ISO C standard; it has
```

```
been added here as a portable way to process backslash escapes expanded in
string operands as provided by the \fBecho\fR utility. See also the USAGE
section of the \fBecho\fR(1) manual page for ways to use \fBprintf\fR as a
replacement for all of the traditional versions of the \fBecho\fR utility.
.sp
.LP
If an argument cannot be parsed correctly for the corresponding conversion
specification, the \fBprintf\fR utility reports an error. Thus, overflow and
extraneous characters at the end of an argument being used for a numeric
conversion are to be reported as errors.
.sp
.LP
It is not considered an error if an argument operand is not completely used for
a \fBc\fR or \fBs\fR conversion or if a string operand's first or second
character is used to get the numeric value of a character.
.SH EXAMPLES
.SS "/usr/bin/printf"
.LP
\fBExample 1 \fRPrinting a Series of Prompts
.sp
.LP
The following example alerts the user, then prints and reads a series of
prompts:

.sp
.in +2
.nf
example% \fBprintf "\eaPlease fill in the following: \enName: "
read name
printf "Phone number: "
read phone\fR
.fi
.in -2
.sp

.LP
\fBExample 2 \fRPrinting a Table of Calculations
.sp
.LP
The following example prints a table of calculations. It reads out a list of
right and wrong answers from a file, calculates the percentage correctly, and
prints them out. The numbers are right-justified and separated by a single tab
character. The percentage is written to one decimal place of accuracy:

.sp
.in +2
.nf
example% \fBwhile read right wrong ; do
   percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)
   printf "%2d right\et%2d wrong\et(%s%%)\en" \e
          $right $wrong $percent
done < database_file\fR
.fi
.in -2
.sp

.LP
\fBExample 3 \fRPrinting number strings
.sp
.LP
The command:

.sp
.in +2
.nf
example% \fBprintf "%5d%4d\en" 1 21 321 4321 54321\fR
```



```
.fi
.in -2
.sp

.sp
.LP
produces:

.sp
.in +2
.nf
    1  21
  3214321
54321   0
.fi
.in -2
.sp

.sp
.LP
The \fIformat\fR operand is used three times to print all of the given strings
and that a \fB0\fR was supplied by \fBprintf\fR to satisfy the last \fB%4d\fR
conversion specification.

.LP
\fBExample 4 \fRTabulating Conversion Errors
.sp
.LP
The following example tabulates conversion errors.

.sp
.LP
The \fBprintf\fR utility tells the user when conversion errors are detected
while producing numeric output. These results would be expected on an
implementation with 32-bit twos-complement integers when \fB%d\fR is specified
as the \fIformat\fR operand:

.sp

.sp
.TS
box;
c c c
l l l .
Arguments       Standard        Diagnostic
5a      5       printf: 5a not completely converted
9999999999      2147483647      printf: 9999999999: Results too large
-9999999999     -2147483648     printf: -9999999999: Results too large
ABC     0       printf: ABC expected numeric value
.TE

.sp
.LP
The value shown on standard output is what would be expected as the return
value from the function \fBstrtol\fR(3C). A similar correspondence exists
between \fB%u\fR and \fBstrtoul\fR(3C), and \fB%e\fR, \fB%f\fR and \fB%g\fR and
\fBstrtod\fR(3C).

.LP
\fBExample 5 \fRPrinting Output for a Specific Locale
.sp
.LP
The following example prints output for a specific locale. In a locale using
the ISO/IEC 646:1991 standard as the underlying codeset, the command:

.sp
```

```
.in +2
.nf
example% \fBprintf "%d\en" 3 +3 -3 \e'3 \e"+3 "'-3"\fR
.fi
.in -2
.sp

.sp
.LP
produces:

.sp

.sp
.TS
box;
l l
l l .
\fB3\fR Numeric value of constant 3
\fB3\fR Numeric value of constant 3
\fB\(mi3\fR     Numeric value of constant \(mi3
\fB51\fR        T{
Numeric value of the character '3' in the ISO/IEC 646:1991 standard codeset
T}
\fB43\fR        T{
Numeric value of the character '+' in the ISO/IEC 646:1991 standard codeset
T}
\fB45\fR        T{
Numeric value of the character '\(mi' in the SO/IEC 646:1991 standard codeset
T}
.TE

.sp
.LP
In a locale with multi-byte characters, the value of a character is intended to
be the value of the equivalent of the \fBwchar_t\fR representation of the
character.

.sp
.LP
If an argument operand cannot be completely converted into an internal value
appropriate to the corresponding conversion specification, a diagnostic message
is written to standard error and the utility does exit with a zero exit status,
but continues processing any remaining operands and writes the value
accumulated at the time the error was detected to standard output.

.LP
\fBExample 6 \fRAlternative floating point representation 1
.sp
.LP
The \fBprintf\fR utility supports an alternative floating point representation
(see \fBprintf\fR(3C) entry for the "\fB%a\fR"/"\fB%A\fR"), which allows the
output of floating-point values in a format that avoids the usual base16 to
base10 rounding errors.

.sp
.in +2
.nf
example% printf "%a\en" 2 3.1 NaN
.fi
.in -2
.sp

.sp
.LP
produces:
```



```
.sp
.in +2
.nf
0x1.0000000000000000000000000000p+01
0x1.8cccccccccccccccccccccccccccdp+01
nan
.fi
.in -2
.sp

.LP
\fBExample 7 \fRAlternative floating point representation 2
.sp
.LP
The following example shows two different representations of the same
floating-point value.

.sp
.in +2
.nf
example% x=2 ; printf "%f == %a\en" x x
.fi
.in -2
.sp

.sp
.LP
produces:

.sp
.in +2
.nf
2.000000 == 0x1.0000000000000000000000000000p+01
.fi
.in -2
.sp

.LP
\fBExample 8 \fROutput of unicode values
.sp
.LP
The following command will print the EURO unicode symbol (code-point 0x20ac).

.sp
.in +2
.nf
example% LC_ALL=en_US.UTF-8 printf "\u[20ac]\en"
.fi
.in -2
.sp

.sp
.LP
produces:

.sp
.in +2
.nf
<euro>
.fi
.in -2
.sp

.sp
.LP
```

```
where "<euro>" represents the EURO currency symbol character.

.LP
\fBExample 9 \fRConvert unicode character to unicode code-point value
.sp
.LP
The following command will print the hexadecimal value of a given character.

.sp
.in +2
.nf
example% export LC_ALL=en_US.UTF-8
example% printf "%x\en" "'<euro>"
.fi
.in -2
.sp

.sp
.LP
where "<euro>" represents the EURO currency symbol character (code-point
0x20ac).

.sp
.LP
produces:

.sp
.in +2
.nf
20ac
.fi
.in -2
.sp

.LP
\fBExample 10 \fRPrint the numeric value of an ASCII character
.sp
.in +2
.nf
example% printf "%d\en" "'A"
.fi
.in -2
.sp

.sp
.LP
produces:

.sp
.in +2
.nf
65
.fi
.in -2
.sp

.LP
\fBExample 11 \fRPrint the language-independent date and time format
.sp
.LP
To print the language-independent date and time format, the following statement
could be used:

.sp
.in +2
.nf
```



```
example% printf "format" weekday month day hour min
.fi
.in -2
.sp

.sp
.LP
For example,

.sp
.in +2
.nf
$ printf format "Sunday" "July" 3 10 2
.fi
.in -2
.sp

.sp
.LP
For American usage, format could be the string:

.sp
.in +2
.nf
"%s, %s %d, %d:%.2d\en"
.fi
.in -2
.sp

.sp
.LP
producing the message:

.sp
.in +2
.nf
Sunday, July 3, 10:02
.fi
.in -2
.sp

.sp
.LP
Whereas for EU usage, format could be the string:

.sp
.in +2
.nf
"%1$s, %3$d. %2$s, %4$d:%5$.2d\en"
.fi
.in -2
.sp

.sp
.LP
Note that the '$' characters must be properly escaped, such as

.sp
.in +2
.nf
"%1\e$s, %3\e$d. %2\e$s, %4\e$d:%5\e$.2d\en" in this case
"%1\$s, %3\$d. %2\$s, %4\$d:%5\$.2d\en" in this case
.fi
.in -2
.sp
```

```
.sp
.LP
producing the message:

.sp
.in +2
.nf
Sunday, 3. July, 10:02
.fi
.in -2
.sp

.SH ENVIRONMENT VARIABLES
.sp
.LP
See \fBenviron\fR(5) for descriptions of the following environment variables
that affect the execution of \fBprintf\fR: \fBLANG\fR, \fBLC_ALL\fR,
\fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, \fBLC_NUMERIC\fR, and \fBNLSPATH\fR.
.SH EXIT STATUS
.sp
.LP
The following exit values are returned:
.sp
.ne 2
.na
\fB\fB0\fR\fR
.ad
.RS 6n
Successful completion.
.RE

.sp
.ne 2
.na
\fB\fB>0\fR\fR
.ad
.RS 6n
An error occurred.
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.SS "/usr/bin/printf"
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
CSI     Enabled
_
Interface Stability     Committed
_
Standard        See \fBstandards\fR(5).
.TE

.SS "ksh93"
.sp

.sp
.TS
```



```
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
Interface Stability     Uncommitted
.TE

.SH SEE ALSO
.sp
.LP
\fBawk\fR(1), \fBbc\fR(1), \fBdate\fR(1), \fBecho\fR(1), \fBksh93\fR(1),
\fBprintf\fR(3C), \fBstrtod\fR(3C), \fBstrtol\fR(3C), \fBstrtoul\fR(3C),
\fBattributes\fR(5), \fBenviron\fR(5), \fBformats\fR(5), \fBstandards\fR(5)
.SH NOTES
.sp
.LP
Using format specifiers (characters following '%') which are not listed in the
\fBprintf\fR(3C) or this manual page will result in undefined behavior.
.sp
.LP
Using escape sequences (the character following a backslash ('\e')) which are
not listed in the \fBprintf\fR(3C) or this manual page will result in undefined
behavior.
.sp
.LP
Floating-point values follow C99, XPG6 and IEEE 754 standard behavior and can
handle values the same way as the platform's |\fBlong double\fR| datatype.
.sp
.LP
Floating-point values handle the sign separately which allows signs for values
like NaN (for example, -nan), Infinite (for example, -inf) and zero (for
example, -0.0).
```

```
**********************************************************
    1236 Mon May 12 22:34:34 2014
new/usr/src/pkg/manifests/system-test-utiltest.mf
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
#

set name=pkg.fmri value=pkg:/system/test/utiltest@$(PKGVERS)
set name=pkg.description value="Miscellaneous Utility Unit Tests"
set name=pkg.summary value="Utility Unit Test Suite"
set name=info.classification \
    value=org.opensolaris.category.2008:Development/System
set name=variant.arch value=$(ARCH)
dir path=opt/util-tests
dir path=opt/util-tests/bin
dir path=opt/util-tests/runfiles
dir path=opt/util-tests/tests
file path=opt/util-tests/README mode=0444
file path=opt/util-tests/bin/utiltest mode=0555
file path=opt/util-tests/runfiles/default.run mode=0444
file path=opt/util-tests/tests/printf_test mode=0555
license lic_CDDL license=lic_CDDL
depend fmri=system/test/testrunner type=require
```

****
613 Mon May 12 22:34:34 2014
new/usr/src/test/Makefile
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
****

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
#

.PARALLEL: $(SUBDIRS)

SUBDIRS = os-tests test-runner util-tests zfs-tests
SUBDIRS = os-tests test-runner zfs-tests

include Makefile.com
```

```
**********************************************************
     552 Mon May 12 22:34:34 2014
new/usr/src/test/util-tests/Makefile
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
#

.PARALLEL: $(SUBDIRS)

SUBDIRS = cmd runfiles tests doc

include $(SRC)/test/Makefile.com
```

```
**********************************************************
     847 Mon May 12 22:34:34 2014
new/usr/src/test/util-tests/cmd/Makefile
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
# Copyright (c) 2012 by Delphix. All rights reserved.
#

include $(SRC)/Makefile.master
include $(SRC)/test/Makefile.com

ROOTOPTPKG = $(ROOT)/opt/util-tests
ROOTBIN = $(ROOTOPTPKG)/bin

PROGS = utiltest

CMDS = $(PROGS:%=$(ROOTBIN)/%)
$(CMDS) := FILEMODE = 0555

all lint clean clobber:

install: $(CMDS)

$(CMDS): $(ROOTBIN)

$(ROOTBIN):
        $(INS.dir)

$(ROOTBIN)/%: %.ksh
        $(INS.rename)
```

```
**********************************************************
    1449 Mon May 12 22:34:34 2014
new/usr/src/test/util-tests/cmd/utiltest.ksh
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
```

```
#!/usr/bin/ksh

#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
#

export MY_TESTS="/opt/util-tests"
runner="/opt/test-runner/bin/run"

function fail
{
        echo $1
        exit ${2:-1}
}

function find_runfile
{
        typeset distro=
        if [[ -d /opt/delphix && -h /etc/delphix/version ]]; then
                distro=delphix
        elif [[ 0 -ne $(grep -c OpenIndiana /etc/release 2>/dev/null) ]]; then
                distro=openindiana
        elif [[ 0 -ne $(grep -c OmniOS /etc/release 2>/dev/null) ]]; then
                distro=omnios
        elif [[ -f $MY_TESTS/runfiles/default.run ]]; then
                # optional catch-all
                distro=default
        fi

        [[ -n $distro ]] && echo $MY_TESTS/runfiles/$distro.run
}

while getopts c: c; do
        case $c in
        'c')
                runfile=$OPTARG
                [[ -f $runfile ]] || fail "Cannot read file: $runfile"
                ;;
        esac
done
shift $((OPTIND - 1))

[[ -z $runfile ]] && runfile=$(find_runfile)
[[ -z $runfile ]] && fail "Couldn't determine distro"
```



```
$runner -c $runfile

exit $?
```

```
**********************************************************
     800 Mon May 12 22:34:34 2014
new/usr/src/test/util-tests/doc/Makefile
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
# Copyright (c) 2012 by Delphix. All rights reserved.
#

include $(SRC)/Makefile.master

README = README

ROOTOPTPKG = $(ROOT)/opt/util-tests

FILES = $(README:%=$(ROOTOPTPKG)/%)
$(FILES) := FILEMODE = 0444

all: $(README)

install: $(ROOTOPTPKG) $(FILES)

clean lint clobber:

$(ROOTOPTPKG):
        $(INS.dir)

$(ROOTOPTPKG)/%: %
        $(INS.file)
```

```
**********************************************************
    2180 Mon May 12 22:34:34 2014
new/usr/src/test/util-tests/doc/README
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
# Copyright (c) 2012 by Delphix. All rights reserved.
#

Utils Unit Test Suite README

1. What the Utils Unit Test Suite tests
2. Building and installing the Utils Unit Test Suite
3. Running the Utils Unit Test Suite
4. Test results

-------------------------------------------------------------------------------

1. What the Utils Unit Test Suite tests

The Utils unit test suite is for testing standard shell / POSIX utilities.
For example utilities such as "printf" are tested.

2. Building and installing the Utils Unit Test Suite

The Utils Unit Test Suite runs under the testrunner framework (which can be
installed as pkg:/system/test/testrunner). To build both the Utils Unit Test
Suite and the testrunner without running a full nightly:

        build_machine$ bldenv [-d] <your_env_file>
        build_machine$ cd $SRC/test
        build_machine$ dmake install
        build_machine$ cd $SRC/pkg
        build_machine$ dmake install

Then set the publisher on the test machine to point to your repository and
install the Utils Unit Test Suite.

        test_machine# pkg install pkg:/system/test/utiltest

Note, the framework will be installed automatically, as the Utils Unit Test
Suite depends on it.

3. Running the Utils Unit Test Suite

The pre-requisites for running the OS Unit Test Suite are:
        - Any user may perform these tests.

Once the pre-requisites are satisfied, simply run the ostest script:

        test_machine$ /opt/util-tests/bin/utiltest
```



```
4. Test results

While the OS Unit Test Suite is running, one informational line is printed at
the end of each test, and a results summary is printed at the end of the run.
The results summary includes the location of the complete logs, which is of the
form /var/tmp/test_results/<ISO 8601 date>.
```

```
**********************************************************
     909 Mon May 12 22:34:34 2014
new/usr/src/test/util-tests/runfiles/Makefile
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
#

include $(SRC)/Makefile.master

SRCS = default.run

ROOTOPTPKG = $(ROOT)/opt/util-tests
RUNFILES = $(ROOTOPTPKG)/runfiles

CMDS = $(SRCS:%=$(RUNFILES)/%)
$(CMDS) := FILEMODE = 0444

all: $(SRCS)

install: $(CMDS)

clean lint clobber:

$(CMDS): $(RUNFILES) $(SRCS)

$(RUNFILES):
        $(INS.dir)

$(RUNFILES)/%: %
        $(INS.file)
```

```
**********************************************************
     654 Mon May 12 22:34:34 2014
new/usr/src/test/util-tests/runfiles/default.run
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copryight 2014 Garrett D'Amore <garrett@damore.org>
#

[DEFAULT]
pre =
verbose = False
quiet = False
timeout = 60
post =
outputdir = /var/tmp/test_results

[/opt/util-tests/tests/printf_test]
```

```
**********************************************************
     567 Mon May 12 22:34:34 2014
new/usr/src/test/util-tests/tests/Makefile
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
#

SUBDIRS = printf

include $(SRC)/test/Makefile.com
```

```
**********************************************************
     867 Mon May 12 22:34:34 2014
new/usr/src/test/util-tests/tests/printf/Makefile
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
#

include $(SRC)/cmd/Makefile.cmd
include $(SRC)/test/Makefile.com

PROG = printf_test

ROOTOPTPKG = $(ROOT)/opt/util-tests
TESTDIR = $(ROOTOPTPKG)/tests

CMDS = $(PROG:%=$(TESTDIR)/%)
$(CMDS) := FILEMODE = 0555

all lint clean clobber:

install: all $(CMDS)

$(CMDS): $(TESTDIR) $(PROG).ksh

$(TESTDIR):
        $(INS.dir)

$(TESTDIR)/%: %.ksh
        $(INS.rename)
```

```
**********************************************************
    4618 Mon May 12 22:34:34 2014
new/usr/src/test/util-tests/tests/printf/printf_test.ksh
4818 printf(1) should support n$ width and precision specifiers
4854 printf(1) doesn't support %b and \c properly
Reviewed by: Keith Wesolowski <keith.wesolowski@joyent.com>
Approved by: TBD
**********************************************************
```

```
#! /usr/bin/ksh
#
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
#

PRINTF=${PRINTF:=/usr/bin/printf}

test_start() {
        print "TEST STARTING ${1}: ${2}"
}

test_pass() {
        print "TEST PASS: ${1}"
}

test_fail() {
        print "TEST FAIL: ${1}: ${2}"
        exit -1
}

checkrv() {
        if [[ $? -ne 0 ]]; then
                test_fail $1 "exit failure"
        fi
}

compare() {
        if [[ "$2" != "$3" ]]; then
                test_fail $1 "compare mismatch, got [$2] expected [$3]"
        fi
}

typeset -A tests=()


typeset -A tests[01]=()
tests[01][desc]="hexadecimal lowercase"
tests[01][format]='%04x'
tests[01][args]="255"
tests[01][result]="00ff"

typeset -A tests[02]=()
tests[02][desc]="hexadecimal 32-bit"
tests[02][format]='%08x'
tests[02][args]='65537'
tests[02][result]=00010001
```



```
typeset -A tests[03]=()
tests[03][desc]="multiple arguments"
tests[03][format]='%d %s '
tests[03][args]="1 one 2 two 3 three"
tests[03][result]='1 one 2 two 3 three '

typeset -A tests[04]=()
tests[04][desc]="variable position parameters"
tests[04][format]='%2$s %1$d '
tests[04][args]="1 one 2 two 3 three"
tests[04][result]='one 1 two 2 three 3 '

typeset -A tests[05]=()
tests[05][desc]="width"
tests[05][format]='%10s'
tests[05][args]="abcdef"
tests[05][result]='    abcdef'

typeset -A tests[06]=()
tests[06][desc]="width and precision"
tests[06][format]='%10.3s'
tests[06][args]="abcdef"
tests[06][result]='       abc'

typeset -A tests[07]=()
tests[07][desc]="variable width and precision"
tests[07][format]='%*.*s'
tests[07][args]="10 3 abcdef"
tests[07][result]='       abc'

typeset -A tests[08]=()
tests[08][desc]="variable position width and precision"
tests[08][format]='%2$*1$.*3$s'
tests[08][args]="10 abcdef 3"
tests[08][result]='       abc'

typeset -A tests[09]=()
tests[09][desc]="multi variable position width and precision"
tests[09][format]='%2$*1$.*3$s'
tests[09][args]="10 abcdef 3 5 xyz 1"
tests[09][result]='       abc    x'

typeset -A tests[10]=()
tests[10][desc]="decimal from hex"
tests[10][format]='%d '
tests[10][args]="0x1000 0XA"
tests[10][result]='4096 10 '

typeset -A tests[11]=()
tests[11][desc]="negative dec (64-bit)"
tests[11][format]='%x'
tests[11][args]="-1"
tests[11][result]='ffffffffffffffff'

typeset -A tests[12]=()
tests[12][desc]="float (basic)"
tests[12][format]='%f'
tests[12][args]="3.14"
tests[12][result]='3.140000'

typeset -A tests[12]=()
tests[12][desc]="float precision"
tests[12][format]='%.2f'
tests[12][args]="3.14159"
tests[12][result]='3.14'
```

```
typeset -A tests[13]=()
tests[13][desc]="left justify"
tests[13][format]='%-5d'
tests[13][args]="45"
tests[13][result]='45   '

typeset -A tests[14]=()
tests[14][desc]="newlines"
tests[14][format]='%s\n%s\n%s'
tests[14][args]="one two three"
tests[14][result]='one
two
three'

typeset -A tests[15]=()
tests[15][desc]="embedded octal escape"
tests[15][format]='%s\41%s'
tests[15][args]="one two"
tests[15][result]='one!two'

typeset -A tests[16]=()
tests[16][desc]="backslash string (%b)"
tests[16][format]='%b'
tests[16][args]='\0101\0102\0103'
tests[16][result]='ABC'

typeset -A tests[17]=()
tests[17][desc]="backslash c in %b"
tests[17][format]='%b%s'
tests[17][args]='\0101\cone two'
tests[17][result]='A'

typeset -A tests[18]=()
tests[18][desc]="backslash octal in format"
tests[18][format]='HI\1120K\0112tabbed\11again'
tests[18][args]=
tests[18][result]='HIJ0K        2tabbed again'

typeset -A tests[19]=()
tests[19][desc]="backslash octal in %b"
tests[19][format]="%b"
tests[19][args]='HI\0112K\011tabbed'
tests[19][result]='HIJK tabbed'

typeset -A tests[20]=()
tests[20][desc]="numeric %d and ASCII conversions"
tests[20][format]='%d '
tests[20][args]="3 +3 -3 \"3 \"+ '-"
tests[20][result]='3 3 -3 51 43 45 '

typeset -A tests[21]=()
tests[21][desc]="verify second arg only"
tests[21][format]='%2$s'
tests[21][args]='abc xyz'
tests[21][result]="xyz"

#debug=yes

for i in "${!tests[@]}"; do
        t=test_$i
        desc=${tests[$i][desc]}
        format=${tests[$i][format]}
        args="${tests[$i][args]}"
        result=${tests[$i][result]}

```



```
        test_start $t "${tests[$i][desc]}"
        [[ -n "$debug" ]] && echo $PRINTF "$format" "${args[@]}"
        comp=$($PRINTF "$format" ${args[@]})
        checkrv $t
        [[ -n "$debug" ]] && echo "got [$comp]"
        good=$result
        compare $t "$comp" "$good"
        test_pass $t
done
```