

new/usr/src/cmd/printf/printf.c

1

```
*****
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
*****
1 /*
2 * Copyright 2014 Garrett D'Amore <garrett@damore.org>
3 * Copyright 2010 Nexenta Systems, Inc. All rights reserved.
4 * Copyright (c) 1989, 1993
5 *      The Regents of the University of California. All rights reserved.
6 *
7 * Redistribution and use in source and binary forms, with or without
8 * modification, are permitted provided that the following conditions
9 * are met:
10 * 1. Redistributions of source code must retain the above copyright
11 *    notice, this list of conditions and the following disclaimer.
12 * 2. Redistributions in binary form must reproduce the above copyright
13 *    notice, this list of conditions and the following disclaimer in the
14 *    documentation and/or other materials provided with the distribution.
15 * 4. Neither the name of the University nor the names of its contributors
16 *    may be used to endorse or promote products derived from this software
17 *    without specific prior written permission.
18 *
19 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
20 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22 * ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
23 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
24 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
25 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
26 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
27 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
28 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
29 * SUCH DAMAGE.
30 */

32 #include <sys/types.h>

34 #include <err.h>
35 #include <errno.h>
36 #include <inttypes.h>
37 #include <limits.h>
38 #include <stdio.h>
39 #include <stdlib.h>
40 #include <string.h>
41 #include <unistd.h>
42 #include <alloca.h>
43 #include <cctype.h>
44 #include <locale.h>
45 #include <note.h>

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

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

53 #define _(x)    gettext(x)

55 #define PF(f, func) do {
56     char *b = NULL;
57     int dollar = 0;
58     if (*f == '$') { \
59         \
60         \
61         \
62         \
63         \
64         \
65         \
66         \
67         \
68         \
69         \
70         \
71         \
72         \
73         \
74         \
75         \
76         \
77         \
78         \
79         \
80         \
81         \
82         \
83         \
84         \
85         \
86         \
87         \
88         \
89         \
90         \
91         \
92         \
93         \
94         \
95         \
96         \
97         \
98         \
99         \
100        \
101        \
102        \
103        \
104        \
105        \
106        \
107        \
108        \
109        \
110        \
111        \
112        \
113        \
114        \
115        \
116        \
117        \
118        \
119        \
120        \
121        \
122        \
123        \
124        \
125        \
126        \
127        \
128        \
129        \
130        \
131        \
132        \
133        \
134        \
135        \
136        \
137        \
138        \
139        \
140        \
141        \
142        \
143        \
144        \
145        \
146        \
147        \
148        \
149        \
150        \
151        \
152        \
153        \
154        \
155        \
156        \
157        \
158        \
159        \
160        \
161        \
162        \
163        \
164        \
165        \
166        \
167        \
168        \
169        \
170        \
171        \
172        \
173        \
174        \
175        \
176        \
177        \
178        \
179        \
180        \
181        \
182        \
183        \
184        \
185        \
186        \
187        \
188        \
189        \
190        \
191        \
192        \
193        \
194        \
195        \
196        \
197        \
198        \
199        \
200        \
201        \
202        \
203        \
204        \
205        \
206        \
207        \
208        \
209        \
210        \
211        \
212        \
213        \
214        \
215        \
216        \
217        \
218        \
219        \
220        \
221        \
222        \
223        \
224        \
225        \
226        \
227        \
228        \
229        \
230        \
231        \
232        \
233        \
234        \
235        \
236        \
237        \
238        \
239        \
240        \
241        \
242        \
243        \
244        \
245        \
246        \
247        \
248        \
249        \
250        \
251        \
252        \
253        \
254        \
255        \
256        \
257        \
258        \
259        \
260        \
261        \
262        \
263        \
264        \
265        \
266        \
267        \
268        \
269        \
270        \
271        \
272        \
273        \
274        \
275        \
276        \
277        \
278        \
279        \
280        \
281        \
282        \
283        \
284        \
285        \
286        \
287        \
288        \
289        \
290        \
291        \
292        \
293        \
294        \
295        \
296        \
297        \
298        \
299        \
300        \
301        \
302        \
303        \
304        \
305        \
306        \
307        \
308        \
309        \
310        \
311        \
312        \
313        \
314        \
315        \
316        \
317        \
318        \
319        \
320        \
321        \
322        \
323        \
324        \
325        \
326        \
327        \
328        \
329        \
330        \
331        \
332        \
333        \
334        \
335        \
336        \
337        \
338        \
339        \
340        \
341        \
342        \
343        \
344        \
345        \
346        \
347        \
348        \
349        \
350        \
351        \
352        \
353        \
354        \
355        \
356        \
357        \
358        \
359        \
360        \
361        \
362        \
363        \
364        \
365        \
366        \
367        \
368        \
369        \
370        \
371        \
372        \
373        \
374        \
375        \
376        \
377        \
378        \
379        \
380        \
381        \
382        \
383        \
384        \
385        \
386        \
387        \
388        \
389        \
390        \
391        \
392        \
393        \
394        \
395        \
396        \
397        \
398        \
399        \
400        \
401        \
402        \
403        \
404        \
405        \
406        \
407        \
408        \
409        \
410        \
411        \
412        \
413        \
414        \
415        \
416        \
417        \
418        \
419        \
420        \
421        \
422        \
423        \
424        \
425        \
426        \
427        \
428        \
429        \
430        \
431        \
432        \
433        \
434        \
435        \
436        \
437        \
438        \
439        \
440        \
441        \
442        \
443        \
444        \
445        \
446        \
447        \
448        \
449        \
450        \
451        \
452        \
453        \
454        \
455        \
456        \
457        \
458        \
459        \
460        \
461        \
462        \
463        \
464        \
465        \
466        \
467        \
468        \
469        \
470        \
471        \
472        \
473        \
474        \
475        \
476        \
477        \
478        \
479        \
480        \
481        \
482        \
483        \
484        \
485        \
486        \
487        \
488        \
489        \
490        \
491        \
492        \
493        \
494        \
495        \
496        \
497        \
498        \
499        \
500        \
501        \
502        \
503        \
504        \
505        \
506        \
507        \
508        \
509        \
510        \
511        \
512        \
513        \
514        \
515        \
516        \
517        \
518        \
519        \
520        \
521        \
522        \
523        \
524        \
525        \
526        \
527        \
528        \
529        \
530        \
531        \
532        \
533        \
534        \
535        \
536        \
537        \
538        \
539        \
540        \
541        \
542        \
543        \
544        \
545        \
546        \
547        \
548        \
549        \
550        \
551        \
552        \
553        \
554        \
555        \
556        \
557        \
558        \
559        \
560        \
561        \
562        \
563        \
564        \
565        \
566        \
567        \
568        \
569        \
570        \
571        \
572        \
573        \
574        \
575        \
576        \
577        \
578        \
579        \
580        \
581        \
582        \
583        \
584        \
585        \
586        \
587        \
588        \
589        \
589        \
590        \
591        \
592        \
593        \
594        \
595        \
596        \
597        \
598        \
599        \
599        \
600        \
601        \
602        \
603        \
604        \
605        \
606        \
607        \
608        \
609        \
609        \
610        \
611        \
612        \
613        \
614        \
615        \
616        \
617        \
618        \
619        \
619        \
620        \
621        \
622        \
623        \
624        \
625        \
626        \
627        \
628        \
629        \
629        \
630        \
631        \
632        \
633        \
634        \
635        \
636        \
637        \
638        \
639        \
639        \
640        \
641        \
642        \
643        \
644        \
645        \
646        \
647        \
648        \
649        \
649        \
650        \
651        \
652        \
653        \
654        \
655        \
656        \
657        \
658        \
659        \
659        \
660        \
661        \
662        \
663        \
664        \
665        \
666        \
667        \
668        \
669        \
669        \
670        \
671        \
672        \
673        \
674        \
675        \
676        \
677        \
678        \
679        \
679        \
680        \
681        \
682        \
683        \
684        \
685        \
686        \
687        \
688        \
689        \
689        \
690        \
691        \
692        \
693        \
694        \
695        \
696        \
697        \
698        \
699        \
699        \
700        \
701        \
702        \
703        \
704        \
705        \
706        \
707        \
708        \
709        \
709        \
710        \
711        \
712        \
713        \
714        \
715        \
716        \
717        \
718        \
719        \
719        \
720        \
721        \
722        \
723        \
724        \
725        \
726        \
727        \
728        \
729        \
729        \
730        \
731        \
732        \
733        \
734        \
735        \
736        \
737        \
738        \
739        \
739        \
740        \
741        \
742        \
743        \
744        \
745        \
746        \
747        \
748        \
749        \
749        \
750        \
751        \
752        \
753        \
754        \
755        \
756        \
757        \
758        \
759        \
759        \
760        \
761        \
762        \
763        \
764        \
765        \
766        \
767        \
768        \
769        \
769        \
770        \
771        \
772        \
773        \
774        \
775        \
776        \
777        \
778        \
779        \
779        \
780        \
781        \
782        \
783        \
784        \
785        \
786        \
787        \
788        \
789        \
789        \
790        \
791        \
792        \
793        \
794        \
795        \
796        \
797        \
798        \
799        \
799        \
800        \
801        \
802        \
803        \
804        \
805        \
806        \
807        \
808        \
809        \
809        \
810        \
811        \
812        \
813        \
814        \
815        \
816        \
817        \
818        \
819        \
819        \
820        \
821        \
822        \
823        \
824        \
825        \
826        \
827        \
828        \
829        \
829        \
830        \
831        \
832        \
833        \
834        \
835        \
836        \
837        \
838        \
839        \
839        \
840        \
841        \
842        \
843        \
844        \
845        \
846        \
847        \
848        \
849        \
849        \
850        \
851        \
852        \
853        \
854        \
855        \
856        \
857        \
858        \
859        \
859        \
860        \
861        \
862        \
863        \
864        \
865        \
866        \
867        \
868        \
869        \
869        \
870        \
871        \
872        \
873        \
874        \
875        \
876        \
877        \
878        \
879        \
879        \
880        \
881        \
882        \
883        \
884        \
885        \
886        \
887        \
888        \
889        \
889        \
890        \
891        \
892        \
893        \
894        \
895        \
896        \
897        \
898        \
899        \
899        \
900        \
901        \
902        \
903        \
904        \
905        \
906        \
907        \
908        \
909        \
909        \
910        \
911        \
912        \
913        \
914        \
915        \
916        \
917        \
918        \
919        \
919        \
920        \
921        \
922        \
923        \
924        \
925        \
926        \
927        \
928        \
929        \
929        \
930        \
931        \
932        \
933        \
934        \
935        \
936        \
937        \
938        \
939        \
939        \
940        \
941        \
942        \
943        \
944        \
945        \
946        \
947        \
948        \
949        \
949        \
950        \
951        \
952        \
953        \
954        \
955        \
956        \
957        \
958        \
959        \
959        \
960        \
961        \
962        \
963        \
964        \
965        \
966        \
967        \
968        \
969        \
969        \
970        \
971        \
972        \
973        \
974        \
975        \
976        \
977        \
978        \
979        \
979        \
980        \
981        \
982        \
983        \
984        \
985        \
986        \
987        \
988        \
989        \
989        \
990        \
991        \
992        \
993        \
994        \
995        \
996        \
997        \
998        \
999        \
999        \
1000       \
1001      \
1002      \
1003      \
1004      \
1005      \
1006      \
1007      \
1008      \
1009      \
10010     \
10011     \
10012     \
10013     \
10014     \
10015     \
10016     \
10017     \
10018     \
10019     \
10020     \
10021     \
10022     \
10023     \
10024     \
10025     \
10026     \
10027     \
10028     \
10029     \
10029     \
10030     \
10031     \
10032     \
10033     \
10034     \
10035     \
10036     \
10037     \
10038     \
10039     \
10039     \
10040     \
10041     \
10042     \
10043     \
10044     \
10045     \
10046     \
10047     \
10048     \
10049     \
10049     \
10050     \
10051     \
10052     \
10053     \
10054     \
10055     \
10056     \
10057     \
10058     \
10059     \
10059     \
10060     \
10061     \
10062     \
10063     \
10064     \
10065     \
10066     \
10067     \
10068     \
10069     \
10069     \
10070     \
10071     \
10072     \
10073     \
10074     \
10075     \
10075     \
10076     \
10077     \
10078     \
10079     \
10079     \
10080     \
10081     \
10082     \
10083     \
10084     \
10085     \
10085     \
10086     \
10087     \
10088     \
10089     \
10089     \
10090     \
10091     \
10092     \
10093     \
10094     \
10095     \
10095     \
10096     \
10097     \
10098     \
10099     \
10099     \
100100    \
100101    \
100102    \
100103    \
100104    \
100105    \
100106    \
100107    \
100108    \
100109    \
100110    \
100111    \
100112    \
100113    \
100114    \
100115    \
100116    \
100117    \
100118    \
100119    \
100120    \
100121    \
100122    \
100123    \
100124    \
100125    \
100126    \
100127    \
100128    \
100129    \
100130    \
100131    \
100132    \
100133    \
100134    \
100135    \
100136    \
100137    \
100138    \
100139    \
100140    \
100141    \
100142    \
100143    \
100144    \
100145    \
100146    \
100147    \
100148    \
100149    \
100150    \
100151    \
100152    \
100153    \
100154    \
100155    \
100156    \
100157    \
100158    \
100159    \
100160    \
100161    \
100162    \
100163    \
100164    \
100165    \
100166    \
100167    \
100168    \
100169    \
100170    \
100171    \
100172    \
100173    \
100174    \
100175    \
100176    \
100177    \
100178    \
100179    \
100179    \
100180    \
100181    \
100182    \
100183    \
100184    \
100185    \
100186    \
100187    \
100188    \
100189    \
100189    \
100190    \
100191    \
100192    \
100193    \
100194    \
100195    \
100196    \
100197    \
100198    \
100199    \
100199    \
100200    \
100201    \
100202    \
100203    \
100204    \
100205    \
100206    \
100207    \
100208    \
100209    \
100209    \
100210    \
100211    \
100212    \
100213    \
100214    \
100215    \
100216    \
100217    \
100218    \
100219    \
100219    \
100220    \
100221    \
100222    \
100223    \
100224    \
100225    \
100226    \
100227    \
100228    \
100229    \
100229    \
100230    \
100231    \
100232    \
100233    \
100234    \
100235    \
100236    \
100237    \
100238    \
100239    \
100239    \
100240    \
100241    \
100242    \
100243    \
100244    \
100245    \
100246    \
100247    \
100248    \
100249    \
100249    \
100250    \
100251    \
100252    \
100253    \
100254    \
100255    \
100256    \
100257    \
100258    \
100259    \
100259    \
100260    \
100261    \
100262    \
100263    \
100264    \
100265    \
100266    \
100267    \
100268    \
100269    \
100269    \
100270    \
100271    \
100272    \
100273    \
100274    \
100275    \
100276    \
100277    \
100278    \
100279    \
100279    \
100280    \
100281    \
100282    \
100283    \
100284    \
100285    \
100286    \
100287    \
100288    \
100289    \
100289    \
100290    \
100291    \
100292    \
100293    \
100294    \
100295    \
100296    \
100297    \
100298    \
100299    \
100299    \
100300    \
100301    \
100302    \
100303    \
100304    \
100305    \
100306    \
100307    \
100308    \
100309    \
100309    \
100310    \
100311    \
100312    \
100313    \
100314    \
100315    \
100316    \
100317    \
100318    \
100319    \
100319    \
100320    \
100321    \
100322    \
100323    \
100324    \
100325    \
100326    \
100327    \
100328    \
100329    \
100329    \
100330    \
100331    \
100332    \
100333    \
100334    \
100335    \
100336    \
100337    \
100338    \
100339    \
100339    \
100340    \
100341    \
100342    \
100343    \
100344    \
100345    \
100346    \
100347    \
100348    \
100349    \
100349    \
100350    \
100351    \
100352    \
100353    \
100354    \
100355    \
100356    \
100357    \
100358    \
100359    \
100359    \
100360    \
100361    \
100362    \
100363    \
100364    \
100365    \
100366    \
100367    \
100368    \
100369    \
100369    \
100370    \
100371    \
100372    \
100373    \
100374    \
100375    \
100376    \
100377    \
100378    \
100379    \
100379    \
100380    \
100381    \
100382    \
100383    \
100384    \
100385    \
100386    \
100387    \
100388    \
100389    \
100389    \
100390    \
100391    \
100392    \
100393    \
100394    \
100395    \
100396    \
100397    \
100398    \
100399    \
100399    \
100400    \
100401    \
100402    \
100403    \
100404    \
100405    \
100406    \
100407    \
100408    \
100409    \
100409    \
100410    \
100411    \
100412    \
100413    \
100414    \
100415    \
100416    \
100417    \
100418    \
100419    \
100419    \
100420    \
100421    \
100422    \
100423    \
100424    \
100425    \
100426    \
100427    \
100428    \
100429    \
100429    \
100430    \
100431    \
100432    \
100433    \
100434    \
100435    \
100436    \
100437    \
100438    \
100439    \
100439    \
100440    \
100441    \
100442    \
100443    \
100444    \
100445    \
100446    \
100447    \
100448    \
100449    \
100449    \
100450    \
100451    \
100452    \
100453    \
100454    \
100455    \
100456    \
100457    \
100458    \
100459    \
100459    \
100460    \
100461    \
100462    \
100463    \
100464    \
100465    \
100466    \
100467    \
100468    \
100469    \
100469    \
100470    \
100471    \
100472    \
100473    \
100474    \
100475    \
100476    \
100477    \
100478    \
100479    \
100479    \
100480    \
100481    \
100482    \
100483    \
100484    \
100485    \
100486    \
100487    \
100488    \
100489    \
100489    \
100490    \
100491    \
100492    \
100493    \
100494    \
100495    \
100496    \
100497    \
100498    \
100499    \
100499    \
100500    \
100501    \
100502    \
100503    \
100504    \
100505    \
100506    \
100507    \
100508    \
100509    \
100509    \
100510    \
100511    \
100512    \
100513    \
100514    \
100515    \
100516    \
100517    \
100518    \
100519    \
100519    \
100520    \
100521    \
100522    \
100523    \
100524    \
100525    \
100526    \
100527    \
100528    \
100529    \
100529    \
100530    \
100531    \
100532    \
100533    \
100534    \
100535    \
100536    \
100537    \
100538    \
100539    \
100539    \
100540    \
100541    \
100542    \
100543    \
100544    \
100545    \
100546    \
100547    \
100548    \
100549    \
100549    \
100550    \
100551    \
100552    \
100553    \
100554    \
100555    \
100556    \
100557    \
100558    \
100559    \
100559    \
100560    \
100561    \
100562    \
100563    \
100564    \
100565    \
100566    \
100567    \
100568    \
100569    \
100569    \
100570    \
100571    \
100572    \
100573    \
100574    \
100575    \
100576    \
100577    \
100578    \
100579    \
100579    \
100580    \
100581    \
100582    \
100583    \
100584    \
100585    \
100586    \
100587    \
100588    \
100589    \
100589    \
100590    \
100591    \
100592    \
100593    \
100594    \
100595    \
100596    \
100597    \
100598    \
100599    \
100599    \
100600    \
100601    \
100602    \
100603    \
100604    \
100605    \
100606    \
100607    \
100608    \
100609    \
100609    \
100610    \
100611    \
100612    \
100613    \
100614    \
100615    \
100616    \
100617    \
100618    \
100619    \
100619    \
100620    \
100621    \
100622    \
100623    \
100624    \
100625    \
100626    \
100627    \
100628    \
100629    \
100629    \
100630    \
100631    \
100632    \
100633    \
100634    \
100635    \
100636    \
100637    \
100638    \
100639    \
100639    \
100640    \
100641    \
100642    \
100643    \
100644    \
100645    \
100646    \
100647    \
100648    \
100649    \
100649    \
100650    \
100651    \
100652    \
100653    \
100654    \
100655    \
100656    \
1006
```

```

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 */
129    rval = end = 0;
130    argv = ++argv;
131
132    for (;;) {
133        maxargv = argv;
134        char **maxargv = argv;
135
136        myargv = argv;
137        for (myargc = 0; argv[myargc]; myargc++)
138            /* nop */;
139        start = fmt;
140        while (fmt < format + len) {
141            if (fmt[0] == '%') {
142                (void) fwrite(start, 1, PTRDIFF(fmt, start),
143                             stdout);
144                if (fmt[1] == '%') {
145                    /* %% prints a % */
146                    (void) putchar('%');
147                    fmt += 2;
148                } else {
149                    fmt = doformat(fmt, &rval);
150                    if (fmt == NULL)
151                        return (1);
152                    end = 0;
153                }
154            }
155            start = fmt;
156            if (argv > maxargv)
157                maxargv = argv;
158        }
159        argv = maxargv;
160
161        if (end == 1) {
162            warnx1_( "missing format character"), NULL, NULL);
163            return (1);
164        }
165        (void) fwrite(start, 1, PTRDIFF(fmt, start), stdout);
166        if (!*argv)
167            if (chopped || !*argv)
168                return (rval);
169        /* Restart at the beginning of the format string. */
170        fmt = format;
171        end = 1;
172    } /* NOTREACHED */
173
174 static char *
175 doformat(char *fmt, int *rval)
176 doformat(char *start, int *rval)
177 {
178     static const char skip1[] = "#'-+ 0";

```

```

180     static const char skip2[] = "0123456789";
181     char *fmt;
182     int fieldwidth, haveprec, havewidth, mod_ldbl, precision;
183     char convch, nextch;
184     char *start;
185     char **fargv;
186     char *dptr;
187     int l;
188
189     start = alloca(strlen(fmt) + 1);
190     fmt = start + 1;
191
192     dptr = start;
193     *dptr++ = '%';
194     *dptr = 0;
195
196     fmt++;
197
198     /* look for "n$" field index specifier */
199     l = strspn(fmt, digits);
200     if ((l > 0) && (fmt[l] == '$')) {
201         int idx = atoi(fmt);
202         fmt += strspn(fmt, skip2);
203         if ((*fmt == '$') && (fmt != (start + 1))) {
204             int idx = atoi(start + 1);
205             if (idx <= myargc) {
206                 argv = &myargv[idx - 1];
207             } else {
208                 argv = &myargv[myargc];
209             }
210             if (argv > maxargv) {
211                 maxargv = argv;
212             }
213             fmt += 1 + 1;
214
215             /* save format argument */
216             fargv = argv;
217             start = fmt;
218             fmt++;
219         } else {
220             fargv = NULL;
221             fmt = start + 1;
222         }
223
224         /* skip to field width */
225         l = strspn(fmt, digits);
226         if ((l > 0) && (fmt[l] == '$')) {
227             int idx = atoi(fmt);
228             if (fargv == NULL) {
229                 warnx1_( "incomplete use of n$"), NULL, NULL);
230                 return (NULL);
231             }
232             if (idx <= myargc) {
233                 argv = &myargv[idx - 1];
234             } else {
235                 argv = &myargv[myargc];
236             }
237         }
238     }
239
240     /* skip to field width */
241     l = strspn(fmt, digits);
242     if ((l > 0) && (fmt[l] == '$')) {
243         int idx = atoi(fmt);
244         if (fargv == NULL) {
245             warnx1_( "incomplete use of n$"), NULL, NULL);
246             return (NULL);
247         }
248         if (idx <= myargc) {
249             argv = &myargv[idx - 1];
250         } else {
251             argv = &myargv[myargc];
252         }
253     }
254
255     /* skip to field width */
256     l = strspn(fmt, digits);
257     if ((l > 0) && (fmt[l] == '$')) {
258         int idx = atoi(fmt);
259         if (fargv == NULL) {
260             warnx1_( "incomplete use of n$"), NULL, NULL);
261             return (NULL);
262         }
263         if (idx <= myargc) {
264             argv = &myargv[idx - 1];
265         } else {
266             argv = &myargv[myargc];
267         }
268     }
269
270     /* skip to field width */
271     l = strspn(fmt, digits);
272     if ((l > 0) && (fmt[l] == '$')) {
273         int idx = atoi(fmt);
274         if (fargv == NULL) {
275             warnx1_( "incomplete use of n$"), NULL, NULL);
276             return (NULL);
277         }
278         if (idx <= myargc) {
279             argv = &myargv[idx - 1];
280         } else {
281             argv = &myargv[myargc];
282         }
283     }
284
285     /* skip to field width */
286     l = strspn(fmt, digits);
287     if ((l > 0) && (fmt[l] == '$')) {
288         int idx = atoi(fmt);
289         if (fargv == NULL) {
290             warnx1_( "incomplete use of n$"), NULL, NULL);
291             return (NULL);
292         }
293         if (idx <= myargc) {
294             argv = &myargv[idx - 1];
295         } else {
296             argv = &myargv[myargc];
297         }
298     }
299
300     /* skip to field width */
301     l = strspn(fmt, digits);
302     if ((l > 0) && (fmt[l] == '$')) {
303         int idx = atoi(fmt);
304         if (fargv == NULL) {
305             warnx1_( "incomplete use of n$"), NULL, NULL);
306             return (NULL);
307         }
308         if (idx <= myargc) {
309             argv = &myargv[idx - 1];
310         } else {
311             argv = &myargv[myargc];
312         }
313     }
314
315     /* skip to field width */
316     l = strspn(fmt, digits);
317     if ((l > 0) && (fmt[l] == '$')) {
318         int idx = atoi(fmt);
319         if (fargv == NULL) {
320             warnx1_( "incomplete use of n$"), NULL, NULL);
321             return (NULL);
322         }
323         if (idx <= myargc) {
324             argv = &myargv[idx - 1];
325         } else {
326             argv = &myargv[myargc];
327         }
328     }
329
330     /* skip to field width */
331     l = strspn(fmt, digits);
332     if ((l > 0) && (fmt[l] == '$')) {
333         int idx = atoi(fmt);
334         if (fargv == NULL) {
335             warnx1_( "incomplete use of n$"), NULL, NULL);
336             return (NULL);
337         }
338         if (idx <= myargc) {
339             argv = &myargv[idx - 1];
340         } else {
341             argv = &myargv[myargc];
342         }
343     }
344
345     /* skip to field width */
346     l = strspn(fmt, digits);
347     if ((l > 0) && (fmt[l] == '$')) {
348         int idx = atoi(fmt);
349         if (fargv == NULL) {
350             warnx1_( "incomplete use of n$"), NULL, NULL);
351             return (NULL);
352         }
353         if (idx <= myargc) {
354             argv = &myargv[idx - 1];
355         } else {
356             argv = &myargv[myargc];
357         }
358     }
359
360     /* skip to field width */
361     l = strspn(fmt, digits);
362     if ((l > 0) && (fmt[l] == '$')) {
363         int idx = atoi(fmt);
364         if (fargv == NULL) {
365             warnx1_( "incomplete use of n$"), NULL, NULL);
366             return (NULL);
367         }
368         if (idx <= myargc) {
369             argv = &myargv[idx - 1];
370         } else {
371             argv = &myargv[myargc];
372         }
373     }
374
375     /* skip to field width */
376     l = strspn(fmt, digits);
377     if ((l > 0) && (fmt[l] == '$')) {
378         int idx = atoi(fmt);
379         if (fargv == NULL) {
380             warnx1_( "incomplete use of n$"), NULL, NULL);
381             return (NULL);
382         }
383         if (idx <= myargc) {
384             argv = &myargv[idx - 1];
385         } else {
386             argv = &myargv[myargc];
387         }
388     }
389
390     /* skip to field width */
391     l = strspn(fmt, digits);
392     if ((l > 0) && (fmt[l] == '$')) {
393         int idx = atoi(fmt);
394         if (fargv == NULL) {
395             warnx1_( "incomplete use of n$"), NULL, NULL);
396             return (NULL);
397         }
398         if (idx <= myargc) {
399             argv = &myargv[idx - 1];
400         } else {
401             argv = &myargv[myargc];
402         }
403     }
404
405     /* skip to field width */
406     l = strspn(fmt, digits);
407     if ((l > 0) && (fmt[l] == '$')) {
408         int idx = atoi(fmt);
409         if (fargv == NULL) {
410             warnx1_( "incomplete use of n$"), NULL, NULL);
411             return (NULL);
412         }
413         if (idx <= myargc) {
414             argv = &myargv[idx - 1];
415         } else {
416             argv = &myargv[myargc];
417         }
418     }
419
420     /* skip to field width */
421     l = strspn(fmt, digits);
422     if ((l > 0) && (fmt[l] == '$')) {
423         int idx = atoi(fmt);
424         if (fargv == NULL) {
425             warnx1_( "incomplete use of n$"), NULL, NULL);
426             return (NULL);
427         }
428         if (idx <= myargc) {
429             argv = &myargv[idx - 1];
430         } else {
431             argv = &myargv[myargc];
432         }
433     }
434
435     /* skip to field width */
436     l = strspn(fmt, digits);
437     if ((l > 0) && (fmt[l] == '$')) {
438         int idx = atoi(fmt);
439         if (fargv == NULL) {
440             warnx1_( "incomplete use of n$"), NULL, NULL);
441             return (NULL);
442         }
443         if (idx <= myargc) {
444             argv = &myargv[idx - 1];
445         } else {
446             argv = &myargv[myargc];
447         }
448     }
449
450     /* skip to field width */
451     l = strspn(fmt, digits);
452     if ((l > 0) && (fmt[l] == '$')) {
453         int idx = atoi(fmt);
454         if (fargv == NULL) {
455             warnx1_( "incomplete use of n$"), NULL, NULL);
456             return (NULL);
457         }
458         if (idx <= myargc) {
459             argv = &myargv[idx - 1];
460         } else {
461             argv = &myargv[myargc];
462         }
463     }
464
465     /* skip to field width */
466     l = strspn(fmt, digits);
467     if ((l > 0) && (fmt[l] == '$')) {
468         int idx = atoi(fmt);
469         if (fargv == NULL) {
470             warnx1_( "incomplete use of n$"), NULL, NULL);
471             return (NULL);
472         }
473         if (idx <= myargc) {
474             argv = &myargv[idx - 1];
475         } else {
476             argv = &myargv[myargc];
477         }
478     }
479
480     /* skip to field width */
481     l = strspn(fmt, digits);
482     if ((l > 0) && (fmt[l] == '$')) {
483         int idx = atoi(fmt);
484         if (fargv == NULL) {
485             warnx1_( "incomplete use of n$"), NULL, NULL);
486             return (NULL);
487         }
488         if (idx <= myargc) {
489             argv = &myargv[idx - 1];
490         } else {
491             argv = &myargv[myargc];
492         }
493     }
494
495     /* skip to field width */
496     l = strspn(fmt, digits);
497     if ((l > 0) && (fmt[l] == '$')) {
498         int idx = atoi(fmt);
499         if (fargv == NULL) {
500             warnx1_( "incomplete use of n$"), NULL, NULL);
501             return (NULL);
502         }
503         if (idx <= myargc) {
504             argv = &myargv[idx - 1];
505         } else {
506             argv = &myargv[myargc];
507         }
508     }
509
510     /* skip to field width */
511     l = strspn(fmt, digits);
512     if ((l > 0) && (fmt[l] == '$')) {
513         int idx = atoi(fmt);
514         if (fargv == NULL) {
515             warnx1_( "incomplete use of n$"), NULL, NULL);
516             return (NULL);
517         }
518         if (idx <= myargc) {
519             argv = &myargv[idx - 1];
520         } else {
521             argv = &myargv[myargc];
522         }
523     }
524
525     /* skip to field width */
526     l = strspn(fmt, digits);
527     if ((l > 0) && (fmt[l] == '$')) {
528         int idx = atoi(fmt);
529         if (fargv == NULL) {
530             warnx1_( "incomplete use of n$"), NULL, NULL);
531             return (NULL);
532         }
533         if (idx <= myargc) {
534             argv = &myargv[idx - 1];
535         } else {
536             argv = &myargv[myargc];
537         }
538     }
539
540     /* skip to field width */
541     l = strspn(fmt, digits);
542     if ((l > 0) && (fmt[l] == '$')) {
543         int idx = atoi(fmt);
544         if (fargv == NULL) {
545             warnx1_( "incomplete use of n$"), NULL, NULL);
546             return (NULL);
547         }
548         if (idx <= myargc) {
549             argv = &myargv[idx - 1];
550         } else {
551             argv = &myargv[myargc];
552         }
553     }
554
555     /* skip to field width */
556     l = strspn(fmt, digits);
557     if ((l > 0) && (fmt[l] == '$')) {
558         int idx = atoi(fmt);
559         if (fargv == NULL) {
560             warnx1_( "incomplete use of n$"), NULL, NULL);
561             return (NULL);
562         }
563         if (idx <= myargc) {
564             argv = &myargv[idx - 1];
565         } else {
566             argv = &myargv[myargc];
567         }
568     }
569
570     /* skip to field width */
571     l = strspn(fmt, digits);
572     if ((l > 0) && (fmt[l] == '$')) {
573         int idx = atoi(fmt);
574         if (fargv == NULL) {
575             warnx1_( "incomplete use of n$"), NULL, NULL);
576             return (NULL);
577         }
578         if (idx <= myargc) {
579             argv = &myargv[idx - 1];
580         } else {
581             argv = &myargv[myargc];
582         }
583     }
584
585     /* skip to field width */
586     l = strspn(fmt, digits);
587     if ((l > 0) && (fmt[l] == '$')) {
588         int idx = atoi(fmt);
589         if (fargv == NULL) {
590             warnx1_( "incomplete use of n$"), NULL, NULL);
591             return (NULL);
592         }
593         if (idx <= myargc) {
594             argv = &myargv[idx - 1];
595         } else {
596             argv = &myargv[myargc];
597         }
598     }
599
600     /* skip to field width */
601     l = strspn(fmt, digits);
602     if ((l > 0) && (fmt[l] == '$')) {
603         int idx = atoi(fmt);
604         if (fargv == NULL) {
605             warnx1_( "incomplete use of n$"), NULL, NULL);
606             return (NULL);
607         }
608         if (idx <= myargc) {
609             argv = &myargv[idx - 1];
610         } else {
611             argv = &myargv[myargc];
612         }
613     }
614
615     /* skip to field width */
616     l = strspn(fmt, digits);
617     if ((l > 0) && (fmt[l] == '$')) {
618         int idx = atoi(fmt);
619         if (fargv == NULL) {
620             warnx1_( "incomplete use of n$"), NULL, NULL);
621             return (NULL);
622         }
623         if (idx <= myargc) {
624             argv = &myargv[idx - 1];
625         } else {
626             argv = &myargv[myargc];
627         }
628     }
629
630     /* skip to field width */
631     l = strspn(fmt, digits);
632     if ((l > 0) && (fmt[l] == '$')) {
633         int idx = atoi(fmt);
634         if (fargv == NULL) {
635             warnx1_( "incomplete use of n$"), NULL, NULL);
636             return (NULL);
637         }
638         if (idx <= myargc) {
639             argv = &myargv[idx - 1];
640         } else {
641             argv = &myargv[myargc];
642         }
643     }
644
645     /* skip to field width */
646     l = strspn(fmt, digits);
647     if ((l > 0) && (fmt[l] == '$')) {
648         int idx = atoi(fmt);
649         if (fargv == NULL) {
650             warnx1_( "incomplete use of n$"), NULL, NULL);
651             return (NULL);
652         }
653         if (idx <= myargc) {
654             argv = &myargv[idx - 1];
655         } else {
656             argv = &myargv[myargc];
657         }
658     }
659
660     /* skip to field width */
661     l = strspn(fmt, digits);
662     if ((l > 0) && (fmt[l] == '$')) {
663         int idx = atoi(fmt);
664         if (fargv == NULL) {
665             warnx1_( "incomplete use of n$"), NULL, NULL);
666             return (NULL);
667         }
668         if (idx <= myargc) {
669             argv = &myargv[idx - 1];
670         } else {
671             argv = &myargv[myargc];
672         }
673     }
674
675     /* skip to field width */
676     l = strspn(fmt, digits);
677     if ((l > 0) && (fmt[l] == '$')) {
678         int idx = atoi(fmt);
679         if (fargv == NULL) {
680             warnx1_( "incomplete use of n$"), NULL, NULL);
681             return (NULL);
682         }
683         if (idx <= myargc) {
684             argv = &myargv[idx - 1];
685         } else {
686             argv = &myargv[myargc];
687         }
688     }
689
690     /* skip to field width */
691     l = strspn(fmt, digits);
692     if ((l > 0) && (fmt[l] == '$')) {
693         int idx = atoi(fmt);
694         if (fargv == NULL) {
695             warnx1_( "incomplete use of n$"), NULL, NULL);
696             return (NULL);
697         }
698         if (idx <= myargc) {
699             argv = &myargv[idx - 1];
700         } else {
701             argv = &myargv[myargc];
702         }
703     }
704
705     /* skip to field width */
706     l = strspn(fmt, digits);
707     if ((l > 0) && (fmt[l] == '$')) {
708         int idx = atoi(fmt);
709         if (fargv == NULL) {
710             warnx1_( "incomplete use of n$"), NULL, NULL);
711             return (NULL);
712         }
713         if (idx <= myargc) {
714             argv = &myargv[idx - 1];
715         } else {
716             argv = &myargv[myargc];
717         }
718     }
719
720     /* skip to field width */
721     l = strspn(fmt, digits);
722     if ((l > 0) && (fmt[l] == '$')) {
723         int idx = atoi(fmt);
724         if (fargv == NULL) {
725             warnx1_( "incomplete use of n$"), NULL, NULL);
726             return (NULL);
727         }
728         if (idx <= myargc) {
729             argv = &myargv[idx - 1];
730         } else {
731             argv = &myargv[myargc];
732         }
733     }
734
735     /* skip to field width */
736     l = strspn(fmt, digits);
737     if ((l > 0) && (fmt[l] == '$')) {
738         int idx = atoi(fmt);
739         if (fargv == NULL) {
740             warnx1_( "incomplete use of n$"), NULL, NULL);
741             return (NULL);
742         }
743         if (idx <= myargc) {
744             argv = &myargv[idx - 1];
745         } else {
746             argv = &myargv[myargc];
747         }
748     }
749
750     /* skip to field width */
751     l = strspn(fmt, digits);
752     if ((l > 0) && (fmt[l] == '$')) {
753         int idx = atoi(fmt);
754         if (fargv == NULL) {
755             warnx1_( "incomplete use of n$"), NULL, NULL);
756             return (NULL);
757         }
758         if (idx <= myargc) {
759             argv = &myargv[idx - 1];
760         } else {
761             argv = &myargv[myargc];
762         }
763     }
764
765     /* skip to field width */
766     l = strspn(fmt, digits);
767     if ((l > 0) && (fmt[l] == '$')) {
768         int idx = atoi(fmt);
769         if (fargv == NULL) {
770             warnx1_( "incomplete use of n$"), NULL, NULL);
771             return (NULL);
772         }
773         if (idx <= myargc) {
774             argv = &myargv[idx - 1];
775         } else {
776             argv = &myargv[myargc];
777         }
778     }
779
780     /* skip to field width */
781     l = strspn(fmt, digits);
782     if ((l > 0) && (fmt[l] == '$')) {
783         int idx = atoi(fmt);
784         if (fargv == NULL) {
785             warnx1_( "incomplete use of n$"), NULL, NULL);
786             return (NULL);
787         }
788         if (idx <= myargc) {
789             argv = &myargv[idx - 1];
790         } else {
791             argv = &myargv[myargc];
792         }
793     }
794
795     /* skip to field width */
796     l = strspn(fmt, digits);
797     if ((l > 0) && (fmt[l] == '$')) {
798         int idx = atoi(fmt);
799         if (fargv == NULL) {
800             warnx1_( "incomplete use of n$"), NULL, NULL);
801             return (NULL);
802         }
803         if (idx <= myargc) {
804             argv = &myargv[idx - 1];
805         } else {
806             argv = &myargv[myargc];
807         }
808     }
809
810     /* skip to field width */
811     l = strspn(fmt, digits);
812     if ((l > 0) && (fmt[l] == '$')) {
813         int idx = atoi(fmt);
814         if (fargv == NULL) {
815             warnx1_( "incomplete use of n$"), NULL, NULL);
816             return (NULL);
817         }
818         if (idx <= myargc) {
819             argv = &myargv[idx - 1];
820         } else {
821             argv = &myargv[myargc];
822         }
823     }
824
825     /* skip to field width */
826     l = strspn(fmt, digits);
827     if ((l > 0) && (fmt[l] == '$')) {
828         int idx = atoi(fmt);
829         if (fargv == NULL) {
830             warnx1_( "incomplete use of n$"), NULL, NULL);
831             return (NULL);
832         }
833         if (idx <= myargc) {
834             argv = &myargv[idx - 1];
835         } else {
836             argv = &myargv[myargc];
837         }
838     }
839
840     /* skip to field width */
841     l = strspn(fmt, digits);
842     if ((l > 0) && (fmt[l] == '$')) {
843         int idx = atoi(fmt);
844         if (fargv == NULL) {
845             warnx1_( "incomplete use of n$"), NULL, NULL);
846             return (NULL);
847         }
848         if (idx <= myargc) {
849             argv = &myargv[idx - 1];
850         } else {
851             argv = &myargv[myargc];
852         }
853     }
854
855     /* skip to field width */
856     l = strspn(fmt, digits);
857     if ((l > 0) && (fmt[l] == '$')) {
858         int idx = atoi(fmt);
859         if (fargv == NULL) {
860             warnx1_( "incomplete use of n$"), NULL, NULL);
861             return (NULL);
862         }
863         if (idx <= myargc) {
864             argv = &myargv[idx - 1];
865         } else {
866             argv = &myargv[myargc];
867         }
868     }
869
870     /* skip to field width */
871     l = strspn(fmt, digits);
872     if ((l > 0) && (fmt[l] == '$')) {
873         int idx = atoi(fmt);
874         if (fargv == NULL) {
875             warnx1_( "incomplete use of n$"), NULL, NULL);
876             return (NULL);
877         }
878         if (idx <= myargc) {
879             argv = &myargv[idx - 1];
880         } else {
881             argv = &myargv[myargc];
882         }
883     }
884
885     /* skip to field width */
886     l = strspn(fmt, digits);
887     if ((l > 0) && (fmt[l] == '$')) {
888         int idx = atoi(fmt);
889         if (fargv == NULL) {
890             warnx1_( "incomplete use of n$"), NULL, NULL);
891             return (NULL);
892         }
893         if (idx <= myargc) {
894             argv = &myargv[idx - 1];
895         } else {
896             argv = &myargv[myargc];
897         }
898     }
899
900     /* skip to field width */
901     l = strspn(fmt, digits);
902     if ((l > 0) && (fmt[l] == '$')) {
903         int idx = atoi(fmt);
904         if (fargv == NULL) {
905             warnx1_( "incomplete use of n$"), NULL, NULL);
906             return (NULL);
907         }
908         if (idx <= myargc) {
909             argv = &myargv[idx - 1];
910         } else {
911             argv = &myargv[myargc];
912         }
913     }
914
915     /* skip to field width */
916     l = strspn(fmt, digits);
917     if ((l > 0) && (fmt[l] == '$')) {
918         int idx = atoi(fmt);
919         if (fargv == NULL) {
920             warnx1_( "incomplete use of n$"), NULL, NULL);
921             return (NULL);
922         }
923         if (idx <= myargc) {
924             argv = &myargv[idx - 1];
925         } else {
926             argv = &myargv[myargc];
927         }
928     }
929
930     /* skip to field width */
931     l = strspn(fmt, digits);
932     if ((l > 0) && (fmt[l] == '$')) {
933         int idx = atoi(fmt);
934         if (fargv == NULL) {
935             warnx1_( "incomplete use of n$"), NULL, NULL);
936             return (NULL);
937         }
938         if (idx <= myargc) {
939             argv = &myargv[idx - 1];
940         } else {
941             argv = &myargv[myargc];
942         }
943     }
944
945     /* skip to field width */
946     l = strspn(fmt, digits);
947     if ((l > 0) && (fmt[l] == '$')) {
948         int idx = atoi(fmt);
949         if (fargv == NULL) {
950             warnx1_( "incomplete use of n$"), NULL, NULL);
951             return (NULL);
952         }
953         if (idx <= myargc) {
954             argv = &myargv[idx - 1];
955         } else {
956             argv = &myargv[myargc];
957         }
958     }
959
960     /* skip to field width */
961     l = strspn(fmt, digits);
962     if ((l > 0) && (fmt[l] == '$')) {
963         int idx = atoi(fmt);
964         if (fargv == NULL) {
965             warnx1_( "incomplete use of n$"), NULL, NULL);
966             return (NULL);
967         }
968         if (idx <= myargc) {
969             argv = &myargv[idx - 1];
970         } else {
971             argv = &myargv[myargc];
972         }
973     }
974
975     /* skip to field width */
976     l = strspn(fmt, digits);
977     if ((l > 0) && (fmt[l] == '$')) {
978         int idx = atoi(fmt);
979         if (fargv == NULL) {
980             warnx1_( "incomplete use of n$"), NULL, NULL);
981             return (NULL);
982         }
983         if (idx <= myargc) {
984             argv = &myargv[idx - 1];
985         } else {
986             argv = &myargv[myargc];
987         }
988     }
989
990     /* skip to field width */
991     l = strspn(fmt, digits);
992     if ((l > 0) && (fmt[l] == '$')) {
993         int idx = atoi(fmt);
994         if (fargv == NULL) {
995             warnx1_( "incomplete use of n$"), NULL, NULL);
996             return (NULL);
997         }
998         if (idx <= myargc) {
999             argv = &myargv[idx - 1];
1000        } else {
1001            argv = &myargv[myargc];
1002        }
1003    }
1004
1005    /* NOTREACHED */
1006
1007    static const char skip1[] = "#'-+ 0";
1008
1009    static const char skip2[] = "0123456789";
1010
1011    static const char skip3[] = "0123456789";
1012
1013    static const char skip4[] = "0123456789";
1014
1015    static const char skip5[] = "0123456789";
1016
1017    static const char skip6[] = "0123456789";
1018
1019    static const char skip7[] = "0123456789";
1020
1021    static const char skip8[] = "0123456789";
1022
1023    static const char skip9[] = "0123456789";
1024
1025    static const char skip10[] = "0123456789";
1026
1027    static const char skip11[] = "0123456789";
1028
1029    static const char skip12[] = "0123456789";
1030
1031    static const char skip13[] = "0123456789";
1032
1033    static const char skip14[] = "0123456789";
1034
1035    static const char skip15[] = "0123456789";
1036
1037    static const char skip16[] = "0123456789";
1038
1039    static const char skip17[] = "0123456789";
1040
1041    static const char skip18[] = "0123456789";
1042
1043    static const char skip19[] = "0123456789";
1044
1045    static const char skip20[] = "0123456789";
1046
1047    static const char skip21[] = "0123456789";
1048
1049    static const char skip22[] = "0123456789";
1050
1051    static const char skip23[] = "0123456789";
1052
1053    static const char skip24[] = "0123456789";
1054
1055    static const char skip25[] = "0123456789";
1056
1057    static const char skip26[] = "0123456789";
1058
1059    static const char skip27[] = "0123456789";
1060
1061    static const char skip28[] = "0123456789";
1062
1063    static const char skip29[] = "0123456789";
1064
1065    static const char skip30[] = "0123456789";
1066
1067    static const char skip31[] = "0123456789";
1068
1069    static const char skip32[] = "0123456789";
1070
1071    static const char skip33[] = "0123456789";
1072
1073    static const char skip34[] = "0123456789";
1074
1075    static const char skip35[] = "0123456789";
1076
1077    static const char skip36[] = "0123456789";
1078
1079    static const char skip37[] = "0123456789";
1080
1081    static const char skip38[] = "0123456789";
1082
1083    static const char skip39[] = "01234567
```

```

235         }
236         fmt += 1 + 1;
237     } else if (fargv != NULL) {
238         warnx1(_("incomplete use of n$"), NULL, NULL);
239         return (NULL);
240     }
241
242     if (getint(&fieldwidth))
243         return (NULL);
244     if (gargv > maxargv) {
245         maxargv = gargv;
246     }
247     havewidth = 1;
248
249     *dptr++ = '*';
250     *dptr = 0;
251     ++fmt;
252 } else {
253     havewidth = 0;
254
255     /* skip to possible '.', get following precision */
256     while (isdigit(*fmt)) {
257         *dptr++ = *fmt++;
258         *dptr = 0;
259     }
260
261     if (*fmt == '.') {
262         /* precision present? */
263         fmt++;
264         *dptr++ = '.';
265
266         ++fmt;
267         if (*fmt == '*') {
268
269             fmt++;
270             l = strspn(fmt, digits);
271             if ((l > 0) && (fmt[l] == '$')) {
272                 int idx = atoi(fmt);
273                 if (fargv == NULL) {
274                     warnx1(_("incomplete use of n$"),
275                           NULL, NULL);
276                     return (NULL);
277                 }
278                 if (idx <= myargc) {
279                     gargv = &myargv[idx - 1];
280                 } else {
281                     gargv = &myargv[myargc];
282                 }
283                 fmt += l + 1;
284             } else if (fargv != NULL) {
285                 warnx1(_("incomplete use of n$"), NULL, NULL);
286                 return (NULL);
287             }
288
289             if (getint(&precision))
290                 return (NULL);
291             if (gargv > maxargv) {
292                 maxargv = gargv;
293             }
294             haveprec = 1;
295             *dptr++ = '*';
296             *dptr = 0;
297             ++fmt;
298 } else {

```

```

297
298         haveprec = 0;
299
300         /* skip to conversion char */
301         while (isdigit(*fmt)) {
302             *dptr++ = *fmt++;
303             *dptr = 0;
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;
313
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     }
334
335     /* save the current arg offset, and set to the format arg */
336     if (fargv != NULL) {
337         gargv = fargv;
338     }
339
340     convch = *fmt;
341     nextch = *++fmt;
342
343     *fmt = '\0';
344     switch (convch) {
345     case 'b':
346         size_t len;
347         char *p;
348         int getout;
349
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);
357         *(fmt - 1) = 's';
358         PF(start, p);
359         *(fmt - 1) = 'b';
360         free(p);
361     }

```

```

359         if (getout)
360             exit(*rval);
361         break;
362     }
363     case 'c': {
364         char p;
365         p = getchr();
366         PF(start, p);
367         break;
368     }
369     case 's': {
370         const char *p;
371         p = getstr();
372         PF(start, p);
373         break;
374     }
375     case 'd': case 'i': case 'o': case 'u': case 'x': case 'X': {
376         char *f;
377         intmax_t val;
378         uintmax_t uval;
379         int signedconv;
380
381         signedconv = (convch == 'd' || convch == 'i');
382         if ((f = mknum(start, convch)) == NULL)
383             return (NULL);
384         if (getnum(&val, &uval, signedconv))
385             *rval = 1;
386         if (signedconv)
387             PF(f, val);
388         else
389             PF(f, uval);
390         break;
391     }
392     case 'e': case 'E':
393     case 'f': case 'F':
394     case 'g': case 'G':
395     case 'a': case 'A': {
396         long double p;
397
398         if (getfloating(&p, mod_ldbl))
399             *rval = 1;
400         if (mod_ldbl)
401             PF(start, p);
402         else
403             PF(start, (double)p);
404         break;
405     }
406     default:
407         warnx2(_("illegal format character %c"), convch, NULL);
408         return (NULL);
409     }
410     fmt = nextch;
411
412     /* return the argv to the next element */
413     return (fmt);
414 }
415
416 unchanged_portion_omitted_
444 static int
445 escape(char *fmt, int percent, size_t *len)
446 {
447     char *save, *store, c;
448     int value;

```

```

450     for (save = store = fmt; ((c = *fmt) != 0); ++fmt, ++store) {
451         if (c != '\\') {
452             *store = c;
453             continue;
454         }
455         switch (++fmt) {
456             case '\0': /* EOS, user error */
457                 *store = '\\';
458                 ++store = '\0';
459                 *len = PTRDIFF(store, save);
460                 return (0);
461             case '\\': /* backslash */
462                 *store = *fmt;
463                 break;
464             case 'a': /* bell/alert */
465                 *store = '\a';
466                 break;
467             case 'b': /* backspace */
468                 *store = '\b';
469                 break;
470             case 'c':
471                 if (!percent) {
472                     *store = '\0';
473                     *len = PTRDIFF(store, save);
474                     return (1);
475                 }
476                 *store = 'c';
477                 break;
478             case 'f': /* form-feed */
479                 *store = '\f';
480                 break;
481             case 'n': /* newline */
482                 *store = '\n';
483                 break;
484             case 'r': /* carriage-return */
485                 *store = '\r';
486                 break;
487             case 't': /* horizontal tab */
488                 *store = '\t';
489                 break;
490             case 'v': /* vertical tab */
491                 *store = '\v';
492                 break;
493             case '0': /* octal constant */
494             case '1': case '2': case '3':
495             case '4': case '5': case '6': case '7':
496                 c = (percent && *fmt == '0') ? 4 : 3;
497                 for (value = 0;
498                     --&& *fmt >= '0' && *fmt <= '7'; ++fmt) {
499                     value <= 3;
500                     value += *fmt - '0';
501                 }
502                 --fmt;
503                 if (percent && value == '%') {
504                     *store++ = '%';
505                     *store = '%';
506                 } else
507                     *store = (char)value;
508                 break;
509             default:
510                 *store = *fmt;
511                 break;
512             case '%':
513                 }
514             }

```

```
515     *store = '\0';
516     *len = PTRDIFF(store, save);
517     return (0);
518 }
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
*****
1 '\\" te
2 .\" Copyright 2014 Garrett D'Amore <garrett@damore.org>
3 .\" Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved
4 .\" Copyright 1992, X/Open Company Limited All Rights Reserved
5 .\" Portions Copyright (c) 1982-2007 AT&T Knowledge Ventures
6 .\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for
7 .\" permission to reproduce portions of its copyrighted documentation.
8 .\" Original documentation from The Open Group can be obtained online at
9 .\" http://www.opengroup.org/bookstore/.
10 .\" The Institute of Electrical and Electronics Engineers and The Open Group,
11 .\" have given us permission to reprint portions of their documentation. In the
12 .\" following statement, the phrase "this text" refers to portions of the
13 .\" system documentation. Portions of this text are reprinted and reproduced in
14 .\" electronic form in the Sun OS Reference Manual, from IEEE Std 1003.1, 2004
15 .\" Edition, Standard for Information Technology -- Portable Operating System
16 .\" Interface (POSIX), The Open Group Base Specifications Issue 6, Copyright
17 .\" (C) 2001-2004 by the Institute of Electrical and Electronics Engineers,
18 .\" Inc and The Open Group. In the event of any discrepancy between these
19 .\" versions and the original IEEE and The Open Group Standard, the original
20 .\" IEEE and The Open Group Standard is the referee document. The original
21 .\" Standard can be obtained online at
22 .\" http://www.opengroup.org/unix/online.html.
23 .\" Sun Microsystems, Inc. gratefully acknowledges The Open Group for permission
24 .\" The Institute of Electrical and Electronics Engineers and The Open Group, ha
25 .\" reprinted and reproduced in electronic form in the Sun OS Reference Manu
26 .\" and Electronics Engineers, Inc and The Open Group. In the event of any discr
27 .\" This notice shall appear on any product containing this material.
28 .\" The contents of this file are subject to the terms of the Common
29 .\" Development and Distribution License (the "License"). You may not use this
30 .\" file except in compliance with the License.
31 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or
32 .\" http://www.opensolaris.org/os/licensing. See the License for the specific
33 .\" language governing permissions and limitations under the License.
34 .\" When distributing Covered Code, include this CDDL HEADER in each file and
35 .\" include the License file at usr/src/OPENSOLARIS.LICENSE. If applicable,
36 .\" add the following below this CDDL HEADER, with the fields enclosed by
37 .\" brackets "[]" replaced with your own identifying information:
38 .\" Portions Copyright [yyyy] [name of copyright owner]
39 .TH PRINTF 1 "May 11, 2014"
40 .\" The contents of this file are subject to the terms of the Common Development
41 .\" See the License for the specific language governing permissions and limitat
42 .\" the fields enclosed by brackets "[]" replaced with your own identifying info
43 .TH PRINTF 1 "Aug 11, 2009"
44 .SH NAME
45 printf \- write formatted output
46 .SH SYNOPSIS
47 .SS "/usr/bin/printf"
48 .LP
49 .nf
50 \fBprintf\fR \fIformat\fR [\fIargument\fR]...
51 .fi
52 .SS "ksh93"
53 .LP
54 .nf
55 \fBprintf\fR \fIformat\fR [\fIstring\fR...]
```

```
56 .SH DESCRIPTION
57 .SS "/usr/bin/printf"
58 .sp
59 .LP
60 The following operands are supported by \fB/usr/bin/printf\fR:
61 .sp
62 .ne 2
63 .na
64 \fB\fIformat\fR\fR
65 .ad
66 .RS 12n
67 A string describing the format to use to write the remaining operands. The
68 \fIformat\fR operand is used as the \fIformat\fR string described on the
69 \fBformats\fR(5) manual page, with the following exceptions:
70 .RS +4
71 .TP
72 .ie t \(\bu
73 .el o
74 .RS +4
75 A \fBSPACE\fR character in the format string, in any context other than a flag
76 of a conversion specification, is treated as an ordinary character that is
77 copied to the output.
78 .RE
79 .RS +4
80 .TP
81 .ie t \(\bu
82 .el o
83 A character in the format string is treated as a character, not as a
84 \fBSPACE\fR character.
85 .RE
86 .RS +4
87 .TP
88 .ie t \(\bu
89 .el o
90 In addition to the escape sequences described on the \fBformats\fR(5) manual
91 page (\fBe\ea\fR, \fB\ea\fR, \fB\eb\fR, \fB\ef\fR, \fB\en\fR, \fB\er\fR,
92 \fB\et\fR, \fB\ev\fR), \fB\ea\fR\fIddd\fR, where \fIddd\fR is a one-, two- or
93 three-digit octal number, is written as a byte with the numeric value specified
94 by the octal number.
95 .RE
96 .RS +4
97 .TP
98 .ie t \(\bu
99 .el o
100 The program does not precede or follow output from the \fBd\fR or \fBu\fR
101 conversion specifications with blank characters not specified by the
102 \fIformat\fR operand.
103 .RE
104 .RS +4
105 .TP
106 .ie t \(\bu
107 .el o
108 The program does not precede output from the \fBo\fR conversion specification
109 with zeros not specified by the \fIformat\fR operand.
110 .RE
111 .RS +4
112 .TP
113 .ie t \(\bu
114 .el o
115 The argument used for the conversion character (or width or precision
116 parameters, see below) may be taken from the \fIn\fRnht argument instead
```

```

117 of the next unused argument, by specifying \fIn\fR\fB$\fR immediately following
118 the \fB%\fR character, or the \fB*\fR character (for width or precision
119 arguments).
120 If \fIn\fR\fB$\fR appears in any conversions in the format string,
121 then it must be used for all conversions, including any variable width or
122 precision specifiers.
123 .RE
124 .RS +4
125 .TP
126 .ie t \(\bu
127 .el o
128 The special character \fB*\fR may be used instead of a string of decimal digits
129 to indicate a minimum field width or a precision. In this case the next
130 available argument is used (or the \fIn\fRth if the form \fIn\fR\fB$\fR is
131 used), treating its value as a decimal string.
132 .RE
133 .RS +4
134 .TP
135 .ie t \(\bu
136 .el o
137 An additional conversion character, \fBb\fR, is supported as follows. The
138 argument is taken to be a string that can contain backslash-escape sequences.
139 The following backslash-escape sequences are supported:
140 .RS +4
141 .TP
142 .ie t \(\bu
143 .el o
144 The escape sequences listed on the \fBformats\fR(5) manual page (\fB\ea\fR,
145 \fB\ea\fR, \fB\eb\fR, \fB\ef\fR, \fB\en\fR, \fB\er\fR, \fB\et\fR, \fB\ev\fR),
146 which are converted to the characters they represent
147 .RE
148 .RS +4
149 .TP
150 .ie t \(\bu
151 .el o
152 \fB\eo\fR\fId\ddd\fR, where \fId\ddd\fR is a zero-, one-, two- or three-digit octal
153 number that is converted to a byte with the numeric value specified by the
154 octal number
155 .RE
156 .RS +4
157 .TP
158 .ie t \(\bu
159 .el o
160 \fB\ec\fR, which is written and causes \fBprintf\fR to ignore any remaining
161 characters in the string operand containing it, any remaining string operands
162 and any additional characters in the \fIformat\fR operand.
163 .RE
164 .RE
165 The interpretation of a backslash followed by any other sequence of characters
166 is unspecified.
167 .sp
168 Bytes from the converted string are written until the end of the string or the
169 number of bytes indicated by the precision specification is reached. If the
170 precision is omitted, it is taken to be infinite, so all bytes up to the end of
171 the converted string are written. For each specification that consumes an
172 argument, the next argument operand is evaluated and converted to the
173 appropriate type for the conversion as specified below. The \fIformat\fR
174 operand is reused as often as necessary to satisfy the argument operands. Any
175 extra \fBc\fR or \fBs\fR conversion specifications are evaluated as if a null
176 string argument were supplied; other extra conversion specifications are
177 evaluated as if a zero argument were supplied.
178 .sp
179 When there are more argument operands than format specifiers, and the
180 format includes \fIn\fR\fB$\fR position indicators, then the format is
181 reprocessed from the beginning as above, but with the argument list starting
182 from the next argument after the highest \fIn\fRth argument previously

```

```

183 encountered.
184 .sp
185 If the \fIformat\fR operand
186 evaluated as if a zero argument were supplied. If the \fIformat\fR operand
187 contains no conversion specifications and \fIargument\fR operands are present,
188 the results are unspecified. If a character sequence in the \fIformat\fR
189 operand begins with a \fB%\fR character, but does not form a valid conversion
190 specification, the behavior is unspecified.
191 .RE
192 .sp
193 .ne 2
194 .na
195 \fB\fIargument\fR\fR
196 .ad
197 .RS 12n
198 The strings to be written to standard output, under the control of
199 \fBformat\fR. The \fIargument\fR operands are treated as strings if the
200 corresponding conversion character is \fBb\fR, \fBc\fR or \fBs\fR. Otherwise,
201 it is evaluated as a C constant, as described by the ISO C standard, with the
202 following extensions:
203 .RS +4
204 .TP
205 .ie t \(\bu
206 .el o
207 A leading plus or minus sign is allowed.
208 .RE
209 .RS +4
210 .TP
211 .ie t \(\bu
212 .el o
213 If the leading character is a single- or double-quote, the value is the numeric
214 value in the underlying codeset of the character following the single- or
215 double-quote.
216 .RE
217 If an argument operand cannot be completely converted into an internal value
218 appropriate to the corresponding conversion specification, a diagnostic message
219 is written to standard error and the utility does not exit with a zero exit
220 status, but continues processing any remaining operands and writes the value
221 accumulated at the time the error was detected to standard output.
222 .RE
223 .SS "ksh93"
224 .sp
225 .LP
226 The \fIformat\fR operands support the full range of ANSI C/C99/XPG6 formatting
227 specifiers as well as additional specifiers:
228 .sp
229 .ne 2
230 .na
231 \fB\fB\ea\fR\fR
232 .ad
233 .RS 6n
234 Each character in the string operand is processed specially, as follows:
235 .sp
236 .ne 2
237 .na
238 \fB\fB\ea\fR\fR
239 .ad
240 .RS 8n
241 Alert character.
242 .RE
243 .sp
244 .ne 2
245 .na

```

```

248 \fB\fB\eb\fR\fR
249 .ad
250 .RS 8n
251 Backspace character.
252 .RE

254 .sp
255 .ne 2
256 .na
257 \fB\fB\ec\fR\fR
258 .ad
259 .RS 8n
260 Terminate output without appending NEWLINE. The remaining string operands are
261 ignored.
262 .RE

264 .sp
265 .ne 2
266 .na
267 \fB\fB\ee\fR\fR
268 .ad
269 .RS 8n
270 Escape character (\fBASCII\fR octal \fB033\fR).
271 .RE

273 .sp
274 .ne 2
275 .na
276 \fB\fB\ef\fR\fR
277 .ad
278 .RS 8n
279 FORM FEED character.
280 .RE

282 .sp
283 .ne 2
284 .na
285 \fB\fB\en\fR\fR
286 .ad
287 .RS 8n
288 NEWLINE character.
289 .RE

291 .sp
292 .ne 2
293 .na
294 \fB\fB\et\fR\fR
295 .ad
296 .RS 8n
297 TAB character.
298 .RE

300 .sp
301 .ne 2
302 .na
303 \fB\fB\ev\fR\fR
304 .ad
305 .RS 8n
306 Vertical tab character.
307 .RE

309 .sp
310 .ne 2
311 .na
312 \fB\fB\ee\fR\fR
313 .ad

```

```

314 .RS 8n
315 Backslash character.
316 .RE

318 .sp
319 .ne 2
320 .na
321 \fB\fB\ee0\fR\fIx\fR\fR
322 .ad
323 .RS 8n
324 The 8-bit character whose \fBASCII\fR code is the \fB1\fR-, \fB2\fR-, or
325 \fB3\fR-digit octal number \fIx\fR.
326 .RE

328 .RE

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

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

350 .sp
351 .ne 2
352 .na
353 \fB\fB%P\fR\fR
354 .ad
355 .RS 6n
356 Treat \fIstring\fR as an extended regular expression and convert it to a shell
357 pattern.
358 .RE

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

371 .sp
372 .ne 2
373 .na
374 \fB\fB%R\fR\fR
375 .ad
376 .RS 6n
377 Treat \fIstring\fR as an shell pattern expression and convert it to an extended
378 regular expression.
379 .RE

```

```

381 .sp
382 .ne 2
383 .na
384 \fB\fB%T\fR\fR
385 .ad
386 .RS 6n
387 Treat \fIstring\fR as a date/time string and format it. The \fBT\fR can be
388 preceded by (\fIdformat\fR), where \fIdformat\fR is a date format as defined by
389 the \fBdate\fR(1) command.
390 .RE

392 .sp
393 .ne 2
394 .na
395 \fB\fB%Z\fR\fR
396 .ad
397 .RS 6n
398 Output a byte whose value is \fB0\fR.
399 .RE

401 .sp
402 .LP
403 When performing conversions of \fIstring\fR to satisfy a numeric format
404 specifier, if the first character of \fIstring\fR is \fB"or'\fR, the value is
405 the numeric value in the underlying code set of the character following the
406 \fB"or'\fR. Otherwise, \fIstring\fR is treated like a shell arithmetic
407 expression and evaluated.
408 .sp
409 .LP
410 If a \fIstring\fR operand cannot be completely converted into a value
411 appropriate for that format specifier, an error occurs, but remaining
412 \fIstring\fR operands continue to be processed.
413 .sp
414 .LP
415 In addition to the format specifier extensions, the following extensions of
416 ANSI C/C99/XPG6 are permitted in format specifiers:
417 .RS +4
418 .TP
419 .ie t \(\bu
420 .el o
421 The escape sequences \fB\ee\fR and \fB\ee\fR expand to the escape character
422 which is octal 033 in ASCII.
423 .RE
424 .RS +4
425 .TP
426 .ie t \(\bu
427 .el o
428 The escape sequence \fB\ecx\fR expands to CTRL-x.
429 .RE
430 .RS +4
431 .TP
432 .ie t \(\bu
433 .el o
434 The escape sequence \fB\ec[.\fR\fIname\fR\fB\&.\fR\fR expands to the collating
435 element \fIname\fR.
436 .RE
437 .RS +4
438 .TP
439 .ie t \(\bu
440 .el o
441 The escape sequence \fB\ex{hex}\fR expands to the character corresponding to
442 the hexadecimal value \fBhex\fR.
439 The escape sequence \fB\ex{hex}\fR expands to the character corresponding to the
440 hexadecimal value \fBhex\fR.
443 .RE

```

```

444 .RS +4
445 .TP
446 .ie t \(\bu
447 .el o
448 The format modifier flag = can be used to center a field to a specified width.
449 When the output is a terminal, the character width is used rather than the
450 number of bytes.
451 .RE
452 .RS +4
453 .TP
454 .ie t \(\bu
455 .el o
456 Each of the integral format specifiers can have a third modifier after width
457 and precision that specifies the base of the conversion from 2 to 64. In this
458 case, the \fB#\fR modifier causes \fIbase\fR\fB#\fR to be prepended to the
459 value.
460 .RE
461 .RS +4
462 .TP
463 .ie t \(\bu
464 .el o
465 The \fB#\fR modifier can be used with the \fBd\fR specifier when no base is
466 specified to cause the output to be written in units of 1000 with a suffix of
467 one of \fBk M G T P E\fR.
468 .RE
469 .RS +4
470 .TP
471 .ie t \(\bu
472 .el o
473 The \fB#\fR modifier can be used with the \fBi\fR specifier to cause the output
474 to be written in units of \fBl024\fR with a suffix of one of \fBK Mi Gi Ti Pi
475 \fE\fR.
476 .RE
477 .sp
478 .LP
479 If there are more \fIstring\fR operands than format specifiers, the format
480 string is reprocessed from the beginning. If there are fewer \fIstring\fR
481 operands than format specifiers, then \fIstring\fR specifiers are treated as if
482 empty strings were supplied, numeric conversions are treated as if \fB0\fR was
483 supplied, and time conversions are treated as if \fBnow\fR was supplied.
484 .sp
485 .LP
486 When there are more argument operands than format specifiers, and the
487 format includes \fIn\fR\fB$\fR position indicators, then the format is
488 reprocessed from the beginning as above, but with the argument list starting
489 from the next argument after the highest \fIn\fRth argument previously
490 encountered.
491 .sp
492 .LP
493 \fB/usr/bin/printf\fR is equivalent to \fBksh93\fR's \fBprintf\fR built-in and
494 \fBprintf -f\fR, which allows additional options to be specified.
495 .SH USAGE
496 .SS "/usr/bin/printf"
497 .sp
498 .LP
499 The \fBprintf\fR utility, like the \fBprintf\fR(3C) function on which it is
500 based, makes no special provision for dealing with multi-byte characters when
501 using the \fB%c\fR conversion specification. Applications should be extremely
502 cautious using either of these features when there are multi-byte characters in
503 the character set.
504 .sp
505 .LP
447 Field widths and precisions cannot be specified as \fB*\fR.
448 .sp
449 .LP
506 The \fB%b\fR conversion specification is not part of the ISO C standard; it has

```

```

507 been added here as a portable way to process backslash escapes expanded in
508 string operands as provided by the \fBecho\fR utility. See also the USAGE
509 section of the \fBecho\fR(1) manual page for ways to use \fBprintf\fR as a
510 replacement for all of the traditional versions of the \fBecho\fR utility.
511 .sp
512 .LP
513 If an argument cannot be parsed correctly for the corresponding conversion
514 specification, the \fBprintf\fR utility reports an error. Thus, overflow and
515 extraneous characters at the end of an argument being used for a numeric
516 conversion are to be reported as errors.
517 .sp
518 .LP
519 It is not considered an error if an argument operand is not completely used for
520 a \fbC\fR or \fbS\fR conversion or if a string operand's first or second
521 character is used to get the numeric value of a character.
522 .SH EXAMPLES
523 .SS "/usr/bin/printf"
524 .LP
525 \fBExample 1\fR Printing a Series of Prompts
526 .sp
527 .LP
528 The following example alerts the user, then prints and reads a series of
529 prompts:
530
531 .sp
532 .in +2
533 .nf
534 example% \fBprintf "\ePlease fill in the following: \enName: "
535 read name
536 printf "Phone number: "
537 read phone\fR
538 .fi
539 .in -2
540 .sp
541 .LP
542 \fBExample 2\fR Printing a Table of Calculations
543 .sp
544 .LP
545 The following example prints a table of calculations. It reads out a list of
546 right and wrong answers from a file, calculates the percentage correctly, and
547 prints them out. The numbers are right-justified and separated by a single tab
548 character. The percentage is written to one decimal place of accuracy:
549
550 .sp
551 .in +2
552 .nf
553 example% \fBwhile read right wrong ; do
554     percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)
555     printf "%2d right\et%2d wrong\et(%s)\n" \e
556     $right $wrong $percent
557 done < database_file\fR
558 .fi
559 .in -2
560 .sp
561 .LP
562 \fBExample 3\fR Printing number strings
563 .sp
564 .LP
565 The command:
566 .sp
567 example% \fBprintf "%5d%4d\n" 1 21 321 4321 54321\fR

```

```

573 .fi
574 .in -2
575 .sp
576 .LP
577 produces:
578 .sp
579 .LP
580
581 .sp
582 .in +2
583 .nf
584     1 21
585     3214321
586 54321 0
587 .fi
588 .in -2
589 .sp
590 .LP
591 The \fIformat\fR operand is used three times to print all of the given strings
592 and that a \fb0\fR was supplied by \fBprintf\fR to satisfy the last \fb%4d\fR
593 conversion specification.
594
595 .LP
596 \fBExample 4\fR \fRTabulating Conversion Errors
597 .sp
598 .LP
599 The following example tabulates conversion errors.
600 .LP
601
602 .sp
603 .LP
604 The \fBprintf\fR utility tells the user when conversion errors are detected
605 while producing numeric output. These results would be expected on an
606 implementation with 32-bit two's-complement integers when \fb%d\fR is specified
607 as the \fIformat\fR operand:
608
609 .sp
610 .LP
611 .TS
612 .sp
613 .TS
614 box;
615 c c c
616 l l l .
617 Arguments Standard Diagnostic
618 5a 5 printf: 5a not completely converted
619 9999999999 2147483647 printf: 9999999999: Results too large
620 -9999999999 -2147483648 printf: -9999999999: Results too large
621 ABC 0 printf: ABC expected numeric value
622 .TE
623 .sp
624 .LP
625 The value shown on standard output is what would be expected as the return
626 value from the function \fbStrtol\fR(3C). A similar correspondence exists
627 between \fb%u\fR and \fbstrtoul\fR(3C), and \fb%e\fR, \fb%f\fR and \fb%g\fR and
628 \fbstrtod\fR(3C).
629
630 .LP
631 \fBExample 5\fR \fRPrinting Output for a Specific Locale
632 .sp
633 .LP
634 The following example prints output for a specific locale. In a locale using
635 the ISO/IEC 646:1991 standard as the underlying codeset, the command:
636
637 .sp

```

```

639 .in +2
640 .nf
641 example% \fBprintf "%d\en" 3 +3 -3 \e'3 \e"+3 "'-3"\fR
642 .fi
643 .in -2
644 .sp
645 .sp
646 .LP
647 produces:
648
649 .sp
650 .TS
651 box;
652 l 1
653 l 1 .
654 \fB3\fR Numeric value of constant 3
655 \fB3\fR Numeric value of constant 3
656 \fB\mi3\fR Numeric value of constant \(\mi3
657 \fB51\fR T{
658 Numeric value of the character '3' in the ISO/IEC 646:1991 standard codeset
659 T}
660 \fB43\fR T{
661 Numeric value of the character '+' in the ISO/IEC 646:1991 standard codeset
662 T}
663 \fB45\fR T{
664 Numeric value of the character '\mi' in the SO/IEC 646:1991 standard codeset
665 T}
666 \fB45\fR T{
667 Numeric value of the character '\mi' in the SO/IEC 646:1991 standard codeset
668 T}
669 .TE
670 .sp
671 .LP
672 In a locale with multi-byte characters, the value of a character is intended to
673 be the value of the equivalent of the \fbwchar_t\fR representation of the
674 character.
675
676 .sp
677 .LP
678 If an argument operand cannot be completely converted into an internal value
679 appropriate to the corresponding conversion specification, a diagnostic message
680 is written to standard error and the utility does exit with a zero exit status,
681 but continues processing any remaining operands and writes the value
682 accumulated at the time the error was detected to standard output.
683
684 .LP
685 \fBExample 6 \fRALternative floating point representation 1
686 .sp
687 .LP
688 The \fBprintf\fR utility supports an alternative floating point representation
689 (see \fBprintf\fR(3C) entry for the "\fb%a\fR"/"\fb%A\fR"), which allows the
690 output of floating-point values in a format that avoids the usual base16 to
691 base10 rounding errors.
692
693 .sp
694 .in +2
695 .nf
696 example% printf "%a\en" 2 3.1 NaN
697 .fi
698 .in -2
699 .sp
700 .sp
701 .LP
702 produces:
703 .LP
704

```

```

705 .sp
706 .in +2
707 .nf
708 0x1.00000000000000000000000000000000p+01
709 0x1.8ccccccccccccccccccccccccccccdp+01
710 1nan
711 .fi
712 .in -2
713 .sp
714 .LP
715 \fBExample 7 \fRALternative floating point representation 2
716 .sp
717 The following example shows two different representations of the same
718 floating-point value.
719 .LP
720
721
722 .sp
723 .in +2
724 .nf
725 example% x=2 ; printf "%f == %a\en" x x
726 .fi
727 .in -2
728 .sp
729 .sp
730 .LP
731 produces:
732
733 .sp
734 .in +2
735 .nf
736 2.000000 == 0x1.00000000000000000000000000000000p+01
737 .fi
738 .in -2
739 .sp
740 .LP
741 .sp
742 .LP
743 \fBExample 8 \fROutput of unicode values
744 .sp
745 .LP
746 The following command will print the EURO unicode symbol (code-point 0x20ac).
747
748 .sp
749 .in +2
750 .nf
751 example% LC_ALL=en_US.UTF-8 printf "\u[20ac]\en"
752 .fi
753 .in -2
754 .sp
755 .sp
756 .LP
757 produces:
758 .sp
759 .LP
760 .sp
761 .in +2
762 .nf
763 <euro>
764 .fi
765 .in -2
766 .sp
767 .sp
768 .LP
769 produces:
770 .LP
771

```

```

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

773 .LP
774 \fBExample 9\fR Convert unicode character to unicode code-point value
775 .sp
776 .LP
777 The following command will print the hexadecimal value of a given character.

779 .sp
780 .in +2
781 .nf
782 example% export LC_ALL=en_US.UTF-8
783 example% printf "%x\n" "'<euro>'"
784 .fi
785 .in -2
786 .sp

788 .sp
789 .LP
790 where "<euro>" represents the EURO currency symbol character (code-point
791 0x20ac).

793 .sp
794 .LP
795 produces:

797 .sp
798 .in +2
799 .nf
800 20ac
801 .fi
802 .in -2
803 .sp

805 .LP
806 \fBExample 10\fR Print the numeric value of an ASCII character
807 .sp
808 .in +2
809 .nf
810 example% printf "%d\n" "'A'"
811 .fi
812 .in -2
813 .sp

815 .sp
816 .LP
817 produces:

819 .sp
820 .in +2
821 .nf
822 65
823 .fi
824 .in -2
825 .sp

827 .LP
828 \fBExample 11\fR Print the language-independent date and time format
829 .sp
830 .LP
831 To print the language-independent date and time format, the following statement
832 could be used:

834 .sp
835 .in +2
836 .nf

```

```

837 example% printf "format" weekday month day hour min
838 .fi
839 .in -2
840 .sp

842 .sp
843 .LP
844 For example,

846 .sp
847 .in +2
848 .nf
849 $ printf format "Sunday" "July" 3 10 2
850 .fi
851 .in -2
852 .sp

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

858 .sp
859 .in +2
860 .nf
861 "%s, %s %d, %d:%.2d\n"
862 .fi
863 .in -2
864 .sp

866 .sp
867 .LP
868 producing the message:

870 .sp
871 .in +2
872 .nf
873 Sunday, July 3, 10:02
874 .fi
875 .in -2
876 .sp

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

882 .sp
883 .in +2
884 .nf
885 "%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
886 .fi
887 .in -2
888 .sp

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

894 .sp
895 .in +2
896 .nf
897 "%1\$s, %3\$d. %2\$s, %4\$d:%5\$2d\n" in this case
898 .fi
899 .in -2
900 .sp

```

```

902 .sp
903 .LP
904 producing the message:

906 .sp
907 .in +2
908 .nf
909 Sunday, 3. July, 10:02
910 .fi
911 .in -2
912 .sp

914 .SH ENVIRONMENT VARIABLES
915 .sp
916 .LP
917 See \fBenviron\fR(5) for descriptions of the following environment variables
918 that affect the execution of \fBprintf\fR: \fBLANG\fR, \fBLC_ALL\fR,
919 \fBLC_CTYPE\fR, \fBLC_MESSAGES\fR, \fBLC_NUMERIC\fR, and \fBNLSPATH\fR.
920 .SH EXIT STATUS
921 .sp
922 .LP
923 The following exit values are returned:
924 .sp
925 .ne 2
926 .na
927 \fB\fB0\fR\fR
928 .ad
929 .RS 6n
930 Successful completion.
931 .RE

933 .sp
934 .ne 2
935 .na
936 \fB\fB>0\fR\fR
937 .ad
938 .RS 6n
939 An error occurred.
940 .RE

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

949 .sp
950 .TS
951 box;
952 c | c
953 l | l .
954 ATTRIBUTE TYPE ATTRIBUTE VALUE
955 -
956 CSI      Enabled
957 -
958 Interface Stability     Committed
959 -
960 Standard      See \fBstandards\fR(5).
961 .TE

963 .SS "ksh93"
964 .sp
966 .sp
967 .TS

```

```

968 box;
969 c | c
970 l | l .
971 ATTRIBUTE TYPE ATTRIBUTE VALUE
972 -
973 Interface Stability     Uncommitted
974 .TE

976 .SH SEE ALSO
977 .sp
978 .LP
979 \fBawk\fR(1), \fBbc\fR(1), \fBdate\fR(1), \fBecho\fR(1), \fBksh93\fR(1),
980 \fBprintf\fR(3C), \fBstrtod\fR(3C), \fBstrtol\fR(3C), \fBstrtoul\fR(3C),
981 \fBattributes\fR(5), \fBenviron\fR(5), \fBformats\fR(5), \fBstandards\fR(5)
982 .SH NOTES
983 .sp
984 .LP
985 Using format specifiers (characters following '%') which are not listed in the
986 \fBprintf\fR(3C) or this manual page will result in undefined behavior.
987 .sp
988 .LP
989 Using escape sequences (the character following a backslash ('\e')) which are
990 not listed in the \fBprintf\fR(3C) or this manual page will result in undefined
991 behavior.
992 .sp
993 .LP
994 Floating-point values follow C99, XPG6 and IEEE 754 standard behavior and can
995 handle values the same way as the platform's |\fBlong double\fR| datatype.
996 .sp
997 .LP
998 Floating-point values handle the sign separately which allows signs for values
999 like NaN (for example, -nan), Infinite (for example, -inf) and zero (for
1000 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
*****
```

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License (" CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #

12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 # Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
15 #

17 set name=pkg.fmri value=(pkg:/system/test/utiltest@$PKGVERS)
18 set name=pkg.description value="Miscellaneous Utility Unit Tests"
19 set name=pkg.summary value="Utility Unit Test Suite"
20 set name=info.classification \
21     value=org.opensolaris.category.2008:Development/System
22 set name=variant.arch value=$(ARCH)
23 dir path=opt/util-tests
24 dir path=opt/util-tests/bin
25 dir path=opt/util-tests/runfiles
26 dir path=opt/util-tests/tests
27 file path=opt/util-tests/README mode=0444
28 file path=opt/util-tests/bin/utiltest mode=0555
29 file path=opt/util-tests/runfiles/default.run mode=0444
30 file path=opt/util-tests/tests/printf_test mode=0555
31 license lic_CDDL license=lic_CDDL
32 depend fmri=system/test/testrunner type=require
```

```
new/usr/src/test/Makefile
```

```
1
```

```
*****
```

```
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
```

```
*****
```

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License (" CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
```

```
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
```

```
10 #
```

```
12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
15 #
```

```
17 .PARALLEL: $(SUBDIRS)
```

```
19 SUBDIRS = os-tests test-runner util-tests zfs-tests
18 SUBDIRS = os-tests test-runner zfs-tests
```

```
21 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
```

```
*****
```

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License (" CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #

12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 #

16 .PARALLEL: ${SUBDIRS}

18 SUBDIRS = cmd runfiles tests doc

20 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
```

```
*****
```

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License (" CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #

12 #
13 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
14 # Copyright (c) 2012 by Delphix. All rights reserved.
15 #

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

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

23 PROGS = utiltest

25 CMDS = $(PROGS:%= $(ROOTBIN)/%)
26 $(CMDS) := FILEMODE = 0555

28 all lint clean clobber:

30 install: $(CMDS)

32 $(CMDS): $(ROOTBIN)

34 $(ROOTBIN):
35     $(INS.dir)

37 $(ROOTBIN)/%: %.ksh
38     $(INS.rename)
```

```

new/usr/src/test/util-tests/cmd/utiltest.ksh
*****
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
*****
1 #!/usr/bin/ksh
3 #
4 # This file and its contents are supplied under the terms of the
5 # Common Development and Distribution License (" CDDL"), version 1.0.
6 # You may only use this file in accordance with the terms of version
7 # 1.0 of the CDDL.
8 #
9 # A full copy of the text of the CDDL should have accompanied this
10 # source. A copy of the CDDL is also available via the Internet at
11 # http://www.illumos.org/license/CDDL.
12 #
14 #
15 # Copyright (c) 2012 by Delphix. All rights reserved.
16 # Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
17 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
18 #
20 export MY_TESTS="/opt/util-tests"
21 runner="/opt/test-runner/bin/run"
23 function fail
24 {
25     echo $1
26     exit ${2:-1}
27 }
29 function find_runfile
30 {
31     typeset distro=
32     if [[ -d /opt/delphix && -h /etc/delphix/version ]]; then
33         distro=delphix
34     elif [[ 0 -ne $(grep -c OpenIndiana /etc/release 2>/dev/null) ]]; then
35         distro=openindiana
36     elif [[ 0 -ne $(grep -c OmniOS /etc/release 2>/dev/null) ]]; then
37         distro=omnios
38     elif [[ -f $MY_TESTS/runfiles/default.run ]]; then
39         # optional catch-all
40         distro=default
41     fi
43     [[ -n $distro ]] && echo $MY_TESTS/runfiles/$distro.run
44 }
46 while getopts c: c; do
47     case $c in
48     'c')
49         runfile=$OPTARG
50         [[ -f $runfile ]] || fail "Cannot read file: $runfile"
51         ;;
52     esac
53 done
54 shift $((OPTIND - 1))
56 [[ -z $runfile ]] && runfile=$(find_runfile)
57 [[ -z $runfile ]] && fail "Couldn't determine distro"

```

```

1
2
new/usr/src/test/util-tests/cmd/utiltest.ksh
59 $runner -c $runfile
61 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
```

```
*****
```

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License (" CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #

12 #
13 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
14 # Copyright (c) 2012 by Delphix. All rights reserved.
15 #

17 include $(SRC)/Makefile.master

19 READMES = README

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

23 FILES = $(READMES:%= $(ROOTOPTPKG)/%)
24 $(FILES) := FILEMODE = 0444

26 all: $(READMES)

28 install: $(ROOTOPTPKG) $(FILES)

30 clean lint clobber:

32 $(ROOTOPTPKG):
33     $(INS.dir)

35 $(ROOTOPTPKG)/%: %
36     $(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
*****
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License (" CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #

12 #
13 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
14 # Copyright (c) 2012 by Delphix. All rights reserved.
15 #
```

17 Utils Unit Test Suite README

- 19 1. What the Utils Unit Test Suite tests
- 20 2. Building and installing the Utils Unit Test Suite
- 21 3. Running the Utils Unit Test Suite
- 22 4. Test results

24 -----

26 1. What the Utils Unit Test Suite tests

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

31 2. Building and installing the Utils Unit Test Suite

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

```
37     build_machine$ bldenv [-d] <your_env_file>
38     build_machine$ cd $SRC/test
39     build_machine$ dmake install
40     build_machine$ cd $SRC/pkg
41     build_machine$ dmake install
```

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

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

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

51 3. Running the Utils Unit Test Suite

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

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

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

60 4. Test results

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

```
new/usr/src/test/util-tests/runfiles/Makefile
```

```
1
```

```
*****
```

```
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
```

```
*****
```

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License (" CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #

12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 # Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
15 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
16 #

18 include $(SRC)/Makefile.master

20 SRCS = default.run

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

25 CMDS = $(SRCS:=%$(RUNFILES)/%)
26 $(CMDS) := FILEMODE = 0444

28 all: $(SRCS)

30 install: $(CMDS)

32 clean lint clobber:

34 $(CMDS): $(RUNFILES) $(SRCS)

36 $(RUNFILES):
37     $(INS.dir)

39 $(RUNFILES)/%: %
40     $(INS.file)
```

```
new/usr/src/test/util-tests/runfiles/default.run
```

```
1
```

```
*****
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
*****
```

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License (" CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #

12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
15 #

17 [DEFAULT]
18 pre =
19 verbose = False
20 quiet = False
21 timeout = 60
22 post =
23 outputdir = /var/tmp/test_results
25 [/opt/util-tests/tests/printf_test]
```

```
new/usr/src/test/util-tests/tests/Makefile
```

```
1
```

```
*****
```

```
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
```

```
*****
```

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License (" CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #

12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
15 #

17 SUBDIRS = printf
19 include $(SRC)/test/Makefile.com
```

```
new/usr/src/test/util-tests/tests/printf/Makefile
```

```
1
```

```
*****
```

```
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
```

```
*****
```

```
1 #
2 # This file and its contents are supplied under the terms of the
3 # Common Development and Distribution License (" CDDL"), version 1.0.
4 # You may only use this file in accordance with the terms of version
5 # 1.0 of the CDDL.
6 #
7 # A full copy of the text of the CDDL should have accompanied this
8 # source. A copy of the CDDL is also available via the Internet at
9 # http://www.illumos.org/license/CDDL.
10 #

12 #
13 # Copyright (c) 2012 by Delphix. All rights reserved.
14 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
15 #

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

20 PROG = printf_test

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

25 CMDS = $(PROG:=%$(TESTDIR)/%)
26 $(CMDS) := FILEMODE = 0555

28 all lint clean clobber:

30 install: all $(CMDS)

32 $(CMDS): $(TESTDIR) $(PROG).ksh

34 $(TESTDIR):
35     $(INS.dir)

37 $(TESTDIR)/%: %.ksh
38     $(INS.rename)
```

```

new/usr/src/test/util-tests/tests/printf/printf_test.ksh
*****
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
*****
1 #! /usr/bin/ksh
2 #
3 #
4 # This file and its contents are supplied under the terms of the
5 # Common Development and Distribution License (" CDDL"), version 1.0.
6 # You may only use this file in accordance with the terms of version
7 # 1.0 of the CDDL.
8 #
9 # A full copy of the text of the CDDL should have accompanied this
10 # source. A copy of the CDDL is also available via the Internet at
11 # http://www.illumos.org/license/CDDL.
12 #

14 #
15 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
16 #

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

20 test_start() {
21     print "TEST STARTING ${1}: ${2}"
22 }

24 test_pass() {
25     print "TEST PASS: ${1}"
26 }

28 test_fail() {
29     print "TEST FAIL: ${1}: ${2}"
30     exit -1
31 }

33 checkrv() {
34     if [[ $? -ne 0 ]]; then
35         test_fail $1 "exit failure"
36     fi
37 }

39 compare() {
40     if [[ "$2" != "$3" ]]; then
41         test_fail $1 "compare mismatch, got [$2] expected [$3]"
42     fi
43 }

45 typeset -A tests=()

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

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

```

1

```

new/usr/src/test/util-tests/tests/printf/printf_test.ksh

```

```

60 typeset -A tests[03]=()
61 tests[03][desc]="multiple arguments"
62 tests[03][format]='$d %s '
63 tests[03][args]="1 one 2 two 3 three"
64 tests[03][result]='1 one 2 two 3 three '

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

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

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

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

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

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

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

108 typeset -A tests[11]=()
109 tests[11][desc]="negative dec (64-bit)"
110 tests[11][format]="%x"
111 tests[11][args]="-1"
112 tests[11][result]='fffffffffffff'

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

120 typeset -A tests[12]=()
121 tests[12][desc]="float precision"
122 tests[12][format]=".2f"
123 tests[12][args]="3.14159"
124 tests[12][result]='3.14'

```

2

```

126 typeset -A tests[13]=()
127 tests[13][desc]="left justify"
128 tests[13][format]='%-5d'
129 tests[13][args]="'45"
130 tests[13][result]="'45      '

132 typeset -A tests[14]=()
133 tests[14][desc]="newlines"
134 tests[14][format]='%s\n%s\n%s'
135 tests[14][args]="one two three"
136 tests[14][result]="'one
137 two
138 three'

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

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

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

158 typeset -A tests[18]=()
159 tests[18][desc]="backslash octal in format"
160 tests[18][format]='H\1120K\0112tabbed\1lagain'
161 tests[18][args]-
162 tests[18][result]='H\1J0K          2tabbed again'

164 typeset -A tests[19]=()
165 tests[19][desc]="backslash octal in %b"
166 tests[19][format]="%b"
167 tests[19][args]='H\0112K\011tabbed'
168 tests[19][result]='HIJK tabbed'

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

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

182 #debug=yes

184 for i in "${!tests[@]}"; do
185     t=test_$i
186     desc=${tests[$i][desc]}
187     format=${tests[$i][format]}
188     args="${tests[$i][args]}"
189     result=${tests[$i][result]}
190

```

```

191     test_start $t "${tests[$i][desc]}"
192     [[ -n "$debug" ]] && echo $PRINTF "$format" "${args[@]}"
193     comp=$($PRINTF "$format" ${args[@]})
194     checkrv $t
195     [[ -n "$debug" ]] && echo "got [$comp]"
196     good=$result
197     compare $t "$comp" "$good"
198     test_pass $t
199 done

```