
6739 Tue Dec 11 09:48:08 2018

new/usr/src/man/man1/pmadvice.1

10057 Man page misspellings output particular overridden

Reviewed by: Gerg Mihály Doma <domag02@gmail.com>

```

1  \" te
2 .\" Copyright (c) 2008, Sun Microsystems, Inc. All Rights Reserved.
3 .\" The contents of this file are subject to the terms of the Common Development
4 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
5 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
6  .TH PMADVICE 1 \"Sep 25, 2008\"
7  .SH NAME
8  pmadvice \- applies advice about memory to a process
9  .SH SYNOPSIS
10 .LP
11 .nf
12 \fBpmadvice\fR \fB-o\fR \fIoption\fR[, \fIoption\fR] [\fB-F\fR] [\fB-l\fR] [\fB-v
13 .fi

15 .SH DESCRIPTION
16 .LP
17 \fBpmadvice\fR applies advice about how memory is used in the specified process
18 using \fBmadvice\fR(3C).
19 .sp
20 .LP
21 \fBpmadvice\fR allows users to apply advice to a specific sub-range at a
22 specific instant in time. \fBpmadvice\fR differs from \fBmadv.so.1\fR(1) in
23 that \fBmadv.so.1\fR(1) applies the advice throughout execution of the target
24 program to all segments of a specified type.
25 .SH OPTIONS
26 .LP
27 The following options are supported:
28 .sp
29 .ne 2
30 .na
31 \fB\F\fR
32 .ad
33 .RS 6n
34 Force by grabbing the target process even if another process has control.
35 .sp
36 You should exercise caution when using the \fB-F\fR option. See \fBproc\fR(1).
37 .RE

39 .sp
40 .ne 2
41 .na
42 \fB-l\fR
43 .ad
44 .RS 6n
45 Show unresolved dynamic linker map names.
46 .RE

48 .sp
49 .ne 2
50 .na
51 \fB-o\fR
52 .ad
53 .RS 6n
54 Specify advice to apply in the following form:
55 .sp
56 .in +2
57 .nf
58 private=\fBadvice\fR
59 shared=\fBadvice\fR
60 heap=\fBadvice\fR

```

```

61 stack=\fBadvice\fR
62 \fBAddress\fR[:\fBlength\fR]=\fBadvice\fR
63 .fi
64 .in -2
65 .sp

67 where the \fBadvice\fR can be one of the following:
68 .sp
69 .in +2
70 .nf
71 normal
72 random
73 sequential
74 willneed
75 dontneed
76 free
77 access_lwp
78 access_many
79 access_default
80 purge
81 .fi
82 .in -2
83 .sp

85 An \fBAddress\fR and \fBlength\fR can be given to specify a subrange to apply
86 the advice. The \fBAddress\fR should be hexadecimal and the \fBlength\fR should
87 be in bytes by default.
88 .sp
89 If \fBlength\fR is not specified and the starting address refers to the start
90 of a segment, the advice is applied to that segment. \fBlength\fR can be
91 qualified by \fBK\fR, \fBM\fR, \fBG\fR, \fBT\fR, \fBP\fR, or \fBE\fR to specify
92 kilobytes, megabytes, gigabytes, terabytes, or exabytes respectively as the
93 unit of measure.
94 .RE

96 .sp
97 .ne 2
98 .na
99 \fB-v\fR
100 .ad
101 .RS 6n
102 Print verbose output. Display output as \fBpmap\fR(1) does, showing what advice
103 is being applied where. This can be useful when the advice is being applied to
104 a named region (for example, private, shared, and so forth) to get feedback on
105 exactly where the advice is being applied.
106 .RE

108 .sp
109 .LP
110 \fBpmadvice\fR tries to process all legal options. If an illegal address range
111 is specified, an error message is printed and the offending option is skipped.
112 \fBpmadvice\fR quits without processing any options and prints a usage message
113 when there is a syntax error.
114 .sp
115 .LP
116 If conflicting advice is given on a region, the order of precedence is from
117 most specific advice to least, that is, most general. In other words, advice
118 specified for a particular address range takes precedence over advice for heap
119 specified for a particular address range takes precedence over advice for heap
120 memory.
121 .sp
122 .LP
123 Moreover, the advice in each of the following groups are mutually exclusive
124 from the other advice within the same group:
125 .sp

```

```

126 .in +2
127 .nf
128 MADV_NORMAL, MADV_RANDOM, MADV_SEQUENTIAL
129 MADV_WILLNEED, MADV_DONTNEED, MADV_FREE, MADV_PURGE
130 MADV_ACCESS_DEFAULT, MADV_ACCESS_LWP, MADV_ACCESS_MANY
131 .fi
132 .in -2
133 .sp

135 .SH OPERANDS
136 .LP
137 The following operands are supported:
138 .sp
139 .ne 2
140 .na
141 \fB\fIpid\fR\fR
142 .ad
143 .RS 7n
144 Process ID.
145 .RE

147 .SH EXAMPLES
148 \fBExample 1\fR Applying Advice to a Segment at Specified Address
149 .sp
150 .LP
151 The following example applies advice to a segment at a specified address:

153 .sp
154 .in +2
155 .nf
156 % pmap $$
157 100666: tcsh
158 00010000 312K r-x-- /usr/bin/tcsh
159 0006C000 48K rwx-- /usr/bin/tcsh
160 00078000 536K rwx-- [ heap ]
161 FF100000 856K r-x-- /lib/libc.so.1
162 FF1E6000 32K rwx-- /lib/libc.so.1
163 FF1EE000 8K rwx-- /lib/libc.so.1
164 FF230000 168K r-x-- /lib/libcurses.so.1
165 FF26A000 32K rwx-- /lib/libcurses.so.1
166 FF272000 8K rwx-- /lib/libcurses.so.1
167 FF280000 576K r-x-- /lib/libnsl.so.1
168 FF310000 40K rwx-- /lib/libnsl.so.1
169 FF31A000 24K rwx-- /lib/libnsl.so.1
170 FF364000 8K rwx-- [ anon ]
171 FF370000 48K r-x-- /lib/libsocket.so.1
172 FF38C000 8K rwx-- /lib/libsocket.so.1
173 FF3A0000 8K r-x-- /platform/sun4u-us3/lib/libc_psr.so.1
174 FF3B0000 176K r-x-- /lib/ld.so.1
175 FF3EC000 8K rwx-- /lib/ld.so.1
176 FF3EE000 8K rwx-- /lib/ld.so.1
177 FFBE6000 104K rw--- [ stack ]
178 %
179 % pmadvise -o 78000=access_lwp $$

181 %
182 .fi
183 .in -2
184 .sp

186 .LP
187 \fBExample 2\fR Using the \fB-v\fR Option
188 .sp
189 .LP
190 The following example displays verbose output from \fBpmadvise\fR:

```

```

192 .sp
193 .in +2
194 .nf

196 % pmadvise -o heap=access_lwp,stack=access_default -v $$
197 1720: -sh
198 00010000 88K r-x-- /sbin/sh
199 00036000 8K rwx-- /sbin/sh
200 00038000 16K rwx-- [ heap ] <= access_lwp
201 FF250000 24K r-x-- /lib/libgen.so.1
202 FF266000 8K rwx-- /lib/libgen.so.1
203 FF272000 8K rwx-- [ anon ]
204 FF280000 840K r-x-- /lib/libc.so.1
205 FF362000 32K rwx-- /lib/libc.so.1
206 FF36A000 16K rwx-- /lib/libc.so.1
207 FF380000 8K r-x-- /platform/sun4u-us3/lib/libc_psr.so.1
208 FF390000 64K rwx-- [ anon ]
209 FF3B0000 168K r-x-- /lib/ld.so.1
210 FF3EA000 8K rwx-- /lib/ld.so.1
211 FF3EC000 8K rwx-- /lib/ld.so.1
212 FFBE0000 8K rw--- [ stack ] <= access_default
213 .fi
214 .in -2
215 .sp

217 .SH EXIT STATUS
218 .LP
219 The following exit values are returned:
220 .sp
221 .ne 2
222 .na
223 \fB\fI0\fR\fR
224 .ad
225 .RS 12n
226 Successful completion.
227 .RE

229 .sp
230 .ne 2
231 .na
232 \fB\fInon-zero\fR\fR
233 .ad
234 .RS 12n
235 An error occurred.
236 .RE

238 .SH FILES
239 .ne 2
240 .na
241 \fB\fIproc/*\fR\fR
242 .ad
243 .RS 19n
244 Process files
245 .RE

247 .sp
248 .ne 2
249 .na
250 \fB\fIusr/prob/lib/*\fR\fR
251 .ad
252 .RS 19n
253 \fB\fIproc\fR tools support files
254 .RE

256 .SH ATTRIBUTES
257 .LP

```

new/usr/src/man/man1/pmadvise.1

5

258 See \fBattributes\fR(5) for descriptions of the following attributes:
259 .sp

261 .sp
262 .TS
263 box;
264 c | c
265 l | l .
266 ATTRIBUTE TYPE ATTRIBUTE VALUE
267 -
268 Interface Stability See below.
269 .TE

271 .sp
272 .LP
273 The command syntax is Evolving. The output formats are Unstable.
274 .SH SEE ALSO
275 .LP
276 \fBmadv.so.1\fR(1), \fBpmap\fR(1), \fBproc\fR(1), \fBmadvise\fR(3C),
277 \fBattributes\fR(5)

```

*****
18942 Tue Dec 11 09:48:08 2018
new/usr/src/man/man1m/fsdb_udfs.1m
10057 Man page misspellings ouput particular overridden
Reviewed by: Gerg Mihajly Doma <domag02@gmail.com>
*****
1 \" te
2.\" Copyright (c) 1999 Sun Microsystems, Inc. All Rights Reserved.
3.\" The contents of this file are subject to the terms of the Common Development
4.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
5.\" When distributing Covered Code, include this CDDL HEADER in each file and in
6.TH FSDB_UDFS 1M "Nov 26, 2017"
7.SH NAME
8 fsdb_udfs \- udfs file system debugger
9.SH SYNOPSIS
10.LP
11.nf
12 \fBfsdb \fR [\fB-F\fR] udfs [\fIgeneric_option\fR] [\fB-o \fR\fIspecific_option\
13 .fi
15.SH DESCRIPTION
16.LP
17 The \fBfsdb_udfs\fR command is an interactive tool that can be used to patch up
18 a damaged \fBudfs\fR file system. \fBfsdb_udfs\fR has conversions to translate
19 block and i-numbers into their corresponding disk addresses. Mnemonic offsets
20 to access different parts of an inode are also included. Mnemonic offsets
21 greatly simplify the process of correcting control block entries or descending
22 the file system tree.
23.sp
24.LP
25 \fBfsdb\fR contains several error-checking routines to verify inode and block
26 addresses. These can be disabled if necessary by invoking \fBfsdb\fR with the
27 \fB-o\fR option or by using the \fBo\fR command.
28.sp
29.LP
30 \fBfsdb\fR reads one block at a time, and therefore works with raw as well as
31 block \fBBI/O\fR devices. A buffer management routine is used to retain commonly
32 used blocks of data in order to reduce the number of read system calls. All
33 assignment operations result in an immediate write-through of the corresponding
34 block. In order to modify any portion of the disk, \fBfsdb\fR must be invoked
35 with the \fB-w\fR option.
36.sp
37.LP
38 Wherever possible, \fBadb\fR-like syntax has been adopted to promote the use of
39 \fBfsdb\fR through familiarity.
40.SH OPTIONS
41.LP
42 The following options are supported:
43.sp
44.ne 2
45.na
46 \fB\F-o\fR \fIspecific_option\fR
47.ad
48.RS 22n
49 Specify \fBudfs\fR file system specific options in a comma-separated list with
50 no intervening spaces. The following specific options are supported:
51.sp
52.ne 2
53.na
54 \fBo\fR
55.ad
56.RS 12n
57 Override some error conditions.
58.RE
60.sp

```

```

61.ne 2
62.na
63 \fBp=\fIstring\fR
64.ad
65.RS 12n
66 Set prompt to \fIstring\fR.
67.RE
69.sp
70.ne 2
71.na
72 \fBw\fR
73.ad
74.RS 12n
75 Open for write.
76.RE
78.sp
79.ne 2
80.na
81 \fB?\fR
82.ad
83.RS 12n
84 Display usage.
85.RE
87.RE
89.SH USAGE
90.LP
91 Numbers are considered hexadecimal by default. The user has control over how
92 data is to be displayed or accepted. The \fBbase\fR command displays or sets
93 the input and output base. Once set, all input defaults to this base and all
94 output displays in this base. The base can be overridden temporarily for input
95 by preceding hexadecimal numbers by \fB0x\fR, preceding decimal numbers with a
96 \fB0t\fR, or octal numbers with a \fB0\fR. Hexadecimal numbers beginning with
97 \fBa\fR-\fBf\fR or \fBA\fR-\fBBF\fR must be preceded with a \fB0x\fR to
98 distinguish them from commands.
99.sp
100.LP
101 Disk addressing by \fBfsdb\fR is at the byte level. However, \fBfsdb\fR offers
102 many commands to convert a desired inode, directory entry, block, and so forth,
103 to a byte address. After the address has been calculated, \fBfsdb\fR records
104 the result in the current address (\fBdot\fR).
105.sp
106.LP
107 Several global values are maintained by \fBfsdb\fR:
108.RS +4
109.TP
110.ie t \(\bu
111.el o
112 Current base (referred to as \fBbase\fR)
113.RE
114.RS +4
115.TP
116.ie t \(\bu
117.el o
118 Current address (referred to as \fBdot\fR)
119.RE
120.RS +4
121.TP
122.ie t \(\bu
123.el o
124 Current inode (referred to as \fBinode\fR)

```

```

125 .RE
126 .RS +4
127 .TP
128 .ie t \(\bu
129 .el o
130 Current count (referred to as \fBcount\fR)
131 .RE
132 .RS +4
133 .TP
134 .ie t \(\bu
135 .el o
136 Current type (referred to as \fBtype\fR)
137 .RE
138 .sp
139 .LP
140 Most commands use the preset value of \fBdot\fR in their execution. For
141 example,
142 .sp
143 .in +2
144 .nf
145 > 2:inode
146 .fi
147 .in -2
148 .sp

150 .sp
151 .LP
152 first sets the value of dot (\fB\&.\fR) to \fB2\fR, colon (\fB:\fR), signifies
153 the start of a command, and the \fBinode\fR command sets \fBinode\fR to
154 \fB2\fR. A count is specified after a comma (\fB,\fR). Once set, count remains
155 at this value until a new command is encountered that resets the value back to
156 \fB1\fR (the default).
157 .sp
158 .LP
159 So, if
160 .sp
161 .in +2
162 .nf
163 > 2000,400/X
164 .fi
165 .in -2
166 .sp

168 .sp
169 .LP
170 is entered, \fB400\fR hex longs are listed from \fB2000\fR, and when completed,
171 the value of dot is \fB2000 + 400 * sizeof\fR (long). If a RETURN is then
172 entered, the output routine uses the current values of \fBdot\fR, \fBcount\fR,
173 and \fBtype\fR and displays \fB400 \fRmore hex longs. An asterisk (\fB*\fR)
174 causes the entire block to be displayed. An example showing several commands
175 and the use of RETURN would be:
176 .sp
177 .in +2
178 .nf
179 > 2:ino; 0:dir?d
180 .fi
181 .in -2
182 .sp

184 .sp
185 .LP
186 or
187 .sp
188 .in +2
189 .nf
190 > 2:ino; 0:db:block?d

```

```

191 .fi
192 .in -2
193 .sp

195 .sp
196 .LP
197 The two examples are synonymous for getting to the first directory entry of the
198 root of the file system. Once there, subsequently entering a RETURN, plus
199 (\fB+\fR), or minus (\fB-\fR) advances to subsequent entries. Notice that
200 .sp
201 .in +2
202 .nf
203 > 2:inode; :ls
204 .fi
205 .in -2
206 .sp

208 .sp
209 .LP
210 or
211 .sp
212 .in +2
213 .nf
214 > :ls /
215 .fi
216 .in -2
217 .sp

219 .sp
220 .LP
221 is again synonymous.
222 .SS "Expressions"
223 .LP
224 The following symbols are recognized by \fBfsdb\fR:
225 .sp
226 .ne 2
227 .na
228 \fBRETURN\fR
229 .ad
230 .RS 13n
231 Update the value of dot by the current value of \fBtype\fR and \fBdisplay\fR
232 using the current value of \fBcount\fR.
233 .RE

235 .sp
236 .ne 2
237 .na
238 \fB\#I#\fR
239 .ad
240 .RS 13n
241 Update the value of dot by specifying a numeric expression. Specify numeric
242 expressions using addition, subtraction, multiplication, and division
243 operators ( \fB+\fR, \fB-\fR, \fB*\fR, and \fB%\fR). Numeric expressions are
244 evaluated from left to right and can use parentheses. After evaluation, the
245 value of dot is updated.
246 .RE

248 .sp
249 .ne 2
250 .na
251 \fB\, \fBcount\fR
252 .ad
253 .RS 13n
254 Update the count indicator. The global value of \fBcount\fR is updated to
255 \fBcount\fR. The value of \fBcount\fR remains until a new command is run. A
256 \fBcount\fR specifier of \fB*\fR attempts to show a blocks's worth of

```

```

257 information. The default for \fIcount\fR is \fB1\fR.
258 .RE

260 .sp
261 .ne 2
262 .na
263 \fB? \fI f\fR\fR
264 .ad
265 .RS 13n
266 Display in structured style with format specifier \fBf\fR. See \fBFormatted
267 Output\fR.
268 .RE

270 .sp
271 .ne 2
272 .na
273 \fB/ \fIf\fR\fR
274 .ad
275 .RS 13n
276 Display in unstructured style with format specifier \fBf\fR. See \fBFormatted
277 Output\fR.
278 .RE

280 .sp
281 .ne 2
282 .na
283 \fB\&.\fR
284 .ad
285 .RS 13n
286 Display the value of dot.
287 .RE

289 .sp
290 .ne 2
291 .na
292 \fB+\fIe\fR\fR
293 .ad
294 .RS 13n
295 Increment the value of dot by the expression \fIe\fR. The amount actually
296 incremented is dependent on the size of type: \fBdot = dot + \fR\fIe \fR\fB*
297 \fR\fBsizeof\fR (\fItype\fR) The default for \fIe\fR is \fB1\fR.
298 .RE

300 .sp
301 .ne 2
302 .na
303 \fB\mi\fIe\fR\fR
304 .ad
305 .RS 13n
306 Decrement the value of dot by the expression \fIe \fR. See \fB+\fR.
307 .RE

309 .sp
310 .ne 2
311 .na
312 \fB*\fIe\fR\fR
313 .ad
314 .RS 13n
315 Multiply the value of dot by the expression \fIe\fR. Multiplication and
316 division don't use \fItype\fR. In the above calculation of dot, consider the
317 \fBsizeof\fR (\fItype\fR) to be \fB1\fR.
318 .RE

320 .sp
321 .ne 2
322 .na

```

```

323 \fB*\fIe\fR\fR
324 .ad
325 .RS 13n
326 Divide the value of dot by the expression \fIe\fR. See \fB*\fR.
327 .RE

329 .sp
330 .ne 2
331 .na
332 \fB< \fIname\fR\fR
333 .ad
334 .RS 13n
335 Restore an address saved in register \fIname\fR. \fIname\fR must be a single
336 letter or digit.
337 .RE

339 .sp
340 .ne 2
341 .na
342 \fB> \fIname\fR\fR
343 .ad
344 .RS 13n
345 Save an address in register \fIname\fR. \fIname\fR must be a single letter or
346 digit.
347 .RE

349 .sp
350 .ne 2
351 .na
352 \fB= \fIf\fR\fR
353 .ad
354 .RS 13n
355 Display indicator. If \fIf\fR is a legitimate format specifier (see
356 \fBFormatted Output\fR), then the value of dot is displayed using format
357 specifier \fIf\fR. Otherwise, assignment is assumed. See \fB= [s] [e]\fR.
358 .RE

360 .sp
361 .ne 2
362 .na
363 \fB= [\fIs\fR] [\fIe\fR]\fR
364 .ad
365 .RS 13n
366 Change the value of dot using an assignment indicator. The address pointed to
367 by dot has its contents changed to the value of the expression \fIe\fR or to
368 the \fBASCII\fR representation of the quoted (\fB"\fR) string \fIs\fR. This can
369 be useful for changing directory names or \fBASCII\fR file information.
370 .RE

372 .sp
373 .ne 2
374 .na
375 \fB+= \fIe\fR\fR
376 .ad
377 .RS 13n
378 Change the value of dot using an incremental assignment. The address pointed to
379 by dot has its contents incremented by expression \fIe\fR.
380 .RE

382 .sp
383 .ne 2
384 .na
385 \fB-= e\fR
386 .ad
387 .RS 13n
388 Change the value of dot using a decremental assignment. Decrement the contents

```

```

389 of the address pointed to by dot by expression \fIe\fR.
390 .RE

392 .SS "Commands"
393 .LP
394 A command must be prefixed by a colon (\fB:\fR). Only enough letters of the
395 command to uniquely distinguish it are needed. Multiple commands can be entered
396 on one line by separating them by a SPACE, TAB, or semicolon (\fB;\fR).
397 .sp
398 .LP
399 To view a potentially unmounted disk in a reasonable manner, \fBfsdb\fR
400 supports the \fBcd\fR, \fBpwd\fR, \fBls\fR, and \fBfind\fR commands. The
401 functionality of each of these commands basically matches that of its UNIX
402 counterpart. See \fBcd\fR(1), \fBpwd\fR(1), \fBls\fR(1), and \fBfind\fR(1) for
403 details. The \fB*\fR, \fB.\fR, \fB?\fR, and \fB-\fR wildcard characters are
404 also supported.
405 .sp
406 .LP
407 The following commands are supported:
408 .sp
409 .ne 2
410 .na
411 \fBbase[=\fIb\fR]\fR
412 .ad
413 .sp .6
414 .RS 4n
415 Display or set the base. All input and output is governed by the current base.
416 Without the \fB=\fR \fIb\fR, displays the current base. Otherwise, sets the
417 current base to \fIb\fR. Base is interpreted using the old value of base, so to
418 ensure correctness use the \fB0\fR, \fB0t\fR, or \fB0x\fR prefix when changing
419 the base. The default for base is hexadecimal.
420 .RE

422 .sp
423 .ne 2
424 .na
425 \fBblock\fR
426 .ad
427 .sp .6
428 .RS 4n
429 Convert the value of dot to a block address.
430 .RE

432 .sp
433 .ne 2
434 .na
435 \fBcd [\fIdir\fR]\fR
436 .ad
437 .sp .6
438 .RS 4n
439 Change the current directory to directory \fIdir\fR. The current values of
440 inode and dot are also updated. If \fBdir\fR is not specified, changes
441 directories to inode 2, root (\fB/\fR).
442 .RE

444 .sp
445 .ne 2
446 .na
447 \fBdirectory\fR
448 .ad
449 .sp .6
450 .RS 4n
451 If the current inode is a directory, converts the value of dot to a directory
452 slot offset in that directory, and dot now points to this entry.
453 .RE

```

```

455 .sp
456 .ne 2
457 .na
458 \fBfile\fR
459 .ad
460 .sp .6
461 .RS 4n
462 Set the value of dot as a relative block count from the beginning of the file.
463 The value of dot is updated to the first byte of this block.
464 .RE

466 .sp
467 .ne 2
468 .na
469 \fBfind \fIdir\fR [\fB-name\fR \fIn\fR] | [\fB-inum\fR \fI i\fR]\fR
470 .ad
471 .sp .6
472 .RS 4n
473 Find files by name or i-number. Recursively searches directory \fIdir\fR and
474 below for file names whose i-number matches \fBi\fR or whose name matches
475 below for file names whose i-number matches \fBi\fR or whose name matches
476 pattern \fIn\fR. Only one of the two options (\fB-name\fR or \fB-inum\fR) can
477 be used at one time. The find \fB-print\fR is not necessary or accepted.
478 .RE

479 .sp
480 .ne 2
481 .na
482 \fBfill=\fIip\fR\fR
483 .ad
484 .sp .6
485 .RS 4n
486 Fill an area of disk with pattern \fIip\fR. The area of disk is delimited by dot
487 and count.
488 .RE

490 .sp
491 .ne 2
492 .na
493 \fBinode\fR
494 .ad
495 .sp .6
496 .RS 4n
497 Convert the value of dot to an inode address. If successful, the current value
498 of inode is updated as well as the value of dot. As a convenient shorthand, if
499 \fB:inode\fR appears at the beginning of the line, the value of dot is set to
500 the current inode and that inode is displayed in inode format.
501 .RE

503 .sp
504 .ne 2
505 .na
506 \fBl [ \fB-R\fR ] [\fB-l\fR ] \fIpat1\fR \fIpat2\fR...\fR
507 .ad
508 .sp .6
509 .RS 4n
510 List directories or files. If no file is specified, the current directory is
511 assumed. Either or both of the options can be used (but, if used, must be
512 specified before the filename specifiers). Wild card characters are available
513 and multiple arguments are acceptable. The long listing shows only the i-number
514 and the name; use the inode command with \fB?i\fR to get more information.
515 .RE

517 .sp
518 .ne 2
519 .na

```

```

520 \fBoverride\fR
521 .ad
522 .sp .6
523 .RS 4n
524 Toggle the value of override. Some error conditions might be overridden if
525 override is toggled to \fBon\fR.
526 .RE

528 .sp
529 .ne 2
530 .na
531 \fBprompt "\fIp\fR"\fR
532 .ad
533 .sp .6
534 .RS 4n
535 Change the \fBfsdb\fR prompt to \fIp\fR. \fIp\fR must be enclosed in quotes.
536 .RE

538 .sp
539 .ne 2
540 .na
541 \fBpwd\fR
542 .ad
543 .sp .6
544 .RS 4n
545 Display the current working directory.
546 .RE

548 .sp
549 .ne 2
550 .na
551 \fBquit\fR
552 .ad
553 .sp .6
554 .RS 4n
555 Quit \fBfsdb\fR.
556 .RE

558 .sp
559 .ne 2
560 .na
561 \fBtag\fR
562 .ad
563 .sp .6
564 .RS 4n
565 Convert the value of dot and if this is a valid tag, print the volume structure
566 according to the tag.
567 .RE

569 .sp
570 .ne 2
571 .na
572 \fB!\fR
573 .ad
574 .sp .6
575 .RS 4n
576 Escape to the shell.
577 .RE

579 .SS "Inode Commands"
580 .LP
581 In addition to the above commands, several other commands deal with inode
582 fields and operate directly on the current inode (they still require the colon
583 (\fB:\fR). They can be used to more easily display or change the particular
584 fields. The value of dot is only used by the \fB:db\fR and \fB:ib\fR commands.
585 Upon completion of the command, the value of dot is changed so that it points

```

```

586 to that particular field. For example,
587 .sp
588 .in +2
589 .nf
590 > :ln+=1
591 .fi
592 .in -2
593 .sp

595 .sp
596 .LP
597 increments the link count of the current inode and sets the value of dot to the
598 address of the link count field.
599 .sp
600 .LP
601 The following inode commands are supported:
602 .sp
603 .ne 2
604 .na
605 \fBat\fR
606 .ad
607 .RS 8n
608 Access time
609 .RE

611 .sp
612 .ne 2
613 .na
614 \fBbs\fR
615 .ad
616 .RS 8n
617 Block size
618 .RE

620 .sp
621 .ne 2
622 .na
623 \fBct\fR
624 .ad
625 .RS 8n
626 Creation time
627 .RE

629 .sp
630 .ne 2
631 .na
632 \fBgid\fR
633 .ad
634 .RS 8n
635 Group id
636 .RE

638 .sp
639 .ne 2
640 .na
641 \fBln\fR
642 .ad
643 .RS 8n
644 Link number
645 .RE

647 .sp
648 .ne 2
649 .na
650 \fBmt\fR
651 .ad

```



```

652 .RS 8n
653 Modification time
654 .RE

656 .sp
657 .ne 2
658 .na
659 \fBmd\fR
660 .ad
661 .RS 8n
662 Mode
663 .RE

665 .sp
666 .ne 2
667 .na
668 \fBmaj\fR
669 .ad
670 .RS 8n
671 Major device number
672 .RE

674 .sp
675 .ne 2
676 .na
677 \fBmin\fR
678 .ad
679 .RS 8n
680 Minor device number
681 .RE

683 .sp
684 .ne 2
685 .na
686 \fBnm\fR
687 .ad
688 .RS 8n
689 This command actually operates on the directory name field. Once      poised at
690 the desired directory entry (using the \fBdirectory\fR command), this command
691 allows you to change or display the directory name. For example,
692 .sp
693 .in +2
694 .nf
695 > 7:dir:nm="foo"
696 .fi
697 .in -2
698 .sp

700 gets the \fB7\fRth directory entry of the current inode and changes its name to
701 \fBfoo\fR. Directory names cannot be made larger than the field allows. If an
702 attempt is made to make a directory name larger than the field allows,, the
703 string is truncated to fit and a warning message is displayed.
704 .RE

706 .sp
707 .ne 2
708 .na
709 \fBsz\fR
710 .ad
711 .RS 8n
712 File size
713 .RE

715 .sp
716 .ne 2
717 .na

```

```

718 \fBuid\fR
719 .ad
720 .RS 8n
721 User \fBID\fR
722 .RE

724 .sp
725 .ne 2
726 .na
727 \fBuniq\fR
728 .ad
729 .RS 8n
730 Unique \fBID\fR
731 .RE

733 .SS "Formatted Output"
734 .LP
735 Formatted output comes in two styles and many format types. The two styles of
736 formatted output are: structured and unstructured. Structured output is used to
737 display inodes, directories, and so forth. Unstructured output displays raw
738 data.
739 .sp
740 .LP
741 Format specifiers are preceded by the slash (\fB/\fR) or question mark
742 (\fB?\fR) character. \fIitype\fR is updated as necessary upon completion.
743 .sp
744 .LP
745 The following format specifiers are preceded by the \fB?\fR character:
746 .sp
747 .ne 2
748 .na
749 \fBi\fR
750 .ad
751 .RS 5n
752 Display as inodes in the current base.
753 .RE

755 .sp
756 .ne 2
757 .na
758 \fBd\fR
759 .ad
760 .RS 5n
761 Display as directories in the current base.
762 .RE

764 .sp
765 .LP
766 The following format specifiers are preceded by the \fB/\fR character:
767 .sp
768 .ne 2
769 .na
770 \fBb\fR
771 .ad
772 .RS 9n
773 Display as bytes in the current base.
774 .RE

776 .sp
777 .ne 2
778 .na
779 \fBc\fR
780 .ad
781 .RS 9n
782 Display as characters.
783 .RE

```

```

785 .sp
786 .ne 2
787 .na
788 \fBo | O\fR
789 .ad
790 .RS 9n
791 Display as octal shorts or longs.
792 .RE

794 .sp
795 .ne 2
796 .na
797 \fBd | D\fR
798 .ad
799 .RS 9n
800 Display as decimal shorts or longs.
801 .RE

803 .sp
804 .ne 2
805 .na
806 \fBx | X\fR
807 .ad
808 .RS 9n
809 Display as hexadecimal shorts or longs.
810 .RE

812 .SH EXAMPLES
813 .LP
814 \fBExample 1 \fRUsing fsdb as a calculator for complex arithmetic
815 .sp
816 .LP
817 The following command displays \fB2010\fR in decimal format, and is an example
818 of using \fBfsdb\fR as a calculator for complex arithmetic.

820 .sp
821 .in +2
822 .nf
823 > 2000+400%(20+20)=D
824 .fi
825 .in -2
826 .sp

828 .LP
829 \fBExample 2 \fRUsing fsdb to display an i-number in inode format
830 \fBExample 2 \fRUsing fsdb to display an i-number in inode format
831 .sp
832 The following command displays the i-number \fB386\fR in inode format.\fB386\fR
833 becomes the current inode.

835 .sp
836 .in +2
837 .nf
838 > 386:ino?i
839 .fi
840 .in -2
841 .sp

843 .LP
844 \fBExample 3 \fRUsing fsdb to change the link count
845 .sp
846 .LP
847 The following command changes the link count for the current inode to \fB4\fR.

```

```

849 .sp
850 .in +2
851 .nf
852 > :ln=4
853 .fi
854 .in -2
855 .sp

857 .LP
858 \fBExample 4 \fRUsing fsdb to increment the link count
859 .sp
860 .LP
861 The following command increments the link count by \fB1\fR.

863 .sp
864 .in +2
865 .nf
866 > :ln+=1
867 .fi
868 .in -2
869 .sp

871 .LP
872 \fBExample 5 \fRUsing fsdb to display the creation time as a hexadecimal long
873 .sp
874 .LP
875 The following command displays the creation time as a hexadecimal long.

877 .sp
878 .in +2
879 .nf
880 > :ct=X
881 .fi
882 .in -2
883 .sp

885 .LP
886 \fBExample 6 \fRUsing fsdb to display the modification time in time format
887 .sp
888 .LP
889 The following command displays the modification time in time format.

891 .sp
892 .in +2
893 .nf
894 > :mt=t
895 .fi
896 .in -2
897 .sp

899 .LP
900 \fBExample 7 \fRUsing fsdb to display in ASCII
901 .sp
902 .LP
903 The following command displays, in \fBASCII\fR, block \fB0\fR of the file
904 associated with the current inode.

906 .sp
907 .in +2
908 .nf
909 > 0:file/c
910 .fi
911 .in -2
912 .sp

914 .LP

```

```

915 \fBExample 8 \fRUsing fsdb to display the directory enteries for the root inode
916 .sp
917 .LP
918 The following command displays the first block's directory entries for the root
919 inode of this file system. This command stops prematurely if the \fBEOF\fR is
920 reached.
922 .sp
923 .in +2
924 .nf
925 > 2:ino,*?d
926 .fi
927 .in -2
928 .sp
930 .LP
931 \fBExample 9 \fRUsing fsdb to change the current inode
932 .sp
933 .LP
934 The following command changes the current inode to that associated with the
935 \fB5\fRth directory entry (numbered from \fB0\fR) of the current inode. The
936 first logical block of the file is then displayed in \fBASCII\fR.
938 .sp
939 .in +2
940 .nf
941 > 5:dir:inode; 0:file,*?c
942 .fi
943 .in -2
944 .sp
946 .LP
947 \fBExample 10 \fRUsing fsdb to change the i-number
948 .sp
949 .LP
950 The following command changes the i-number for the \fB7\fRth directory slot in
951 the root directory to \fB3\fR.
953 .sp
954 .in +2
955 .nf
956 > 2:inode; 7:dir=3
957 .fi
958 .in -2
959 .sp
961 .LP
962 \fBExample 11 \fRUsing fsdb to change the name field
963 .sp
964 .LP
965 The following command changes the \fBname\fR field in the directory slot to
966 \fBname\fR.
968 .sp
969 .in +2
970 .nf
971 > 7:dir:nm="name"
972 .fi
973 .in -2
974 .sp
976 .LP
977 \fBExample 12 \fRUsing fsdb to display the a block
978 .sp
979 .LP
980 The following command displays the \fB3\fRrd block of the current inode as

```

```

981 directory entries.
983 .LP
984 \fBExample 13 \fRUsing fsdb to set the contents of address
985 .sp
986 .LP
987 The following command sets the contents of address \fB2050\fR to
988 \fB0xffffffff\fR. \fB0xffffffff\fR can be truncated, depending on the current
989 type.
991 .sp
992 .in +2
993 .nf
994 > 2050=0xffff
995 .fi
996 .in -2
997 .sp
999 .LP
1000 \fBExample 14 \fRUsing fsdb to place an ASCII string at an address
1001 .sp
1002 .LP
1003 The following command places the \fBASCII\fR string \fBthis is some text\fR at
1004 address \fB1c92434\fR.
1006 .sp
1007 .in +2
1008 .nf
1009 > 1c92434="this is some text"
1010 .fi
1011 .in -2
1012 .sp
1014 .SH SEE ALSO
1015 .LP
1016 \fBbelri\fR(1M), \fBfsc_udfs\fR(1M), \fBdir\fR(4), \fBattributes\fR(5)

```

```

*****
22547 Tue Dec 11 09:48:08 2018
new/usr/src/man/man1m/fsdb_ufs.1m
10057 Man page misspellings ouput particular overridden
Reviewed by: Gerg Mihajly Doma <domag02@gmail.com>
*****
1 \" te
2.\" Copyright (c) 1988 Regents of the University
3.\" of California. All rights reserved. Copyright (c) 2003 Sun Microsystems,
4.\" Inc. All Rights Reserved.
5.TH FSDB_UFS 1M \"Apr 14, 2003\"
6.SH NAME
7 fsdb_ufs \- ufs file system debugger
8.SH SYNOPSIS
9.LP
10.nf
11 \fBfsdb\fR \fB-F\fR ufs [\fIgeneric_options\fR] [\fIspecific_options\fR] \fIspec
12.fi

14.SH DESCRIPTION
15.sp
16 The \fBfsdb_ufs\fR command is an interactive tool that can be used to patch up
17 a damaged \fBUFS\fR file system. It has conversions to translate block and
18 i-numbers into their corresponding disk addresses. Also included are mnemonic
19 offsets to access different parts of an inode. These greatly simplify the
20 process of correcting control block entries or descending the file system tree.
21.sp
22.LP
23 \fBfsdb\fR contains several error-checking routines to verify inode and block
24 addresses. These can be disabled if necessary by invoking \fBfsdb\fR with the
25 \fB-o\fR option or by the use of the \fBo\fR command.
26.sp
27.LP
28 \fBfsdb\fR reads a block at a time and will therefore work with raw as well as
29 block \fBBI/O\fR devices. A buffer management routine is used to retain commonly
30 used blocks of data in order to reduce the number of read system calls. All
31 assignment operations result in an immediate write-through of the corresponding
32 block. Note that in order to modify any portion of the disk, \fBfsdb\fR must be
33 invoked with the \fBw\fR option.
34.sp
35.LP
36 Wherever possible, \fBbadb-\fRlike syntax was adopted to promote the use of
37 \fBfsdb\fR through familiarity.
38.SH OPTIONS
39.LP
40 The following option is supported:
41.sp
42.ne 2
43.na
44 \fB\fb-o\fR
45.ad
46.RS 6n
47 Specify \fBUFS\fR file system specific options. These options can be any
48 combination of the following separated by commas (with no intervening spaces).
49 The options available are:
50.sp
51.ne 2
52.na
53 \fB\fb\F?\fR
54.ad
55.RS 14n
56 Display usage
57.RE

```

```

59 .sp
60 .ne 2
61 .na
62 \fB\fbBo\fR
63 .ad
64 .RS 14n
65 Override some error conditions
66 .RE

68 .sp
69 .ne 2
70 .na
71 \fB\fbBp='string'\fR
72 .ad
73 .RS 14n
74 set prompt to string
75 .RE

77 .sp
78 .ne 2
79 .na
80 \fB\fbBw\fR
81 .ad
82 .RS 14n
83 open for write
84 .RE

86 .RE

88.SH USAGE
91.sp
89.LP
90 Numbers are considered hexadecimal by default. However, the user has control
91 over how data is to be displayed or accepted. The \fBbase\fR command will
92 display or set the input/output base. Once set, all input will default to this
93 base and all output will be shown in this base. The base can be overridden
94 temporarily for input by preceding hexadecimal numbers with \&'fB0x\fR',
95 preceding decimal numbers with '\fB0t\fR', or octal numbers with '\fB0\fR'.
96 Hexadecimal numbers beginning with \fBa-f\fR or \fBA-F\fR must be preceded with
97 \&'fB0x\fR' to distinguish them from commands.
98.sp
99.LP
100 Disk addressing by \fBfsdb\fR is at the byte level. However, \fBfsdb\fR offers
101 many commands to convert a desired inode, directory entry, block, superblock
102 and so forth to a byte address. Once the address has been calculated,
103 \fBfsdb\fR will record the result in dot (\fB&.\fR).
104.sp
105.LP
106 Several global values are maintained by \fBfsdb\fR:
107.RS +4
108.TP
109.ie t \(\bu
110.el o
111 the current base (referred to as \fBbase\fR),
112.RE
113.RS +4
114.TP
115.ie t \(\bu
116.el o
117 the current address (referred to as \fBdot\fR),
118.RE
119.RS +4
120.TP
121.ie t \(\bu
122.el o
123 the current inode (referred to as \fBinode\fR),

```

```

124 .RE
125 .RS +4
126 .TP
127 .ie t \(\bu
128 .el o
129 the current count (referred to as \fBcount\fR),
130 .RE
131 .RS +4
132 .TP
133 .ie t \(\bu
134 .el o
135 and the current type (referred to as \fBtype\fR).
136 .RE
137 .sp
138 .LP
139 Most commands use the preset value of \fBdot\fR in their execution. For
140 example,
141 .sp
142 .LP
143 \fB> 2:inode\fR
144 .sp
145 .LP
146 will first set the value of \fBdot\fR to 2, ':', will alert the start of a
147 command, and the \fBinode\fR command will set \fBinode\fR to 2. A count is
148 specified after a ','. Once set, \fBcount\fR will remain at this value until a
149 new command is encountered which will then reset the value back to 1 (the
150 default). So, if
151 .sp
152 .LP
153 \fB> 2000,400/X\fR
154 .sp
155 .LP
156 is typed, 400 hex longs are listed from 2000, and when completed, the value of
157 \fBdot\fR will be \fB2000 + 400 * sizeof (long)\fR. If a \fBRETURN\fR is then
158 typed, the output routine will use the current values of \fBdot\fR,
159 \fBcount\fR, and \fBtype\fR and display 400 more hex longs. A '*' will cause
160 the entire block to be displayed.
161 .sp
162 .LP
163 End of fragment, block and file are maintained by \fBfsdb\fR. When displaying
164 data as fragments or blocks, an error message will be displayed when the end of
165 fragment or block is reached. When displaying data using the \fBdb\fR,
166 \fBbib\fR, \fBdirectory\fR, or \fBfile\fR commands an error message is displayed
167 if the end of file is reached. This is mainly needed to avoid passing the end
168 of a directory or file and getting unknown and unwanted results.
169 .sp
170 .LP
171 An example showing several commands and the use of \fBRETURN\fR would be:
172 .sp
173 .in +2
174 .nf
175 \fB> 2:ino; 0:dir?d\fR
176 or
177 \fB> 2:ino; 0:db:block?d\fR
178 .fi
179 .in -2
180 .sp
182 .sp
183 .LP
184 The two examples are synonymous for getting to the first directory entry of the
185 root of the file system. Once there, any subsequent \fBRETURN\fR (or +, -)
186 will advance to subsequent entries. Note that
187 .sp
188 .in +2
189 .nf

```

```

190 \fB> 2:inode; :ls\fR
191 or
192 \fB> :ls /\fR
193 .fi
194 .in -2
195 .sp
197 .sp
198 .LP
199 is again synonymous.
200 .SS "Expressions"
204 .sp
201 .LP
202 The symbols recognized by \fBfsdb\fR are:
203 .sp
204 .ne 2
205 .na
206 \fB\FBRETURN\fR\fR
207 .ad
208 .RS 13n
209 update the value of \fBdot\fR by the current value of \fBtype\fR and display
210 using the current value of \fBcount\fR.
211 .RE
213 .sp
214 .ne 2
215 .na
216 \fB\FB#\fR\fR
217 .ad
218 .RS 13n
219 numeric expressions may be composed of +, -, *, and % operators (evaluated left
220 to right) and may use parentheses. Once evaluated, the value of \fBdot\fR is
221 updated.
222 .RE
224 .sp
225 .ne 2
226 .na
227 \fB\FB,\fR\fR\fI count\fR\fR
228 .ad
229 .RS 13n
230 count indicator. The global value of \fBcount\fR will be updated to
231 \fBcount\fR. The value of \fBcount\fR will remain until a new command is run. A
232 count specifier of '*' will attempt to show a \fBblocks's\fR worth of
233 information. The default for \fBcount\fR is 1.
234 .RE
236 .sp
237 .ne 2
238 .na
239 \fB\FB?\fR\fR\fI f\fR\fR
240 .ad
241 .RS 13n
242 display in structured style with format specifier \fRif\fR. See
243 \fBFormatted Output\fR.
244 \fBFormatted\fROutput\fB&.\fR
244 .RE
246 .sp
247 .ne 2
248 .na
249 \fB\FB/\fR\fR\fI f\fR\fR
250 .ad
251 .RS 13n
252 display in unstructured style with format specifier \fRif\fR. See
253 \fBFormatted Output\fR.

```

```

256 display in unstructured style with format specifier \fIf\fR See
257 \fBFormatted\fROutput\fB\&.\fR
254 .RE

256 .sp
257 .ne 2
258 .na
259 \fB\fB\&.\fR\fR
260 .ad
261 .RS 13n
262 the value of \fBdot\fR.
263 .RE

265 .sp
266 .ne 2
267 .na
268 \fB\fB+\fR\fIe\fR\fR
269 .ad
270 .RS 13n
271 increment the value of \fBdot\fR by the expression \fIe.\fR The amount actually
272 incremented is dependent on the size of \fBtype\fR:
273 .sp
274 \fBdot = dot + e * sizeof (type)\fR
275 .sp
276 The default for \fIe\fR is \fB1\fR.
277 .RE

279 .sp
280 .ne 2
281 .na
282 \fB\fB-\fR\fIe\fR\fR
283 .ad
284 .RS 13n
285 decrement the value of \fBdot\fR by the expression \fIe\fR. See \fB+\fR.
286 .RE

288 .sp
289 .ne 2
290 .na
291 \fB\fB*\fR\fIe\fR\fR
292 .ad
293 .RS 13n
294 multiply the value of \fBdot\fR by the expression \fIe.\fR Multiplication and
295 division don't use \fBtype\fR. In the above calculation of \fBdot\fR, consider
296 the \fBsizeof(type)\fR to be \fB1\fR.
297 .RE

299 .sp
300 .ne 2
301 .na
302 \fB\fB%\fR\fIe\fR\fR
303 .ad
304 .RS 13n
305 divide the value of \fBdot\fR by the expression \fIe\fR. See \fB*\fR.
306 .RE

308 .sp
309 .ne 2
310 .na
311 \fB\fB<\fR\fI name\fR\fR
312 .ad
313 .RS 13n
314 restore an address saved in register \fIname\fR. \fIname\fR must be a single
315 letter or digit.
316 .RE

```

```

318 .sp
319 .ne 2
320 .na
321 \fB\fB>\fR\fI name\fR\fR
322 .ad
323 .RS 13n
324 save an address in register \fIname\fR. \fIname\fR must be a single letter or
325 digit.
326 .RE

328 .sp
329 .ne 2
330 .na
331 \fB\fB=\fR\fI f\fR\fR
332 .ad
333 .RS 13n
334 display indicator. If \fIf\fR is a legitimate format specifier, then the value
335 display indicator. If \fIf\fR is a legitimate format specifier, then the value
336 of \fBdot\fR is displayed using the format specifier \fIf\fR. See
337 \fBFormatted Output\fR. Otherwise, assignment is assumed. See \fB=\fR.
338 \fBFormatted Output\fR. Otherwise, assignment is assumed See \fB=\fR.
339 .RE

339 .sp
340 .ne 2
341 .na
342 \fB\fB= [\fR\fIis\fR\fB] [\fR\fIe\fR\fB]\fR\fR
343 .ad
344 .RS 13n
345 assignment indicator. The address pointed to by \fBdot\fR has its contents
346 changed to the value of the expression \fIe\fR or to the \fBASCII\fR
347 representation of the quoted (") string \fIis\fR. This may be useful for
348 changing directory names or \fBASCII\fR file information.
349 .RE

351 .sp
352 .ne 2
353 .na
354 \fB\fB+=\fR\fI e\fR\fR
355 .ad
356 .RS 13n
357 incremental assignment. The address pointed to by \fBdot\fR has its contents
358 incremented by expression \fIe\fR.
359 .RE

361 .sp
362 .ne 2
363 .na
364 \fB\fB=-\fR\fI e\fR\fR
365 .ad
366 .RS 13n
367 decremental assignment. The address pointed to by \fBdot\fR has its contents
368 decremented by expression \fIe\fR.
369 .RE

371 .SS "Commands"
376 .sp
377 .LP
378 A command must be prefixed by a ':' character. Only enough letters of the
379 command to uniquely distinguish it are needed. Multiple commands may be entered
380 on one line by separating them by a \fBSPACE\fR, \fBTAB\fR or ';'.
381 .sp
382 .LP
383 In order to view a potentially unmounted disk in a reasonable manner,
384 \fBfsdb\fR offers the \fBcd\fR, \fBpwd\fR, \fBls\fR and \fBfind\fR commands.
385 The functionality of these commands substantially matches those of its UNIX

```

```

381 counterparts. See individual commands for details. The '*', '?', and '[' wild
382 card characters are available.
383 .sp
384 .ne 2
385 .na
386 \fB\fBbase=b\fR\fR
387 .ad
388 .sp .6
389 .RS 4n
390 display or set base. As stated above, all input and output is governed by the
391 current \fBbase\fR. If the \fB=b\fR is omitted, the current \fBbase\fR is
392 displayed. Otherwise, the current \fBbase\fR is set to \fBb.\fR Note that this
393 is interpreted using the old value of \fBbase\fR, so to ensure correctness use
394 the '0', '0t', or '0x' prefix when changing the \fBbase\fR. The default for
395 \fBbase\fR is hexadecimal.
396 .RE

398 .sp
399 .ne 2
400 .na
401 \fB\fBblock\fR\fR
402 .ad
403 .sp .6
404 .RS 4n
405 convert the value of \fBdot\fR to a block address.
406 .RE

408 .sp
409 .ne 2
410 .na
411 \fB\fBbcd \fR\fR\fR
412 .ad
413 .sp .6
414 .RS 4n
415 change the current directory to directory \fR. The current values of
416 \fBbinode\fR and \fBdot\fR are also updated. If no \fR is specified, then
417 change directories to inode \fB2\fR ("/").
418 .RE

420 .sp
421 .ne 2
422 .na
423 \fB\fBbcg\fR\fR
424 .ad
425 .sp .6
426 .RS 4n
427 convert the value of \fBdot\fR to a cylinder group.
428 .RE

430 .sp
431 .ne 2
432 .na
433 \fB\fBdirectory\fR\fR
434 .ad
435 .sp .6
436 .RS 4n
437 If the current \fBbinode\fR is a directory, then the value of \fBdot\fR is
438 converted to a directory slot offset in that directory and \fBdot\fR now points
439 to this entry.
440 .RE

442 .sp
443 .ne 2
444 .na
445 \fB\fBfile\fR\fR
446 .ad

```

```

447 .sp .6
448 .RS 4n
449 the value of \fBdot\fR is taken as a relative block count from the beginning of
450 the file. The value of \fBdot\fR is updated to the first byte of this block.
451 .RE

453 .sp
454 .ne 2
455 .na
456 \fB\fBfind\fR \fR [ \fB-name\fR \fR] [\fB-inum\fR \fR]\fR
457 .ad
458 .sp .6
459 .RS 4n
460 find files by name or i-number. \fBfind\fR recursively searches directory
461 \fBdir\fR and below for filenames whose i-number matches \fR or whose name
462 matches pattern \fR. Note that only one of the two options (-name or -inum)
463 may be used at one time. Also, the -print is not needed or accepted.
464 .RE

466 .sp
467 .ne 2
468 .na
469 \fB\fBfill\fR \fR=\fR\fR
470 .ad
471 .sp .6
472 .RS 4n
473 fill an area of disk with pattern \fR. The area of disk is delimited by
474 \fBdot\fR and \fBcount\fR.
475 .RE

477 .sp
478 .ne 2
479 .na
480 \fB\fBfragment\fR\fR
481 .ad
482 .sp .6
483 .RS 4n
484 convert the value of \fBdot\fR to a fragment address. The only difference
485 between the \fBfragment\fR command and the \fBblock\fR command is the amount
486 that is able to be displayed.
487 .RE

489 .sp
490 .ne 2
491 .na
492 \fB\fBbinode\fR\fR
493 .ad
494 .sp .6
495 .RS 4n
496 convert the value of \fBdot\fR to an inode address. If successful, the current
497 value of \fBbinode\fR will be updated as well as the value of \fBdot\fR. As a
498 convenient shorthand, if ':inode' appears at the beginning of the line, the
499 value of \fBdot\fR is set to the current \fBbinode\fR and that inode is
500 displayed in inode format.
501 .RE

503 .sp
504 .ne 2
505 .na
506 \fB\fBlog_chk\fR\fR
507 .ad
508 .sp .6
509 .RS 4n
510 run through the valid log entries without printing any information and verify
511 the layout.
512 .RE

```

```

514 .sp
515 .ne 2
516 .na
517 \fB\fBlog_delta\fR\fR
518 .ad
519 .sp .6
520 .RS 4n
521 count the number of deltas into the log, using the value of dot as an offset
522 into the log. No checking is done to make sure that offset is within the
523 head/tail offsets.
524 .RE

526 .sp
527 .ne 2
528 .na
529 \fB\fBlog_head\fR\fR
530 .ad
531 .sp .6
532 .RS 4n
533 display the header information about the file system logging. This shows the
534 block allocation for the log and the data structures on the disk.
535 .RE

537 .sp
538 .ne 2
539 .na
540 \fB\fBlog_otodb\fR\fR
541 .ad
542 .sp .6
543 .RS 4n
544 return the physical disk block number, using the value of dot as an offset into
545 the log.
546 .RE

548 .sp
549 .ne 2
550 .na
551 \fB\fBlog_show\fR\fR
552 .ad
553 .sp .6
554 .RS 4n
555 display all deltas between the beginning of the log (BOL) and the end of the
556 log (EOL).
557 .RE

559 .sp
560 .ne 2
561 .na
562 \fB\fBls\fR\fR
563 .ad
564 .sp .6
565 .RS 4n
566 [ \fB-R\fR ] [ \fB-l\fR ] \fIpat1 pat2\fR\|.|\|. list directories or files.
567 If no file is specified, the current directory is assumed. Either or both of
568 the options may be used (but, if used, \fImust\fR be specified before the
569 filename specifiers). Also, as stated above, wild card characters are available
570 and multiple arguments may be given. The long listing shows only the i-number
571 and the name; use the \fBinode\fR command with '?i' to get more information.
572 .RE

574 .sp
575 .ne 2
576 .na
577 \fB\fBoverride\fR\fR
578 .ad

```

```

579 .sp .6
580 .RS 4n
581 toggle the value of override. Some error conditions may be overridden if
582 toggle the value of override. Some error conditions may be overridden if
583 .RE

585 .sp
586 .ne 2
587 .na
588 \fB\fBprompt\fR\fR \fI p\fR\fR
589 .ad
590 .sp .6
591 .RS 4n
592 change the \fBfsdb\fR prompt to \fI p\fR. \fI p\fR must be surrounded by (")s.
593 .RE

595 .sp
596 .ne 2
597 .na
598 \fB\fBpwd\fR\fR
599 .ad
600 .sp .6
601 .RS 4n
602 display the current working directory.
603 .RE

605 .sp
606 .ne 2
607 .na
608 \fB\fBquit\fR\fR
609 .ad
610 .sp .6
611 .RS 4n
612 quit \fBfsdb\fR.
613 .RE

615 .sp
616 .ne 2
617 .na
618 \fB\fBsb\fR\fR
619 .ad
620 .sp .6
621 .RS 4n
622 the value of \fIdot\fR is taken as a cylinder group number and then converted
623 to the address of the superblock in that cylinder group. As a shorthand, ':sb'
624 at the beginning of a line will set the value of \fIdot\fR to \fIthe\fR
625 superblock and display it in superblock format.
626 .RE

628 .sp
629 .ne 2
630 .na
631 \fB\fBshadow\fR\fR
632 .ad
633 .sp .6
634 .RS 4n
635 if the current inode is a shadow inode, then the value of \fIdot\fR is set to
636 the beginning of the shadow inode data.
637 .RE

639 .sp
640 .ne 2
641 .na
642 \fB\fB!\fR\fR
643 .ad

```



```
644 .sp .6
645 .RS 4n
646 escape to shell
647 .RE
```

```
649 .SS "Inode Commands"
```

```
655 .sp
650 .LP
651 In addition to the above commands, there are several commands that deal with
652 inode fields and operate directly on the current \fBinode\fR (they still
653 require the ':'). They may be used to more easily display or change the
654 particular fields. The value of \fIdot\fR is only used by the '\fB:db\fR'
655 and '\fB:ib\fR' commands. Upon completion of the command, the value of \fIdot\fR
656 changed to point to that particular field. For example,
```

```
657 .sp
658 .LP
659 \fB> :ln+=1\fR
660 .sp
661 .LP
662 would increment the link count of the current \fBinode\fR and set the value of
663 \fIdot\fR to the address of the link count field.
```

```
664 .sp
665 .ne 2
666 .na
667 \fB\fBat\fR\fR
668 .ad
669 .RS 7n
670 access time.
671 .RE
```

```
673 .sp
674 .ne 2
675 .na
676 \fB\fBbs\fR\fR
677 .ad
678 .RS 7n
679 block size.
680 .RE
```

```
682 .sp
683 .ne 2
684 .na
685 \fB\fBct\fR\fR
686 .ad
687 .RS 7n
688 creation time.
689 .RE
```

```
691 .sp
692 .ne 2
693 .na
694 \fB\fBdb\fR\fR
695 .ad
696 .RS 7n
697 use the current value of \fIdot\fR as a direct block index, where direct blocks
698 number from 0 - 11. In order to display the block itself, you need to 'pipe'
699 this result into the \fBblock\fR or \fBfragment\fR command. For example,
```

```
700 .sp
701 .in +2
702 .nf
703 \fB      > 1:db:block,20/X\fR
704 .fi
705 .in -2
706 .sp
```

708 would get the contents of data block field 1 from the inode and convert it to a

709 block address. 20 longs are then displayed in hexadecimal. See

```
710 \fBFormatted Output\fR.
716 \fBFormatted\fROutput\fB&.\fR
711 .RE
```

```
713 .sp
714 .ne 2
715 .na
716 \fB\fBgid\fR\fR
717 .ad
718 .RS 7n
719 group id.
720 .RE
```

```
722 .sp
723 .ne 2
724 .na
725 \fB\fBib\fR\fR
726 .ad
727 .RS 7n
728 use the current value of \fIdot\fR as an indirect block index where indirect
729 blocks number from 0 - 2. This will only get the indirect block itself (the
730 block containing the pointers to the actual blocks). Use the \fBfile\fR command
731 and start at block 12 to get to the actual blocks.
732 .RE
```

```
734 .sp
735 .ne 2
736 .na
737 \fB\fBln\fR\fR
738 .ad
739 .RS 7n
740 link count.
741 .RE
```

```
743 .sp
744 .ne 2
745 .na
746 \fB\fBmt\fR\fR
747 .ad
748 .RS 7n
749 modification time.
750 .RE
```

```
752 .sp
753 .ne 2
754 .na
755 \fB\fBmd\fR\fR
756 .ad
757 .RS 7n
758 mode.
759 .RE
```

```
761 .sp
762 .ne 2
763 .na
764 \fB\fBmaj\fR\fR
765 .ad
766 .RS 7n
767 major device number.
768 .RE
```

```
770 .sp
771 .ne 2
772 .na
773 \fB\fBmin\fR\fR
```

```

774 .ad
775 .RS 7n
776 minor device number.
777 .RE

779 .sp
780 .ne 2
781 .na
782 \fB\fBnm\fR\fR
783 .ad
784 .RS 7n
785 although listed here, this command actually operates on the directory name
786 field. Once poised at the desired directory entry (using the \fIdirectory\fR
787 command), this command will allow you to change or display the directory name.
788 For example,
789 .sp
790 \fB> 7:dir:nm="foo"\fR
791 .sp
792 will get the \fB7\fRth directory entry of the current \fBinode\fR and change
793 its name to foo. Note that names cannot be made larger than the field is set up
794 for. If an attempt is made, the string is truncated to fit and a warning
795 message to this effect is displayed.
796 .RE

798 .sp
799 .ne 2
800 .na
801 \fB\fBsi\fR\fR
802 .ad
803 .RS 7n
804 shadow inode.
805 .RE

807 .sp
808 .ne 2
809 .na
810 \fB\fBsz\fR\fR
811 .ad
812 .RS 7n
813 file size.
814 .RE

816 .sp
817 .ne 2
818 .na
819 \fB\fBuid\fR\fR
820 .ad
821 .RS 7n
822 user id.
823 .RE

825 .SS "Formatted Output"
826 .LP
827 There are two styles and many format types. The two styles are structured and
828 unstructured. Structured output is used to display inodes, directories,
829 superblocks and the like. Unstructured displays raw data. The following shows
830 the different ways of displaying:
831 .sp
832 .ne 2
833 .na
834 \fB\fB?\fR\fR
835 .ad
836 .RS 5n
837 .sp
838 .ne 2

```

```

839 .na
840 \fB\fBc\fR\fR
841 .ad
842 .RS 5n
843 display as cylinder groups
844 .RE

846 .sp
847 .ne 2
848 .na
849 \fB\fBi\fR\fR
850 .ad
851 .RS 5n
852 display as inodes
853 .RE

855 .sp
856 .ne 2
857 .na
858 \fB\fBd\fR\fR
859 .ad
860 .RS 5n
861 display as directories
862 .RE

864 .sp
865 .ne 2
866 .na
867 \fB\fBs\fR\fR
868 .ad
869 .RS 5n
870 display as superblocks
871 .RE

873 .sp
874 .ne 2
875 .na
876 \fB\fBS\fR\fR
877 .ad
878 .RS 5n
879 display as shadow inode data
880 .RE

882 .RE

884 .sp
885 .ne 2
886 .na
887 \fB\fB/\fR\fR
888 .ad
889 .RS 5n
890 .sp
891 .ne 2
892 .na
893 \fB\fBb\fR\fR
894 .ad
895 .RS 7n
896 display as bytes
897 .RE

899 .sp
900 .ne 2
901 .na
902 \fB\fBc\fR\fR
903 .ad
904 .RS 7n

```

```

905 display as characters
906 .RE

908 .sp
909 .ne 2
910 .na
911 \fB\fBo O\fR\fR
912 .ad
913 .RS 7n
914 display as octal shorts or longs
915 .RE

917 .sp
918 .ne 2
919 .na
920 \fB\fBd D\fR\fR
921 .ad
922 .RS 7n
923 display as decimal shorts or longs
924 .RE

926 .sp
927 .ne 2
928 .na
929 \fB\fBx X\fR\fR
930 .ad
931 .RS 7n
932 display as hexadecimal shorts or longs
933 .RE

935 The format specifier immediately follows the '/' or '?' character. The values
936 displayed by '/b' and all '?' formats are displayed in the current \fBbase\fR.
937 Also, \fBtype\fR is appropriately updated upon completion.
938 .RE

940 .SH EXAMPLES
941 .LP
942 \fBExample 1 \fRDisplaying in Decimal
943 .sp
944 .LP
945 The following command displays \fB2010\fR in decimal (use of \fBfsdb\fR as a
946 calculator for complex arithmetic):

948 .sp
949 .in +2
950 .nf
951 > 2000+400%(20+20)=D
952 .fi
953 .in -2
954 .sp

956 .LP
957 \fBExample 2 \fRDisplaying an i-number in Inode Format
958 .sp
959 .LP
960 The following command displays i-number \fB386\fR in an inode format. This now
961 becomes the current \fBinode\fR:

963 .sp
964 .in +2
965 .nf
966 > 386:ino?i
967 .fi
968 .in -2
969 .sp

```

```

971 .LP
972 \fBExample 3 \fRChanging the Link Count
973 .sp
974 .LP
975 The following command changes the link count for the current \fBinode\fR to
976 \fB4\fR:

978 .sp
979 .in +2
980 .nf
981 > :ln=4
982 .fi
983 .in -2
984 .sp

986 .LP
987 \fBExample 4 \fRIncrementing the Link Count
988 .sp
989 .LP
990 The following command increments the link count by \fB1\fR:

992 .sp
993 .in +2
994 .nf
995 > :ln+=1
996 .fi
997 .in -2
998 .sp

1000 .LP
1001 \fBExample 5 \fRDisplaying the Creation Time
1002 .sp
1003 .LP
1004 The following command displays the creation time as a hexadecimal long:

1006 .sp
1007 .in +2
1008 .nf
1009 > :ct=X
1010 .fi
1011 .in -2
1012 .sp

1014 .LP
1015 \fBExample 6 \fRDisplaying the Modification Time
1016 .sp
1017 .LP
1018 The following command displays the modification time in time format:

1020 .sp
1021 .in +2
1022 .nf
1023 > :mt=t
1024 .fi
1025 .in -2
1026 .sp

1028 .LP
1029 \fBExample 7 \fRDisplaying in ASCII
1030 .sp
1031 .LP
1032 The following command displays in \fBASCII,\fR block zero of the file
1033 associated with the current \fBinode\fR:

1035 .sp
1036 .in +2

```

```

1037 .nf
1038 > 0:file/c
1039 .fi
1040 .in -2
1041 .sp

1043 .LP
1044 \fBExample 8 \fRDisplaying the First Block's Worth of Directory Entries
1045 .sp
1046 .LP
1047 The following command displays the first block's worth of directory entries for
1048 the root inode of this file system. It will stop prematurely if the \fBEOF\fR
1049 is reached:

1051 .sp
1052 .in +2
1053 .nf
1054 > 2:ino,*?d
1055 .fi
1056 .in -2
1057 .sp

1059 .LP
1060 \fBExample 9 \fRDisplaying Changes to the Current Inode
1061 .sp
1062 .LP
1063 The following command displays changes the current inode to that associated
1064 with the \fB5\fRth directory entry (numbered from zero) of the current
1065 \fBInode\fR. The first logical block of the file is then displayed in
1066 \fBBASCII\fR:

1068 .sp
1069 .in +2
1070 .nf
1071 > 5:dir:inode; 0:file,*/c
1072 .fi
1073 .in -2
1074 .sp

1076 .LP
1077 \fBExample 10 \fRDisplaying the Superblock
1078 .sp
1079 .LP
1080 The following command displays the superblock of this file system:

1082 .sp
1083 .in +2
1084 .nf
1085 > :sb
1086 .fi
1087 .in -2
1088 .sp

1090 .LP
1091 \fBExample 11 \fRDisplaying the Cylinder Group
1092 .sp
1093 .LP
1094 The following command displays cylinder group information and summary for
1095 cylinder group \fB1\fR:

1097 .sp
1098 .in +2
1099 .nf
1100 > 1:cg?c
1101 .fi
1102 .in -2

```

```

1103 .sp

1105 .LP
1106 \fBExample 12 \fRChanging the i-number
1107 .sp
1108 .LP
1109 The following command changes the i-number for the seventh directory slot in
1110 the root directory to \fB3\fR:

1112 .sp
1113 .in +2
1114 .nf
1115 > 2:inode; 7:dir=3
1116 .fi
1117 .in -2
1118 .sp

1120 .LP
1121 \fBExample 13 \fRDisplaying as Directory Entries
1122 .sp
1123 .LP
1124 The following command displays the third block of the current \fBInode\fR as
1125 directory entries:

1127 .sp
1128 .in +2
1129 .nf
1130 > 2:db:block,*?d
1131 .fi
1132 .in -2
1133 .sp

1135 .LP
1136 \fBExample 14 \fRChanging the Name Field
1137 .sp
1138 .LP
1139 The following command changes the name field in the directory slot to
1140 \fBfname\fR:

1142 .sp
1143 .in +2
1144 .nf
1145 > 7:dir:nm="name"
1146 .fi
1147 .in -2
1148 .sp

1150 .LP
1151 \fBExample 15 \fRGetting and Filling Elements
1152 .sp
1153 .LP
1154 The following command gets fragment \fB3c3\fR and fill \fB20\fR \fBtype\fR
1155 elements with \fB0x20\fR:

1157 .sp
1158 .in +2
1159 .nf
1160 > 3c3:fragment,20:fill=0x20
1161 .fi
1162 .in -2
1163 .sp

1165 .LP
1166 \fBExample 16 \fRSetting the Contents of an Address
1167 .sp
1168 .LP

```

```
1169 The following command sets the contents of address \fB2050\fR to
1170 \fB0xffffffff\fR. \fB0xffffffff\fR may be truncated depending on the current
1171 \fBtype\fR:

1173 .sp
1174 .in +2
1175 .nf
1176 > 2050=0xffff
1177 .fi
1178 .in -2
1179 .sp

1181 .LP
1182 \fBExample 17 \fRPlacing ASCII
1183 .sp
1184 .LP
1185 The following command places the \fBASCII\fR for the string at \fB1c92434\fR:

1187 .sp
1188 .in +2
1189 .nf
1190 > 1c92434="this is some text"
1191 .fi
1192 .in -2
1193 .sp

1195 .LP
1196 \fBExample 18 \fRDisplaying Shadow Inode Data
1197 .sp
1198 .LP
1199 The following command displays all of the shadow inode data in the shadow inode
1200 associated with the root inode of this file system:

1202 .sp
1203 .in +2
1204 .nf
1205 > 2:ino:si:ino;0:shadow,*?S
1206 .fi
1207 .in -2
1208 .sp

1210 .SH SEE ALSO
1218 .sp
1211 .LP
1212 \fBclri\fR(1M), \fBfsck_ufs\fR(1M), \fBdir_ufs\fR(4), \fBattributes\fR(5),
1213 \fBbufs\fR(7FS)
1214 .SH WARNINGS
1223 .sp
1215 .LP
1216 Since \fBfsdb\fR reads the disk raw, extreme caution is advised in determining
1217 its availability of \fBfsdb\fR on the system. Suggested permissions are 600 and
1218 owned by bin.
1219 .SH NOTES
1229 .sp
1220 .LP
1221 The old command line syntax for clearing i-nodes using the ufs-specific
1222 \fB\&'-z i-number'\fR option is still supported by the new debugger, though it
1223 is obsolete and will be removed in a future release. Use of this flag will
1224 result in correct operation, but an error message will be printed warning of
1225 the impending obsolescence of this option to the command. The equivalent
1226 functionality is available using the more flexible \fBclri\fR(1M) command.
```

```

*****
19056 Tue Dec 11 09:48:08 2018
new/usr/src/man/man1m/ikeadm.1m
10057 Man page misspellings output particular overridden
Reviewed by: Gerg Mihaly Doma <domag02@gmail.com>
*****
1 \" te
2 .\" Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved.
3 .\" The contents of this file are subject to the terms of the Common Development
4 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
5 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
6 .TH IKEADM 1M \"Jan 27, 2009\"
7 .SH NAME
8 ikeadm \- manipulate Internet Key Exchange (IKE) parameters and state
9 .SH SYNOPSIS
10 .LP
11 .nf
12 \fBikeadm\fR [\fB-mp\fR]
13 .fi

15 .LP
16 .nf
17 \fBikeadm\fR [\fB-mp\fR] get [debug | priv | stats | defaults]
18 .fi

20 .LP
21 .nf
22 \fBikeadm\fR [\fB-mp\fR] set [debug | priv] [level] [file]
23 .fi

25 .LP
26 .nf
27 \fBikeadm\fR [\fB-mp\fR] [get | del] [pl | rule | preshared] [id]
28 .fi

30 .LP
31 .nf
32 \fBikeadm\fR [\fB-mp\fR] add [rule | preshared] { \fBdescription\fR }
33 .fi

35 .LP
36 .nf
37 \fBikeadm\fR [\fB-mp\fR] token [login | logout] \fBIPKCS#11_Token_Object\fR
38 .fi

40 .LP
41 .nf
42 \fBikeadm\fR [\fB-mp\fR] [read | write] [rule | preshared | certcache] \fBifile\fR
43 .fi

45 .LP
46 .nf
47 \fBikeadm\fR [\fB-mp\fR] [dump | pls | rule | preshared]
48 .fi

50 .LP
51 .nf
52 \fBikeadm\fR [\fB-mp\fR] flush [pl | certcache]
53 .fi

55 .LP
56 .nf
57 \fBikeadm\fR help
58 [get | set | add | del | read | write | dump | flush | token]
59 .fi

```

```

61 .SH DESCRIPTION
62 .sp
63 .LP
64 The \fBikeadm\fR utility retrieves information from and manipulates the
65 configuration of the Internet Key Exchange (\fBIKE\fR) protocol daemon,
66 \fBin.iked\fR(1M).
67 .sp
68 .LP
69 \fBikeadm\fR supports a set of operations, which may be performed on one or
70 more of the supported object types. When invoked without arguments,
71 \fBikeadm\fR enters interactive mode which prints a prompt to the standard
72 output and accepts commands from the standard input until the end-of-file is
73 reached.
74 .sp
75 .LP
76 Because \fBikeadm\fR manipulates sensitive keying information, you must be
77 superuser to use this command. Additionally, some of the commands available
78 require that the daemon be running in a privileged mode, which is established
79 when the daemon is started.
80 .sp
81 .LP
82 For details on how to use this command securely see .
83 .SH OPTIONS
84 .sp
85 .LP
86 The following options are supported:
87 .sp
88 .ne 2
89 .na
90 \fBn\fR
91 .ad
92 .sp .6
93 .RS 4n
94 Prevent attempts to print host and network names symbolically when reporting
95 actions. This is useful, for example, when all name servers are down or are
96 otherwise unreachable.
97 .RE

98 .sp
99 .ne 2
100 .na
101 \fBp\fR
102 .ad
103 .sp .6
104 .RS 4n
105 Paranoid. Do not print any keying material, even if saving Security
106 Associations. Instead of an actual hexadecimal digit, print an \fBX\fR when
107 this flag is turned on.
108 .RE

109 .SH USAGE
110 .SS "Commands"
111 .sp
112 .LP
113 The following commands are supported:
114 .sp
115 .ne 2
116 .na
117 \fBadd\fR
118 .ad
119 .sp .6
120 .RS 4n
121 Add the specified object. This option can be used to add a new policy rule or a
122 new preshared key to the current (running) \fBin.iked\fR configuration. When adding a
123 new preshared key, the command cannot be invoked from the command line, as it
124 will contain keying material. The rule or key being added is specified using

```

```

124 appropriate id-value pairs as described in the \fBID FORMATS\fR section.
125 .RE

127 .sp
128 .ne 2
129 .na
130 \fB\fBdel\fR\fR
131 .ad
132 .sp .6
133 .RS 4n
134 Delete a specific object or objects from \fBin.iked\fR's current configuration.
135 This operation is available for \fBIKE\fR (Phase 1) \fBSA\fRs, policy rules,
136 and preshared keys. The object to be deleted is specified as described in the
137 \fBId Formats\fR.
138 .RE

140 .sp
141 .ne 2
142 .na
143 \fB\fBdump\fR\fR
144 .ad
145 .sp .6
146 .RS 4n
147 Display all objects of the specified type known to \fBin.iked\fR. This option
148 can be used to display all Phase 1 \fBSA\fRs, policy rules, preshared keys, or
149 the certificate cache. A large amount of output may be generated by this
150 command.
151 .RE

153 .sp
154 .ne 2
155 .na
156 \fB\fBflush\fR\fR
157 .ad
158 .sp .6
159 .RS 4n
160 Remove all \fBIKE\fR (Phase 1) \fBSA\fRs or cached certificates from
161 \fBin.iked\fR.
162 .sp
163 Note that flushing the \fBcertcache\fR will also (as a side-effect) update IKE
164 with any new certificates added or removed.
165 .RE

167 .sp
168 .ne 2
169 .na
170 \fB\fBget\fR\fR
171 .ad
172 .sp .6
173 .RS 4n
174 Lookup and display the specified object. May be used to view the current debug
175 or privilege level, global statistics and default values for the daemon, or a
176 specific \fBIKE\fR (Phase 1) \fBSA\fR, policy rule, or preshared key. The
177 latter three object types require that identifying information be passed in;
178 the appropriate specification for each object type is described below.
179 .RE

181 .sp
182 .ne 2
183 .na
184 \fB\fBhelp\fR\fR
185 .ad
186 .sp .6
187 .RS 4n
188 Print a brief summary of commands, or, when followed by a command, prints
189 information about that command.

```

```

190 .RE

192 .sp
193 .ne 2
194 .na
195 \fB\fBread\fR\fR
196 .ad
197 .sp .6
198 .RS 4n
199 Update the current \fBin.iked\fR configuration by reading the policy rules or
200 preshared keys from either the default location or from the file specified.
201 .RE

203 .sp
204 .ne 2
205 .na
206 \fB\fBset\fR\fR
207 .ad
208 .sp .6
209 .RS 4n
210 Adjust the current debug or privilege level. If the debug level is being
211 modified, an output file may optionally be specified; the output file
212 \fBmust\fR be specified if the daemon is running in the background and is not
213 currently printing to a file. When changing the privilege level, adjustments
214 may only be made to lower the access level; it cannot be increased using
215 ikeadm.
216 .RE

218 .sp
219 .ne 2
220 .na
221 \fB\fBwrite\fR\fR
222 .ad
223 .sp .6
224 .RS 4n
225 Write the current \fBin.iked\fR policy rule set or preshared key set to the
226 specified file. A destination file must be specified. This command should not
227 be used to overwrite the existing configuration files.
228 .RE

230 .sp
231 .ne 2
232 .na
233 \fB\fBtoken\fR\fR
234 .ad
235 .sp .6
236 .RS 4n
237 Log into a PKCS#11 token object and grant access to keying material or log out
238 and invalidate access to keying material.
239 .sp
240 \fBtoken\fR can be run as a normal user with the following authorizations:
241 .RS +4
242 .TP
243 .ie t \ (bu
244 .el o
245 \fBtoken\fR login: \fBsolaris.network.ipsec.ike.token.login\fR
246 .RE
247 .RS +4
248 .TP
249 .ie t \ (bu
250 .el o
251 \fBtoken\fR logout: \fBsolaris.network.ipsec.ike.token.logout\fR
252 .RE
253 .RE

255 .SS "Object Types"

```

```

259 .sp
256 .ne 2
257 .na
258 \fBdebug\fR
259 .ad
260 .sp .6
261 .RS 4n
262 Specifies the daemon's debug level. This determines the amount and type of
263 output provided by the daemon about its operations. The debug level is actually
264 a bitmask, with individual bits enabling different types of information.
265 .sp

```

```

267 .sp
268 .TS
269 c c c
270 l l l .
271 Description      Flag      Nickname
272 -
273 Certificate management 0x0001 cert
274 Key management      0x0002 key
275 Operational        0x0004 op
276 Phase 1 SA creation 0x0008 phase1
277 Phase 2 SA creation 0x0010 phase2
278 PF_KEY interface   0x0020 pfkey
279 Policy management  0x0040 policy
280 Proposal construction 0x0080 prop
281 Door interface     0x0100 door
282 Config file processing 0x0200 config
283 All debug flags    0x3ff  all
284 .TE

```

286 When specifying the debug level, either a number (decimal or hexadecimal) or a
 287 string of nicknames may be given. For example, \fB88\fR, \fB0x58\fR, and
 288 \fBphase1\fR+\fBphase2\fR+\fBpolicy\fR are all equivalent, and will turn on
 289 debug for \fBphase 1\fR \fBsa\fR creation, \fBphase 2 sa\fR creation, and
 290 policy management. A string of nicknames may also be used to remove certain
 291 types of information; \fBall-op\fR has the effect of turning on all debug
 292 \fBexcept\fR for operational messages; it is equivalent to the numbers
 293 \fB1019\fR or \fB0x3fb\fR.
 294 .RE

```

296 .sp
297 .ne 2
298 .na
299 \fBpriv\fR
300 .ad
301 .sp .6
302 .RS 4n
303 Specifies the daemon's access privilege level. The possible values are:
304 .sp
305 .in +2
306 .nf
307 Description          Level  Nickname
308 Base level           0      base
309 Access to preshared key info 1      modkeys
310 Access to keying material 2      keymat
311 .fi
312 .in -2
313 .sp

```

315 By default, \fBin.iked\fR is started at the base level. A command-line option
 316 can be used to start the daemon at a higher level. \fBikeadm\fR can be used to
 317 lower the level, but it cannot be used to raise the level.
 318 .sp
 319 Either the numerical level or the nickname may be used to specify the target
 320 privilege level.

```

321 .sp
322 In order to get, add, delete, dump, read, or write preshared keys, the
323 privilege level must at least give access to preshared key information.
324 However, when viewing preshared keys (either using the get or dump command),
325 the key itself will only be available if the privilege level gives access to
326 keying material. This is also the case when viewing Phase 1 \fBSA\fRs.
327 .RE

329 .sp
330 .ne 2
331 .na
332 \fBstats\fR
333 .ad
334 .sp .6
335 .RS 4n
336 Global statistics from the daemon, covering both successful and failed Phase 1
337 \fBSA\fR creation.
338 .sp
339 Reported statistics include:
340 .RS +4
341 .TP
342 .ie t \(\bu
343 .el o
344 Count of current P1 \fBSA\fRs which the local entity initiated
345 .RE
346 .RS +4
347 .TP
348 .ie t \(\bu
349 .el o
350 Count of current P1 \fBSA\fRs where the local entity was the responder
351 .RE
352 .RS +4
353 .TP
354 .ie t \(\bu
355 .el o
356 Count of all P1 \fBSA\fRs which the local entity initiated since boot
357 .RE
358 .RS +4
359 .TP
360 .ie t \(\bu
361 .el o
362 Count of all P1 \fBSA\fRs where the local entity was the responder since boot
363 .RE
364 .RS +4
365 .TP
366 .ie t \(\bu
367 .el o
368 Count of all attempted \fBP1\fR \fBSA\fRs since boot, where the local entity
369 was the initiator; includes failed attempts
370 .RE
371 .RS +4
372 .TP
373 .ie t \(\bu
374 .el o
375 Count of all attempted P1 \fBSA\fRs since boot, where the local entity was the
376 responder; includes failed attempts
377 .RE
378 .RS +4
379 .TP
380 .ie t \(\bu
381 .el o
382 Count of all failed attempts to initiate a \fBP1\fR \fBSA\fR, where the failure
383 occurred because the peer did not respond
384 .RE
385 .RS +4
386 .TP

```



```

387 .ie t \(\bu
388 .el o
389 Count of all failed attempts to initiate a P1 \fBSA\fR, where the peer
390 responded
391 .RE
392 .RS +4
393 .TP
394 .ie t \(\bu
395 .el o
396 Count of all failed \fBP1\fR \fBSA\fRs where the peer was the initiator
397 .RE
398 .RS +4
399 .TP
400 .ie t \(\bu
401 .el o
402 Whether a PKCS#11 library is in use, and if applicable, the PKCS#11 library
403 that is loaded. See .
404 .RE
405 .RE

407 .sp
408 .ne 2
409 .na
410 \fBdefaults\fR
411 .ad
412 .sp .6
413 .RS 4n
414 Display default values used by the \fBin.iked\fR daemon. Some values can be
415 overridden in the daemon configuration file (see \fBike.config\fR(4)); for these
416 overridden in the daemon configuration file (see \fBike.config\fR(4)); for these
417 values, the token name is displayed in the \fBget defaults\fR output. The
418 output will reflect where a configuration token has changed the default.
419 .sp
420 Default values might be ignored in the event a peer system makes a valid
421 alternative proposal or they can be overridden by per-rule values established in
422 alternative proposal or they can be overridden by per-rule values established in
423 \fBike.config\fR. In such instances, a \fBget defaults\fR command continues to
424 display the default values, not the values used to override the defaults.
425 .RE

425 .sp
426 .ne 2
427 .na
428 \fBP1\fR
429 .ad
430 .sp .6
431 .RS 4n
432 An \fBIKE\fR Phase 1 \fBSA\fR. A \fBP1\fR object is identified by an \fBIP\fR
433 address pair or a cookie pair; identification formats are described below.
434 .RE

436 .sp
437 .ne 2
438 .na
439 \fBBrule\fR
440 .ad
441 .sp .6
442 .RS 4n
443 An \fBIKE\fR policy rule, defining the acceptable security characteristics for
444 Phase 1 \fBSA\fRs between specified local and remote identities. A rule is
445 identified by its label; identification formats are described below.
446 .RE

448 .sp
449 .ne 2
450 .na

```

```

451 \fBpreshared\fR
452 .ad
453 .sp .6
454 .RS 4n
455 A preshared key, including the local and remote identification and applicable
456 \fBIKE\fR mode. A preshared key is identified by an \fBIP\fR address pair or an
457 identity pair; identification formats are described below.
458 .RE

460 .SS "Id Formats"
465 .sp
466 .LP
467 .LP
468 Commands like \fBadd\fR, \fBdel\fR, and \fBget\fR require that additional
469 information be specified on the command line. In the case of the delete and get
470 commands, all that is required is to minimally identify a given object; for the
471 add command, the full object must be specified.
472 .sp
473 Minimal identification is accomplished in most cases by a pair of values. For
474 \fBIP\fR addresses, the local addr and then the remote addr are specified,
475 either in dot-notation for IPv4 addresses, colon-separated hexadecimal format
476 for IPv6 addresses, or a host name present in the host name database. If a host
477 name is given that expands to more than one address, the requested operation
478 will be performed multiple times, once for each possible combination of
479 addresses.
480 .sp
481 .LP
482 Identity pairs are made up of a local type-value pair, followed by the remote
483 type-value pair. Valid types are:
484 .sp
485 .ne 2
486 .na
487 \fBprefix\fR
488 .ad
489 .sp .6
490 .RS 4n
491 An address prefix.
492 .RE

489 .sp
490 .ne 2
491 .na
492 \fBfqdn\fR
493 .ad
494 .sp .6
495 .RS 4n
496 A fully-qualified domain name.
497 .RE

499 .sp
500 .ne 2
501 .na
502 \fBdomain\fR
503 .ad
504 .sp .6
505 .RS 4n
506 Domain name, synonym for fqdn.
507 .RE

509 .sp
510 .ne 2
511 .na
512 \fBuser_fqdn\fR
513 .ad
514 .sp .6
515 .RS 4n

```

```

516 User identity of the form \fIuser\fR@fqdn.
517 .RE

519 .sp
520 .ne 2
521 .na
522 \fBmailbox\fR
523 .ad
524 .sp .6
525 .RS 4n
526 Synonym for \fBuser_fqdn\fR.
527 .RE

529 .sp
530 .LP
531 A cookie pair is made up of the two cookies assigned to a Phase 1 Security
532 Association (\fBSA\fR) when it is created; first is the initiator's, followed
533 by the responder's. A cookie is a 64-bit number.
534 .sp
535 .LP
536 Finally, a label (which is used to identify a policy rule) is a character
537 string assigned to the rule when it is created.
538 .sp
539 .LP
540 Formatting a rule or preshared key for the add command follows the format rules
541 for the in.iked configuration files. Both are made up of a series of id-value
542 pairs, contained in curly braces (\fB{\fR and \fB}\fR). See \fBike.config\fR(4)
543 and \fBike.preshared\fR(4) for details on the formatting of rules and preshared
544 keys.
545 .SH SECURITY
551 .sp
546 .LP
547 The \fBikeadm\fR command allows a privileged user to enter cryptographic keying
548 information. If an adversary gains access to such information, the security of
549 IPsec traffic is compromised. The following issues should be taken into account
550 when using the \fBikeadm\fR command.
551 .RS +4
552 .TP
553 .ie t \(\bu
554 .el o
555 Is the \fBTTY\fR going over a network (interactive mode)?
556 .sp
557 If it is, then the security of the keying material is the security of the
558 network path for this \fBTTY\fR's traffic. Using \fBikeadm\fR over a clear-text
559 telnet or rlogin session is risky. Even local windows may be vulnerable to
560 attacks where a concealed program that reads window events is present.
561 .RE
562 .RS +4
563 .TP
564 .ie t \(\bu
565 .el o
566 Is the file accessed over the network or readable to the world (read/write
567 commands)?
568 .sp
569 A network-mounted file can be sniffed by an adversary as it is being read. A
570 world-readable file with keying material in it is also risky.
571 .RE
572 .sp
573 .LP
574 If your source address is a host that can be looked up over the network, and
575 your naming system itself is compromised, then any names used will no longer be
576 trustworthy.
577 .sp
578 .LP
579 Security weaknesses often lie in misapplication of tools, not the tools
580 themselves. It is recommended that administrators are cautious when using the

```

```

581 \fBikeadm\fR command. The safest mode of operation is probably on a console, or
582 other hard-connected \fBTTY\fR.
583 .sp
584 .LP
585 For additional information regarding this subject, see the afterward by Matt
586 Blaze in Bruce Schneier's \fIApplied Cryptography: Protocols, Algorithms, and
587 Source Code in C.\fR
588 .SH EXAMPLES
589 .LP
590 \fBExample 1\fR \fREmptying out all Phase 1 Security Associations
591 .sp
592 .LP
593 The following command empties out all Phase 1 Security Associations:

595 .sp
596 .in +2
597 .nf
598 example# \fBikeadm flush pl\fR
599 .fi
600 .in -2
601 .sp

603 .LP
604 \fBExample 2\fR \fRDisplaying all Phase 1 Security Associations
605 .sp
606 .LP
607 The following command displays all Phase 1 Security Associations:

609 .sp
610 .in +2
611 .nf
612 example# \fBikeadm dump pl\fR
613 .fi
614 .in -2
615 .sp

617 .LP
618 \fBExample 3\fR \fRDeleting a Specific Phase 1 Security Association
619 .sp
620 .LP
621 The following command deletes the specified Phase 1 Security Associations:

623 .sp
624 .in +2
625 .nf
626 example# \fBikeadm del pl local_ip remote_ip\fR
627 .fi
628 .in -2
629 .sp

631 .LP
632 \fBExample 4\fR \fRAdding a Rule From a File
633 .sp
634 .LP
635 The following command adds a rule from a file:

637 .sp
638 .in +2
639 .nf
640 example# \fBikeadm add rule rule_file\fR
641 .fi
642 .in -2
643 .sp

645 .LP
646 \fBExample 5\fR \fRAdding a Preshared Key

```

```

647 .sp
648 .LP
649 The following command adds a preshared key:

651 .sp
652 .in +2
653 .nf
654 example# \fBikeadm\fR
655     ikeadm> \fBadd preshared { localidtype ip localid local_ip
656         remoteidtype ip remoteid remote_ip ike_mode main
657         key 1234567890abcdef1234567890abcdef }\fR
658 .fi
659 .in -2
660 .sp

662 .LP
663 \fBExample 6 \fRSaving All Preshared Keys to a File
664 .sp
665 .LP
666 The following command saves all preshared keys to a file:

668 .sp
669 .in +2
670 .nf
671 example# \fBikeadm write preshared target_file\fR
672 .fi
673 .in -2
674 .sp

676 .LP
677 \fBExample 7 \fRViewing a Particular Rule
678 .sp
679 .LP
680 The following command views a particular rule:

682 .sp
683 .in +2
684 .nf
685 example# \fBikeadm get rule rule_label\fR
686 .fi
687 .in -2
688 .sp

690 .LP
691 \fBExample 8 \fRReading in New Rules from \fBike.config\fR
692 .sp
693 .LP
694 The following command reads in new rules from the ike.config file:

696 .sp
697 .in +2
698 .nf
699 example# \fBikeadm read rules\fR
700 .fi
701 .in -2
702 .sp

704 .LP
705 \fBExample 9 \fRLowering the Privilege Level
706 .sp
707 .LP
708 The following command lowers the privilege level:

710 .sp
711 .in +2
712 .nf

```

```

713 example# \fBikeadm set priv base\fR
714 .fi
715 .in -2
716 .sp

718 .LP
719 \fBExample 10 \fRViewing the Debug Level
720 .sp
721 .LP
722 The following command shows the current debug level

724 .sp
725 .in +2
726 .nf
727 example# \fBikeadm get debug\fR
728 .fi
729 .in -2
730 .sp

732 .LP
733 \fBExample 11 \fRUsing stats to Verify Hardware Accelerator
734 .sp
735 .LP
736 The following example shows how stats may include an optional line at the end
737 to indicate if IKE is using a PKCS#11 library to accelerate public-key
738 operations, if applicable.

740 .sp
741 .in +2
742 .nf
743 example# \fBikeadm get stats\fR
744 Phase 1 SA counts:
745 Current:  initiator:      0  responder:      0
746 Total:    initiator:     21  responder:     27
747 Attempted: initiator:     21  responder:     27
748 Failed:   initiator:      0  responder:      0
749                                     initiator fails include 0 time-out(s)
750 PKCS#11 library linked in from /opt/SUNWconn/lib/libpkcs11.so
751 example#
752 .fi
753 .in -2
754 .sp

756 .LP
757 \fBExample 12 \fRDisplaying the Certificate Cache
758 .sp
759 .LP
760 The following command shows the certificate cache and the status of associated
761 private keys, if applicable:

763 .sp
764 .in +2
765 .nf
766 example# \fBikeadm dump certcache\fR
767 .fi
768 .in -2
769 .sp

771 .LP
772 \fBExample 13 \fRLogging into a PKCS#11 Token
773 .sp
774 .LP
775 The following command shows logging into a PKCS#11 token object and unlocking
776 private keys:

778 .sp

```

```
779 .in +2
780 .nf
781 example# \fBikeadm token login "Sun Metaslot"\fR
782 Enter PIN for PKCS#11 token:
783 ikeadm: PKCS#11 operation successful
784 .fi
785 .in -2
786 .sp

788 .SH EXIT STATUS
795 .sp
789 .LP
790 The following exit values are returned:
791 .sp
792 .ne 2
793 .na
794 \fB\fb0\fR\fR
795 .ad
796 .RS 12n
797 Successful completion.
798 .RE

800 .sp
801 .ne 2
802 .na
803 \fB\fbnon-zero\fR\fR
804 .ad
805 .RS 12n
806 An error occurred. Writes an appropriate error message to standard error.
807 .RE

809 .SH ATTRIBUTES
817 .sp
810 .LP
811 See \fBattributes\fR(5) for descriptions of the following attributes:
812 .sp

814 .sp
815 .TS
816 box;
817 c | c
818 l | l .
819 ATTRIBUTE TYPE ATTRIBUTE VALUE
820 -
821 Interface Stability Not an Interface
822 .TE

824 .SH SEE ALSO
833 .sp
825 .LP
826 \fBin.iked\fR(1M), \fBike.config\fR(4), \fBike.preshared\fR(4),
827 \fBattributes\fR(5), \fBipsec\fR(7P)
828 .sp
829 .LP
830 Schneier, Bruce, \fIApplied Cryptography: Protocols, Algorithms, and Source
831 Code in C\fR, Second Edition, John Wiley & Sons, New York, NY, 1996.
832 .SH NOTES
842 .sp
833 .LP
834 As \fBin.iked\fR can run only in the global zone and exclusive-IP zones, this
835 command is not useful in shared-IP zones.
```

```

*****
4539 Tue Dec 11 09:48:08 2018
new/usr/src/man/man1m/mount_tmpfs.1m
10057 Man page misspellings output particular overridden
Reviewed by: Gerg Mihajly Doma <domag02@gmail.com>
*****
1 \" te
2 .\" Copyright (c) 2003, Sun Microsystems, Inc. All Rights Reserved
3 .\" Copyright 2015 Joyent, Inc.
4 .\" The contents of this file are subject to the terms of the Common Development
5 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
6 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
7 .TH MOUNT_TMPFS 1M \"Mar 18, 2015\"
8 .SH NAME
9 mount_tmpfs \- mount tmpfs file systems
10 .SH SYNOPSIS
11 .LP
12 .nf
13 \fBmount\fR [\fB-F\fR tmpfs] [\fB-o\fR \fIspecific_options\fR] [\fB-O\fR] \fIспе
14 .fi

16 .SH DESCRIPTION
17 .LP
18 \fBtmpfs\fR is a memory based file system which uses kernel resources relating
19 to the \fBVM\fR system and page cache as a file system.
20 .sp
21 .LP
22 \fBmount\fR attaches a \fBtmpfs\fR file system to the file system hierarchy at
23 the pathname location \fImount_point\fR, which must already exist. If
24 \fImount_point\fR has any contents prior to the \fBmount\fR operation, these
25 remain hidden until the file system is once again unmounted. The attributes
26 (mode, owner, and group) of the root of the \fBtmpfs\fR filesystem are
27 inherited from the underlying \fImount_point\fR, provided that those attributes
28 are determinable. If not, the root's attributes are set to their default
29 values. The mode may also be overridden by the \fBmode\fR mount option, which
30 values. The mode may also be overridden by the \fBmode\fR mount option, which
31 takes precedence if set.
32 .sp
33 .LP
34 The \fIspecial\fR argument is usually specified as \fBswap\fR but is in fact
35 disregarded and assumed to be the virtual memory resources within the system.
36 .SH OPTIONS
37 .ne 2
38 .na
39 \fB\F-o\fR \fIspecific_options\fR\fR
40 .ad
41 .RS 23n
42 Specify \fBtmpfs\fR file system specific options in a comma-separated list with
43 no intervening spaces. If invalid options are specified, a warning message is
44 printed and the invalid options are ignored. The following options are
45 available:
46 .sp
47 .ne 2
48 .na
49 \fB\Fbremount\fR\fR
50 .ad
51 .sp .6
52 .RS 19n
53 Remounts a file system with a new size. A size not explicitly
54 set with \fBbremount\fR reverts to no limit.
55 .RE

56 .sp
57 .ne 2
58 .na
59 \fBmode=\fIoctalmode\fR\fR

```

```

60 .ad
61 .RS 19n
62 The \fImode\fR argument controls the permissions of the \fBtmpfs\fR mount
63 point. The argument must be an octal number, of the form passed to
64 \fBchmod\fR(1). Only the access mode, setuid, setgid, and sticky bits (a mask
65 of \fB0777\fR) may be set. If this option is not provided then the default
66 mode behaviour, as described above, applies.
67 .RE

69 .sp
70 .sp
71 .ne 2
72 .na
73 \fBsize=\fIsize\fR\fR
74 .ad
75 .RS 19n
76 The \fIsize\fR argument controls the size of this particular \fBtmpfs\fR file
77 system. If the argument is has a 'k' suffix, the number will be interpreted as
78 a number of kilobytes. An 'm' suffix will be interpreted as a number of
79 megabytes. A 'g' suffix will be interpreted as a number of gigabytes. A '%'
80 suffix will be interpreted as a percentage of the swap space available to the
81 zone. No suffix is interpreted as bytes. In all cases, the actual size of
82 the file system is the number of bytes specified, rounded up to the physical
83 page size of the system.
84 .RE

86 .sp
87 .ne 2
88 .na
89 \fB\Bxattr\fR | \fBnoxattr\fR\fR
90 .ad
91 .RS 19n
92 Allow or disallow the creation and manipulation of extended attributes. The
93 default is \fBxattr\fR. See \fBfsattr\fR(5) for a description of extended
94 attributes.
95 .RE

97 .RE

99 .sp
100 .ne 2
101 .na
102 \fB\F-O\fR\fR
103 .ad
104 .RS 23n
105 Overlay mount. Allow the file system to be mounted over an existing mount
106 point, making the underlying file system inaccessible. If a mount is attempted
107 on a pre-existing mount point without setting this flag, the mount will fail,
108 producing the error: \f(CWdevice busy\fR.
109 .RE

111 .SH FILES
112 .ne 2
113 .na
114 \fB\F/etc/mnttab\fR\fR
115 .ad
116 .RS 15n
117 Table of mounted file systems
118 .RE

120 .SH SEE ALSO
121 .LP
122 \fBmount\fR(1M), \fBmknod\fR(2), \fBmount\fR(2), \fBopen\fR(2),
123 \fBumount\fR(2), \fBmnttab\fR(4), \fBattributes\fR(5), \fBfsattr\fR(5),
124 \fBtmpfs\fR(7FS)
125 .SH NOTES

```

126 .LP
127 If the directory on which a file system is to be mounted is a symbolic link,
128 the file system is mounted on the directory to which the symbolic link refers,
129 rather than on top of the symbolic link itself.

```

*****
13174 Tue Dec 11 09:48:08 2018
new/usr/src/man/man1/nfsstat.1m
10057 Man page misspellings ouput particular overridden
Reviewed by: Gerg Mihály Doma <domag02@gmail.com>
*****
1 \" te
2.\" Copyright (c) 2007, Sun Microsystems, Inc. All Rights Reserved
3.\" The contents of this file are subject to the terms of the Common Development
4.\" See the License for the specific language governing permissions and limitat
5.\" the fields enclosed by brackets \"[]\" replaced with your own identifying info
6.TH NFSSTAT 1M \"Jun 16, 2009\"
7.SH NAME
8 nfsstat \- NFS statistics
9.SH SYNOPSIS
10.LP
11.nf
12 \fBnfsstat\fR [\fB-cnrsza\fR] [\fB-T\fR u | d ] [\fB-v\fR \fIversion\fR] [\fIint
13 .fi

15.LP
16.nf
17 \fBnfsstat\fR \fB-m\fR [\fIpathname\fR]...
18 .fi

20.SH DESCRIPTION
21.sp
21.LP
22 \fBnfsstat\fR displays statistical information about the \fBNFS\fR and
23 \fBRPC\fR (Remote Procedure Call), interfaces to the kernel. It can also be
24 used to reinitialize this information. If no options are given the default is
25 as follows:
26.sp
27.LP
28 \fBnfsstat\fR \fB-csnra\fR
29.sp
30.LP
31 The default displays everything, but reinitializes nothing.
32.SH OPTIONS
33.sp
33.ne 2
34.na
35 \fB\fB-a\fR\fR
36.ad
37.sp .6
38.RS 4n
39 Display \fBNFS_ACL\fR information.
40.RE

42.sp
43.ne 2
44.na
45 \fB\fB-c\fR\fR
46.ad
47.sp .6
48.RS 4n
49 Display client information. Only the client side \fBNFS\fR, \fBRPC\fR, and
50 \fBNFS_ACL\fR information is printed. Can be combined with the \fB-n\fR,
51 \fB-r\fR, and \fB-a\fR options to print client side \fBNFS\fR, \fBRPC\fR, and
52 \fBNFS_ACL\fR information only.
53.RE

55.sp
56.ne 2
57.na
58 \fB\fB-m\fR [\fIpathname...\fR]\fR

```

```

59 .ad
60 .sp .6
61 .RS 4n
62 Display statistics for each \fBNFS\fR mounted file system. If \fIpathname\fR is
63 not specified, displays statistics for all NFS mounted file systems. If
64 \fIpathname\fR is specified, displays statistics for the NFS mounted file
65 systems indicated by \fIpathname\fR.
66 .sp
67 This includes the server name and address, mount flags, current read and write
68 sizes, the retransmission count, the attribute cache timeout values, failover
69 information, and the timers used for dynamic retransmission. The dynamic
70 retransmission timers are displayed only where dynamic retransmission is in
71 use. By default, \fBNFS\fR mounts over the \fBTCP\fR protocols and \fBNFS\fR
72 Version 3 mounts over either \fBTCP\fR or \fBUDP\fR do not use dynamic
73 retransmission.
74 .sp
75 If you specify the \fB-m\fR option, this is the only option that \fBnfsstat\fR
76 uses. If you specify other options with \fB-m\fR, you receive an error message
77 alerting that the \fB-m\fR flag cannot be combined with other options.
78 .RE

80 .sp
81.ne 2
82.na
83 \fB\fB-n\fR\fR
84 .ad
85.sp .6
86.RS 4n
87 Display \fBNFS\fR information. \fBNFS\fR information for both the client and
88 server side are printed. Can be combined with the \fB-c\fR and \fB-s\fR options
89 to print client or server \fBNFS\fR information only.
90.RE

92.sp
93.ne 2
94.na
95 \fB\fB-r\fR\fR
96 .ad
97.sp .6
98.RS 4n
99 Display \fBRPC\fR information.
100.RE

102.sp
103.ne 2
104.na
105 \fB\fB-s\fR\fR
106 .ad
107.sp .6
108.RS 4n
109 Display server information.
110.RE

112.sp
113.ne 2
114.na
115 \fB\fB-T\fR \fBu\fR | \fBd\fR
116 .ad
117.sp .6
118.RS 4n
119 Display a time stamp.
120.sp
121 Specify \fBu\fR for a printed representation of the internal representation of
122 time. See \fBtime\fR(2). Specify \fBd\fR for standard date format. See
123 \fBdate\fR(1).
124 .RE

```

```

126 .sp
127 .ne 2
128 .na
129 \fB\fB-v\fR \fIversion\fR\fR
130 .ad
131 .sp .6
132 .RS 4n
133 Specify which NFS version for which to print statistics. When followed by the
134 optional \fIversion\fR argument, (\fB2\fR|\fB3\fR|\fB4\fR), specifies
135 statistics for that version. By default, prints statistics for all versions.
136 .RE

138 .sp
139 .ne 2
140 .na
141 \fB\fB-z\fR\fR
142 .ad
143 .sp .6
144 .RS 4n
145 Zero (reinitialize) statistics. This option is for use by the super user only,
146 and can be combined with any of the above options to zero particular sets of
147 statistics after printing them.
148 .RE

150 .SH OPERANDS
153 .sp
151 .LP
152 The following operands are supported:
153 .sp
154 .ne 2
155 .na
156 \fB\fIcount\fR\fR
157 .ad
158 .sp .6
159 .RS 4n
160 Display only count reports
161 .RE

163 .sp
164 .ne 2
165 .na
166 \fB\fIinterval\fR\fR
167 .ad
168 .sp .6
169 .RS 4n
170 Report once each interval seconds.
171 .RE

173 .sp
174 .ne 2
175 .na
176 \fB\fIpathname\fR\fR
177 .ad
178 .sp .6
179 .RS 4n
180 Specify the pathname of a file in an NFS mounted file system for which
181 statistics are to be displayed.
182 .RE

184 .SH DISPLAYS
188 .sp
185 .LP
186 The server \fBRPC\fR display includes the following fields:
187 .sp
188 .ne 2

```

```

189 .na
190 \fB\fBbadcalls\fR\fR
191 .ad
192 .sp .6
193 .RS 4n
194 The total number of calls rejected by the \fBRPC\fR layer (the sum of
195 \fBbadlen\fR and \fBxdrcall\fR as defined below).
196 .RE

198 .sp
199 .ne 2
200 .na
201 \fB\fBbadlen\fR\fR
202 .ad
203 .sp .6
204 .RS 4n
205 The number of \fBRPC\fR calls with a length shorter than a minimum-sized
206 \fBRPC\fR call.
207 .RE

209 .sp
210 .ne 2
211 .na
212 \fB\fBcalls\fR\fR
213 .ad
214 .sp .6
215 .RS 4n
216 The total number of \fBRPC\fR calls received.
217 .RE

219 .sp
220 .ne 2
221 .na
222 \fB\fBdupchecks\fR\fR
223 .ad
224 .sp .6
225 .RS 4n
226 The number of \fBRPC\fR calls that looked up in the duplicate request cache.
227 .RE

229 .sp
230 .ne 2
231 .na
232 \fB\fBdupreqs\fR\fR
233 .ad
234 .sp .6
235 .RS 4n
236 The number of \fBRPC\fR calls that were found to be duplicates.
237 .RE

239 .sp
240 .ne 2
241 .na
242 \fB\fBnullrecv\fR\fR
243 .ad
244 .sp .6
245 .RS 4n
246 The number of times an \fBRPC\fR call was not available when it was thought to
247 be received.
248 .RE

250 .sp
251 .ne 2
252 .na
253 \fB\fBxdrcall\fR\fR
254 .ad

```



```

255 .sp .6
256 .RS 4n
257 The number of \fBRPC\fR calls whose header could not be \fBXDR\fR decoded.
258 .RE

260 .sp
261 .LP
262 The server \fBNFS\fR display shows the number of \fBNFS\fR calls received
263 (\fBcalls\fR) and rejected (\fBbadcalls\fR), and the counts and percentages for
264 the various calls that were made.
265 .sp
266 .LP
267 The server \fBNFS_ACL\fR display shows the counts and percentages for the
268 various calls that were made.
269 .sp
270 .LP
271 The client \fBRPC\fR display includes the following fields:
272 .sp
273 .ne 2
274 .na
275 \fB\fBcalls\fR\fR
276 .ad
277 .sp .6
278 .RS 4n
279 The total number of \fBRPC\fR calls made.
280 .RE

282 .sp
283 .ne 2
284 .na
285 \fB\fBbadcalls\fR\fR
286 .ad
287 .sp .6
288 .RS 4n
289 The total number of calls rejected by the \fBRPC\fR layer.
290 .RE

292 .sp
293 .ne 2
294 .na
295 \fB\fBbadverfs\fR\fR
296 .ad
297 .sp .6
298 .RS 4n
299 The number of times the call failed due to a bad verifier in the response.
300 .RE

302 .sp
303 .ne 2
304 .na
305 \fB\fBbadxids\fR\fR
306 .ad
307 .sp .6
308 .RS 4n
309 The number of times a reply from a server was received which did not correspond
310 to any outstanding call.
311 .RE

313 .sp
314 .ne 2
315 .na
316 \fB\fBcantconn\fR\fR
317 .ad
318 .sp .6
319 .RS 4n
320 The number of times the call failed due to a failure to make a connection to

```

```

321 the server.
322 .RE

324 .sp
325 .ne 2
326 .na
327 \fB\fBcantsend\fR\fR
328 .ad
329 .sp .6
330 .RS 4n
331 The number of times a client was unable to send an \fBRPC\fR request over a
332 connectionless transport when it tried to do so.
333 .RE

335 .sp
336 .ne 2
337 .na
338 \fB\fBinterrupts\fR\fR
339 .ad
340 .sp .6
341 .RS 4n
342 The number of times the call was interrupted by a signal before completing.
343 .RE

345 .sp
346 .ne 2
347 .na
348 \fB\fBnewcreds\fR\fR
349 .ad
350 .sp .6
351 .RS 4n
352 The number of times authentication information had to be refreshed.
353 .RE

355 .sp
356 .ne 2
357 .na
358 \fB\fBnomem\fR\fR
359 .ad
360 .sp .6
361 .RS 4n
362 The number of times the call failed due to a failure to allocate memory.
363 .RE

365 .sp
366 .ne 2
367 .na
368 \fB\fBretrans\fR\fR
369 .ad
370 .sp .6
371 .RS 4n
372 The number of times a call had to be retransmitted due to a timeout while
373 waiting for a reply from the server. Applicable only to \fBRPC\fR over
374 connection-less transports.
375 .RE

377 .sp
378 .ne 2
379 .na
380 \fB\fBtimeouts\fR\fR
381 .ad
382 .sp .6
383 .RS 4n
384 The number of times a call timed out while waiting for a reply from the server.
385 .RE

```

```

387 .sp
388 .ne 2
389 .na
390 \fB\fBtimers\fR\fR
391 .ad
392 .sp .6
393 .RS 4n
394 The number of times the calculated time-out value was greater than or equal to
395 the minimum specified time-out value for a call.
396 .RE

398 .sp
399 .LP
400 The client \fBNFS\fR display shows the number of calls sent and rejected, as
401 well as the number of times a \fBCLIENT\fR handle was received (\fBclgets\fR),
402 the number of times the \fBCLIENT\fR handle cache had no unused entries
403 (\fBcltoomany\fR), as well as a count of the various calls and their respective
404 percentages.
405 .sp
406 .LP
407 The client \fBNFS_ACL\fR display shows the counts and percentages for the
408 various calls that were made.
409 .sp
410 .LP
411 The \fB-m\fR option includes information about mount flags set by mount
412 options, mount flags internal to the system, and other mount information. See
413 \fBmount_nfs\fR(1M).
414 .sp
415 .LP
416 The following mount flags are set by mount options:
417 .sp
418 .ne 2
419 .na
420 \fB\fBgrpid\fR\fR
421 .ad
422 .sp .6
423 .RS 4n
424 System V group id inheritance.
425 .RE

427 .sp
428 .ne 2
429 .na
430 \fB\fBhard\fR\fR
431 .ad
432 .sp .6
433 .RS 4n
434 Hard mount.
435 .RE

437 .sp
438 .ne 2
439 .na
440 \fB\fBintr\fR\fR
441 .ad
442 .sp .6
443 .RS 4n
444 Interrupts allowed on hard mount.
445 .RE

447 .sp
448 .ne 2
449 .na
450 \fB\fBlllock\fR\fR
451 .ad
452 .sp .6

```

```

453 .RS 4n
454 Local locking being used (no lock manager).
455 .RE

457 .sp
458 .ne 2
459 .na
460 \fB\fBnoac\fR\fR
461 .ad
462 .sp .6
463 .RS 4n
464 Client is not caching attributes.
465 .RE

467 .sp
468 .ne 2
469 .na
470 \fB\fBnointr\fR\fR
471 .ad
472 .sp .6
473 .RS 4n
474 No interrupts allowed on hard mount.
475 .RE

477 .sp
478 .ne 2
479 .na
480 \fB\fBnocto\fR\fR
481 .ad
482 .sp .6
483 .RS 4n
484 No close-to-open consistency.
485 .RE

487 .sp
488 .ne 2
489 .na
490 \fB\fBrettrans\fR\fR
491 .ad
492 .sp .6
493 .RS 4n
494 \fBNFS\fR retransmissions.
495 .RE

497 .sp
498 .ne 2
499 .na
500 \fB\fBpctimesync\fR\fR
501 .ad
502 .sp .6
503 .RS 4n
504 \fB\RPC\fR time sync.
505 .RE

507 .sp
508 .ne 2
509 .na
510 \fB\fBbrsize\fR\fR
511 .ad
512 .sp .6
513 .RS 4n
514 Read buffer size in bytes.
515 .RE

517 .sp
518 .ne 2

```

```

519 .na
520 \fB\fBsec\fR\fR
521 .ad
522 .sp .6
523 .RS 4n
524 \fBsec\fR has one of the following values:
525 .sp
526 .ne 2
527 .na
528 \fB\fBdh\fR\fR
529 .ad
530 .sp .6
531 .RS 4n
532 \fBdes\fR-style authentication (encrypted timestamps).
533 .RE

535 .sp
536 .ne 2
537 .na
538 \fB\fBkrb5\fR\fR
539 .ad
540 .sp .6
541 .RS 4n
542 \fBkerberos v5\fR-style authentication.
543 .RE

545 .sp
546 .ne 2
547 .na
548 \fB\fBkrb5i\fR\fR
549 .ad
550 .sp .6
551 .RS 4n
552 \fBkerberos v5\fR-style authentication with integrity.
553 .RE

555 .sp
556 .ne 2
557 .na
558 \fB\fBkrb5p\fR\fR
559 .ad
560 .sp .6
561 .RS 4n
562 \fBkerberos v5\fR-style authentication with privacy.
563 .RE

565 .sp
566 .ne 2
567 .na
568 \fB\fBnone\fR\fR
569 .ad
570 .sp .6
571 .RS 4n
572 No authentication.
573 .RE

575 .sp
576 .ne 2
577 .na
578 \fB\fBshort\fR\fR
579 .ad
580 .sp .6
581 .RS 4n
582 Short hand UNIX-style authentication.
583 .RE

```

```

585 .sp
586 .ne 2
587 .na
588 \fB\fBsys\fR\fR
589 .ad
590 .sp .6
591 .RS 4n
592 UNIX-style authentication (UID, GID).
593 .RE

595 .RE

597 .sp
598 .ne 2
599 .na
600 \fB\fBsoft\fR\fR
601 .ad
602 .sp .6
603 .RS 4n
604 Soft mount.
605 .RE

607 .sp
608 .ne 2
609 .na
610 \fB\fBtimeo\fR\fR
611 .ad
612 .sp .6
613 .RS 4n
614 Initial \fBNFS\fR timeout, in tenths of a second.
615 .RE

617 .sp
618 .ne 2
619 .na
620 \fB\fBwsize\fR\fR
621 .ad
622 .sp .6
623 .RS 4n
624 Write buffer size in bytes.
625 .RE

627 .sp
628 .LP
629 The following mount flags are internal to the system:
630 .sp
631 .ne 2
632 .na
633 \fB\fBacl\fR\fR
634 .ad
635 .sp .6
636 .RS 4n
637 Server supports \fBNFS_ACL\fR.
638 .RE

640 .sp
641 .ne 2
642 .na
643 \fB\fBdown\fR\fR
644 .ad
645 .sp .6
646 .RS 4n
647 Server is down.
648 .RE

650 .sp

```

```

651 .ne 2
652 .na
653 \fB\fBdynamic\fR\fR
654 .ad
655 .sp .6
656 .RS 4n
657 Dynamic transfer size adjustment.
658 .RE

660 .sp
661 .ne 2
662 .na
663 \fB\fBblink\fR\fR
664 .ad
665 .sp .6
666 .RS 4n
667 Server supports links.
668 .RE

670 .sp
671 .ne 2
672 .na
673 \fB\fBmirrormount\fR\fR
674 .ad
675 .sp .6
676 .RS 4n
677 Mounted automatically by means of the \fBmirrormount\fR mechanism.
678 .RE

680 .sp
681 .ne 2
682 .na
683 \fB\fBprinted\fR\fR
684 .ad
685 .sp .6
686 .RS 4n
687 "Not responding" message printed.
688 .RE

690 .sp
691 .ne 2
692 .na
693 \fB\fBbreaddir\fR\fR
694 .ad
695 .sp .6
696 .RS 4n
697 Use \fBbreaddir\fR instead of \fBbreaddirplus\fR.
698 .RE

700 .sp
701 .ne 2
702 .na
703 \fB\fBsymlink\fR\fR
704 .ad
705 .sp .6
706 .RS 4n
707 Server supports symbolic links.
708 .RE

710 .sp
711 .LP
712 The following flags relate to additional mount information:
713 .sp
714 .ne 2
715 .na
716 \fB\fBproto\fR\fR

```

```

717 .ad
718 .sp .6
719 .RS 4n
720 Protocol.
721 .RE

723 .sp
724 .ne 2
725 .na
726 \fB\fBvers\fR\fR
727 .ad
728 .sp .6
729 .RS 4n
730 \fB\fBNFS\fR version.
731 .RE

733 .sp
734 .LP
735 The \fB-m\fR option also provides attribute cache timeout values. The following
736 fields in \fB-m\fR output provide timeout values for attribute cache:
740 fields in \fB-m\fR output provide timeout values for attribute cache:
737 .sp
738 .ne 2
739 .na
740 \fB\fBacdirmax\fR\fR
741 .ad
742 .sp .6
743 .RS 4n
744 Maximum seconds to hold cached directory attributes.
745 .RE

747 .sp
748 .ne 2
749 .na
750 \fB\fBacdirmin\fR\fR
751 .ad
752 .sp .6
753 .RS 4n
754 Minimum seconds to hold cached directory attributes.
755 .RE

757 .sp
758 .ne 2
759 .na
760 \fB\fBacregmax\fR\fR
761 .ad
762 .sp .6
763 .RS 4n
764 Maximum seconds to hold cached file attributes.
765 .RE

767 .sp
768 .ne 2
769 .na
770 \fB\fBacregmin\fR\fR
771 .ad
772 .sp .6
773 .RS 4n
774 Minimum seconds to hold cached file attributes.
775 .RE

777 .sp
778 .LP
779 The following fields in \fB-m\fR output provide failover information:
780 .sp
781 .ne 2

```

```

782 .na
783 \fB\fBcurrserver\fR\fR
784 .ad
785 .sp .6
786 .RS 4n
787 Which server is currently providing \fBNFS\fR service. See the \fI\fR for
788 additional details.
789 .RE

791 .sp
792 .ne 2
793 .na
794 \fB\fBfailover\fR\fR
795 .ad
796 .sp .6
797 .RS 4n
798 How many times a new server has been selected.
799 .RE

801 .sp
802 .ne 2
803 .na
804 \fB\fBnoresponse\fR\fR
805 .ad
806 .sp .6
807 .RS 4n
808 How many times servers have failed to respond.
809 .RE

811 .sp
812 .ne 2
813 .na
814 \fB\fBremap\fR\fR
815 .ad
816 .sp .6
817 .RS 4n
818 How many times files have been re-evaluated to the new server.
819 .RE

821 .sp
822 .LP
823 The fields in \fB-m\fR output shown below provide information on dynamic
824 retransmissions. These items are displayed only where dynamic retransmission is
825 in use.
826 .sp
827 .ne 2
828 .na
829 \fB\fBcur\fR\fR
830 .ad
831 .sp .6
832 .RS 4n
833 Current backed-off retransmission value, in milliseconds.
834 .RE

836 .sp
837 .ne 2
838 .na
839 \fB\fBdev\fR\fR
840 .ad
841 .sp .6
842 .RS 4n
843 Estimated deviation, in milliseconds.
844 .RE

846 .sp
847 .ne 2

```

```

848 .na
849 \fB\fBsrvt\fR\fR
850 .ad
851 .sp .6
852 .RS 4n
853 The value for the smoothed round-trip time, in milliseconds.
854 .RE

856 .SH EXIT STATUS
861 .sp
857 .LP
858 The following exit values are returned:
859 .sp
860 .ne 2
861 .na
862 \fB\fB0\fR\fR
863 .ad
864 .sp .6
865 .RS 4n
866 Successful completion.
867 .RE

869 .sp
870 .ne 2
871 .na
872 \fB\fB>0\fR\fR
873 .ad
874 .sp .6
875 .RS 4n
876 An error occurred.
877 .RE

879 .SH SEE ALSO
885 .sp
880 .LP
881 \fBmount_nfs\fR(1M), \fBattributes\fR(5)
882 .sp
883 .LP
884 \fI\fR
885 .sp
886 .LP
887 \fI\fR

```

13599 Tue Dec 11 09:48:09 2018

new/usr/src/man/man1m/smbadm.1m

10057 Man page misspellings output particular overridden

Reviewed by: Gerg Mihály Doma <domag02@gmail.com>

```

1 "\" te
2.\" Copyright 2014 Nexenta Systems, Inc. All rights reserved.
3.\" Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved.
4.\" The contents of this file are subject to the terms of the Common Development
5.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
6.\" When distributing Covered Code, include this CDDL HEADER in each file and in
7.TH SMBADM 1M "April 9, 2016"
8.SH NAME
9 smbadm \- configure and manage CIFS local groups and users, and manage domain
10 membership
11.SH SYNOPSIS
12.LP
13.nf
14 \fBsmbadm add-member\fR -m \fImember\fR [[-m \fImember\fR] \&.\|.|.] \fIgroup\f
15 .fi

17.LP
18.nf
19 \fBsmbadm create\fR [-d \fIdescription\fR] \fIgroup\fR
20 .fi

22.LP
23.nf
24 \fBsmbadm delete\fR \fIgroup\fR
25 .fi

27.LP
28.nf
29 \fBsmbadm disable-user\fR \fIusername\fR
30 .fi

32.LP
33.nf
34 \fBsmbadm enable-user\fR \fIusername\fR
35 .fi

37.LP
38.nf
39 \fBsmbadm get\fR [[-p \fIproperty\fR] \&.\|.|.] \fIgroup\fR
40 .fi

42.LP
43.nf
44 \fBsmbadm join\fR [-y] -u \fIusername\fR \fIdomain\fR
45 .fi

47.LP
48.nf
49 \fBsmbadm join\fR [-y] -w \fIworkgroup\fR
50 .fi

52.LP
53.nf
54 \fBsmbadm list\fR
55 .fi

57.LP
58.nf
59 \fBsmbadm lookup\fR \fIaccount-name\fR [\fIaccount-name\fR [\&.\|.|.]]
60 .fi

```

```

62.LP
63.nf
64 \fBsmbadm remove-member\fR -m \fImember\fR [[-m \fImember\fR] \&.\|.|.] \fIgroup
65 .fi

67.LP
68.nf
69 \fBsmbadm rename\fR \fIgroup\fR \fInew-group\fR
70 .fi

72.LP
73.nf
74 \fBsmbadm set\fR -p \fIproperty\fR=\fIvalue\fR [[-p \fIproperty\fR=\fIvalue\fR]
75 .fi

77.LP
78.nf
79 \fBsmbadm show\fR [-m] [-p] [\fIgroup\fR]
80 .fi

82.SH DESCRIPTION
83.LP
84 The \fBsmbadm\fR command is used to configure \fBCIFS\fR local groups and to
85 manage domain membership. You can also use the \fBsmbadm\fR command to enable
86 or disable SMB password generation for individual local users.
87.sp
88.LP
89 \fBCIFS\fR local groups can be used when Windows accounts must be members of
90 some local groups and when Windows style privileges must be granted. Solaris
91 local groups cannot provide these functions.
92.sp
93.LP
94 There are two types of local groups: user defined and built-in. Built-in local
95 groups are predefined local groups to support common administration tasks.
96.sp
97.LP
98 In order to provide proper identity mapping between \fBCIFS\fR local groups and
99 Solaris groups, a \fBCIFS\fR local group must have a corresponding Solaris
100 group. This requirement has two consequences: first, the group name must
101 conform to the intersection of the Windows and Solaris group name rules. Thus,
102 a \fBCIFS\fR local group name can be up to eight (8) characters long and
103 contain only lowercase characters and numbers. Second, a Solaris local group
104 has to be created before a \fBCIFS\fR local group can be created.
105.sp
106.LP
107 Built-in groups are standard Windows groups and are predefined by the
108 \fBCIFS\fR service. The built-in groups cannot be added, removed, or renamed,
109 and these groups do not follow the \fBCIFS\fR local group naming conventions.
110.sp
111.LP
112 When the \fBCIFS\fR server is started, the following built-in groups are
113 available:
114.sp
115.ne 2
116.na
117 \fBAdministrators\fR
118.ad
119.sp .6
120.RS 4n
121 Group members can administer the system.
122.RE

124.sp
125.ne 2
126.na

```

```

127 \fBBackup Operators\fR
128 .ad
129 .sp .6
130 .RS 4n
131 Group members can bypass file access controls to back up and restore files.
132 .RE

134 .sp
135 .ne 2
136 .na
137 \fBPower Users\fR
138 .ad
139 .sp .6
140 .RS 4n
141 Group members can share directories.
142 .RE

144 .sp
145 .LP
146 Solaris local users must have an SMB password for authentication and to gain
147 access to CIFS resources. This password is created by using the \fBpasswd\fR(1)
148 command when the \fBpam_smb_password\fR module is added to the system's PAM
149 configuration. See the \fBpam_smb_password\fR(5) man page.
150 .sp
151 .LP
152 The \fBdisable-user\fR and \fBenable-user\fR subcommands control SMB
153 password-generation for a specified local user. When disabled, the user is
154 prevented from connecting to the Solaris CIFS service. By default, SMB
155 password-generation is enabled for all local users.
156 .sp
157 .LP
158 To reenable a disabled user, you must use the \fBenable-user\fR subcommand and
159 then reset the user's password by using the \fBpasswd\fR command. The
160 \fBpam_smb_password.so.1\fR module must be added to the system's PAM
161 configuration to generate an SMB password.
162 .SS "Escaping Backslash Character"
163 .LP
164 For the \fBadd-member\fR, \fBremove-member\fR, and \fBjoin\fR (with \fB-u\fR)
165 subcommands, the backslash character (\fB\e\fR) is a valid separator between
166 member or user names and domain names. The backslash character is a shell
167 special character and must be quoted. For example, you might escape the
168 backslash character with another backslash character:
169 \fB\domain\fR\fB\e\e\fR\fIusername\fR. For more information about handling shell
170 special characters, see the man page for your shell.
171 .SH OPERANDS
172 .LP
173 The \fBsmbadm\fR command uses the following operands:
174 .sp
175 .ne 2
176 .na
177 \fB\domain\fR
178 .ad
179 .sp .6
180 .RS 4n
181 Specifies the name of an existing Windows domain to join.
182 .RE

184 .sp
185 .ne 2
186 .na
187 \fB\group\fR
188 .ad
189 .sp .6
190 .RS 4n
191 Specifies the name of the \fBCIFS\fR local group.
192 .RE

```

```

194 .sp
195 .ne 2
196 .na
197 \fB\username\fR
198 .ad
199 .sp .6
200 .RS 4n
201 Specifies the name of a Solaris local user.
202 .RE

204 .SH SUBCOMMANDS
205 .LP
206 The \fBsmbadm\fR command includes these subcommands:
207 .sp
208 .ne 2
209 .na
210 \fB\add-member\fR -m \fImember\fR [[-m \fImember\fR] \&.\|.|.]
211 \fB\group\fR
212 .ad
213 .sp .6
214 .RS 4n
215 Adds the specified member to the specified \fBCIFS\fR local group. The \fB-m\fR
216 \fImember\fR option specifies the name of a \fBCIFS\fR local group member. The
217 member name must include an existing user name and an optional domain name.
218 .sp
219 Specify the member name in either of the following formats:
220 .sp
221 .in +2
222 .nf
223 [\fIdomain\fR\e]\fIusername\fR
224 [\fIdomain\fR/]\fIusername\fR
225 .fi
226 .in -2
227 .sp

229 For example, a valid member name might be \fBsales\eterry\fR or
230 \fBsales/terry\fR, where \fBsales\fR is the Windows domain name and \fBterry\fR
231 is the name of a user in the \fBsales\fR domain.
232 .RE

234 .sp
235 .ne 2
236 .na
237 \fB\create\fR [\fB-d\fR \fBdescription\fR] \fB\group\fR
238 .ad
239 .sp .6
240 .RS 4n
241 Creates a \fBCIFS\fR local group with the specified name. You can optionally
242 specify a description of the group by using the \fB-d\fR option.
243 .RE

245 .sp
246 .ne 2
247 .na
248 \fB\delete\fR \fB\group\fR
249 .ad
250 .sp .6
251 .RS 4n
252 Deletes the specified \fBCIFS\fR local group. The built-in groups cannot be
253 deleted.
254 .RE

256 .sp
257 .ne 2
258 .na

```

```

259 \fB\fBdisable\fR \fIusername\fR\fR
260 .ad
261 .sp .6
262 .RS 4n
263 Disables SMB password-generation capabilities for the specified local user. A
264 disabled local user is prevented from accessing the system by means of the CIFS
265 service. When a local user account is disabled, you cannot use the \fBpasswd\fR
266 command to modify the user's SMB password until the user account is reenabled.
267 .RE

269 .sp
270 .ne 2
271 .na
272 \fB\fBenable\fR \fIusername\fR\fR
273 .ad
274 .sp .6
275 .RS 4n
276 Enables SMB password-generation capabilities for the specified local user.
277 After the password-generation capabilities are reenabled, you must use the
278 \fBpasswd\fR command to generate the SMB password for the local user before he
279 can connect to the CIFS service.
280 .sp
281 The \fBpasswd\fR command manages both the Solaris password and SMB password for
282 this user if the \fBpam_smb_passwd\fR module has been added to the system's PAM
283 configuration.
284 .RE

286 .sp
287 .ne 2
288 .na
289 \fB\fBget\fR [[\fB-p\fR \fIproperty\fR=\fIvalue\fR] \&.\|.|\|.] \fIgroup\fR\fR
290 .ad
291 .sp .6
292 .RS 4n
293 Retrieves property values for the specified group. If no property is specified,
294 all property values are shown.
295 .RE

297 .sp
298 .ne 2
299 .na
300 \fB\fBjoin\fR \fB[-y] -u\fR \fIusername\fR \fIdomain\fR\fR
301 .ad
302 .sp .6
303 .RS 4n
304 Joins a Windows domain or a workgroup.
305 .sp
306 The default mode for the \fBCIFS\fR service is workgroup mode, which uses the
307 default workgroup name, \fBWORKGROUP\fR.
308 .sp
309 An authenticated user account is required to join a domain, so you must specify
310 the Windows administrative user name with the \fB-u\fR option. If the password
311 is not specified on the command line, the user is prompted for it. This user
312 should be the domain administrator or any user who has administrative
313 privileges for the target domain.
314 .sp
315 \fIusername\fR and \fIdomain\fR can be entered in any of the following formats:
316 .sp
317 .in +2
318 .nf
319 \fIusername\fR[+\fIpassword\fR] \fIdomain\fR
320 \fIdomain\fR\e\fIusername\fR[+\fIpassword\fR]
321 \fIdomain\fR/\fIusername\fR[+\fIpassword\fR]
322 \fIusername\fR@\fIdomain\fR
323 .fi
324 .in -2

```

```

325 .sp

327 \&...where \fIdomain\fR can be the NetBIOS or DNS domain name.
328 .sp
329 If a machine trust account for the system already exists on a domain
330 controller, any authenticated user account can be used when joining the domain.
331 However, if the machine trust account does \fBnot\fR already exist, an account
332 that has administrative privileges on the domain is required to join the
333 domain.
334 Specifying \fB-y\fR will bypass the smb service restart prompt.
335 .RE

337 .sp
338 .ne 2
339 .na
340 \fB\fBjoin\fR \fB[-y] -w\fR \fIworkgroup\fR\fR
341 .ad
342 .sp .6
343 .RS 4n
344 Joins a Windows domain or a workgroup.
345 .sp
346 The \fB-w\fR \fIworkgroup\fR option specifies the name of the workgroup to join
347 when using the \fBjoin\fR subcommand.
348 Specifying \fB-y\fR will bypass the smb service restart prompt.
349 .RE

351 .sp
352 .ne 2
353 .na
354 \fB\fBlist\fR\fR
355 .ad
356 .sp .6
357 .RS 4n
358 Shows information about the current workgroup or domain. The information
359 typically includes the workgroup name or the primary domain name. When in
360 domain mode, the information includes domain controller names and trusted
361 domain names.
362 .sp
363 Each entry in the output is identified by one of the following tags:
364 Each entry in the output is identified by one of the following tags:
365 .sp
366 .na
367 \fB\fB- [*] -\fR\fR
368 .ad
369 .RS 11n
370 Primary domain
371 .RE

373 .sp
374 .ne 2
375 .na
376 \fB\fB- [.] -\fR\fR
377 .ad
378 .RS 11n
379 Local domain
380 .RE

382 .sp
383 .ne 2
384 .na
385 \fB\fB- [-] -\fR\fR
386 .ad
387 .RS 11n
388 Other domains
389 .RE

```



```

391 .sp
392 .ne 2
393 .na
394 \fB\fB- [+ ] -\fR\fR
395 .ad
396 .RS 11n
397 Selected domain controller
398 .RE

400 .RE

402 .sp
403 .ne 2
404 .na
405 \fB\fBlookup\fR\fR \fIaccount-name\fR [\fIaccount-name\fR [\&.\|.|.]]

407 .ad
408 .sp .6
409 .RS 4n
410 Lookup the SID for the given \fIaccount-name\fR, or lookup the
411 \fIaccount-name\fR for the given SID. This subcommand is
412 primarily for diagnostic use, to confirm whether the server
413 can lookup domain accounts and/or SIDs.
414 .RE

416 .sp
417 .ne 2
418 .na
419 \fB\fBremove-member\fR -m \fImember\fR [[-m \fImember\fR] \&.\|.|.]
420 \fIgroup\fR\fR
421 .ad
422 .sp .6
423 .RS 4n
424 Removes the specified member from the specified \fBCIFS\fR local group. The
425 \fB-m\fR \fImember\fR option specifies the name of a \fBCIFS\fR local group
426 member. The member name must include an existing user name and an optional
427 domain name.
428 .sp
429 Specify the member name in either of the following formats:
430 .sp
431 .in +2
432 .nf
433 [\fIdomain\fR\e]\fIusername\fR
434 [\fIdomain\fR/]\fIusername\fR
435 .fi
436 .in -2
437 .sp

439 For example, a valid member name might be \fBsales\eterry\fR or
440 \fBsales/terry\fR, where \fBsales\fR is the Windows domain name and \fBterry\fR
441 is the name of a user in the \fBsales\fR domain.
442 .RE

444 .sp
445 .ne 2
446 .na
447 \fB\fBrename\fR \fIgroup\fR \fInew-group\fR\fR
448 .ad
449 .sp .6
450 .RS 4n
451 Renames the specified \fBCIFS\fR local group. The group must already exist. The
452 built-in groups cannot be renamed.
453 .RE

455 .sp

```

```

456 .ne 2
457 .na
458 \fB\fBset\fR \fB-p\fR \fIproperty\fR=\fIvalue\fR [[\fB-p\fR
459 \fIproperty\fR=\fIvalue\fR] \&.\|.|.] \fIgroup\fR\fR
460 .ad
461 .sp .6
462 .RS 4n
463 Sets configuration properties for a \fBCIFS\fR local group. The description and
464 the privileges for the built-in groups cannot be changed.
465 .sp
466 The \fB-p\fR \fIproperty\fR=\fB=\fR\fIvalue\fR option specifies the list of
467 properties to be set on the specified group.
468 .sp
469 The group-related properties are as follows:
470 .sp
471 .ne 2
472 .na
473 \fB\fBbackup=[on|off]\fR\fR
474 .ad
475 .sp .6
476 .RS 4n
477 Specifies whether members of the \fBCIFS\fR local group can bypass file access
478 controls to back up file system objects.
479 .RE

481 .sp
482 .ne 2
483 .na
484 \fB\fBdescription=\fR\fIdescription-text\fR\fR
485 .ad
486 .sp .6
487 .RS 4n
488 Specifies a text description for the \fBCIFS\fR local group.
489 .RE

491 .sp
492 .ne 2
493 .na
494 \fB\fBrestore=[on|off]\fR\fR
495 .ad
496 .sp .6
497 .RS 4n
498 Specifies whether members of the \fBCIFS\fR local group can bypass file access
499 controls to restore file system objects.
500 .RE

502 .sp
503 .ne 2
504 .na
505 \fB\fBtake-ownership=[on|off]\fR\fR
506 .ad
507 .sp .6
508 .RS 4n
509 Specifies whether members of the \fBCIFS\fR local group can take ownership of
510 file system objects.
511 .RE

513 .RE

515 .sp
516 .ne 2
517 .na
518 \fB\fBshow\fR [\fB-m\fR] [\fB-p\fR] [\fIgroup\fR]\fR
519 .ad
520 .sp .6
521 .RS 4n

```

```
522 Shows information about the specified \fBCIFS\fR local group or groups. If no
523 group is specified, information is shown for all groups. If the \fB-m\fR option
524 is specified, the group members are also shown. If the \fB-p\fR option is
525 specified, the group privileges are also shown.
526 .RE

528 .SH EXIT STATUS
529 .LP
530 The following exit values are returned:
531 .sp
532 .ne 2
533 .na
534 \fB0\fR
535 .ad
536 .RS 13n
537 Successful completion.
538 .RE

540 .sp
541 .ne 2
542 .na
543 \fB>0\fR
544 .ad
545 .RS 13n
546 An error occurred.
547 .RE

549 .SH ATTRIBUTES
550 .LP
551 See the \fBattributes\fR(5) man page for descriptions of the following
552 attributes:
553 .sp

555 .sp
556 .TS
557 box;
558 c | c
559 l | l .
560 ATTRIBUTE TYPE ATTRIBUTE VALUE
561 -
562 Utility Name and Options Uncommitted
563 -
564 Utility Output Format Not-An-Interface
565 -
566 \fBsmbadm join\fR Obsolete
567 .TE

569 .SH SEE ALSO
570 .LP
571 \fBpasswd\fR(1), \fBgroupadd\fR(1M), \fBbidmap\fR(1M), \fBbidmapd\fR(1M),
572 \fBkclient\fR(1M), \fBshare\fR(1M), \fBsharectl\fR(1M), \fBsharemgr\fR(1M),
573 \fBsmbd\fR(1M), \fB smbstat\fR(1M), \fB smb\fR(4), \fBsmbautohome\fR(4),
574 \fBattributes\fR(5), \fBpam_smb_passwd\fR(5), \fBsmf\fR(5)
```

```

*****
4185 Tue Dec 11 09:48:09 2018
new/usr/src/man/man1m/uucleanup.lm
10057 Man page misspellings ouput particular overridden
Reviewed by: Gerg Mihajly Doma <domag02@gmail.com>
*****
1 \" te
2 .\" Copyright (c) 1996, Sun Microsystems, Inc. All Rights Reserved.
3 .\" Copyright 1989 AT&T
4 .\" The contents of this file are subject to the terms of the Common Development
5 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
6 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
7 .TH UUCLEANUP 1M \"May 19, 1993\"
8 .SH NAME
9 uucleanup \- uucp spool directory clean-up
10 .SH SYNOPSIS
11 .LP
12 .nf
13 \fB/usr/lib/uucp/uucleanup\fR [\fB-C\fR\fR\fR] [\fB-D\fR\fR\fR] [\fB-m\fR
14 [\fB-o\fR\fR\fR] [\fB-s\fR\fR\fR] [\fB-W\fR\fR\fR] [\fB-x\fR
15 .fi

17 .SH DESCRIPTION
18 .sp
19 \fBuucleanup\fR will scan the spool directories for old files and take
20 appropriate action to remove them in a useful way:
21 .RS +4
22 .TP
23 .ie t \(\bu
24 .el o
25 Inform the requester of send/receive requests for systems that can not be
26 reached.
27 .RE
28 .RS +4
29 .TP
30 .ie t \(\bu
31 .el o
32 Return undeliverable mail to the sender.
33 .RE
34 .RS +4
35 .TP
36 .ie t \(\bu
37 .el o
38 Deliver \fBbrnews\fR files addressed to the local system.
39 .RE
40 .RS +4
41 .TP
42 .ie t \(\bu
43 .el o
44 Remove all other files.
45 .RE
46 .sp
47 .LP
48 In addition, there is a provision to warn users of requests that have been
49 waiting for a given number of days (default 1 day). Note: \fBuucleanup\fR will
50 process as if all option \fBtimes\fR were specified to the default values
51 unless \fBtime\fR is specifically set.
52 .sp
53 .LP
54 This program is typically started by the shell \fBuudemon.cleanup\fR, which
55 should be started by \fBcron\fR(1M).
56 .SH OPTIONS
57 .sp
58 .ne 2
59 .na

```

```

59 \fB\fB-C\fR\fR\fR
60 .ad
61 .RS 17n
62 Remove any \fBC.\fR files greater or equal to \fBtime\fR days old and send
63 appropriate information to the requester (default 7 days).
64 .RE

66 .sp
67 .ne 2
68 .na
69 \fB\fB-D\fR\fR\fR
70 .ad
71 .RS 17n
72 Remove any \fBD.\fR files greater or equal to \fBtime\fR days old, make an
73 attempt to deliver mail messages, and execute \fBbrnews\fR when appropriate
74 (default 7 days).
75 .RE

77 .sp
78 .ne 2
79 .na
80 \fB\fB-m\fR\fR\fR
81 .ad
82 .RS 17n
83 Include \fBstring\fR in the warning message generated by the \fB-W\fR option.
84 The default line is "See your local administrator to locate the problem".
85 .RE

87 .sp
88 .ne 2
89 .na
90 \fB\fB-o\fR\fR\fR
91 .ad
92 .RS 17n
93 Delete other files whose age is more than \fBtime\fR days (default 2 days).
94 .RE

96 .sp
97 .ne 2
98 .na
99 \fB\fB-s\fR\fR\fR
100 .ad
101 .RS 17n
102 Execute for \fBsystem\fR spool directory only.
103 .RE

105 .sp
106 .ne 2
107 .na
108 \fB\fB-W\fR\fR\fR
109 .ad
110 .RS 17n
111 Any \fBC.\fR files equal to \fBtime\fR days old will cause a mail message to be
112 sent to the requester warning about the delay in contacting the remote. The
113 message includes the \fBJOBID\fR, and in the case of mail, the mail message.
114 The administrator may include a message line telling whom to call to check the
115 problem (\fB-m\fR option) (default 1 day).
116 .RE

118 .sp
119 .ne 2
120 .na
121 \fB\fB-x\fR\fR\fR
122 .ad
123 .RS 17n
124 Produce debugging output on standard output. \fBdebug-level\fR is a single digit

```

126 Produce debugging output on standard output. `\fIdebug-level\fR` is a single digit
125 between 0 and 9; higher numbers give more detailed debugging information. (This
126 option may not be available on all systems.)
127 .RE

129 .sp
130 .ne 2
131 .na
132 `\fB\fB-X\fR\fBtime\fR\fR`
133 .ad
134 .RS 17n
135 Any `\fBX.\fR` files greater or equal to `\fBtime\fR` days old will be removed. The
136 `\fBD.\fR` files are probably not present (if they were, the `\fBX.\fR` could get
137 executed). But if there are `\fBD.\fR` files, they will be taken care of by D.
138 processing (default 2 days).
139 .RE

141 .SH FILES

144 .sp
142 .ne 2
143 .na
144 `\fB\fB/usr/lib/uucp\fR\fR`
145 .ad
146 .RS 19n
147 directory with commands used by `\fBuucleanup\fR` internally
148 .RE

150 .sp
151 .ne 2
152 .na
153 `\fB\fB/var/spool/uucp\fR\fR`
154 .ad
155 .RS 19n
156 spool directory
157 .RE

159 .SH SEE ALSO

163 .sp
160 .LP
161 `\fBuucp\fR(1C)`, `\fBuux\fR(1C)`, `\fBcron\fR(1M)`, `\fBattributes\fR(5)`

```

*****
6144 Tue Dec 11 09:48:09 2018
new/usr/src/man/man3lgrp/lgrp_affinity_get.3lgrp
10057 Man page misspellings output particular overridden
Reviewed by: Gerg Mihály Doma <domag02@gmail.com>
*****
1  \" te
2 .\" Copyright (c) 2003, Sun Microsystems, Inc. All Rights Reserved.
3 .\" The contents of this file are subject to the terms of the Common Development
4 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
5 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
6  .TH LGRP_AFFINITY_GET 3LGRP \"Apr 16, 2003\"
7  .SH NAME
8  lgrp_affinity_get, lgrp_affinity_set \- get of set lgroup affinity
9  .SH SYNOPSIS
10 .LP
11 .nf
12 cc [ \fIflag \&.\|.|\.\fR ] \fIfile\fR\&.\|.|\.\fR \fB-llgrp\fR [ \fIlibrary \&.\|.
13 #include <sys/lgrp_user.h>

15 \fBlgrp_affinity_t\fR \fBlgrp_affinity_get\fR(\fBidtype_t\fR \fIidtype\fR, \fBid
16 \fBlgrp_id_t\fR \fIid\fR);
17 .fi

19 .LP
20 .nf
21 \fBint\fR \fBlgrp_affinity_set\fR(\fBidtype_t\fR \fIidtype\fR, \fBid_t\fR \fIid\
22 \fBlgrp_affinity_t\fR \fIaffinity\fR);
23 .fi

25 .SH DESCRIPTION
26 .sp
27 .LP
28 The \fBlgrp_affinity_get()\fR function returns the affinity that the LWP or set
29 of LWPs specified by the \fIidtype\fR and \fIid\fR arguments have for the given
30 lgroup.
31 .sp
32 .LP
33 The \fBlgrp_affinity_set()\fR function sets the affinity that the LWP or set of
34 LWPs specified by \fIidtype\fR and \fIid\fR have for the given lgroup. The
35 lgroup affinity can be set to \fBBLGRP_AFF_STRONG\fR, \fBBLGRP_AFF_WEAK\fR, or
36 \fBBLGRP_AFF_NONE\fR.
37 .sp
38 .LP
39 If the \fIidtype\fR is \fBFBP_PID\fR, the affinity is retrieved for one of the
40 LWPs in the process or set for all the LWPs of the process with process ID
41 (PID) \fIid\fR. The affinity is retrieved or set for the LWP of the current
42 process with LWP ID \fIid\fR if \fIidtype\fR is \fBFBP_LWPID\fR. If \fIid\fR is
43 \fBFBP_MYID\fR, then the current LWP or process is specified.
44 .sp
45 .LP
46 The operating system uses the lgroup affinities as advice on where to run a
47 thread and allocate its memory and factors this advice in with other
48 constraints. Processor binding and processor sets can restrict which lgroups a
49 thread can run on, but do not change the lgroup affinities.
50 .sp
51 .LP
52 Each thread can have an affinity for an lgroup in the system such that the
53 thread will tend to be scheduled to run on that lgroup and allocate memory from
54 there whenever possible. If the thread has affinity for more than one lgroup,
55 the operating system will try to run the thread and allocate its memory on the
56 lgroup for which it has the strongest affinity, then the next strongest, and so
57 on up through some small, system-dependent number of these lgroup affinities.
58 When multiple lgroups have the same affinity, the order of preference among
59 them is unspecified and up to the operating system to choose. The lgroup with
60 the strongest affinity that the thread can run on is known as its "home lgroup"

```

```

60 (see \fBlgrp_home\fR(3LGRP)) and is usually the operating system's first choice
61 of where to run the thread and allocate its memory.
62 .sp
63 .LP
64 There are different levels of affinity that can be specified by a thread for a
65 particular lgroup. The levels of affinity are the following from strongest to
66 weakest:
67 .sp
68 .in +2
69 .nf
70 LGRP_AFF_STRONG /* strong affinity */
71 LGRP_AFF_WEAK /* weak affinity */
72 LGRP_AFF_NONE /* no affinity */
73 .fi
74 .in -2

76 .sp
77 .LP
78 The \fBBLGRP_AFF_STRONG\fR affinity serves as a hint to the operating system
79 that the calling thread has a strong affinity for the given lgroup. If this is
80 the thread's home lgroup, the operating system will avoid rehomeing it to
81 another lgroup if possible. However, dynamic reconfiguration, processor
82 offlining, processor binding, and processor set binding and manipulation are
83 examples of events that can cause the operating system to change the thread's
84 home lgroup for which it has a strong affinity.
85 .sp
86 .LP
87 The \fBBLGRP_AFF_WEAK\fR affinity is a hint to the operating system that the
88 calling thread has a weak affinity for the given lgroup. If a thread has a
89 weak affinity for its home lgroup, the operating system interprets this to mean
90 that thread does not mind whether it is rehomeed, unlike \fBBLGRP_AFF_STRONG\fR.
91 Load balancing, dynamic reconfiguration, processor binding, or processor set
92 binding and manipulation are examples of events that can cause the operating
93 system to change a thread's home lgroup for which it has a weak affinity.
94 .sp
95 .LP
96 The \fBBLGRP_AFF_NONE\fR affinity signifies no affinity and can be used to
97 remove a thread's affinity for a particular lgroup. Initially, each thread
98 remove a thread's affinity for a particular lgroup. Initially, each thread
99 has no affinity to any lgroup. If a thread has no lgroup affinities set, the
100 operating system chooses a home lgroup for the thread with no affinity set.
101 .SH RETURN VALUES
102 .sp
103 .LP
104 Upon successful completion, \fBlgrp_affinity_get()\fR returns the affinity for
105 the given lgroup.
106 .sp
107 .LP
108 Upon successful completion, \fBlgrp_affinity_set()\fR return 0.
109 .sp
110 .LP
111 Otherwise, both functions return \fB(0)\fR and set \fBerrno\fR to indicate the
112 error.
113 .SH ERRORS
114 .sp
115 .LP
116 The \fBlgrp_affinity_get()\fR and \fBlgrp_affinity_set()\fR functions will fail
117 if:
118 .sp
119 .na
120 \fBFBEINVAL\fR
121 .ad
122 .RS 10n

```

```
121 The specified lgroup, affinity, or ID type is not valid.
122 .RE

124 .sp
125 .ne 2
126 .na
127 \fB\FBEPERM\fR\fR
128 .ad
129 .RS 10n
130 The effective user of the calling process does not have appropriate privileges,
131 and its real or effective user ID does not match the real or effective user ID
132 of one of the LWPs.
133 .RE

135 .sp
136 .ne 2
137 .na
138 \fB\FBESRCH\fR\fR
139 .ad
140 .RS 10n
141 The specified lgroup or LWP(s) was not found.
142 .RE

144 .SH ATTRIBUTES
148 .sp
145 .LP
146 See \fBAttributes\fR(5) for descriptions of the following attributes:
147 .sp

149 .sp
150 .TS
151 box;
152 c | c
153 l | l .
154 ATTRIBUTE TYPE ATTRIBUTE VALUE
155 _
156 Interface Stability Evolving
157 _
158 MT-Level MT-Safe
159 .TE

161 .SH SEE ALSO
166 .sp
162 .LP
163 \fBBlgrp_home\fR(3LGRP), \fBBlblgrp\fR(3LIB), \fBAttributes\fR(5)
```

26503 Tue Dec 11 09:48:09 2018

new/usr/src/man/man3perl/Lgrp.3perl

10057 Man page misspellings ouput particular overridden

Reviewed by: Gerg Mih^ly Doma <domag02@gmail.com>

```

1  \" te
2  .\" Copyright (c) 2008, Sun Microsystems, Inc. All Rights Reserved.
3  .\" The contents of this file are subject to the terms of the Common Development
4  .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
5  .\" When distributing Covered Code, include this CDDL HEADER in each file and in
6  .TH LGRP 3PERL \"April 9, 2016\"
7  .SH NAME
8  Lgrp \- Perl interface to Solaris liblgrp library
9  .SH SYNOPSIS
10 .LP
11 .nf
12 use Sun::Solaris::Lgrp qw(:ALL);

14 # initialize lgroup interface
15 my $cookie = lgrp_init(LGRP_VIEW_OS | LGRP_VIEW_CALLER);
16 my $l = Sun::Solaris::Lgrp->new(LGRP_VIEW_OS |
17     LGRP_VIEW_CALLER);

19 my $version = lgrp_version(LGRP_VER_CURRENT | LGRP_VER_NONE);
20 $version = $l->version(LGRP_VER_CURRENT | LGRP_VER_NONE);

22 $home = lgrp_home(P_PID, P_MYID);
23 $home = $l->home(P_PID, P_MYID);

25 lgrp_affinity_set(P_PID, $fIpid\fR, $fIlgrp\fR,
26     LGRP_AFF_STRONG | LGRP_AFF_WEAK | LGRP_AFF_NONE);
27 $l->affinity_set(P_PID, $fIpid\fR, $fIlgrp\fR,
28     LGRP_AFF_STRONG | LGRP_AFF_WEAK | LGRP_AFF_NONE);

30 my $affinity = lgrp_affinity_get(P_PID, $fIpid\fR, $fIlgrp\fR);
31 $affinity = $l->affinity_get(P_PID, $fIpid\fR, $fIlgrp\fR);

33 my $nlgtps = lgrp_nlgtps($fIcookie\fR);
34 $nlgtps = $l->nlgtps();

36 my $root = lgrp_root($fIcookie\fR);
37 $root = $l->root();

39 $latency = lgrp_latency($fIlgrp1\fR, $fIlgrp2\fR);
40 $latency = $l->latency($fIlgrp1\fR, $fIlgrp2\fR);

42 my @children = lgrp_children($fIcookie\fR, $fIlgrp\fR);
43 @children = $l->children($lgrp);

45 my @parents = lgrp_parents($fIcookie\fR, $fIlgrp\fR);
46 @parents = $l->parents($fIlgrp\fR);

48 my @lgrps = lgrp_lgrps($fIcookie\fR);
49 @lgrps = $l->lgrps();

51 @lgrps = lgrp_lgrps($fIcookie\fR, $fIlgrp\fR);
52 @lgrps = $l->lgrps($fIlgrp\fR);

54 my @leaves = lgrp_leaves($fIcookie\fR);
55 @leaves = $l->leaves();

57 my $is_leaf = lgrp_isleaf($fIcookie\fR, $fIlgrp\fR);
58 $is_leaf = $l->is_leaf($fIlgrp\fR);

60 my @cpus = lgrp_cpus($fIcookie\fR, $fIlgrp\fR,
```

```

61     LGRP_CONTENT_HIERARCHY | LGRP_CONTENT_DIRECT);
62 @cpus = $l->cpus($fIlgrp\fR, LGRP_CONTENT_HIERARCHY |
63     LGRP_CONTENT_DIRECT);

65 my $memsize = lgrp_mem_size($fIcookie\fR, $fIlgrp\fR,
66     LGRP_MEM_SZ_INSTALLED | LGRP_MEM_SZ_FREE,
67     LGRP_CONTENT_HIERARCHY | LGRP_CONTENT_DIRECT);
68 $memsize = $l->mem_size($fIlgrp\fR,
69     LGRP_MEM_SZ_INSTALLED | LGRP_MEM_SZ_FREE,
70     LGRP_CONTENT_HIERARCHY | LGRP_CONTENT_DIRECT);

72 my $is_stale = lgrp_cookie_stale($fIcookie\fR);
73 $is_stale = $l->stale();

75 lgrp_fini($fIcookie\fR);

77 # The following is available for API version greater than 1:
78 my @lgrps = lgrp_resources($fIcookie\fR, $fIlgrp\fR, LGRP_RSRC_CPU);

80 # Get latencies from cookie
81 $latency = lgrp_latency_cookie($fIcookie\fR, $fIfrom\fR, $fIto\fR);
82 .fi

84 .SH DESCRIPTION
85 .LP
86 This module provides access to the \fBliblgrp\fR(3LIB) library and to various
87 constants and functions defined in <\fBsys/lgrp_sys.h\fR>. It provides both the
88 procedural and object interface to the library. The procedural interface
89 requires (in most cases) passing around a transparent cookie. The object
90 interface hides all the cookie manipulations from the user.
91 .sp
92 .LP
93 Functions returning a scalar value indicate an error by returning \fBundef\fR.
94 The caller can examine the \fB$! \fR variable to get the error value.
95 .sp
96 .LP
97 Functions returning a list value return the number of elements in the list when
98 called in scalar context. In the event of error, the empty list is returned in
99 the array context and \fBundef\fR is returned in the scalar context.
100 .SS \"Constants\"
101 .LP
102 The constants are exported with \fB:CONSTANTS \fR or \fB:ALL \fR tags:
103 .sp
104 .in +2
105 .nf
106 use Sun::Solaris::Lgrp ':ALL';
107 .fi
108 .in -2

110 .sp
111 .LP
112 or
113 .sp
114 .in +2
115 .nf
116 use Sun::Solaris::Lgrp ':CONSTANTS';
117 .fi
118 .in -2

120 .sp
121 .LP
122 The following constants are available for use in Perl programs:
123 .br
124 .in +2
125 \fBLGRP_NONE \fR
126 .in -2
```

```

127 .br
128 .in +2
129 \fBLGRP_VER_CURRENT\fR
130 .in -2
131 .br
132 .in +2
133 \fBLGRP_VER_NONE\fR
134 .in -2
135 .br
136 .in +2
137 \fBLGRP_VIEW_CALLER\fR
138 .in -2
139 .br
140 .in +2
141 \fBLGRP_VIEW_OS\fR
142 .in -2
143 .br
144 .in +2
145 \fBLGRP_AFF_NONE\fR
146 .in -2
147 .br
148 .in +2
149 \fBLGRP_AFF_STRONG\fR
150 .in -2
151 .br
152 .in +2
153 \fBLGRP_AFF_WEAK\fR
154 .in -2
155 .br
156 .in +2
157 \fBLGRP_CONTENT_DIRECT\fR
158 .in -2
159 .br
160 .in +2
161 \fBLGRP_CONTENT_HIERARCHY\fR
162 .in -2
163 .br
164 .in +2
165 \fBLGRP_MEM_SZ_FREE\fR
166 .in -2
167 .br
168 .in +2
169 \fBLGRP_MEM_SZ_FREE\fR
170 .in -2
171 .br
172 .in +2
173 \fBLGRP_RSRC_CPU\fR (1)
174 .in -2
175 .br
176 .in +2
177 \fBLGRP_RSRC_MEM\fR (1)
178 .in -2
179 .br
180 .in +2
181 \fBLGRP_CONTENT_ALL\fR (1)
182 .in -2
183 .br
184 .in +2
185 \fBLGRP_LAT_CPU_TO_MEM\fR (1)
186 .in -2
187 .br
188 .in +2
189 \fBP_PID\fR
190 .in -2
191 .br
192 .in +2

```

```

193 \fBP_LWPID\fR
194 .in -2
195 .br
196 .in +2
197 \fBP_MYID\fR
198 .in -2
199 .sp
200 .LP
201 (1) Available for versions of the \fBliblgrp\fR(3LIB) API greater than 1.
202 .SS "Functions"
203 .LP
204 A detailed description of each function follows. Since this module is intended
205 to provide a Perl interface to the functions in \fBliblgrp\fR(3LIB), a very
206 short description is given for the corresponding functions in this module and a
207 reference is given to the complete description in the \fBliblgrp\fR manual
208 pages. Any differences or additional functionality in the Perl module are
209 highlighted and fully documented here.
210 .sp
211 .ne 2
212 .na
213 \fB\fBlgrp_init([LGRP_VIEW_CALLER | LGRP_VIEW_OS])\fR\fR
214 .ad
215 .sp .6
216 .RS 4n
217 This function initializes the lgroup interface and takes a snapshot of the
218 lgroup hierarchy with the given view. Given the view, \fBlgrp_init()\fR returns
219 a cookie representing this snapshot of the lgroup hierarchy. This cookie should
220 be used with other routines in the lgroup interface needing the lgroup
221 hierarchy. The \fBlgrp_fini()\fR function should be called with the cookie when
222 it is no longer needed. Unlike \fBlgrp_init\fR(3LGRP), \fBLGRP_VIEW_OS\fR is
223 assumed as the default if no view is provided.
224 .sp
225 Upon successful completion, \fBlgrp_init()\fR returns a cookie. Otherwise it
226 returns \fBundef\fR and sets \fB$!\fR to indicate the error.
227 .sp
228 See \fBlgrp_init\fR(3LGRP) for more information.
229 .RE

231 .sp
232 .ne 2
233 .na
234 \fB\fBlgrp_fini\fR($\fIcookie\fR)\fR
235 .ad
236 .sp .6
237 .RS 4n
238 This function takes a cookie, frees the snapshot of the lgroup hierarchy
239 created by \fBlgrp_init()\fR, and cleans up anything else set up by
240 \fBlgrp_init()\fR. After this function is called, the cookie returned by the
241 lgroup interface might no longer be valid and should not be used.
242 .sp
243 Upon successful completion, 1 is returned. Otherwise, \fBundef\fR is returned
244 and \fB$!\fR is set to indicate the error.
245 .sp
246 See \fBlgrp_fini\fR(3LGRP) for more information.
247 .RE

249 .sp
250 .ne 2
251 .na
252 \fB\fBlgrp_view\fR($\fIcookie\fR)\fR
253 .ad
254 .sp .6
255 .RS 4n
256 This function takes a cookie representing the snapshot of the lgroup hierarchy
257 and returns the snapshot's view of the lgroup hierarchy.
258 .sp

```


259 If the given view is `\fBLGRP_VIEW_CALLER`, the snapshot contains only the
 260 resources that are available to the caller (such as those with respect to
 261 processor sets). When the view is `\fBLGRP_VIEW_OS`, the snapshot contains
 262 what is available to the operating system.

263 .sp
 264 Upon successful completion, the function returns the view for the snapshot of
 265 the lgroup hierarchy represented by the given cookie. Otherwise, `\fBundef`
 266 is returned and `\fB$!` is set to indicate the error.

267 .sp
 268 See `\fBlgrp_view` for more information.

269 .RE

271 .sp
 272 .ne 2
 273 .na
 274 `\fB\fBlgrp_home`(`\fR($\fIidtype`, `\fR($\fIid`)`\fR`
 275 .ad
 276 .sp .6
 277 .RS 4n
 278 This function returns the home lgroup for the given process or thread. The
 279 `\fIidtype` argument should be `\fBP_PID` to specify a process and the
 280 `\fIid` argument should be its process ID. Otherwise, the `\fIidtype`
 281 argument should be `\fBP_LWPID` to specify a thread and the `\fIid`
 282 argument should be its LWP ID. The value `\fBP_MYID` can be used for the
 283 argument to specify the current process or thread.

284 .sp
 285 Upon successful completion, `\fBlgrp_home` returns the ID of the home lgroup
 286 of the specified process or thread. Otherwise, `\fBundef` is returned and
 287 `\fB$!` is set to indicate the error.

288 .sp
 289 See `\fBlgrp_home` for more information.

290 .RE

292 .sp
 293 .ne 2
 294 .na
 295 `\fB\fBlgrp_cookie_stale`(`\fR($\fIcookie`)`\fR`
 296 .ad
 297 .sp .6
 298 .RS 4n
 299 Upon successful completion, this function returns whether the cookie is stale.
 300 Otherwise, it returns `\fBundef` and sets `\fB$!` to indicate the error.

301 .sp
 302 The `\fBlgrp_cookie_stale` function will fail with `\fBEINVAL` if the
 303 cookie is not valid.

304 .sp
 305 See `\fBlgrp_cookie_stale` for more information.

306 .RE

308 .sp
 309 .ne 2
 310 .na
 311 `\fB\fBlgrp_cpus`(`\fR($\fIcookie`, `\fR($\fIlgrp`, `\fR($\fIcontext`)`\fR`
 312 .ad
 313 .sp .6
 314 .RS 4n
 315 This function takes a cookie representing a snapshot of the lgroup hierarchy
 316 and returns the list of CPUs in the lgroup specified by `\fIlgrp`. The
 317 `\fIcontext` argument should be set to one of the following values to specify
 318 whether the direct contents or everything in this lgroup including its children
 319 should be returned:

320 .sp
 321 .ne 2
 322 .na
 323 `\fB\fBLGRP_CONTENT_HIERARCHY``\fR`
 324 .ad

325 .RS 26n
 326 everything within this hierarchy
 327 .RE

329 .sp
 330 .ne 2
 331 .na
 332 `\fB\fBLGRP_CONTENT_DIRECT``\fR`
 333 .ad
 334 .RS 26n
 335 directly contained in lgroup
 336 .RE

338 When called in scalar context, `\fBlgrp_cpus` function returns the number of
 339 CPUs contained in the specified lgroup.

340 .sp
 341 In the event of error, `\fBundef` is returned in scalar context and `\fB$!`
 342 is set to indicate the error. In list context, the empty list is returned and
 343 `\fB$!` is set.

344 .sp
 345 See `\fBlgrp_cpus` for more information.

346 .RE

348 .sp
 349 .ne 2
 350 .na
 351 `\fB\fBlgrp_children`(`\fR($\fIcookie`, `\fR($\fIlgrp`)`\fR`
 352 .ad
 353 .sp .6
 354 .RS 4n
 355 This function takes a cookie representing a snapshot of the lgroup hierarchy
 356 and returns the list of lgroups that are children of the specified lgroup.

357 .sp
 358 When called in scalar context, `\fBlgrp_children` returns the number of
 359 children lgroups for the specified lgroup.

360 .sp
 361 In the event of error, `\fBundef` or empty list is returned and `\fB$!` is
 362 set to indicate the error.

363 .sp
 364 See `\fBlgrp_children` for more information.

365 .RE

367 .sp
 368 .ne 2
 369 .na
 370 `\fB\fBlgrp_parents`(`\fR($\fIcookie`, `\fR($\fIlgrp`)`\fR`
 371 .ad
 372 .sp .6
 373 .RS 4n
 374 This function takes a cookie representing a snapshot of the lgroup hierarchy
 375 and returns the list of parents of the specified lgroup.

376 .sp
 377 When called in scalar context, `\fBlgrp_parents` returns the number of
 378 parent lgroups for the specified lgroup.

379 .sp
 380 In the event of error, `\fBundef` or an empty list is returned and `\fB$!` is
 381 set to indicate the error.

382 .sp
 383 See `\fBlgrp_parents` for more information.

384 .RE

386 .sp
 387 .ne 2
 388 .na
 389 `\fB\fBlgrp_nlgrps`(`\fR($\fIcookie`)`\fR`
 390 .ad

```

391 .sp .6
392 .RS 4n
393 This function takes a cookie representing a snapshot of the lgroup hierarchy.
394 It returns the number of lgroups in the hierarchy, where the number is always
395 at least one.
396 .sp
397 In the event of error, \fBundef\fR is returned and \fB!\fR is set to
398 \fBEINVAL\fR, indicating that the cookie is not valid.
399 .sp
400 See \fBlgrp_nlgrps\fR(3LGRP) for more information.
401 .RE

403 .sp
404 .ne 2
405 .na
406 \fB\fBlgrp_root\fR($\fIcookie\fR)\fR
407 .ad
408 .sp .6
409 .RS 4n
410 This function returns the root lgroup ID.
411 .sp
412 In the event of error, \fBundef\fR is returned and \fB!\fR is set to
413 \fBEINVAL\fR, indicatng that the cookie is not valid.
414 .sp
415 See \fBlgrp_root\fR(3LGRP) for more information.
416 .RE

418 .sp
419 .ne 2
420 .na
421 \fB\fBlgrp_mem_size\fR($\fIcookie\fR, $\fIlgrp\fR, $\fItype\fR,
422 $\fIcontent\fR)\fR
423 .ad
424 .sp .6
425 .RS 4n
426 This function takes a cookie representing a snapshot of the lgroup hierarchy.
427 The function returns the memory size of the given lgroup in bytes. The
428 $\fItype\fR argument should be set to one of the following values:
429 .sp
430 .ne 2
431 .na
432 \fB\fBLGRP_MEM_SZ_FREE\fR\fR
433 .ad
434 .RS 25n
435 free memory
436 .RE

438 .sp
439 .ne 2
440 .na
441 \fB\fBLGRP_MEM_SZ_INSTALLED\fR\fR
442 .ad
443 .RS 25n
444 installed memory
445 .RE

447 The $\fIcontent\fR argument should be set to one of the following values to
448 specify whether the direct contents or everything in this lgroup including its
449 children should be returned:
450 .sp
451 .ne 2
452 .na
453 \fB\fBLGRP_CONTENT_HIERARCHY\fR\fR
454 .ad
455 .RS 26n
456 Return everything within this hierarchy.

```

```

457 .RE

459 .sp
460 .ne 2
461 .na
462 \fB\fBLGRP_CONTENT_DIRECT\fR\fR
463 .ad
464 .RS 26n
465 Return that which is directly contained in this lgroup.
466 .RE

468 The total sizes include all the memory in the lgroup including its children,
469 while the others reflect only the memory contained directly in the given
470 lgroup.
471 .sp
472 Upon successful completion, the size in bytes is returned. Otherwise,
473 \fBundef\fR is returned and \fB!\fR is set to indicate the error.
474 .sp
475 See \fBlgrp_mem_size\fR(3LGRP) for more information.
476 .RE

478 .sp
479 .ne 2
480 .na
481 \fB\fBlgrp_version\fR([\fIversion\fR])\fR
482 .ad
483 .sp .6
484 .RS 4n
485 This function takes an interface version number, $\fIversion\fR, as an argument
486 and returns an lgroup interface version. The $\fIversion\fR argument should be
487 the value of \fBLGRP_VER_CURRENT\fR or \fBLGRP_VER_NONE\fR to find out the
488 current lgroup interface version on the running system.
489 .sp
490 If $\fIversion\fR is still supported by the implementation, then
491 \fBlgrp_version()\fR returns the requested version. If \fBLGRP_VER_NONE\fR is
492 returned, the implementation cannot support the requested version.
493 .sp
494 If $\fIversion\fR is \fBLGRP_VER_NONE\fR, \fBlgrp_version()\fR returns the
495 current version of the library.
496 .sp
497 The following example tests whether the version of the interface used by the
498 caller is supported:
499 .sp
500 .in +2
501 .nf
502 lgrp_version(LGRP_VER_CURRENT) == LGRP_VER_CURRENT or
503 die("Built with unsupported lgroup interface");
504 .fi
505 .in -2

507 See \fBlgrp_version\fR(3LGRP) for more information.
508 .RE

510 .sp
511 .ne 2
512 .na
513 \fB\fBlgrp_affinity_set\fR($\fIidtype\fR, $\fIid\fR, $\fIlgrp\fR,
514 $\fIaffinity\fR)\fR
515 .ad
516 .sp .6
517 .RS 4n
518 This function sets the affinity that the LWP or set of LWPs specified by
519 $\fIidtype\fR and $\fIid\fR have for the given lgroup. The lgroup affinity can
520 be set to \fBLGRP_AFF_STRONG\fR, \fBLGRP_AFF_WEAK\fR, or \fBLGRP_AFF_NONE\fR.
521 .sp
522 If the $\fIidtype\fR is \fBFP_PID\fR, the affinity is retrieved for one of the

```

523 LWPs in the process or set for all the LWPs of the process with process ID
 524 (PID) `$(fiid)`. The affinity is retrieved or set for the LWP of the current
 525 process with LWP ID `$(fiid)` if `$(fiidtype)` is `\fBP_LWPID`. If `$(fiid)`
 526 is `\fBP_MYID`, then the current LWP or process is specified.

527 .sp
 528 There are different levels of affinity that can be specified by a thread for a
 529 **particular lgroup. The levels of affinity are the following from strongest to**
 529 **particular lgroup. The levels of affinity are the following from strongest to**
 530 **weakest:**

531 .sp
 532 .ne 2
 533 .na
 534 `\fB\fBLGRP_AFF_STRONG`
 535 .ad
 536 .RS 19n
 537 strong affinity
 538 .RE

540 .sp
 541 .ne 2
 542 .na
 543 `\fB\fBLGRP_AFF_WEAK`
 544 .ad
 545 .RS 19n
 546 weak affinity
 547 .RE

549 .sp
 550 .ne 2
 551 .na
 552 `\fB\fBLGRP_AFF_NONE`
 553 .ad
 554 .RS 19n
 555 no affinity
 556 .RE

558 Upon successful completion, `\fBlgrp_affinity_set()` returns 1. Otherwise, it
 559 returns `\fBundef` and set `\fB$!` to indicate the error.

560 .sp
 561 See `\fBlgrp_affinity_set` for more information.
 562 .RE

564 .sp
 565 .ne 2
 566 .na
 567 `\fB\fBlgrp_affinity_get``(``$(fiidtype)``,` `$(fiid)``,` `$(flgrp)``)`
 568 .ad
 569 .sp .6
 570 .RS 4n
 571 This function returns the affinity that the LWP has to a given lgroup.
 572 .sp
 573 See `\fBlgrp_affinity_get` for more information.
 574 .RE

576 .sp
 577 .ne 2
 578 .na
 579 `\fB\fBlgrp_latency_cookie``(``$(fcookie)``,` `$(fifrom)``,` `$(fito)``,`
 580 `[$\fIbetween=\fBLGRP_LAT_CPU_TO_MEM])`
 581 .ad
 582 .sp .6
 583 .RS 4n
 584 This function takes a cookie representing a snapshot of the lgroup hierarchy
 585 and returns the latency value between a hardware resource in the `$(fifrom)`
 586 lgroup to a hardware resource in the `$(fito)` lgroup. If `$(fifrom)` is the
 587 same lgroup as `$(fito)`, the latency value within that lgroup is returned.

588 .sp
 589 The optional `$(fIbetween)` argument should be set to `\fBLGRP_LAT_CPU_TO_MEM`
 590 to specify between which hardware resources the latency should be measured. The
 591 only valid value is `\fBLGRP_LAT_CPU_TO_MEM`, which represents latency from
 592 CPU to memory.

593 .sp
 594 Upon successful completion, `\fBlgrp_latency_cookie()` return 1. Otherwise,
 595 it returns `\fBundef` and set `\fB$!` to indicate the error. For LGRP API
 596 version 1, the `\fBlgrp_latency_cookie()` is an alias for
 597 `\fBlgrp_latency.`
 598 .sp
 599 See `\fBlgrp_latency_cookie` for more information.
 600 .RE

602 .sp
 603 .ne 2
 604 .na
 605 `\fB\fBlgrp_latency``(``$(fifrom)``,` `$(fito)``)`
 606 .ad
 607 .sp .6
 608 .RS 4n
 609 This function is similar to the `\fBlgrp_latency_cookie()` function, but
 610 returns the latency between the given lgroups at the given instant in time.
 611 Since lgroups can be freed and reallocated, this function might not be able to
 612 provide a consistent answer across calls. For that reason,
 613 `\fBlgrp_latency_cookie()` should be used in its place.
 614 .sp
 615 See `\fBlgrp_latency` for more information.
 616 .RE

618 .sp
 619 .ne 2
 620 .na
 621 `\fB\fBlgrp_resources``(``$(fIcookie)``,` `$(fIgrp)``,` `$(fItype)``)`
 622 .ad
 623 .sp .6
 624 .RS 4n
 625 This function returns the list of lgroups directly containing resources of the
 626 specified type. The resources are represented by a set of lgroups in which each
 627 lgroup directly contains CPU and/or memory resources.
 628 .sp
 629 The `$(fItype)` can be specified as:

630 .sp
 631 .ne 2
 632 .na
 633 `\fB\fBLGRP_RSRC_CPU`
 634 .ad
 635 .RS 17n
 636 CPU resources
 637 .RE

639 .sp
 640 .ne 2
 641 .na
 642 `\fB\fBLGRP_RSRC_MEM`
 643 .ad
 644 .RS 17n
 645 memory resources
 646 .RE

648 In the event of error, `\fBundef` or an empty list is returned and `\fB$!` is
 649 set to indicate the error.

650 .sp
 651 This function is available only for API version 2 and returns `\fBundef` or an
 652 empty list for API version 1 and sets `\fB$!` to `\fBEINVAL`.
 653 .sp

```

654 See \fBlgrp_resources\fR(3LGRP) for more information.
655 .RE

657 .sp
658 .ne 2
659 .na
660 \fB\fBlgrp_lgrps\fR(\fIcookie\fR, [\fIilgrp\fR])\fR
661 .ad
662 .sp .6
663 .RS 4n
664 This function returns a list of all lgroups in a hierarchy starting from
665 \fIilgrp\fR. If \fIilgrp\fR is not specified, uses the value of
666 \fBlgrp_root\fR(\fIcookie\fR). This function returns the empty list on
667 failure.
668 .sp
669 When called in scalar context, this function returns the total number of
670 lgroups in the system.
671 .RE

673 .sp
674 .ne 2
675 .na
676 \fB\fBlgrp_leaves\fR(\fIcookie\fR, [\fIilgrp\fR])\fR
677 .ad
678 .sp .6
679 .RS 4n
680 This function returns a list of all leaf lgroups in a hierarchy starting from
681 \fIilgrp\fR. If \fIilgrp\fR is not specified, this function uses the value of
682 \fBlgrp_root\fR(\fIcookie\fR). It returns \fBundef\fR or an empty list on
683 failure.
684 .sp
685 When called in scalar context, this function returns the total number of leaf
686 lgroups in the system.
687 .RE

689 .sp
690 .ne 2
691 .na
692 \fB\fBlgrp_isleaf\fR(\fIcookie\fR, \fIilgrp\fR)\fR
693 .ad
694 .sp .6
695 .RS 4n
696 This function returns True if \fIilgrp\fR is a leaf (has no children).
697 Otherwise it returns False.
698 .RE

700 .SS "Object methods"
701 .ne 2
702 .na
703 \fB\fBnew\fR([\fIview\fR])\fR
704 .ad
705 .sp .6
706 .RS 4n
707 This method creates a new \fBSun::Solaris::Lgrp\fR object. An optional argument
708 is passed to the \fBlgrp_init()\fR function. By default this method uses
709 \fBLGRP_VIEW_OS\fR.
710 .RE

712 .sp
713 .ne 2
714 .na
715 \fB\fBcookie()\fR\fR
716 .ad
717 .sp .6
718 .RS 4n
719 This method returns a transparent cookie that can be passed to functions

```

```

720 accepting the cookie.
721 .RE

723 .sp
724 .ne 2
725 .na
726 \fB\fBversion\fR([\fIversion\fR])\fR
727 .ad
728 .sp .6
729 .RS 4n
730 Without the argument, this method returns the current version of the
731 \fBliblgrp\fR(3LIB) library. This method is a wrapper for \fBlgrp_version()\fR
732 with \fBLGRP_VER_NONE\fR as the default version argument.
733 .RE

735 .sp
736 .ne 2
737 .na
738 \fB\fBstale()\fR\fR
739 .ad
740 .sp .6
741 .RS 4n
742 This method returns T if the lgroup information in the object is stale and F
743 otherwise. It is a wrapper for \fBlgrp_cookie_stale()\fR.
744 .RE

746 .sp
747 .ne 2
748 .na
749 \fB\fBview()\fR\fR
750 .ad
751 .sp .6
752 .RS 4n
753 This method returns the snapshot's view of the lgroup hierarchy. It is a
754 wrapper for \fBlgrp_view()\fR.
755 .RE

757 .sp
758 .ne 2
759 .na
760 \fB\fBroot()\fR\fR
761 .ad
762 .sp .6
763 .RS 4n
764 This method returns the root lgroup. It is a wrapper for \fBlgrp_root()\fR.
765 .RE

767 .sp
768 .ne 2
769 .na
770 \fB\fBchildren\fR(\fIilgrp\fR)\fR
771 .ad
772 .sp .6
773 .RS 4n
774 This method returns the list of lgroups that are children of the specified
775 lgroup. It is a wrapper for \fBlgrp_children()\fR.
776 .RE

778 .sp
779 .ne 2
780 .na
781 \fB\fBparents\fR(\fIilgrp\fR)\fR
782 .ad
783 .sp .6
784 .RS 4n
785 This method returns the list of lgroups that are parents of the specified

```

```

786 lgroup. It is a wrapper for \fBlgrp_parents()\fR.
787 .RE

789 .sp
790 .ne 2
791 .na
792 \fB\fBnlgrps()\fR\fR
793 .ad
794 .sp .6
795 .RS 4n
796 This method returns the number of lgroups in the hierarchy. It is a wrapper for
797 \fBlgrp_nlgrps()\fR.
798 .RE

800 .sp
801 .ne 2
802 .na
803 \fB\fBmem_size\fR($\fIlgrp\fR, $\fItype\fR, $\fIcontent\fR)\fR
804 .ad
805 .sp .6
806 .RS 4n
807 This method returns the memory size of the given lgroup in bytes. It is a
808 wrapper for \fBlgrp_mem_size()\fR.
809 .RE

811 .sp
812 .ne 2
813 .na
814 \fB\fBcpus\fR($\fIlgrp\fR, $\fIcontext\fR)\fR
815 .ad
816 .sp .6
817 .RS 4n
818 This method returns the list of CPUs in the lgroup specified by $lgrp. It is a
819 wrapper for \fBlgrp_cpus()\fR.
820 .RE

822 .sp
823 .ne 2
824 .na
825 \fB\fBresources\fR($\fIlgrp\fR, $\fItype\fR)\fR
826 .ad
827 .sp .6
828 .RS 4n
829 This method returns the list of lgroups directly containing resources of the
830 specified type. It is a wrapper for \fBlgrp_resources()\fR.
831 .RE

833 .sp
834 .ne 2
835 .na
836 \fB\fBhome\fR($\fIidtype\fR, $\fIid\fR)\fR
837 .ad
838 .sp .6
839 .RS 4n
840 This method returns the home lgroup for the given process or thread. It is a
841 wrapper for \fBlgrp_home()\fR.
842 .RE

844 .sp
845 .ne 2
846 .na
847 \fB\fBaffinity_get\fR($\fIidtype\fR, $\fIid\fR, $\fIlgrp\fR)\fR
848 .ad
849 .sp .6
850 .RS 4n
851 This method returns the affinity that the LWP has to a given lgrp. It is a

```

```

852 wrapper for \fBlgrp_affinity_get()\fR.
853 .RE

855 .sp
856 .ne 2
857 .na
858 \fB\fBaffinity_set\fR($\fIidtype\fR, $\fIid\fR, $\fIlgrp\fR,
859 $\fIaffinity\fR)\fR
860 .ad
861 .sp .6
862 .RS 4n
863 This method sets the affinity that the LWP or set of LWPs specified by
864 $\fIidtype\fR and $\fIid\fR have for the given lgroup. It is a wrapper for
865 lgrp_affinity_set.
866 .RE

868 .sp
869 .ne 2
870 .na
871 \fB\fBBlgrps\fR([\fIlgrp\fR])\fR
872 .ad
873 .sp .6
874 .RS 4n
875 This method returns list of all lgroups in a hierarchy starting from
876 $\fIlgrp\fR or the \fBlgrp_root()\fR if $\fIlgrp\fR is not specified. It is a
877 wrapper for \fBlgrp_lgrps()\fR.
878 .RE

880 .sp
881 .ne 2
882 .na
883 \fB\fBleaves\fR([\fIlgrp\fR])\fR
884 .ad
885 .sp .6
886 .RS 4n
887 This method returns a list of all leaf lgroups in a hierarchy starting from
888 $\fIlgrp\fR. If $\fIlgrp\fR is not specified, this method uses the value of
889 \fBlgrp_root()\fR. It is a wrapper for \fBlgrp_leaves()\fR.
890 .RE

892 .sp
893 .ne 2
894 .na
895 \fB\fBisleaf\fR($\fIlgrp\fR)\fR
896 .ad
897 .sp .6
898 .RS 4n
899 This method returns True if $\fIlgrp\fR is leaf (has no children) and False
900 otherwise. It is a wrapper for \fBlgrp_isleaf()\fR.
901 .RE

903 .sp
904 .ne 2
905 .na
906 \fB\fBlatency\fR($\fIfrom\fR, $\fIto\fR)\fR
907 .ad
908 .sp .6
909 .RS 4n
910 This method returns the latency value between a hardware resource in the
911 $\fIfrom\fR lgroup to a hardware resource in the $\fIto\fR lgroup. It uses
912 \fBlgrp_latency()\fR for version 1 of \fBlblgrp\fR and
913 \fBlgrp_latency_cookie()\fR for newer versions.
914 .RE

916 .SS "Exports"
917 .LP

```

918 By default nothing is exported from this module. The following tags can be used
919 to selectively import constants and functions defined in this module:

```
920 .sp
921 .ne 2
922 .na
923 \fB\FB:LGRP_CONSTANTS\FR\FR
924 .ad
925 .RS 19n
926 \fB\FB:LGRP_AFF_NONE\FR, \fB\FB:LGRP_AFF_STRONG\FR, \fB\FB:LGRP_AFF_WEAK\FR,
927 \fB\FB:LGRP_CONTENT_DIRECT\FR, \fB\FB:LGRP_CONTENT_HIERARCHY\FR,
928 \fB\FB:LGRP_MEM_SZ_FREE\FR, \fB\FB:LGRP_MEM_SZ_INSTALLED\FR, \fB\FB:LGRP_VER_CURRENT\FR,
929 \fB\FB:LGRP_VER_NONE\FR, \fB\FB:LGRP_VIEW_CALLER\FR, \fB\FB:LGRP_VIEW_OS\FR,
930 \fB\FB:LGRP_NONE\FR, \fB\FB:LGRP_RSRC_CPU\FR, \fB\FB:LGRP_RSRC_MEM\FR,
931 \fB\FB:LGRP_CONTENT_ALL\FR, \fB\FB:LGRP_LAT_CPU_TO_MEM\FR
932 .RE
```

```
934 .sp
935 .ne 2
936 .na
937 \fB\FB:PROC_CONSTANTS\FR\FR
938 .ad
939 .RS 19n
940 \fB\FB:BP_PID\FR, \fB\FB:BP_LWPID\FR, \fB\FB:BP_MYID\FR
941 .RE
```

```
943 .sp
944 .ne 2
945 .na
946 \fB\FB:CONSTANTS\FR\FR
947 .ad
948 .RS 19n
949 \fB\FB:LGRP_CONSTANTS\FR, \fB\FB:PROC_CONSTANTS\FR
950 .RE
```

```
952 .sp
953 .ne 2
954 .na
955 \fB\FB:FUNCTIONS\FR\FR
956 .ad
957 .RS 19n
958 \fB\FB:Lgrp_affinity_get()\FR, \fB\FB:Lgrp_affinity_set()\FR, \fB\FB:Lgrp_children()\FR,
959 \fB\FB:Lgrp_cookie_stale()\FR, \fB\FB:Lgrp_cpus()\FR, \fB\FB:Lgrp_fini()\FR,
960 \fB\FB:Lgrp_home()\FR, \fB\FB:Lgrp_init()\FR, \fB\FB:Lgrp_latency()\FR,
961 \fB\FB:Lgrp_latency_cookie()\FR, \fB\FB:Lgrp_mem_size()\FR, \fB\FB:Lgrp_nlgrps()\FR,
962 \fB\FB:Lgrp_parents()\FR, \fB\FB:Lgrp_root()\FR, \fB\FB:Lgrp_version()\FR,
963 \fB\FB:Lgrp_view()\FR, \fB\FB:Lgrp_resources()\FR, \fB\FB:Lgrp_lgrps()\FR,
964 \fB\FB:Lgrp_leaves()\FR, \fB\FB:Lgrp_isleaf()\FR
965 .RE
```

```
967 .sp
968 .ne 2
969 .na
970 \fB\FB:ALL\FR\FR
971 .ad
972 .RS 19n
973 \fB\FB:CONSTANTS\FR, \fB\FB:FUNCTIONS\FR
974 .RE
```

```
976 .SS "Error values"
977 .LP
978 The functions in this module return \fB\FB:undef\FR or an empty list when an
979 underlying library function fails. The \fB\FB:$!\FR is set to provide more
980 information values for the error. The following error codes are possible:
```

```
981 .sp
982 .ne 2
983 .na
```

```
984 \fB\FB:EINVAL\FR\FR
985 .ad
986 .RS 10n
987 The value supplied is not valid.
988 .RE
```

```
990 .sp
991 .ne 2
992 .na
993 \fB\FB:BENOMEM\FR\FR
994 .ad
995 .RS 10n
996 There was not enough system memory to complete an operation.
997 .RE
```

```
999 .sp
1000 .ne 2
1001 .na
1002 \fB\FB:EPERM\FR\FR
1003 .ad
1004 .RS 10n
1005 The effective user of the calling process does not have appropriate privileges,
1006 and its real or effective user ID does not match the real or effective user ID
1007 of one of the threads.
1008 .RE
```

```
1010 .sp
1011 .ne 2
1012 .na
1013 \fB\FB:ESRCH\FR\FR
1014 .ad
1015 .RS 10n
1016 The specified process or thread was not found.
1017 .RE
```

```
1019 .SS "Difference in the API versions"
1020 .LP
1021 The \fB\FB:lgrp(3LIB) library is versioned. The exact version that was used
1022 to compile a module is available through the \fB\FB:Lgrp_version()\FR function.
1023 .sp
1024 .LP
1025 Version 2 of the \fB\FB:Lgrp_user\FR API introduced the following constants and
1026 functions not present in version 1:
1027 .br
1028 .in +2
1029 \fB\FB:LGRP_RSRC_CPU\FR constant
1030 .in -2
1031 .br
1032 .in +2
1033 \fB\FB:LGRP_RSRC_MEM\FR constant
1034 .in -2
1035 .br
1036 .in +2
1037 \fB\FB:LGRP_CONTENT_ALL\FR constant
1038 .in -2
1039 .br
1040 .in +2
1041 \fB\FB:LGRP_LAT_CPU_TO_MEM\FR constant
1042 .in -2
1043 .br
1044 .in +2
1045 \fB\FB:Lgrp_resources()\FR function
1046 .in -2
1047 .br
1048 .in +2
1049 \fB\FB:Lgrp_latency_cookie()\FR function
```

```
1050 .in -2
1051 .sp
1052 .LP
1053 The \fBLGRP_RSRC_CPU\fR and \fBLGRP_RSRC_MEM\fR constants are not defined for
1054 version 1. The \fBlgrp_resources()\fR function is defined for version 1 but
1055 always returns an empty list. The \fBlgrp_latency_cookie()\fR function is an
1056 alias for \fBlgrp_latency()\fR for version 1.
1057 .SH ATTRIBUTES
1058 .LP
1059 See \fBattributes\fR(5) for descriptions of the following attributes:
1060 .sp
1062 .sp
1063 .TS
1064 box:
1065 c | c
1066 l | l .
1067 ATTRIBUTE TYPE ATTRIBUTE VALUE
1068 _
1069 Interface Stability Unstable
1070 .TE
1072 .SH SEE ALSO
1073 .LP
1074 \fBlgrp_affinity_get\fR(3LGRP), \fBlgrp_affinity_set\fR(3LGRP),
1075 \fBlgrp_children\fR(3LGRP), \fBlgrp_cookie_stale\fR(3LGRP),
1076 \fBlgrp_cpus\fR(3LGRP), \fBlgrp_fini\fR(3LGRP), \fBlgrp_home\fR(3LGRP),
1077 \fBlgrp_init\fR(3LGRP), \fBlgrp_latency\fR(3LGRP),
1078 \fBlgrp_latency_cookie\fR(3LGRP), \fBlgrp_mem_size\fR(3LGRP),
1079 \fBlgrp_nlgrps\fR(3LGRP), \fBlgrp_parents\fR(3LGRP),
1080 \fBlgrp_resources\fR(3LGRP), \fBlgrp_root\fR(3LGRP), \fBlgrp_version\fR(3LGRP),
1081 \fBlgrp_view\fR(3LGRP), \fBlblgrp\fR(3LIB), \fBattributes\fR(5)
```

```

*****
85737 Tue Dec 11 09:48:09 2018
new/usr/src/man/man3tecla/gl_get_line.3tecla
10057 Man page misspellings output particular overridden
Reviewed by: Gerg Mihajly Doma <domag02@gmail.com>
*****
unchanged portion omitted
308 .fi
309 .in -2

311 .sp
312 .LP
313 In the example, first the resources needed by the \fBgl_get_line()\fR function
314 are created by calling \fBnew_GetLine()\fR. This allocates the memory used in
315 subsequent calls to the \fBgl_get_line()\fR function, including the history
316 buffer for recording previously entered lines. Then one or more lines are read
317 from the user, until either an error occurs, or the user types exit. Then
318 finally the resources that were allocated by \fBnew_GetLine()\fR, are returned
319 to the system by calling \fBdel_GetLine()\fR. Note the use of the \fBINULL\fR
320 return value of \fBdel_GetLine()\fR to make \fIgl\fR \fBINULL\fR. This is a
321 safety precaution. If the program subsequently attempts to pass \fIgl\fR to
322 \fBgl_get_line()\fR, said function will complain, and return an error, instead
323 of attempting to use the deleted resource object.
324 .SS "The Functions Used In The Example"
325 .LP
326 The \fBnew_GetLine()\fR function creates the resources used by the
327 \fBgl_get_line()\fR function and returns an opaque pointer to the object that
328 contains them. The maximum length of an input line is specified by the
329 \fIlinelen\fR argument, and the number of bytes to allocate for storing history
330 lines is set by the \fIhistlen\fR argument. History lines are stored
331 back-to-back in a single buffer of this size. Note that this means that the
332 number of history lines that can be stored at any given time, depends on the
333 lengths of the individual lines. If you want to place an upper limit on the
334 number of lines that can be stored, see the description of the
335 \fBgl_limit_history()\fR function. If you do not want history at all, specify
336 \fIhistlen\fR as zero, and no history buffer will be allocated.
337 .sp
338 .LP
339 On error, a message is printed to \fBstderr\fR and \fBINULL\fR is returned.
340 .sp
341 .LP
342 The \fBdel_GetLine()\fR function deletes the resources that were returned by a
343 previous call to \fBnew_GetLine()\fR. It always returns \fBINULL\fR (for
344 example, a deleted object). It does nothing if the \fIgl\fR argument is
345 \fBINULL\fR.
346 .sp
347 .LP
348 The \fBgl_get_line()\fR function can be called any number of times to read
349 input from the user. The \fIgl\fR argument must have been previously returned by a
350 call to \fBnew_GetLine()\fR. The \fIprompt\fR argument should be a normal
351 null-terminated string, specifying the prompt to present the user with. By
352 default prompts are displayed literally, but if enabled with the
353 \fBgl_prompt_style()\fR function, prompts can contain directives to do
354 underlining, switch to and from bold fonts, or turn highlighting on and off.
355 .sp
356 .LP
357 If you want to specify the initial contents of the line for the user to edit,
358 pass the desired string with the \fIstart_line\fR argument. You can then
359 specify which character of this line the cursor is initially positioned over by
360 using the \fIstart_pos\fR argument. This should be -1 if you want the cursor to
361 follow the last character of the start line. If you do not want to preload the
362 line in this manner, send \fIstart_line\fR as \fBINULL\fR, and set
363 \fIstart_pos\fR to -1.
364 .sp
365 .LP
366 The \fBgl_get_line()\fR function returns a pointer to the line entered by the

```

```

367 user, or \fBINULL\fR on error or at the end of the input. The returned pointer
368 is part of the specified \fIgl\fR resource object, and thus should not be freed
369 by the caller, or assumed to be unchanging from one call to the next. When
370 reading from a user at a terminal, there will always be a newline character at
371 the end of the returned line. When standard input is being taken from a pipe or
372 a file, there will similarly be a newline unless the input line was too long to
373 store in the internal buffer. In the latter case you should call
374 \fBgl_get_line()\fR again to read the rest of the line. Note that this behavior
375 makes \fBgl_get_line()\fR similar to \fBfgets(3C)\fR. When \fBstdin\fR is not
376 connected to a terminal, \fBgl_get_line()\fR simply calls \fBfgets()\fR.
377 .SS "The Return Status Of \fBgl_get_line()\fR"
378 .LP
379 The \fBgl_get_line()\fR function has two possible return values: a pointer to
380 the completed input line, or \fBINULL\fR. Additional information about what
381 caused \fBgl_get_line()\fR to return is available both by inspecting
382 \fBerrno\fR and by calling the \fBgl_return_status()\fR function.
383 .sp
384 .LP
385 The following are the possible enumerated values returned by
386 \fBgl_return_status()\fR:
387 .sp
388 .ne 2
389 .na
390 \fBFBGLR_NEWLINE\fR
391 .ad
392 .RS 15n
393 The last call to \fBgl_get_line()\fR successfully returned a completed input
394 line.
395 .RE

397 .sp
398 .ne 2
399 .na
400 \fBFBGLR_BLOCKED\fR
401 .ad
402 .RS 15n
403 The \fBgl_get_line()\fR function was in non-blocking server mode, and returned
404 early to avoid blocking the process while waiting for terminal I/O. The
405 \fBgl_pending_io()\fR function can be used to see what type of I/O
406 \fBgl_get_line()\fR was waiting for. See the \fBgl_io_mode(3TECLA)\fR.
407 .RE

409 .sp
410 .ne 2
411 .na
412 \fBFBGLR_SIGNAL\fR
413 .ad
414 .RS 15n
415 A signal was caught by \fBgl_get_line()\fR that had an after-signal disposition
416 of \fBGLS_ABORT\fR. See \fBgl_trap_signal()\fR.
417 .RE

419 .sp
420 .ne 2
421 .na
422 \fBFBGLR_TIMEOUT\fR
423 .ad
424 .RS 15n
425 The inactivity timer expired while \fBgl_get_line()\fR was waiting for input,
426 and the timeout callback function returned \fBGLTO_ABORT\fR. See
427 \fBgl_inactivity_timeout()\fR for information about timeouts.
428 .RE

430 .sp
431 .ne 2
432 .na

```



```

433 \fB\fBGLR_FDABORT\fR\fR
434 .ad
435 .RS 15n
436 An application I/O callback returned \fBGLFD_ABORT\fR. Ssee
437 \fBgl_watch_fd()\fR.
438 .RE

440 .sp
441 .ne 2
442 .na
443 \fB\fBGLR_EOF\fR\fR
444 .ad
445 .RS 15n
446 End of file reached. This can happen when input is coming from a file or a
447 pipe, instead of the terminal. It also occurs if the user invokes the
448 list-or-eof or del-char-or-list-or-eof actions at the start of a new line.
449 .RE

451 .sp
452 .ne 2
453 .na
454 \fB\fBGLR_ERROR\fR\fR
455 .ad
456 .RS 15n
457 An unexpected error caused \fBgl_get_line()\fR to abort (consult \fBerrno\fR
458 and/or \fBgl_error_message()\fR for details.
459 .RE

461 .sp
462 .LP
463 When \fBgl_return_status()\fR returns \fBGLR_ERROR\fR and the value of
464 \fBerrno\fR is not sufficient to explain what happened, you can use the
465 \fBgl_error_message()\fR function to request a description of the last error
466 that occurred.
467 .sp
468 .LP
469 The return value of \fBgl_error_message()\fR is a pointer to the message that
470 occurred. If the \fBgl_error_message()\fR argument is \fBNULL\fR, this will be a pointer to a
471 buffer within \fBgl_get_line()\fR whose value will probably change on the next call to any
472 function associated with \fBgl_get_line()\fR. Otherwise, if a non-null
473 \fBgl_error_message()\fR argument is provided, the error message, including a '\e0'
474 terminator, will be written within the first \fBgl_error_message()\fR elements of this buffer,
475 and the return value will be a pointer to the first element of this buffer. If
476 the message will not fit in the provided buffer, it will be truncated to fit.
477 .SS "Optional Prompt Formatting"
478 .LP
479 Whereas by default the prompt string that you specify is displayed literally
480 without any special interpretation of the characters within it, the
481 \fBgl_prompt_style()\fR function can be used to enable optional formatting
482 directives within the prompt.
483 .sp
484 .LP
485 The \fBgl_prompt_style()\fR argument, which specifies the formatting style, can take any of
486 the following values:
487 .sp
488 .ne 2
489 .na
490 \fB\fBGL_FORMAT_PROMPT\fR\fR
491 .ad
492 .RS 21n
493 In this style, the formatting directives described below, when included in
494 prompt strings, are interpreted as follows:
495 .sp
496 .ne 2
497 .na
498 \fB\fBGL_FORMAT_PROMPT\fR\fR

```

```

499 .ad
500 .RS 6n
501 Display subsequent characters with a bold font.
502 .RE

504 .sp
505 .ne 2
506 .na
507 \fB\fBGL_FORMAT_BOLD\fR\fR
508 .ad
509 .RS 6n
510 Stop displaying characters with the bold font.
511 .RE

513 .sp
514 .ne 2
515 .na
516 \fB\fBGL_FORMAT_FLASH\fR\fR
517 .ad
518 .RS 6n
519 Make subsequent characters flash.
520 .RE

522 .sp
523 .ne 2
524 .na
525 \fB\fBGL_FORMAT_FLASH\fR\fR
526 .ad
527 .RS 6n
528 Turn off flashing characters.
529 .RE

531 .sp
532 .ne 2
533 .na
534 \fB\fBGL_FORMAT_UNDERLINE\fR\fR
535 .ad
536 .RS 6n
537 Underline subsequent characters.
538 .RE

540 .sp
541 .ne 2
542 .na
543 \fB\fBGL_FORMAT_UNDERLINE\fR\fR
544 .ad
545 .RS 6n
546 Stop underlining characters.
547 .RE

549 .sp
550 .ne 2
551 .na
552 \fB\fBGL_FORMAT_PALE\fR\fR
553 .ad
554 .RS 6n
555 Switch to a pale (half brightness) font.
556 .RE

558 .sp
559 .ne 2
560 .na
561 \fB\fBGL_FORMAT_PALE\fR\fR
562 .ad
563 .RS 6n
564 Stop using the pale font.

```

```

565 .RE

567 .sp
568 .ne 2
569 .na
570 \fB\fB%S\fR\fR
571 .ad
572 .RS 6n
573 Highlight subsequent characters (also known as standout mode).
574 .RE

576 .sp
577 .ne 2
578 .na
579 \fB\fB%s\fR\fR
580 .ad
581 .RS 6n
582 Stop highlighting characters.
583 .RE

585 .sp
586 .ne 2
587 .na
588 \fB\fB%V\fR\fR
589 .ad
590 .RS 6n
591 Turn on reverse video.
592 .RE

594 .sp
595 .ne 2
596 .na
597 \fB\fB%v\fR\fR
598 .ad
599 .RS 6n
600 Turn off reverse video.
601 .RE

603 .sp
604 .ne 2
605 .na
606 \fB\fB%#\fR\fR
607 .ad
608 .RS 6n
609 Display a single % character.
610 .RE

612 For example, in this mode, a prompt string like "%UOKu$" would display the
613 prompt "OK$", but with the OK part underlined.
614 .sp
615 Note that although a pair of characters that starts with a % character, but
616 does not match any of the above directives is displayed literally, if a new
617 directive is subsequently introduced which does match, the displayed prompt
618 will change, so it is better to always use %% to display a literal %.
619 .sp
620 Also note that not all terminals support all of these text attributes, and that
621 some substitute a different attribute for missing ones.
622 .RE

624 .sp
625 .ne 2
626 .na
627 \fB\fBGL_LITERAL_PROMPT\fR\fR
628 .ad
629 .RS 21n
630 In this style, the prompt string is printed literally. This is the default

```

```

631 style.
632 .RE

634 .SS "Alternate Configuration Sources"
635 .LP
636 By default users have the option of configuring the behavior of
637 \fBgl_get_line()\fR with a configuration file called \fB&.teclarc\fR in their
638 home directories. The fact that all applications share this same configuration
639 file is both an advantage and a disadvantage. In most cases it is an advantage,
640 since it encourages uniformity, and frees the user from having to configure
641 each application separately. In some applications, however, this single means
642 of configuration is a problem. This is particularly true of embedded software,
643 where there's no filesystem to read a configuration file from, and also in
644 applications where a radically different choice of keybindings is needed to
645 emulate a legacy keyboard interface. To cater for such cases, the
646 \fBgl_configure_getline()\fR function allows the application to control where
647 configuration information is read from.
648 .sp
649 .LP
650 The \fBgl_configure_getline()\fR function allows the configuration commands
651 that would normally be read from a user's \fB~/.teclarc\fR file, to be read
652 from any or none of, a string, an application specific configuration file,
653 and/or a user-specific configuration file. If this function is called before
654 the first call to \fBgl_get_line()\fR, the default behavior of reading
655 \fB~/.teclarc\fR on the first call to \fBgl_get_line()\fR is disabled, so all
656 configurations must be achieved using the configuration sources specified with
657 this function.
658 .sp
659 .LP
660 If \fIapp_string\fR != \fINULL\fR, then it is interpreted as a string
661 containing one or more configuration commands, separated from each other in the
662 string by embedded newline characters. If \fIapp_file\fR != \fINULL\fR then it
663 is interpreted as the full pathname of an application-specific configuration
664 file. If user_file != \fINULL\fR then it is interpreted as the full path name
665 of a user-specific configuration file, such as \fB~/.teclarc\fR. For example,
666 in the call
667 .sp
668 .in +2
669 .nf
670 gl_configure_getline(gl, "edit-mode vi \en nobeep",
671                      "/usr/share/myapp/teclarc", "~/.teclarc");
672 .fi
673 .in -2

675 .sp
676 .LP
677 The \fIapp_string\fR argument causes the calling application to start in
678 \fBvi\fR(1) edit-mode, instead of the default \fBemacs\fR mode, and turns off
679 the use of the terminal bell by the library. It then attempts to read
680 system-wide configuration commands from an optional file called
681 \fB/usr/share/myapp/teclarc\fR, then finally reads user-specific configuration
682 commands from an optional \fB&.teclarc\fR file in the user's home directory.
683 Note that the arguments are listed in ascending order of priority, with the
684 contents of \fIapp_string\fR being potentially over ridden by commands in
685 \fIapp_file\fR, and commands in \fIapp_file\fR potentially being overridden by
685 \fIapp_file\fR, and commands in \fIapp_file\fR potentially being overridden by
686 commands in \fIuser_file\fR.
687 .sp
688 .LP
689 You can call this function as many times as needed, the results being
690 cumulative, but note that copies of any file names specified with the
691 \fIapp_file\fR and \fIuser_file\fR arguments are recorded internally for
692 subsequent use by the read-init-files key-binding function, so if you plan to
693 call this function multiple times, be sure that the last call specifies the
694 filenames that you want re-read when the user requests that the configuration
695 files be re-read.

```

```

696 .sp
697 .LP
698 Individual key sequences can also be bound and unbound using the
699 \fBgl_bind_keyseq()\fR function. The \fIorigin\fR argument specifies the
700 priority of the binding, according to whom it is being established for, and
701 must be one of the following two values.
702 .sp
703 .ne 2
704 .na
705 \fB\fbgl_user_key\fR\fR
706 .ad
707 .RS 15n
708 The user requested this key-binding.
709 .RE

711 .sp
712 .ne 2
713 .na
714 \fB\fbgl_app_key\fR\fR
715 .ad
716 .RS 15n
717 This is a default binding set by the application.
718 .RE

720 .sp
721 .LP
722 When both user and application bindings for a given key sequence have been
723 specified, the user binding takes precedence. The application's binding is
724 subsequently reinstated if the user's binding is later unbound with either
725 another call to this function, or a call to \fBgl_configure_getline()\fR.
726 .sp
727 .LP
728 The \fIkeyseq\fR argument specifies the key sequence to be bound or unbound,
729 and is expressed in the same way as in a \fB-/teclarc\fR configuration file.
730 The \fIaction\fR argument must either be a string containing the name of the
731 action to bind the key sequence to, or it must be \fINULL\fR or \fB""\fR to
732 unbind the key sequence.
733 .SS "Customized Word Completion"
734 .LP
735 If in your application you would like to have TAB completion complete other
736 things in addition to or instead of filenames, you can arrange this by
737 registering an alternate completion callback function with a call to the
738 \fBgl_customize_completion()\fR function.
739 .sp
740 .LP
741 The \fIdata\fR argument provides a way for your application to pass arbitrary,
742 application-specific information to the callback function. This is passed to
743 the callback every time that it is called. It might for example point to the
744 symbol table from which possible completions are to be sought. The
745 \fImatch_fn\fR argument specifies the callback function to be called. The
746 \fICplMatchFn\fR function type is defined in <\fBlibtecla.h\fR>, as is a
747 \fBcpl_match_fn()\fR macro that you can use to declare and prototype callback
748 functions. The declaration and responsibilities of callback functions are
749 described in depth on the \fBcpl_complete_word\fR(3TECLA) manual page.
750 .sp
751 .LP
752 The callback function is responsible for looking backwards in the input line
753 from the point at which the user pressed TAB, to find the start of the word
754 being completed. It then must lookup possible completions of this word, and
755 record them one by one in the \fBWordCompletion\fR object that is passed to it
756 as an argument, by calling the \fBcpl_add_completion()\fR function. If the
757 callback function wants to provide filename completion in addition to its own
758 specific completions, it has the option of itself calling the builtin filename
759 completion callback. This also is documented on the
760 \fBcpl_complete_word\fR(3TECLA) manual page.
761 .sp

```

```

762 .LP
763 If you would like \fBgl_get_line()\fR to return the current input line when a
764 successful completion is been made, you can arrange this when you call
765 \fBcpl_add_completion()\fR by making the last character of the continuation
766 suffix a newline character. The input line will be updated to display the
767 completion, together with any continuation suffix up to the newline character,
768 and \fBgl_get_line()\fR will return this input line.
769 .sp
770 .LP
771 If your callback function needs to write something to the terminal, it must
772 call \fBgl_normal_io()\fR before doing so. This will start a new line after the
773 input line that is currently being edited, reinstate normal terminal I/O, and
774 notify \fBgl_get_line()\fR that the input line will need to be redrawn when the
775 callback returns.
776 .SS "Adding Completion Actions"
777 .LP
778 In the previous section the ability to customize the behavior of the only
779 default completion action, complete-word, was described. In this section the
780 ability to install additional action functions, so that different types of word
781 completion can be bound to different key sequences, is described. This is
782 achieved by using the \fBgl_completion_action()\fR function.
783 .sp
784 .LP
785 The \fIdata\fR and \fImatch_fn\fR arguments are as described on the
786 \fBcpl_complete_word\fR(3TECLA) manual page, and specify the callback function
787 that should be invoked to identify possible completions. The \fIlist_only\fR
788 argument determines whether the action that is being defined should attempt to
789 complete the word as far as possible in the input line before displaying any
790 possible ambiguous completions, or whether it should simply display the list of
791 possible completions without touching the input line. The former option is
792 selected by specifying a value of 0, and the latter by specifying a value of 1.
793 The \fIname\fR argument specifies the name by which configuration files and
794 future invocations of this function should refer to the action. This must
795 either be the name of an existing completion action to be changed, or be a new
796 unused name for a new action. Finally, the \fIkeyseq\fR argument specifies the
797 default key sequence to bind the action to. If this is \fINULL\fR, no new key
798 sequence will be bound to the action.
799 .sp
800 .LP
801 Beware that in order for the user to be able to change the key sequence that is
802 bound to actions that are installed in this manner, you should call
803 \fBgl_completion_action()\fR to install a given action for the first time
804 between calling \fBnew_GetLine()\fR and the first call to \fBgl_get_line()\fR.
805 Otherwise, when the user's configuration file is read on the first call to
806 \fBgl_get_line()\fR, the name of the your additional action will not be known,
807 and any reference to it in the configuration file will generate an error.
808 .sp
809 .LP
810 As discussed for \fBgl_customize_completion()\fR, if your callback function
811 needs to write anything to the terminal, it must call \fBgl_normal_io()\fR
812 before doing so.
813 .SS "Defining Custom Actions"
814 .LP
815 Although the built-in key-binding actions are sufficient for the needs of most
816 applications, occasionally a specialized application may need to define one or
817 more custom actions, bound to application-specific key sequences. For example,
818 a sales application would benefit from having a key sequence that displayed the
819 part name that corresponded to a part number preceding the cursor. Such a
820 feature is clearly beyond the scope of the built-in action functions. So for
821 such special cases, the \fBgl_register_action()\fR function is provided.
822 .sp
823 .LP
824 The \fBgl_register_action()\fR function lets the application register an
825 external function, \fIfn\fR, that will thereafter be called whenever either the
826 specified key sequence, \fIkeyseq\fR, is entered by the user, or the user
827 enters any other key sequence that the user subsequently binds to the specified

```

```

828 action name, \fIname\fR, in their configuration file. The \fIdata\fR argument
829 can be a pointer to anything that the application wants to have passed to the
830 action function, \fIfn\fR, whenever that function is invoked.
831 .sp
832 .LP
833 The action function, \fIfn\fR, should be declared using the
834 \fBGL_ACTION_FN()\fR macro, which is defined in <\fBlibtecla.h\fR>.
835 .sp
836 .in +2
837 .nf
838 #define GL_ACTION_FN(fn) GlAfterAction (fn)(GetLine *gl, \e
839         void *data, int count, size_t curpos, \e
840         const char *line)
841 .fi
842 .in -2

844 .sp
845 .LP
846 The \fIgl\fR and \fIdata\fR arguments are those that were previously passed to
847 \fBgl_register_action()\fR when the action function was registered. The
848 \fIcount\fR argument is a numeric argument which the user has the option of
849 entering using the digit-argument action, before invoking the action. If the
850 user does not enter a number, then the \fIcount\fR argument is set to 1.
851 Nominally this argument is interpreted as a repeat count, meaning that the
852 action should be repeated that many times. In practice however, for some
853 actions a repeat count makes little sense. In such cases, actions can either
854 simply ignore the \fIcount\fR argument, or use its value for a different
855 purpose.
856 .sp
857 .LP
858 A copy of the current input line is passed in the read-only \fIline\fR
859 argument. The current cursor position within this string is given by the index
860 contained in the \fIcurpos\fR argument. Note that direct manipulation of the
861 input line and the cursor position is not permitted because the rules dictated
862 by various modes (such as \fBvi\fR mode versus \fBemacs\fR mode, no-echo mode,
863 and insert mode versus overstrike mode) make it too complex for an application
864 writer to write a conforming editing action, as well as constrain future
865 changes to the internals of \fBgl_get_line()\fR. A potential solution to this
866 dilemma would be to allow the action function to edit the line using the
867 existing editing actions. This is currently under consideration.
868 .sp
869 .LP
870 If the action function wishes to write text to the terminal without this
871 getting mixed up with the displayed text of the input line, or read from the
872 terminal without having to handle raw terminal I/O, then before doing either of
873 these operations, it must temporarily suspend line editing by calling the
874 \fBgl_normal_io()\fR function. This function flushes any pending output to the
875 terminal, moves the cursor to the start of the line that follows the last
876 terminal line of the input line, then restores the terminal to a state that is
877 suitable for use with the C \fBstdio\fR facilities. The latter includes such
878 things as restoring the normal mapping of \en to \er\en, and, when in server
879 mode, restoring the normal blocking form of terminal I/O. Having called this
880 function, the action function can read from and write to the terminal without
881 the fear of creating a mess. It is not necessary for the action function to
882 restore the original editing environment before it returns. This is done
883 automatically by \fBgl_get_line()\fR after the action function returns. The
884 following is a simple example of an action function which writes the sentence
885 "Hello world" on a new terminal line after the line being edited. When this
886 function returns, the input line is redrawn on the line that follows the "Hello
887 world" line, and line editing resumes.
888 .sp
889 .in +2
890 .nf
891 static GL_ACTION_FN(say_hello_fn)
892 {
893     if(gl_normal_io(gl)) /* Temporarily suspend editing */

```

```

894         return GLA_ABORT;
895         printf("Hello world\n");
896         return GLA_CONTINUE;
897     }
898 .fi
899 .in -2

901 .sp
902 .LP
903 Action functions must return one of the following values, to tell
904 \fBgl_get_line()\fR how to proceed.
905 .sp
906 .ne 2
907 .na
908 \fBGLA_ABORT\fR
909 .ad
910 .RS 16n
911 Cause \fBgl_get_line()\fR to return \fINULL\fR.
912 .RE

914 .sp
915 .ne 2
916 .na
917 \fBGLA_RETURN\fR
918 .ad
919 .RS 16n
920 Cause \fBgl_get_line()\fR to return the completed input line
921 .RE

923 .sp
924 .ne 2
925 .na
926 \fBGLA_CONTINUE\fR
927 .ad
928 .RS 16n
929 Resume command-line editing.
930 .RE

932 .sp
933 .LP
934 Note that the \fIname\fR argument of \fBgl_register_action()\fR specifies the
935 name by which a user can refer to the action in their configuration file. This
936 allows them to re-bind the action to an alternate key-sequence. In order for
937 this to work, it is necessary to call \fBgl_register_action()\fR between
938 calling \fBnew_GetLine()\fR and the first call to \fBgl_get_line()\fR.
939 .SS "History Files"
940 .LP
941 To save the contents of the history buffer before quitting your application and
942 subsequently restore them when you next start the application, the
943 \fBgl_save_history()\fR and \fBgl_load_history()\fR functions are provided.
944 .sp
945 .LP
946 The \fIfilename\fR argument specifies the name to give the history file when
947 saving, or the name of an existing history file, when loading. This may contain
948 home directory and environment variable expressions, such as
949 \fB~/myapp_history\fR or \fB$HOME/.myapp_history\fR.
950 .sp
951 .LP
952 Along with each history line, additional information about it, such as its
953 nesting level and when it was entered by the user, is recorded as a comment
954 preceding the line in the history file. Writing this as a comment allows the
955 history file to double as a command file, just in case you wish to replay a
956 whole session using it. Since comment prefixes differ in different languages,
957 the comment argument is provided for specifying the comment prefix. For
958 example, if your application were a UNIX shell, such as the Bourne shell, you
959 would specify "#" here. Whatever you choose for the comment character, you must

```

```

960 specify the same prefix to \fBgl_load_history()\fR that you used when you
961 called \fBgl_save_history()\fR to write the history file.
962 .sp
963 .LP
964 The \fImax_lines\fR argument must be either -1 to specify that all lines in the
965 history list be saved, or a positive number specifying a ceiling on how many of
966 the most recent lines should be saved.
967 .sp
968 .LP
969 Both fuctions return non-zero on error, after writing an error message to
970 \fBstderr\fR. Note that \fBgl_load_history()\fR does not consider the
971 non-existence of a file to be an error.
972 .SS "Multiple History Lists"
973 .LP
974 If your application uses a single \fBGetLine\fR object for entering many
975 different types of input lines, you might want \fBgl_get_line()\fR to
976 distinguish the different types of lines in the history list, and only recall
977 lines that match the current type of line. To support this requirement,
978 \fBgl_get_line()\fR marks lines being recorded in the history list with an
979 integer identifier chosen by the application. Initially this identifier is set
980 to 0 by \fBnew_GetLine()\fR, but it can be changed subsequently by calling
981 \fBgl_group_history()\fR.
982 .sp
983 .LP
984 The integer identifier ID can be any number chosen by the application, but note
985 that \fBgl_save_history()\fR and \fBgl_load_history()\fR preserve the
986 association between identifiers and historical input lines between program
987 invocations, so you should choose fixed identifiers for the different types of
988 input line used by your application.
989 .sp
990 .LP
991 Whenever \fBgl_get_line()\fR appends a new input line to the history list, the
992 current history identifier is recorded with it, and when it is asked to recall
993 a historical input line, it only recalls lines that are marked with the current
994 identifier.
995 .SS "Displaying History"
996 .LP
997 The history list can be displayed by calling \fBgl_show_history()\fR. This
998 function displays the current contents of the history list to the \fBstdio\fR
999 output stream \fIfp\fR. If the \fImax_lines\fR argument is greater than or
1000 equal to zero, then no more than this number of the most recent lines will be
1001 displayed. If the \fIall_groups\fR argument is non-zero, lines from all history
1002 groups are displayed. Otherwise only those of the currently selected history
1003 group are displayed. The format string argument, \fIfmt\fR, determines how the
1004 line is displayed. This can contain arbitrary characters which are written
1005 verbatim, interleaved with any of the following format directives:
1006 .sp
1007 .ne 2
1008 .na
1009 \fB\F%D\fR
1010 .ad
1011 .RS 6n
1012 The date on which the line was originally entered, formatted like 2001-11-20.
1013 .RE

```

```

1015 .sp
1016 .ne 2
1017 .na
1018 \fB\F%T\fR
1019 .ad
1020 .RS 6n
1021 The time of day when the line was entered, formatted like 23:59:59.
1022 .RE

```

```

1024 .sp
1025 .ne 2

```

```

1026 .na
1027 \fB\F%N\fR
1028 .ad
1029 .RS 6n
1030 The sequential entry number of the line in the history buffer.
1031 .RE

```

```

1033 .sp
1034 .ne 2
1035 .na
1036 \fB\F%G\fR
1037 .ad
1038 .RS 6n
1039 The number of the history group which the line belongs to.
1040 .RE

```

```

1042 .sp
1043 .ne 2
1044 .na
1045 \fB\F%#\fR
1046 .ad
1047 .RS 6n
1048 A literal % character.
1049 .RE

```

```

1051 .sp
1052 .ne 2
1053 .na
1054 \fB\F%H\fR
1055 .ad
1056 .RS 6n
1057 The history line itself.
1058 .RE

```

```

1060 .sp
1061 .LP
1062 Thus a format string like "%D %T %H0" would output something like:
1063 .sp
1064 .in +2
1065 .nf
1066 2001-11-20 10:23:34 Hello world
1067 .fi
1068 .in -2

```

```

1070 .sp
1071 .LP
1072 Note the inclusion of an explicit newline character in the format string.
1073 .SS "Looking Up History"
1074 .LP
1075 The \fBgl_lookup_history()\fR function allows the calling application to look
1076 up lines in the history list.
1077 .sp
1078 .LP
1079 The \fIid\fR argument indicates which line to look up, where the first line
1080 that was entered in the history list after \fBnew_GetLine()\fR was called is
1081 denoted by 0, and subsequently entered lines are denoted with successively
1082 higher numbers. Note that the range of lines currently preserved in the history
1083 list can be queried by calling the \fBgl_range_of_history()\fR function. If the
1084 requested line is in the history list, the details of the line are recorded in
1085 the variable pointed to by the \fIhline\fR argument, and 1 is returned.
1086 Otherwise 0 is returned, and the variable pointed to by \fIhline\fR is left
1087 unchanged.
1088 .sp
1089 .LP
1090 Beware that the string returned in \fIhline\fR->\fIline\fR is part of the
1091 history buffer, so it must not be modified by the caller, and will be recycled

```

1092 on the next call to any function that takes `\fIgl`\fR as its argument. Therefore
 1093 you should make a private copy of this string if you need to keep it.
 1094 .SS "Manual History Archival"
 1095 .LP
 1096 By default, whenever a line is entered by the user, it is automatically
 1097 appended to the history list, just before `\fBgl_get_line()`\fR returns the line
 1098 to the caller. This is convenient for the majority of applications, but there
 1099 are also applications that need finer-grained control over what gets added to
 1100 the history list. In such cases, the automatic addition of entered lines to the
 1101 history list can be turned off by calling the `\fBgl_automatic_history()`\fR
 1102 function.
 1103 .sp
 1104 .LP
 1105 If this function is called with its `\fIenable`\fR argument set to 0,
 1106 `\fBgl_get_line()`\fR will not automatically archive subsequently entered lines.
 1107 Automatic archiving can be reenabled at a later time by calling this function
 1108 again, with its `\fIenable`\fR argument set to 1. While automatic history
 1109 archiving is disabled, the calling application can use the
 1110 `\fBgl_append_history()`\fR to append lines to the history list as needed.
 1111 .sp
 1112 .LP
 1113 The `\fIline`\fR argument specifies the line to be added to the history list.
 1114 This must be a normal `'\e0 '` terminated string. If this string contains any
 1115 newline characters, the line that gets archived in the history list will be
 1116 terminated by the first of these. Otherwise it will be terminated by the `'\e0 '`
 1117 terminator. If the line is longer than the maximum input line length that was
 1118 specified when `\fBnew_GetLine()`\fR was called, it will be truncated to the
 1119 actual `\fBgl_get_line()`\fR line length when the line is recalled.
 1120 .sp
 1121 .LP
 1122 If successful, `\fBgl_append_history()`\fR returns 0. Otherwise it returns
 1123 non-zero and sets `\fBerrno`\fR to one of the following values.
 1124 .sp
 1125 .ne 2
 1126 .na
 1127 `\fB\FBEINVAL`\fR
 1128 .ad
 1129 .RS 10n
 1130 One of the arguments passed to `\fBgl_append_history()`\fR was `\fINULL`\fR.
 1131 .RE
 1132 .sp
 1133 .sp
 1134 .ne 2
 1135 .na
 1136 `\fB\FBENOMEM`\fR
 1137 .ad
 1138 .RS 10n
 1139 The specified line was longer than the allocated size of the history buffer (as
 1140 specified when `\fBnew_GetLine()`\fR was called), so it could not be archived.
 1141 .RE
 1142 .sp
 1143 .sp
 1144 .LP
 1145 A textual description of the error can optionally be obtained by calling
 1146 `\fBgl_error_message()`\fR. Note that after such an error, the history list
 1147 remains in a valid state to receive new history lines, so there is little harm
 1148 in simply ignoring the return status of `\fBgl_append_history()`\fR.
 1149 .SS "Miscellaneous History Configuration"
 1150 .LP
 1151 If you wish to change the size of the history buffer that was originally
 1152 specified in the call to `\fBnew_GetLine()`\fR, you can do so with the
 1153 `\fBgl_resize_history()`\fR function.
 1154 .sp
 1155 .LP
 1156 The `\fIhistlen`\fR argument specifies the new size in bytes, and if you specify
 1157 this as 0, the buffer will be deleted.

1158 .sp
 1159 .LP
 1160 As mentioned in the discussion of `\fBnew_GetLine()`\fR, the number of lines that
 1161 can be stored in the history buffer, depends on the lengths of the individual
 1162 lines. For example, a 1000 byte buffer could equally store 10 lines of average
 1163 length 100 bytes, or 20 lines of average length 50 bytes. Although the buffer
 1164 is never expanded when new lines are added, a list of pointers into the buffer
 1165 does get expanded when needed to accommodate the number of lines currently
 1166 stored in the buffer. To place an upper limit on the number of lines in the
 1167 buffer, and thus a ceiling on the amount of memory used in this list, you can
 1168 call the `\fBgl_limit_history()`\fR function.
 1169 .sp
 1170 .LP
 1171 The `\fImax_lines`\fR should either be a positive number ≥ 0 , specifying an
 1172 upper limit on the number of lines in the buffer, or be -1 to cancel any
 1173 previously specified limit. When a limit is in effect, only the `\fImax_lines`\fR
 1174 most recently appended lines are kept in the buffer. Older lines are discarded.
 1175 .sp
 1176 .LP
 1177 To discard lines from the history buffer, use the `\fBgl_clear_history()`\fR
 1178 function.
 1179 .sp
 1180 .LP
 1181 The `\fIall_groups`\fR argument tells the function whether to delete just the
 1182 lines associated with the current history group (see `\fBgl_group_history()`\fR)
 1183 or all historical lines in the buffer.
 1184 .sp
 1185 .LP
 1186 The `\fBgl_toggle_history()`\fR function allows you to toggle history on and off
 1187 without losing the current contents of the history list.
 1188 .sp
 1189 .LP
 1190 Setting the `\fIenable`\fR argument to 0 turns off the history mechanism, and
 1191 setting it to 1 turns it back on. When history is turned off, no new lines will
 1192 be added to the history list, and history lookup key-bindings will act as
 1193 though there is nothing in the history buffer.
 1194 .SS "Querying History Information"
 1195 .LP
 1196 The configured state of the history list can be queried with the
 1197 `\fBgl_history_state()`\fR function. On return, the status information is
 1198 recorded in the variable pointed to by the `\fIstate`\fR argument.
 1199 .sp
 1200 .LP
 1201 The `\fBgl_range_of_history()`\fR function returns the number and range of lines
 1202 in the history list. The return values are recorded in the variable pointed to
 1203 by the range argument. If the `\fInlines`\fR member of this structure is greater
 1204 than zero, then the oldest and newest members report the range of lines in the
 1205 list, and `\fInewest`\fR=`\fIoldest`\fR+`\fInlines`\fR-1. Otherwise they are both
 1206 zero.
 1207 .sp
 1208 .LP
 1209 The `\fBgl_size_of_history()`\fR function returns the total size of the history
 1210 buffer and the amount of the buffer that is currently occupied.
 1211 .sp
 1212 .LP
 1213 On return, the size information is recorded in the variable pointed to by the
 1214 `\fIsize`\fR argument.
 1215 .SS "Changing Terminals"
 1216 .LP
 1217 The `\fBnew_GetLine()`\fR constructor function assumes that input is to be read
 1218 from `\fBstdin`\fR and output written to `\fBstdout`\fR. The following function
 1219 allows you to switch to different input and output streams.
 1220 .sp
 1221 .LP
 1222 The `\fIgl`\fR argument is the object that was returned by `\fBnew_GetLine()`\fR.
 1223 The `\fIinput_fp`\fR argument specifies the stream to read from, and

1224 `\fIoutput_fp`\fR specifies the stream to be written to. Only if both of these
 1225 refer to a terminal, will interactive terminal input be enabled. Otherwise
 1226 `\fBgl_get_line()`\fR will simply call `\fBfgets()`\fR to read command input. If
 1227 both streams refer to a terminal, then they must refer to the same terminal,
 1228 and the type of this terminal must be specified with the `\fIterm`\fR argument.
 1229 The value of the `\fIterm`\fR argument is looked up in the terminal information
 1230 database (`\fBterminfo`\fR or `\fBtermcap`\fR), in order to determine which special
 1231 control sequences are needed to control various aspects of the terminal.
 1232 `\fBnew_GetLine()`\fR for example, passes the return value of
 1233 `\fBgetenv`\fR("TERM") in this argument. Note that if one or both of
 1234 `\fIinput_fp`\fR and `\fIoutput_fp`\fR do not refer to a terminal, then it is legal
 1235 to pass `\fINULL`\fR instead of a terminal type.
 1236 .sp
 1237 .LP
 1238 Note that if you want to pass file descriptors to `\fBgl_change_terminal()`\fR,
 1239 you can do this by creating `\fBstdio`\fR stream wrappers using the POSIX
 1240 `\fBfdopen`\fR(3C) function.
 1241 .SS "External Event Handling"
 1242 .LP
 1243 By default, `\fBgl_get_line()`\fR does not return until either a complete input
 1244 line has been entered by the user, or an error occurs. In programs that need to
 1245 watch for I/O from other sources than the terminal, there are two options.
 1246 .RS +4
 1247 .TP
 1248 .ie t \(\bu
 1249 .el o
 1250 Use the functions described in the `\fBgl_io_mode`\fR(3TECLA) manual page to
 1251 switch `\fBgl_get_line()`\fR into non-blocking server mode. In this mode,
 1252 `\fBgl_get_line()`\fR becomes a non-blocking, incremental line-editing function
 1253 that can safely be called from an external event loop. Although this is a very
 1254 versatile method, it involves taking on some responsibilities that are normally
 1255 performed behind the scenes by `\fBgl_get_line()`\fR.
 1256 .RE
 1257 .RS +4
 1258 .TP
 1259 .ie t \(\bu
 1260 .el o
 1261 While `\fBgl_get_line()`\fR is waiting for keyboard input from the user, you can
 1262 ask it to also watch for activity on arbitrary file descriptors, such as
 1263 network sockets or pipes, and have it call functions of your choosing when
 1264 activity is seen. This works on any system that has the `select` system call,
 1265 which is most, if not all flavors of UNIX.
 1266 .RE
 1267 .sp
 1268 .LP
 1269 Registering a file descriptor to be watched by `\fBgl_get_line()`\fR involves
 1270 calling the `\fBgl_watch_fd()`\fR function. If this returns non-zero, then it
 1271 means that either your arguments are invalid, or that this facility is not
 1272 supported on the host system.
 1273 .sp
 1274 .LP
 1275 The `\fIfd`\fR argument is the file descriptor to be watched. The event argument
 1276 specifies what type of activity is of interest, chosen from the following
 1277 enumerated values:
 1278 .sp
 1279 .ne 2
 1280 .na
 1281 `\fBFBGLFD_READ`\fR
 1282 .ad
 1283 .RS 15n
 1284 Watch for the arrival of data to be read.
 1285 .RE
 1287 .sp
 1288 .ne 2
 1289 .na

1290 `\fBFBGLFD_WRITE`\fR
 1291 .ad
 1292 .RS 15n
 1293 Watch for the ability to write to the file descriptor without blocking.
 1294 .RE
 1296 .sp
 1297 .ne 2
 1298 .na
 1299 `\fBFBGLFD_URGENT`\fR
 1300 .ad
 1301 .RS 15n
 1302 Watch for the arrival of urgent out-of-band data on the file descriptor.
 1303 .RE
 1305 .sp
 1306 .LP
 1307 The `\fIcallback`\fR argument is the function to call when the selected activity
 1308 is seen. It should be defined with the following macro, which is defined in
 1309 `libtecla.h`.
 1310 .sp
 1311 .in +2
 1312 .nf
 1313 #define GL_FD_EVENT_FN(fn) GlFdStatus (fn)(GetLine *gl, \
 1314 void *data, int fd, GlFdEvent event)
 1315 .fi
 1316 .in -2
 1318 .sp
 1319 .LP
 1320 The data argument of the `\fBgl_watch_fd()`\fR function is passed to the callback
 1321 function for its own use, and can point to anything you like, including
 1322 `\fINULL`\fR. The file descriptor and the event argument are also passed to the
 1323 callback function, and this potentially allows the same callback function to be
 1324 registered to more than one type of event and/or more than one file descriptor.
 1325 The return value of the callback function should be one of the following
 1326 values.
 1327 .sp
 1328 .ne 2
 1329 .na
 1330 `\fBFBGLFD_ABORT`\fR
 1331 .ad
 1332 .RS 17n
 1333 Tell `\fBgl_get_line()`\fR to abort. When this happens, `\fBgl_get_line()`\fR
 1334 returns `\fINULL`\fR, and a following call to `\fBgl_return_status()`\fR will
 1335 return `\fBGLR_FDABORT`\fR. Note that if the application needs `\fBerrno`\fR always
 1336 to have a meaningful value when `\fBgl_get_line()`\fR returns `\fINULL`\fR, the
 1337 callback function should set `\fBerrno`\fR appropriately.
 1338 .RE
 1340 .sp
 1341 .ne 2
 1342 .na
 1343 `\fBFBGLFD_REFRESH`\fR
 1344 .ad
 1345 .RS 17n
 1346 Redraw the input line then continue waiting for input. Return this if your
 1347 callback wrote to the terminal.
 1348 .RE
 1350 .sp
 1351 .ne 2
 1352 .na
 1353 `\fBFBGLFD_CONTINUE`\fR
 1354 .ad
 1355 .RS 17n

```

1356 Continue to wait for input, without redrawing the line.
1357 .RE

1359 .sp
1360 .LP
1361 Note that before calling the callback, \fBgl_get_line()\fR blocks most signals
1362 and leaves its own signal handlers installed, so if you need to catch a
1363 particular signal you will need to both temporarily install your own signal
1364 handler, and unblock the signal. Be sure to re-block the signal (if it was
1365 originally blocked) and reinstate the original signal handler, if any, before
1366 returning.
1367 .sp
1368 .LP
1369 Your callback should not try to read from the terminal, which is left in raw
1370 mode as far as input is concerned. You can write to the terminal as usual,
1371 since features like conversion of newline to carriage-return/linefeed are
1372 re-enabled while the callback is running. If your callback function does write
1373 to the terminal, be sure to output a newline first, and when your callback
1374 returns, tell \fBgl_get_line()\fR that the input line needs to be redrawn, by
1375 returning the \fBGLFD_REFRESH\fR status code.
1376 .sp
1377 .LP
1378 To remove a callback function that you previously registered for a given file
1379 descriptor and event, simply call \fBgl_watch_fd()\fR with the same \fIfd\fR
1380 and \fIevent\fR arguments, but with a \fIcallback\fR argument of 0. The
1381 \fIdata\fR argument is ignored in this case.
1382 .SS "Setting An Inactivity Timeout"
1383 .LP
1384 The \fBgl_inactivity_timeout()\fR function can be used to set or cancel an
1385 inactivity timeout. Inactivity in this case refers both to keyboard input, and
1386 to I/O on any file descriptors registered by prior and subsequent calls to
1387 \fBgl_watch_fd()\fR.
1388 .sp
1389 .LP
1390 The timeout is specified in the form of an integral number of seconds and an
1391 integral number of nanoseconds, specified by the \fIsec\fR and \fInsec\fR
1392 arguments, respectively. Subsequently, whenever no activity is seen for this
1393 time period, the function specified by the \fIcallback\fR argument is called.
1394 The \fIdata\fR argument of \fBgl_inactivity_timeout()\fR is passed to this
1395 callback function whenever it is invoked, and can thus be used to pass
1396 arbitrary application-specific information to the callback. The following macro
1397 is provided in <\fBlibtecla.h\fR> for applications to use to declare and
1398 prototype timeout callback functions.
1399 .sp
1400 .in +2
1401 .nf
1402 #define GL_TIMEOUT_FN(fn) GAfterTimeout (fn)(GetLine *gl, void *data)
1403 .fi
1404 .in -2

1406 .sp
1407 .LP
1408 On returning, the application's callback is expected to return one of the
1409 following enumerators to tell \fBgl_get_line()\fR how to proceed after the
1410 timeout has been handled by the callback.
1411 .sp
1412 .ne 2
1413 .na
1414 \fB\FBGLTO_ABORT\fR
1415 .ad
1416 .RS 17n
1417 Tell \fBgl_get_line()\fR to abort. When this happens, \fBgl_get_line()\fR will
1418 return \fFINULL\fR, and a following call to \fBgl_return_status()\fR will return
1419 \fBGLR_TIMEOUT\fR. Note that if the application needs \fBerrno\fR always to
1420 have a meaningful value when \fBgl_get_line()\fR returns \fFINULL\fR, the
1421 callback function should set \fBerrno\fR appropriately.

```

```

1422 .RE

1424 .sp
1425 .ne 2
1426 .na
1427 \fB\FBGLTO_REFRESH\fR
1428 .ad
1429 .RS 17n
1430 Redraw the input line, then continue waiting for input. You should return this
1431 value if your callback wrote to the terminal.
1432 .RE

1434 .sp
1435 .ne 2
1436 .na
1437 \fB\FBGLTO_CONTINUE\fR
1438 .ad
1439 .RS 17n
1440 In normal blocking-I/O mode, continue to wait for input, without redrawing the
1441 user's input line. In non-blocking server I/O mode (see
1442 \fBgl_io_mode\fR(3TECLA)), \fBgl_get_line()\fR acts as though I/O blocked. This
1443 means that \fBgl_get_line()\fR will immediately return \fFINULL\fR, and a
1444 following call to \fBgl_return_status()\fR will return \fBGLR_BLOCKED\fR.
1445 .RE

1447 .sp
1448 .LP
1449 Note that before calling the callback, \fBgl_get_line()\fR blocks most signals
1450 and leaves its own signal handlers installed, so if you need to catch a
1451 particular signal you will need to both temporarily install your own signal
1452 handler and unblock the signal. Be sure to re-block the signal (if it was
1453 originally blocked) and reinstate the original signal handler, if any, before
1454 returning.
1455 .sp
1456 .LP
1457 Your callback should not try to read from the terminal, which is left in raw
1458 mode as far as input is concerned. You can however write to the terminal as
1459 usual, since features like conversion of newline to carriage-return/linefeed
1460 are re-enabled while the callback is running. If your callback function does
1461 write to the terminal, be sure to output a newline first, and when your
1462 callback returns, tell \fBgl_get_line()\fR that the input line needs to be
1463 redrawn, by returning the \fBGLTO_REFRESH\fR status code.
1464 .sp
1465 .LP
1466 Finally, note that although the timeout arguments include a nanosecond
1467 component, few computer clocks presently have resolutions that are finer than a
1468 few milliseconds, so asking for less than a few milliseconds is equivalent to
1469 requesting zero seconds on many systems. If this would be a problem, you should
1470 base your timeout selection on the actual resolution of the host clock (for
1471 example, by calling \fBsysconf\fR(\fB_SC_CLK_TCK\fR)).
1472 .sp
1473 .LP
1474 To turn off timeouts, simply call \fBgl_inactivity_timeout()\fR with a
1475 \fIcallback\fR argument of 0. The \fIdata\fR argument is ignored in this case.
1476 .SS "Signal Handling Defaults"
1477 .LP
1478 By default, the \fBgl_get_line()\fR function intercepts a number of signals.
1479 This is particularly important for signals that would by default terminate the
1480 process, since the terminal needs to be restored to a usable state before this
1481 happens. This section describes the signals that are trapped by default and how
1482 \fBgl_get_line()\fR responds to them. Changing these defaults is the topic of
1483 the following section.
1484 .sp
1485 .LP
1486 When the following subset of signals are caught, \fBgl_get_line()\fR first
1487 restores the terminal settings and signal handling to how they were before

```


1488 \fBgl_get_line()\fR was called, resends the signal to allow the calling
 1489 application's signal handlers to handle it, then, if the process still exists,
 1490 returns \fINULL\fR and sets \fBerrno\fR as specified below.
 1491 .sp
 1492 .ne 2
 1493 .na
 1494 \fB\fBSIGINT\fR\fR
 1495 .ad
 1496 .RS 11n
 1497 This signal is generated both by the keyboard interrupt key (usually \fB^C\fR),
 1498 and the keyboard break key. The \fBerrno\fR value is \fBEBINTR\fR.
 1499 .RE

1501 .sp
 1502 .ne 2
 1503 .na
 1504 \fB\fBSIGHUP\fR\fR
 1505 .ad
 1506 .RS 11n
 1507 This signal is generated when the controlling terminal exits. The \fBerrno\fR
 1508 value is \fBENOTTY\fR.
 1509 .RE

1511 .sp
 1512 .ne 2
 1513 .na
 1514 \fB\fBSIGPIPE\fR\fR
 1515 .ad
 1516 .RS 11n
 1517 This signal is generated when a program attempts to write to a pipe whose
 1518 remote end is not being read by any process. This can happen for example if you
 1519 have called \fBgl_change_terminal()\fR to redirect output to a pipe hidden
 1520 under a pseudo terminal. The \fBerrno\fR value is \fBEBEPIPE\fR.
 1521 .RE

1523 .sp
 1524 .ne 2
 1525 .na
 1526 \fB\fBSIGQUIT\fR\fR
 1527 .ad
 1528 .RS 11n
 1529 This signal is generated by the keyboard quit key (usually \fB^e\fR). The
 1530 \fBerrno\fR value is \fBEBINTR\fR.
 1531 .RE

1533 .sp
 1534 .ne 2
 1535 .na
 1536 \fB\fBSIGABRT\fR\fR
 1537 .ad
 1538 .RS 11n
 1539 This signal is generated by the standard C, abort function. By default it both
 1540 terminates the process and generates a core dump. The \fBerrno\fR value is
 1541 \fBEBINTR\fR.
 1542 .RE

1544 .sp
 1545 .ne 2
 1546 .na
 1547 \fB\fBSIGTERM\fR\fR
 1548 .ad
 1549 .RS 11n
 1550 This is the default signal that the UNIX kill command sends to processes. The
 1551 \fBerrno\fR value is \fBEBINTR\fR.
 1552 .RE

1554 .sp
 1555 .LP
 1556 Note that in the case of all of the above signals, POSIX mandates that by
 1557 default the process is terminated, with the addition of a core dump in the case
 1558 of the \fBSIGQUIT\fR signal. In other words, if the calling application does
 1559 not override the default handler by supplying its own signal handler, receipt
 1560 of the corresponding signal will terminate the application before
 1561 \fBgl_get_line()\fR returns.
 1562 .sp
 1563 .LP
 1564 If \fBgl_get_line()\fR aborts with \fBerrno\fR set to \fBEBINTR\fR, you can find
 1565 out what signal caused it to abort, by calling the \fBgl_last_signal()\fR
 1566 function. This returns the numeric code (for example, \fBSIGINT\fR) of the last
 1567 signal that was received during the most recent call to \fBgl_get_line()\fR, or
 1568 -1 if no signals were received.
 1569 .sp
 1570 .LP
 1571 On systems that support it, when a \fBSIGWINCH\fR (window change) signal is
 1572 received, \fBgl_get_line()\fR queries the terminal to find out its new size,
 1573 redraws the current input line to accommodate the new size, then returns to
 1574 waiting for keyboard input from the user. Unlike other signals, this signal is
 1575 not resent to the application.
 1576 .sp
 1577 .LP
 1578 Finally, the following signals cause \fBgl_get_line()\fR to first restore the
 1579 terminal and signal environment to that which prevailed before
 1580 \fBgl_get_line()\fR was called, then resend the signal to the application. If
 1581 the process still exists after the signal has been delivered, then
 1582 \fBgl_get_line()\fR then re-establishes its own signal handlers, switches the
 1583 terminal back to raw mode, redisplay the input line, and goes back to awaiting
 1584 terminal input from the user.
 1585 .sp
 1586 .ne 2
 1587 .na
 1588 \fB\fBSIGCONT\fR\fR
 1589 .ad
 1590 .RS 13n
 1591 This signal is generated when a suspended process is resumed.
 1592 .RE

1594 .sp
 1595 .ne 2
 1596 .na
 1597 \fB\fBSIGPOLL\fR\fR
 1598 .ad
 1599 .RS 13n
 1600 On SVR4 systems, this signal notifies the process of an asynchronous I/O event.
 1601 Note that under 4.3+BSD, \fBSIGIO\fR and \fBSIGPOLL\fR are the same. On other
 1602 systems, \fBSIGIO\fR is ignored by default, so \fBgl_get_line()\fR does not
 1603 trap it by default.
 1604 .RE

1606 .sp
 1607 .ne 2
 1608 .na
 1609 \fB\fBSIGPWR\fR\fR
 1610 .ad
 1611 .RS 13n
 1612 This signal is generated when a power failure occurs (presumably when the
 1613 system is on a UPS).
 1614 .RE

1616 .sp
 1617 .ne 2
 1618 .na
 1619 \fB\fBSIGALRM\fR\fR

```

1620 .ad
1621 .RS 13n
1622 This signal is generated when a timer expires.
1623 .RE

1625 .sp
1626 .ne 2
1627 .na
1628 \fB\fBSIGUSR1\fR\fR
1629 .ad
1630 .RS 13n
1631 An application specific signal.
1632 .RE

1634 .sp
1635 .ne 2
1636 .na
1637 \fB\fBSIGUSR2\fR\fR
1638 .ad
1639 .RS 13n
1640 Another application specific signal.
1641 .RE

1643 .sp
1644 .ne 2
1645 .na
1646 \fB\fBSIGVTALRM\fR\fR
1647 .ad
1648 .RS 13n
1649 This signal is generated when a virtual timer expires. See \fBsetitimer\fR(2).
1650 .RE

1652 .sp
1653 .ne 2
1654 .na
1655 \fB\fBSIGXCPU\fR\fR
1656 .ad
1657 .RS 13n
1658 This signal is generated when a process exceeds its soft CPU time limit.
1659 .RE

1661 .sp
1662 .ne 2
1663 .na
1664 \fB\fBSIGXFSZ\fR\fR
1665 .ad
1666 .RS 13n
1667 This signal is generated when a process exceeds its soft file-size limit.
1668 .RE

1670 .sp
1671 .ne 2
1672 .na
1673 \fB\fBSIGTSTP\fR\fR
1674 .ad
1675 .RS 13n
1676 This signal is generated by the terminal suspend key, which is usually
1677 \fB^Z\fR, or the delayed terminal suspend key, which is usually \fB^Y\fR.
1678 .RE

1680 .sp
1681 .ne 2
1682 .na
1683 \fB\fBSIGTTIN\fR\fR
1684 .ad
1685 .RS 13n

```

```

1686 This signal is generated if the program attempts to read from the terminal
1687 while the program is running in the background.
1688 .RE

1690 .sp
1691 .ne 2
1692 .na
1693 \fB\fBSIGTTOU\fR\fR
1694 .ad
1695 .RS 13n
1696 This signal is generated if the program attempts to write to the terminal while
1697 the program is running in the background.
1698 .RE

1700 .sp
1701 .LP
1702 Obviously not all of the above signals are supported on all systems, so code to
1703 support them is conditionally compiled into the tecla library.
1704 .sp
1705 .LP
1706 Note that if \fB\fBSIGKILL\fR or \fB\fBSIGPOLL\fR, which by definition cannot be
1707 caught, or any of the hardware generated exception signals, such as
1708 \fB\fBSIGSEGV\fR, \fB\fBSIGBUS\fR, and \fB\fBSIGFPE\fR, are received and unhandled while
1709 \fBgl_get_line()\fR has the terminal in raw mode, the program will be
1710 terminated without the terminal having been restored to a usable state. In
1711 practice, job-control shells usually reset the terminal settings when a process
1712 relinquishes the controlling terminal, so this is only a problem with older
1713 shells.
1714 .SS "Customized Signal Handling"
1715 .LP
1716 The previous section listed the signals that \fBgl_get_line()\fR traps by
1717 default, and described how it responds to them. This section describes how to
1718 both add and remove signals from the list of trapped signals, and how to
1719 specify how \fBgl_get_line()\fR should respond to a given signal.
1720 .sp
1721 .LP
1722 If you do not need \fBgl_get_line()\fR to do anything in response to a signal
1723 that it normally traps, you can tell to \fBgl_get_line()\fR to ignore that
1724 signal by calling \fBgl_ignore_signal()\fR.
1725 .sp
1726 .LP
1727 The \fBisigno\fR argument is the number of the signal (for example,
1728 \fB\fBSIGINT\fR) that you want to have ignored. If the specified signal is not
1729 currently one of those being trapped, this function does nothing.
1730 .sp
1731 .LP
1732 The \fBgl_trap_signal()\fR function allows you to either add a new signal to
1733 the list that \fBgl_get_line()\fR traps or modify how it responds to a signal
1734 that it already traps.
1735 .sp
1736 .LP
1737 The \fBisigno\fR argument is the number of the signal that you want to have
1738 trapped. The \fBiflags\fR argument is a set of flags that determine the
1739 environment in which the application's signal handler is invoked. The
1740 \fBifafter\fR argument tells \fBgl_get_line()\fR what to do after the
1741 application's signal handler returns. The \fBiterrno_value\fR tells
1742 \fBgl_get_line()\fR what to set \fBerrno\fR to if told to abort.
1743 .sp
1744 .LP
1745 The \fBiflags\fR argument is a bitwise OR of zero or more of the following
1746 enumerators:
1747 .sp
1748 .ne 2
1749 .na
1750 \fB\fBGLS_RESTORE_SIG\fR
1751 .ad

```

```

1752 .RS 20n
1753 Restore the caller's signal environment while handling the signal.
1754 .RE

1756 .sp
1757 .ne 2
1758 .na
1759 \fB\bGSLs_RESTORE_TTY\fR\fR
1760 .ad
1761 .RS 20n
1762 Restore the caller's terminal settings while handling the signal.
1763 .RE

1765 .sp
1766 .ne 2
1767 .na
1768 \fB\bGSLs_RESTORE_LINE\fR\fR
1769 .ad
1770 .RS 20n
1771 Move the cursor to the start of the line following the input line before
1772 invoking the application's signal handler.
1773 .RE

1775 .sp
1776 .ne 2
1777 .na
1778 \fB\bGSLs_REDRAW_LINE\fR\fR
1779 .ad
1780 .RS 20n
1781 Redraw the input line when the application's signal handler returns.
1782 .RE

1784 .sp
1785 .ne 2
1786 .na
1787 \fB\bGSLs_UNBLOCK_SIG\fR\fR
1788 .ad
1789 .RS 20n
1790 Normally, if the calling program has a signal blocked (see
1791 \fB\bSigprocmask\fR(2)), \fB\bGl_get_line()\fR does not trap that signal. This flag
1792 tells \fB\bGl_get_line()\fR to trap the signal and unblock it for the duration of
1793 the call to \fB\bGl_get_line()\fR.
1794 .RE

1796 .sp
1797 .ne 2
1798 .na
1799 \fB\bGSLs_DONT_FORWARD\fR\fR
1800 .ad
1801 .RS 20n
1802 If this flag is included, the signal will not be forwarded to the signal
1803 handler of the calling program.
1804 .RE

1806 .sp
1807 .LP
1808 Two commonly useful flag combinations are also enumerated as follows:
1809 .sp
1810 .ne 2
1811 .na
1812 \fB\bGSLs_RESTORE_ENV\fR\fR
1813 .ad
1814 .RS 21n
1815 \fB\bGSLs_RESTORE_SIG\fR | \fB\bGSLs_RESTORE_TTY\fR | \fB\bGSLs_REDRAW_LINE\fR
1816 .RE

```

```

1818 .sp
1819 .ne 2
1820 .na
1821 \fB\bGSLs_SUSPEND_INPUT\fR\fR
1822 .ad
1823 .RS 21n
1824 \fB\bGSLs_RESTORE_ENV\fR | \fB\bGSLs_RESTORE_LINE\fR
1825 .RE

1827 .sp
1828 .LP
1829 If your signal handler, or the default system signal handler for this signal,
1830 if you have not overridden it, never either writes to the terminal, nor
1831 suspends or terminates the calling program, then you can safely set the
1832 \fB\bIflags\fR argument to 0.
1833 .RS +4
1834 .TP
1835 .ie t \(\bu
1836 .el o
1837 The cursor does not get left in the middle of the input line.
1838 .RE
1839 .RS +4
1840 .TP
1841 .ie t \(\bu
1842 .el o
1843 So that the user can type in input and have it echoed.
1844 .RE
1845 .RS +4
1846 .TP
1847 .ie t \(\bu
1848 .el o
1849 So that you do not need to end each output line with \er\en, instead of just
1850 \en.
1851 .RE
1852 .sp
1853 .LP
1854 The \fB\bGSLs_RESTORE_ENV\fR combination is the same as \fB\bGSLs_SUSPEND_INPUT\fR,
1855 except that it does not move the cursor. If your signal handler does not read
1856 or write anything to the terminal, the user will not see any visible indication
1857 that a signal was caught. This can be useful if you have a signal handler that
1858 only occasionally writes to the terminal, where using \fB\bGSLs_SUSPEND_LINE\fR
1859 would cause the input line to be unnecessarily duplicated when nothing had been
1860 written to the terminal. Such a signal handler, when it does write to the
1861 terminal, should be sure to start a new line at the start of its first write,
1862 by writing a new line before returning. If the signal arrives while the user is
1863 entering a line that only occupies a signal terminal line, or if the cursor is
1864 on the last terminal line of a longer input line, this will have the same
1865 effect as \fB\bGSLs_SUSPEND_INPUT\fR. Otherwise it will start writing on a line
1866 that already contains part of the displayed input line. This does not do any
1867 harm, but it looks a bit ugly, which is why the \fB\bGSLs_SUSPEND_INPUT\fR
1868 combination is better if you know that you are always going to be writing to
1869 the terminal.
1870 .sp
1871 .LP
1872 The \fB\bIafter\fR argument, which determines what \fB\bGl_get_line()\fR does after
1873 the application's signal handler returns (if it returns), can take any one of
1874 the following values:
1875 .sp
1876 .ne 2
1877 .na
1878 \fB\bGSLs_RETURN\fR\fR
1879 .ad
1880 .RS 16n
1881 Return the completed input line, just as though the user had pressed the return
1882 key.
1883 .RE

```

```

1885 .sp
1886 .ne 2
1887 .na
1888 \fB\fBGLS_ABORT\fR\fR
1889 .ad
1890 .RS 16n
1891 Cause \fBgl_get_line()\fR to abort. When this happens, \fBgl_get_line()\fR
1892 returns \fINULL\fR, and a following call to \fBgl_return_status()\fR will
1893 return \fBGLR_SIGNAL\fR. Note that if the application needs \fBerrno\fR always
1894 to have a meaningful value when \fBgl_get_line()\fR returns \fINULL\fR, the
1895 callback function should set \fBerrno\fR appropriately.
1896 .RE

1898 .sp
1899 .ne 2
1900 .na
1901 \fB\fBGLS_CONTINUE\fR\fR
1902 .ad
1903 .RS 16n
1904 Resume command line editing.
1905 .RE

1907 .sp
1908 .LP
1909 The \fIerrno_value\fR argument is intended to be combined with the
1910 \fBGLS_ABORT\fR option, telling \fBgl_get_line()\fR what to set the standard
1911 \fBerrno\fR variable to before returning \fINULL\fR to the calling program. It
1912 can also, however, be used with the \fBGL_RETURN\fR option, in case you want to
1913 have a way to distinguish between an input line that was entered using the
1914 return key, and one that was entered by the receipt of a signal.
1915 .SS "Reliable Signal Handling"
1916 .LP
1917 Signal handling is surprisingly hard to do reliably without race conditions. In
1918 \fBgl_get_line()\fR a lot of care has been taken to allow applications to
1919 perform reliable signal handling around \fBgl_get_line()\fR. This section
1920 explains how to make use of this.
1921 .sp
1922 .LP
1923 As an example of the problems that can arise if the application is not written
1924 correctly, imagine that one's application has a \fBSIGINT\fR signal handler
1925 that sets a global flag. Now suppose that the application tests this flag just
1926 before invoking \fBgl_get_line()\fR. If a \fBSIGINT\fR signal happens to be
1927 received in the small window of time between the statement that tests the value
1928 of this flag, and the statement that calls \fBgl_get_line()\fR, then
1929 \fBgl_get_line()\fR will not see the signal, and will not be interrupted. As a
1930 result, the application will not be able to respond to the signal until the
1931 user gets around to finishing entering the input line and \fBgl_get_line()\fR
1932 returns. Depending on the application, this might or might not be a disaster,
1933 but at the very least it would puzzle the user.
1934 .sp
1935 .LP
1936 The way to avoid such problems is to do the following.
1937 .RS +4
1938 .TP
1939 1.
1940 If needed, use the \fBgl_trap_signal()\fR function to configure
1941 \fBgl_get_line()\fR to abort when important signals are caught.
1942 .RE
1943 .RS +4
1944 .TP
1945 2.
1946 Configure \fBgl_get_line()\fR such that if any of the signals that it
1947 catches are blocked when \fBgl_get_line()\fR is called, they will be unblocked
1948 automatically during times when \fBgl_get_line()\fR is waiting for I/O. This
1949 can be done either on a per signal basis, by calling the \fBgl_trap_signal()\fR

```

```

1950 function, and specifying the \fBGLS_UNBLOCK\fR attribute of the signal, or
1951 globally by calling the \fBgl_catch_blocked()\fR function. This function simply
1952 adds the \fBGLS_UNBLOCK\fR attribute to all of the signals that it is currently
1953 configured to trap.
1954 .RE
1955 .RS +4
1956 .TP
1957 3.
1958 Just before calling \fBgl_get_line()\fR, block delivery of all of the
1959 signals that \fBgl_get_line()\fR is configured to trap. This can be done using
1960 the POSIX \fBsigprocmask\fR function in conjunction with the \fBgl_list_signals()\fR
1961 function. This function returns the set of signals that it is currently
1962 configured to catch in the set argument, which is in the form required by
1963 \fBsigprocmask(2)\fR.
1964 .RE
1965 .RS +4
1966 .TP
1967 4.
1968 In the example, one would now test the global flag that the signal handler
1969 sets, knowing that there is now no danger of this flag being set again until
1970 \fBgl_get_line()\fR unblocks its signals while performing I/O.
1971 .RE
1972 .RS +4
1973 .TP
1974 5.
1975 Eventually \fBgl_get_line()\fR returns, either because a signal was caught,
1976 an error occurred, or the user finished entering their input line.
1977 .RE
1978 .RS +4
1979 .TP
1980 6.
1981 Now one would check the global signal flag again, and if it is set, respond
1982 to it, and zero the flag.
1983 .RE
1984 .RS +4
1985 .TP
1986 7.
1987 Use \fBsigprocmask()\fR to unblock the signals that were blocked in step 3.
1988 .RE
1989 .sp
1990 .LP
1991 The same technique can be used around certain POSIX signal-aware functions,
1992 such as \fBsigsetjmp(3C)\fR and \fBsigsuspend(2)\fR, and in particular, the
1993 former of these two functions can be used in conjunction with
1994 \fBsiglongjmp(3C)\fR to implement race-condition free signal handling around
1995 other long-running system calls. The \fBgl_get_line()\fR function manages to
1996 reliably trap signals around calls to functions like \fBread(2)\fR and
1997 \fBselect(3C)\fR without race conditions.
1998 .sp
1999 .LP
2000 The \fBgl_get_line()\fR function first uses the POSIX \fBsigprocmask()\fR
2001 function to block the delivery of all of the signals that it is currently
2002 configured to catch. This is redundant if the application has already blocked
2003 them, but it does no harm. It undoes this step just before returning.
2004 .sp
2005 .LP
2006 Whenever \fBgl_get_line()\fR needs to call read or select to wait for input
2007 from the user, it first calls the POSIX \fBsigsetjmp()\fR function, being sure
2008 to specify a non-zero value for its \fBisavemask\fR argument.
2009 .sp
2010 .LP
2011 If \fBsigsetjmp()\fR returns zero, \fBgl_get_line()\fR then does the following.
2012 .RS +4
2013 .TP
2014 1.
2015 It uses the POSIX \fBsigaction(2)\fR function to register a temporary signal

```

2016 handler to all of the signals that it is configured to catch. This signal
 2017 handler does two things.
 2018 .RS +4
 2019 .TP
 2020 a.
 2021 It records the number of the signal that was received in a file-scope
 2022 variable.
 2023 .RE
 2024 .RS +4
 2025 .TP
 2026 b.
 2027 It then calls the POSIX `\fBsiglongjmp()\fR` function using the buffer that
 2028 was passed to `\fBsigsetjmp()\fR` for its first argument and a non-zero value for
 2029 its second argument.
 2030 .RE
 2031 When this signal handler is registered, the `\fBisa_mask\fR` member of the
 2032 `\fBstruct sigaction\fR` `\fBiact\fR` argument of the call to `\fBsigaction()\fR` is
 2033 configured to contain all of the signals that `\fBgl_get_line()\fR` is catching.
 2034 This ensures that only one signal will be caught at once by our signal handler,
 2035 which in turn ensures that multiple instances of our signal handler do not
 2036 tread on each other's toes.
 2037 .RE
 2038 .RS +4
 2039 .TP
 2040 2.
 2041 Now that the signal handler has been set up, `\fBgl_get_line()\fR` unblocks
 2042 all of the signals that it is configured to catch.
 2043 .RE
 2044 .RS +4
 2045 .TP
 2046 3.
 2047 It then calls the `\fBread()\fR` or `\fBselect()\fR` function to wait for
 2048 keyboard input.
 2049 .RE
 2050 .RS +4
 2051 .TP
 2052 4.
 2053 If this function returns (that is, no signal is received),
 2054 `\fBgl_get_line()\fR` blocks delivery of the signals of interest again.
 2055 .RE
 2056 .RS +4
 2057 .TP
 2058 5.
 2059 It then reinstates the signal handlers that were displaced by the one that
 2060 was just installed.
 2061 .RE
 2062 .sp
 2063 .LP
 2064 Alternatively, if `\fBsigsetjmp()\fR` returns non-zero, this means that one of
 2065 the signals being trapped was caught while the above steps were executing. When
 2066 this happens, `\fBgl_get_line()\fR` does the following.
 2067 .sp
 2068 .LP
 2069 First, note that when a call to `\fBsiglongjmp()\fR` causes `\fBsigsetjmp()\fR` to
 2070 return, provided that the `\fBisavemask\fR` argument of `\fBsigsetjmp()\fR` was
 2071 non-zero, the signal process mask is restored to how it was when
 2072 `\fBsigsetjmp()\fR` was called. This is the important difference between
 2073 `\fBsigsetjmp()\fR` and the older problematic `\fBsetjmp(3C)\fR`, and is the
 2074 essential ingredient that makes it possible to avoid signal handling race
 2075 conditions. Because of this we are guaranteed that all of the signals that we
 2076 blocked before calling `\fBsigsetjmp()\fR` are blocked again as soon as any
 2077 signal is caught. The following statements, which are then executed, are thus
 2078 guaranteed to be executed without any further signals being caught.
 2079 .RS +4
 2080 .TP
 2081 1.

2082 If so instructed by the `\fBgl_get_line()\fR` configuration attributes of the
 2083 signal that was caught, `\fBgl_get_line()\fR` restores the terminal attributes to
 2084 the state that they had when `\fBgl_get_line()\fR` was called. This is
 2085 particularly important for signals that suspend or terminate the process, since
 2086 otherwise the terminal would be left in an unusable state.
 2087 .RE
 2088 .RS +4
 2089 .TP
 2090 2.
 2091 It then reinstates the application's signal handlers.
 2092 .RE
 2093 .RS +4
 2094 .TP
 2095 3.
 2096 Then it uses the C standard-library `\fBbraise()\fR(3C)` function to re-send the
 2097 application the signal that was caught.
 2098 .RE
 2099 .RS +4
 2100 .TP
 2101 4.
 2102 Next it unblocks delivery of the signal that we just sent. This results in
 2103 the signal that was just sent by `\fBraise()\fR` being caught by the
 2104 application's original signal handler, which can now handle it as it sees fit.
 2105 .RE
 2106 .RS +4
 2107 .TP
 2108 5.
 2109 If the signal handler returns (that is, it does not terminate the process),
 2110 `\fBgl_get_line()\fR` blocks delivery of the above signal again.
 2111 .RE
 2112 .RS +4
 2113 .TP
 2114 6.
 2115 It then undoes any actions performed in the first of the above steps and
 2116 redisplay the line, if the signal configuration calls for this.
 2117 .RE
 2118 .RS +4
 2119 .TP
 2120 7.
 2121 `\fBgl_get_line()\fR` then either resumes trying to read a character, or
 2122 aborts, depending on the configuration of the signal that was caught.
 2123 .RE
 2124 .sp
 2125 .LP
 2126 What the above steps do in essence is to take asynchronously delivered signals
 2127 and handle them synchronously, one at a time, at a point in the code where
 2128 `\fBgl_get_line()\fR` has complete control over its environment.
 2129 .SS "The Terminal Size"
 2130 .LP
 2131 On most systems the combination of the `\fBtIOCGWINSZ\fR` ioctl and the
 2132 `\fBSIGWINCH\fR` signal is used to maintain an accurate idea of the terminal
 2133 size. The terminal size is newly queried every time that `\fBgl_get_line()\fR` is
 2134 called and whenever a `\fBSIGWINCH\fR` signal is received.
 2135 .sp
 2136 .LP
 2137 On the few systems where this mechanism is not available, at startup
 2138 `\fBNew_GetLine()\fR` first looks for the `\fBBLINES\fR` and `\fBCOLUMNS\fR`
 2139 environment variables. If these are not found, or they contain unusable values,
 2140 then if a terminal information database like `\fBterminfo\fR` or `\fBtermcap\fR` is
 2141 available, the default size of the terminal is looked up in this database. If
 2142 this too fails to provide the terminal size, a default size of 80 columns by 24
 2143 lines is used.
 2144 .sp
 2145 .LP
 2146 Even on systems that do support ioctl(`\fBtIOCGWINSZ\fR`), if the terminal is on
 2147 the other end of a serial line, the terminal driver generally has no way of

2148 detecting when a resize occurs or of querying what the current size is. In such
 2149 cases no `\fBSIGWINCH` is sent to the process, and the dimensions returned by
 2150 `ioctl(\fBTIOCGWINSZ)` are not correct. The only way to handle such instances
 2151 is to provide a way for the user to enter a command that tells the remote
 2152 system what the new size is. This command would then call the
 2153 `\fBgl_set_term_size()` function to tell `\fBgl_get_line()` about the change
 2154 in size.
 2155 .sp
 2156 .LP
 2157 The `\fIncolumn` and `\fInline` arguments are used to specify the new
 2158 dimensions of the terminal, and must not be less than 1. On systems that do
 2159 support `ioctl(\fBTIOCGWINSZ)`, this function first calls
 2160 `ioctl(\fBTIOCGWINSZ)` to tell the terminal driver about the change in size.
 2161 In non-blocking server-I/O mode, if a line is currently being input, the input
 2162 line is then redrawn to accommodate the changed size. Finally the new values are
 2163 recorded in `\fIgl` for future use by `\fBgl_get_line()`.
 2164 .sp
 2165 .LP
 2166 The `\fBgl_terminal_size()` function allows you to query the current size of
 2167 the terminal, and install an alternate fallback size for cases where the size
 2168 is not available. Beware that the terminal size will not be available if
 2169 reading from a pipe or a file, so the default values can be important even on
 2170 systems that do support ways of finding out the terminal size.
 2171 .sp
 2172 .LP
 2173 This function first updates `\fBgl_get_line()`'s fallback terminal dimensions,
 2174 then records its findings in the return value.
 2175 .sp
 2176 .LP
 2177 The `\fIdef_ncolumn` and `\fIdef_nline` arguments specify the default number
 2178 of terminal columns and lines to use if the terminal size cannot be determined
 2179 by `ioctl(\fBTIOCGWINSZ)` or environment variables.
 2180 .SS "Hiding What You Type"
 2181 .LP
 2182 When entering sensitive information, such as passwords, it is best not to have
 2183 the text that you are entering echoed on the terminal. Furthermore, such text
 2184 should not be recorded in the history list, since somebody finding your
 2185 terminal unattended could then recall it, or somebody snooping through your
 2186 directories could see it in your history file. With this in mind, the
 2187 `\fBgl_echo_mode()` function allows you to toggle on and off the display and
 2188 archival of any text that is subsequently entered in calls to
 2189 `\fBgl_get_line()`.
 2190 .sp
 2191 .LP
 2192 The `\fIenable` argument specifies whether entered text should be visible or
 2193 not. If it is 0, then subsequently entered lines will not be visible on the
 2194 terminal, and will not be recorded in the history list. If it is 1, then
 2195 subsequent input lines will be displayed as they are entered, and provided that
 2196 history has not been turned off with a call to `\fBgl_toggle_history()`, then
 2197 they will also be archived in the history list. Finally, if the enable argument
 2198 is -1, then the echoing mode is left unchanged, which allows you to
 2199 non-destructively query the current setting through the return value. In all
 2200 cases, the return value of the function is 0 if echoing was disabled before the
 2201 function was called, and 1 if it was enabled.
 2202 .sp
 2203 .LP
 2204 When echoing is turned off, note that although tab completion will invisibly
 2205 complete your prefix as far as possible, ambiguous completions will not be
 2206 displayed.
 2207 .SS "Single Character Queries"
 2208 .LP
 2209 Using `\fBgl_get_line()` to query the user for a single character reply, is
 2210 inconvenient for the user, since they must hit the enter or return key before
 2211 the character that they typed is returned to the program. Thus the
 2212 `\fBgl_query_char()` function has been provided for single character queries
 2213 like this.

2214 .sp
 2215 .LP
 2216 This function displays the specified prompt at the start of a new line, and
 2217 waits for the user to type a character. When the user types a character,
 2218 `\fBgl_query_char()` displays it to the right of the prompt, starts a newline,
 2219 then returns the character to the calling program. The return value of the
 2220 function is the character that was typed. If the read had to be aborted for
 2221 some reason, EOF is returned instead. In the latter case, the application can
 2222 call the previously documented `\fBgl_return_status()`, to find out what went
 2223 wrong. This could, for example, have been the reception of a signal, or the
 2224 optional inactivity timer going off.
 2225 .sp
 2226 .LP
 2227 If the user simply hits enter, the value of the `\fIdefchar` argument is
 2228 substituted. This means that when the user hits either newline or return, the
 2229 character specified in `\fIdefchar`, is displayed after the prompt, as though
 2230 the user had typed it, as well as being returned to the calling application. If
 2231 such a replacement is not important, simply pass `'\n'` as the value of
 2232 `\fIdefchar`.
 2233 .sp
 2234 .LP
 2235 If the entered character is an unprintable character, it is displayed
 2236 symbolically. For example, control-A is displayed as `\fB^A`, and characters
 2237 beyond 127 are displayed in octal, preceded by a backslash.
 2238 .sp
 2239 .LP
 2240 As with `\fBgl_get_line()`, echoing of the entered character can be disabled
 2241 using the `\fBgl_echo_mode()` function.
 2242 .sp
 2243 .LP
 2244 If the calling process is suspended while waiting for the user to type their
 2245 response, the cursor is moved to the line following the prompt line, then when
 2246 the process resumes, the prompt is redisplayed, and `\fBgl_query_char()`
 2247 resumes waiting for the user to type a character.
 2248 .sp
 2249 .LP
 2250 Note that in non-blocking server mode, if an incomplete input line is in the
 2251 process of being read when `\fBgl_query_char()` is called, the partial input
 2252 line is discarded, and erased from the terminal, before the new prompt is
 2253 displayed. The next call to `\fBgl_get_line()` will thus start editing a new
 2254 line.
 2255 .SS "Reading Raw Characters"
 2256 .LP
 2257 Whereas the `\fBgl_query_char()` function visibly prompts the user for a
 2258 character, and displays what they typed, the `\fBgl_read_char()` function
 2259 reads a signal character from the user, without writing anything to the
 2260 terminal, or perturbing any incompletely entered input line. This means that it
 2261 can be called not only from between calls to `\fBgl_get_line()`, but also from
 2262 callback functions that the application has registered to be called by
 2263 `\fBgl_get_line()`.
 2264 .sp
 2265 .LP
 2266 On success, the return value of `\fBgl_read_char()` is the character that was
 2267 read. On failure, EOF is returned, and the `\fBgl_return_status()` function
 2268 can be called to find out what went wrong. Possibilities include the optional
 2269 inactivity timer going off, the receipt of a signal that is configured to abort
 2270 `\fBgl_get_line()`, or terminal I/O blocking, when in non-blocking server-I/O
 2271 mode.
 2272 .sp
 2273 .LP
 2274 Beware that certain keyboard keys, such as function keys, and cursor keys,
 2275 usually generate at least three characters each, so a single call to
 2276 `\fBgl_read_char()` will not be enough to identify such keystrokes.
 2277 .SS "Clearing The Terminal"
 2278 .LP
 2279 The calling program can clear the terminal by calling

2280 `\fBgl_erase_terminal()\fR`. In non-blocking server-I/O mode, this function also
 2281 arranges for the current input line to be redrawn from scratch when
 2282 `\fBgl_get_line()\fR` is next called.
 2283 .SS "Displaying Text Dynamically"
 2284 .LP
 2285 Between calls to `\fBgl_get_line()\fR`, the `\fBgl_display_text()\fR` function
 2286 provides a convenient way to display paragraphs of text, left-justified and
 2287 split over one or more terminal lines according to the constraints of the
 2288 current width of the terminal. Examples of the use of this function may be
 2289 found in the demo programs, where it is used to display introductions. In those
 2290 examples the advanced use of optional prefixes, suffixes and filled lines to
 2291 draw a box around the text is also illustrated.
 2292 .sp
 2293 .LP
 2294 If `\fIgl\fR` is not currently connected to a terminal, for example if the output
 2295 of a program that uses `\fBgl_get_line()\fR` is being piped to another program or
 2296 redirected to a file, then the value of the `\fIdef_width\fR` parameter is used
 2297 as the terminal width.
 2298 .sp
 2299 .LP
 2300 The `\fIindentation\fR` argument specifies the number of characters to use to
 2301 **indent each line of output. The `\fIfill_char\fR` argument specifies the character**
 2301 *indent each line of output. The `\fIfill_char\fR`* argument specifies the character
 2302 that will be used to perform this indentation.
 2303 .sp
 2304 .LP
 2305 The `\fIprefix\fR` argument can be either `\fINULL\fR` or a string to place at the
 2306 beginning of each new line (after any indentation). Similarly, the `\fIsuffix\fR`
 2307 argument can be either `\fINULL\fR` or a string to place at the end of each line.
 2308 The suffix is placed flush against the right edge of the terminal, and any
 2309 space between its first character and the last word on that line is filled with
 2310 the character specified by the `\fIfill_char\fR` argument. Normally the
 2311 fill-character is a space.
 2312 .sp
 2313 .LP
 2314 The `\fIstart\fR` argument tells `\fBgl_display_text()\fR` how many characters have
 2315 already been written to the current terminal line, and thus tells it the
 2316 starting column index of the cursor. Since the return value of
 2317 `\fBgl_display_text()\fR` is the ending column index of the cursor, by passing
 2318 the return value of one call to the start argument of the next call, a
 2319 paragraph that is broken between more than one string can be composed by
 2320 calling `\fBgl_display_text()\fR` for each successive portion of the paragraph.
 2321 Note that literal newline characters are necessary at the end of each paragraph
 2322 to force a new line to be started.
 2323 .sp
 2324 .LP
 2325 On error, `\fBgl_display_text()\fR` returns -1.
 2326 .SS "Callback Function Facilities"
 2327 .LP
 2328 Unless otherwise stated, callback functions such as tab completion callbacks
 2329 and event callbacks should not call any functions in this module. The following
 2330 functions, however, are designed specifically to be used by callback functions.
 2331 .sp
 2332 .LP
 2333 Calling the `\fBgl_replace_prompt()\fR` function from a callback tells
 2334 `\fBgl_get_line()\fR` to display a different prompt when the callback returns.
 2335 Except in non-blocking server mode, it has no effect if used between calls to
 2336 `\fBgl_get_line()\fR`. In non-blocking server mode, when used between two calls
 2337 to `\fBgl_get_line()\fR` that are operating on the same input line, the current
 2338 input line will be re-drawn with the new prompt on the following call to
 2339 `\fBgl_get_line()\fR`.
 2340 .SS "International Character Sets"
 2341 .LP
 2342 Since `\fBlibtecla\fR(3LIB) version 1.4.0, \fBgl_get_line()\fR has been 8-bit
 2343 clean. This means that all 8-bit characters that are printable in the user's
 2344 current locale are now displayed verbatim and included in the returned input`

2345 line. Assuming that the calling program correctly contains a call like the
 2346 following,
 2347 .sp
 2348 .in +2
 2349 .nf
 2350 `setlocale(LC_CTYPE, "")`
 2351 .fi
 2352 .in -2
 2354 .sp
 2355 .LP
 2356 then the current locale is determined by the first of the environment variables
 2357 `\fBLC_CTYPE\fR`, `\fBLC_ALL\fR`, and `\fBLANG\fR` that is found to contain a valid
 2358 locale name. If none of these variables are defined, or the program neglects to
 2359 call `\fBsetlocale\fR(3C)`, then the default C locale is used, which is US 7-bit
 2360 ASCII. On most UNIX-like platforms, you can get a list of valid locales by
 2361 typing the command:
 2362 .sp
 2363 .in +2
 2364 .nf
 2365 `locale -a`
 2366 .fi
 2367 .in -2
 2368 .sp
 2370 .sp
 2371 .LP
 2372 at the shell prompt. Further documentation on how the user can make use of this
 2373 to enter international characters can be found in the `\fBtecla\fR(5)` man page.
 2374 .SS "Thread Safety"
 2375 .LP
 2376 Unfortunately neither `\fBterminfo\fR` nor `\fBtermcap\fR` were designed to be
 2377 reentrant, so you cannot safely use the functions of the `getline` module in
 2378 multiple threads (you can use the separate file-expansion and word-completion
 2379 modules in multiple threads, see the corresponding man pages for details).
 2380 However due to the use of POSIX reentrant functions for looking up home
 2381 directories, it is safe to use this module from a single thread of a
 2382 multi-threaded program, provided that your other threads do not use any
 2383 `\fBtermcap\fR` or `\fBterminfo\fR` functions.
 2384 .SH ATTRIBUTES
 2385 .LP
 2386 See `\fBattributes\fR(5)` for descriptions of the following attributes:
 2387 .sp
 2389 .sp
 2390 .TS
 2391 box;
 2392 c | c
 2393 l | l .
 2394 ATTRIBUTE TYPE ATTRIBUTE VALUE
 2395 _
 2396 Interface Stability Committed
 2397 _
 2398 MT-Level MT-Safe
 2399 .TE
 2401 .SH SEE ALSO
 2402 .LP
 2403 `\fBcpl_complete_word\fR(3TECLA)`, `\fBef_expand_file\fR(3TECLA)`,
 2404 `\fBgl_io_mode\fR(3TECLA)`, `\fBlibtecla\fR(3LIB)`, `\fBpca_lookup_file\fR(3TECLA)`,
 2405 `\fBattributes\fR(5)`, `\fBtecla\fR(5)`

```

*****
53473 Tue Dec 11 09:48:09 2018
new/usr/src/man/man5/tecla.5
10057 Man page misspellings output particular overridden
Reviewed by: Gerg Mihajly Doma <domag02@gmail.com>
*****
1 \" te
2.\" Copyright (c) 2000, 2001, 2002, 2003, 2004 by Martin C. Shepherd. All Rights
3.\" Permission is hereby granted, free of charge, to any person obtaining a copy
4.\" \"Software\"), to deal in the Software without restriction, including
5.\" without limitation the rights to use, copy, modify, merge, publish,
6.\" distribute, and/or sell copies of the Software, and to permit persons
7.\" to whom the Software is furnished to do so, provided that the above
8.\" copyright notice(s) and this permission notice appear in all copies of
9.\" the Software and that both the above copyright notice(s) and this
10.\" permission notice appear in supporting documentation.
11.\"
12.\" THE SOFTWARE IS PROVIDED \"AS IS\", WITHOUT WARRANTY OF ANY KIND, EXPRESS
13.\" OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
14.\" MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT
15.\" OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
16.\" HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL
17.\" INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING
18.\" FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
19.\" NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION
20.\" WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
21.\"
22.\" Except as contained in this notice, the name of a copyright holder
23.\" shall not be used in advertising or otherwise to promote the sale, use
24.\" or other dealings in this Software without prior written authorization
25.\" of the copyright holder.
26.\" Portions Copyright (c) 2005, Sun Microsystems, Inc. All Rights Reserved.
27.TH TECLA 5 \"April 9, 2016\"
28.SH NAME
29 tecla, teclarc \- User interface provided by the tecla library.
30.SH DESCRIPTION
31.LP
32 This man page describes the command-line editing features that are available to
33 users of programs that read keyboard input via the tecla library. Users of the
34 \fBtcsh\fR shell will find the default key bindings very familiar. Users of the
35 \fBtcsh shell will find the default key bindings very familiar. Users of the
36 \fBbash\fR shell will also find it quite familiar, but with a few minor
37 differences, most notably in how forward and backward searches through the list
38 of historical commands are performed. There are two major editing modes, one
39 with \fBemacs\fR-like key bindings and another with \fBvi\fR-like key bindings.
40 By default \fBemacs\fR mode is enabled, but \fBvi\fR(1) mode can alternatively
41 be selected via the user's configuration file. This file can also be used to
42 change the bindings of individual keys to suit the user's preferences. By
43 default, tab completion is provided. If the application hasn't reconfigured
44 this to complete other types of symbols, then tab completion completes file
45 names.
46.LP
47 In the rest of this man page, and also in all tecla configuration files, key
48 sequences are expressed as follows.
49 .sp
50 .ne 2
51 .na
52 \fB\^A\fR or \fB\^a\fR
53 .ad
54 .RS 13n
55 This is a 'CONTROL-A', entered by pressing the CONTROL key at the same time as
56 the 'A' key.
57 .RE
58 .sp
59 .sp

```

```

60 .ne 2
61 .na
62 \fB\^E\fR or \fB\^e\fR
63 .ad
64 .RS 13n
65 In key sequences, both of these notations can be entered either by pressing the
66 ESCAPE key, then the following key, or by pressing the META key at the same
67 time as the following key. Thus the key sequence \fB\^p\fR can be typed in two
68 ways, by pressing the ESCAPE key, followed by pressing 'P', or by pressing the
69 META key at the same time as 'P'.
70 .RE
71 .sp
72 .sp
73 .ne 2
74 .na
75 \fB\^B\fR
76 .ad
77 .RS 13n
78 This refers to the up-arrow key.
79 .RE
80 .sp
81 .sp
82 .ne 2
83 .na
84 \fB\^D\fR
85 .ad
86 .RS 13n
87 This refers to the down-arrow key.
88 .RE
89 .sp
90 .sp
91 .ne 2
92 .na
93 \fB\^L\fR
94 .ad
95 .RS 13n
96 This refers to the left-arrow key.
97 .RE
98 .sp
99 .sp
100 .ne 2
101 .na
102 \fB\^R\fR
103 .ad
104 .RS 13n
105 This refers to the right-arrow key.
106 .RE
107 .sp
108 .sp
109 .ne 2
110 .na
111 \fB\^A\fR
112 .ad
113 .RS 13n
114 This is just a normal 'A' key.
115 .RE
116 .sp
117 .SS \"The Tecla Configuration File\"
118 .LP
119 By default, tecla looks for a file called \fB&.teclarc\fR in your home
120 directory (ie. \fB~/teclarc\fR). If it finds this file, it reads it,
121 interpreting each line as defining a new key binding or an editing
122 configuration option. Since the \fBemacs\fR key-bindings are installed by
123 default, if you want to use the non-default \fBvi\fR editing mode, the most
124 important item to go in this file is the following line:
125 .sp

```



```

126 .in +2
127 .nf
128 edit-mode vi
129 .fi
130 .in -2

132 .sp
133 .LP
134 This will re-configure the default bindings for \fBvi\fR-mode. The complete set
135 of arguments that this command accepts are:
136 .sp
137 .ne 2
138 .na
139 \fBvi\fR
140 .ad
141 .RS 9n
142 Install key bindings like those of the \fBvi\fR editor.
143 .RE

145 .sp
146 .ne 2
147 .na
148 \fBemacs\fR
149 .ad
150 .RS 9n
151 Install key bindings like those of the \fBemacs\fR editor. This is the default.
152 .RE

154 .sp
155 .ne 2
156 .na
157 \fBnone\fR
158 .ad
159 .RS 9n
160 Use just the native line editing facilities provided by the terminal driver.
161 .RE

163 .sp
164 .LP
165 To prevent the terminal bell from being rung, such as when an unrecognized
166 control-sequence is typed, place the following line in the configuration file:
167 .sp
168 .in +2
169 .nf
170 nobeep
171 .fi
172 .in -2

174 .sp
175 .LP
176 An example of a key binding line in the configuration file is the following.
177 .sp
178 .in +2
179 .nf
180 bind M-[2~ insert-mode
181 .fi
182 .in -2

184 .sp
185 .LP
186 On many keyboards, the above key sequence is generated when one presses the
187 insert key, so with this key binding, one can toggle between the
188 \fBemacs\fR-mode insert and overwrite modes by hitting one key. One could also
189 do it by typing out the above sequence of characters one by one. As explained
190 above, the \fBm-\fR part of this sequence can be typed either by pressing the
191 ESCAPE key before the following key, or by pressing the META key at the same

```

```

192 time as the following key. Thus if you had set the above key binding, and the
193 insert key on your keyboard didn't generate the above key sequence, you could
194 still type it in either of the following 2 ways.
195 .RS +4
196 .TP
197 1.
198 Hit the ESCAPE key momentarily, then press '[' , then '2' , then finally '~'.
199 .RE
200 .RS +4
201 .TP
202 2.
203 Press the META key at the same time as pressing the '[' key, then press '2' ,
204 then '~'.
205 .RE
206 .sp
207 .LP
208 If you set a key binding for a key sequence that is already bound to a
209 function, the new binding overrides the old one. If in the new binding you omit
210 the name of the new function to bind to the key sequence, the original binding
211 becomes undefined.
212 .sp
213 .LP
214 Starting with versions of \fBlibtecla\fR later than 1.3.3 it is now possible to
215 bind key sequences that begin with a printable character. Previously key
216 sequences were required to start with a CONTROL or META character.
217 .sp
218 .LP
219 Note that the special keywords "up", "down", "left", and "right" refer to the
220 arrow keys, and are thus not treated as key sequences. So, for example, to
221 rebound the up and down arrow keys to use the history search mechanism instead
222 of the simple history recall method, you could place the following in your
223 configuration file:
224 .sp
225 .in +2
226 .nf
227 bind up history-search-backwards
228 bind down history-search-backwards
229 .fi
230 .in -2

232 .sp
233 .LP
234 To unbind an existing binding, you can do this with the bind command by
235 omitting to name any action to rebound the key sequence to. For example, by not
236 specifying an action function, the following command unbinds the default
237 beginning-of-line action from the \fB^A\fR key sequence:
238 .sp
239 .in +2
240 .nf
241 bind ^A
242 .fi
243 .in -2

245 .sp
246 .LP
247 If you create a \fB~/teclarc\fR configuration file, but it appears to have no
248 effect on the program, check the documentation of the program to see if the
249 author chose a different name for this file.
250 .SS "Filename and Tilde Completion"
251 .LP
252 With the default key bindings, pressing the TAB key (aka. \fB^I\fR) results in
253 tecla attempting to complete the incomplete file name that precedes the cursor.
254 Tecla searches backwards from the cursor, looking for the start of the file
255 name, stopping when it hits either a space or the start of the line. If more
256 than one file has the specified prefix, then tecla completes the file name up
257 to the point at which the ambiguous matches start to differ, then lists the

```

```

258 possible matches.
259 .sp
260 .LP
261 In addition to literally written file names, tecla can complete files that
262 start with \fB~/\fR and \fB-user/\fR expressions and that contain \fB$envvar\fR
263 expressions. In particular, if you hit TAB within an incomplete \fB-user\fR,
264 expression, tecla will attempt to complete the username, listing any ambiguous
265 matches.
266 .sp
267 .LP
268 The completion binding is implemented using the \fBcpl_complete_word()\fR
269 function, which is also available separately to users of this library. See the
270 \fBcpl_complete_word\fR(3TECLA) man page for more details.
271 .SS "Filename Expansion"
272 .LP
273 With the default key bindings, pressing \fB^X*\fR causes tecla to expand the
274 file name that precedes the cursor, replacing \fB~/\fR and \fB-user/\fR
275 expressions with the corresponding home directories, and replacing
276 \fB$envvar\fR expressions with the value of the specified environment variable,
277 then if there are any wildcards, replacing the so far expanded file name with a
278 space-separated list of the files which match the wild cards.
279 .sp
280 .LP
281 The expansion binding is implemented using the \fBef_expand_file()\fR function.
282 See the \fBef_expand_file\fR(3TECLA) man page for more details.
283 .SS "Recalling Previously Typed Lines"
284 .LP
285 Every time that a new line is entered by the user, it is appended to a list of
286 historical input lines maintained within the \fBGetLine\fR resource object. You
287 can traverse up and down this list using the up and down arrow keys.
288 Alternatively, you can do the same with the \fB^P\fR, and \fB^N\fR keys, and in
289 \fBvi\fR command mode you can alternatively use the k and j characters. Thus
290 pressing up-arrow once, replaces the current input line with the previously
291 entered line. Pressing up-arrow again, replaces this with the line that was
292 entered before it, etc.. Having gone back one or more lines into the history
293 list, one can return to newer lines by pressing down-arrow one or more times.
294 If you do this sufficient times, you will return to the original line that you
295 were entering when you first hit up-arrow.
296 .sp
297 .LP
298 Note that in \fBvi\fR mode, all of the history recall functions switch the
299 library into command mode.
300 .sp
301 .LP
302 In \fBemacs\fR mode the \fBM-p\fR and \fBM-n\fR keys work just like the
303 \fB^P\fR and \fB^N\fR keys, except that they skip all but those historical
304 lines which share the prefix that precedes the cursor. In \fBvi\fR command mode
305 the upper case 'K' and 'J' characters do the same thing, except that the string
306 that they search for includes the character under the cursor as well as what
307 precedes it.
308 .sp
309 .LP
310 Thus for example, suppose that you were in \fBemacs\fR mode, and you had just
311 entered the following list of commands in the order shown:
312 .sp
313 .in +2
314 .nf
315 ls ~/tecla/
316 cd ~/tecla
317 ls -l getline.c
318 \fBemacs\fR ~/tecla/getline.c
319 .fi
320 .in -2

322 .sp
323 .LP

```

```

324 If you next typed:
325 .sp
326 .in +2
327 .nf
328 ls
329 .fi
330 .in -2

332 .sp
333 .LP
334 and then hit \fBM-p\fR, then rather than returning the previously typed
335 \fBemacs\fR line, which doesn't start with "ls", tecla would recall the "ls -l
336 getline.c" line. Pressing \fBM-p\fR again would recall the "ls ~/tecla/" line.
337 .sp
338 .LP
339 Note that if the string that you are searching for, contains any of the special
340 characters, *, ?, or '[', then it is interpreted as a pattern to be matched.
341 Thus, continuing with the above example, after typing in the list of commands
342 shown, if you then typed:
343 .sp
344 .in +2
345 .nf
346 *tecla*
347 .fi
348 .in -2

350 .sp
351 .LP
352 and hit \fBM-p\fR, then the "\fBemacs\fR ~/tecla/getline.c" line would be
353 recalled first, since it contains the word tecla somewhere in the line,
354 Similarly, hitting \fBM-p\fR again, would recall the "ls ~/tecla/" line, and
355 hitting it once more would recall the "ls ~/tecla/" line. The pattern syntax is
356 the same as that described for file name expansion, in the
357 \fBef_expand_file\fR(3TECLA).
358 .SS "History Files"
359 .LP
360 Authors of programs that use the tecla library have the option of saving
361 historical command-lines in a file before exiting, and subsequently reading
362 them back in from this file when the program is next started. There is no
363 standard name for this file, since it makes sense for each application to use
364 its own history file, so that commands from different applications don't get
365 mixed up.
366 .SS "International Character Sets"
367 .LP
368 Since \fBlibtecla\fR version 1.4.0, tecla has been 8-bit clean. This means that
369 all 8-bit characters that are printable in the user's current locale are now
370 displayed verbatim and included in the returned input line. Assuming that the
371 calling program correctly contains a call like the following,
372 .sp
373 .in +2
374 .nf
375 setlocale(LC_CTYPE, "");
376 .fi
377 .in -2

379 .sp
380 .LP
381 then the current locale is determined by the first of the environment variables
382 \fBLC_CTYPE\fR, \fBLC_ALL\fR, and \fBBLANG\fR, that is found to contain a valid
383 locale name. If none of these variables are defined, or the program neglects to
384 call \fBsetlocale\fR, then the default C locale is used, which is US 7-bit
385 ASCII. On most unix-like platforms, you can get a list of valid locales by
386 typing the command:
387 .sp
388 .in +2
389 .nf

```

```

390 locale -a
391 .fi
392 .in -2

394 .sp
395 .LP
396 at the shell prompt.
397 .SS "Meta Keys and Locales"
398 .LP
399 Beware that in most locales other than the default C locale, META characters
400 become printable, and they are then no longer considered to match \fBm-c\fR
401 style key bindings. This allows international characters to be entered with the
402 compose key without unexpectedly triggering META key bindings. You can still
403 invoke META bindings, since there are actually two ways to do this. For example
404 the binding \fBm-c\fR can also be invoked by pressing the ESCAPE key
405 momentarily, then pressing the c key, and this will work regardless of locale.
406 Moreover, many modern terminal emulators, such as gnome's gnome-terminal's and
407 KDE's konsole terminals, already generate escape pairs like this when you use
408 the META key, rather than a real meta character, and other emulators usually
409 have a way to request this behavior, so you can continue to use the META key on
410 most systems.
411 .sp
412 .LP
413 For example, although xterm terminal emulators generate real 8-bit meta
414 characters by default when you use the META key, they can be configured to
415 output the equivalent escape pair by setting their \fBEightBitInput\fR X
416 resource to False. You can either do this by placing a line like the following
417 in your \fB~/.Xdefaults\fR file,
418 .sp
419 .in +2
420 .nf
421 XTerm*EightBitInput: False
422 .fi
423 .in -2

425 .sp
426 .LP
427 or by starting an \fBxterm\fR with an \fB-xrm\fR \&'*EightBitInput: False'
428 command-line argument. In recent versions of xterm you can toggle this feature
429 on and off with the 'Meta Sends Escape' option in the menu that is displayed
430 when you press the left mouse button and the CONTROL key within an xterm
431 window. In CDE, dtterms can be similarly coerced to generate escape pairs in
432 place of meta characters, by setting the \fBDtterm*KshMode\fR resource to True.
433 .SS "Entering International Characters"
434 .LP
435 If you don't have a keyboard that generates all of the international characters
436 that you need, there is usually a compose key that will allow you to enter
437 special characters, or a way to create one. For example, under X windows on
438 unix-like systems, if your keyboard doesn't have a compose key, you can
439 designate a redundant key to serve this purpose with the xmodmap command. For
440 example, on many PC keyboards there is a microsoft-windows key, which is
441 otherwise useless under Linux. On a laptop, for example, the \fBxev\fR program
442 might report that pressing this key generates keycode 115. To turn this key
443 into a COMPOSE key, do the following:
444 .sp
445 .in +2
446 .nf
447 xmodmap -e 'keycode 115 = Multi_key'
448 .fi
449 .in -2

451 .sp
452 .LP
453 Type this key followed by a " character to enter an 'I' with a umlaut over it.
454 .SS "The Available Key Binding Functions"
455 .LP

```

```

456 The following is a list of the editing functions provided by the tecla library.
457 The names in the leftmost column of the list can be used in configuration files
458 to specify which function a given key or combination of keys should invoke.
459 They are also used in the next two sections to list the default key bindings in
460 \fBemacsfR and \fBvifR modes.
461 .sp
462 .ne 2
463 .na
464 \fBuser-interrupt\fR
465 .ad
466 .RS 30n
467 Send a SIGINT signal to the parent process.
468 .RE

470 .sp
471 .ne 2
472 .na
473 \fBsuspend\fR
474 .ad
475 .RS 30n
476 Suspend the parent process.
477 .RE

479 .sp
480 .ne 2
481 .na
482 \fBstop-output\fR
483 .ad
484 .RS 30n
485 Pause terminal output.
486 .RE

488 .sp
489 .ne 2
490 .na
491 \fBstart-output\fR
492 .ad
493 .RS 30n
494 Resume paused terminal output.
495 .RE

497 .sp
498 .ne 2
499 .na
500 \fBliteral-next\fR
501 .ad
502 .RS 30n
503 Arrange for the next character to be treated as a normal character. This allows
504 control characters to be entered.
505 .RE

507 .sp
508 .ne 2
509 .na
510 \fBcursor-right\fR
511 .ad
512 .RS 30n
513 Move the cursor one character right.
514 .RE

516 .sp
517 .ne 2
518 .na
519 \fBcursor-left\fR
520 .ad
521 .RS 30n

```

```

522 Move the cursor one character left.
523 .RE

525 .sp
526 .ne 2
527 .na
528 \fBinsert-mode\fR
529 .ad
530 .RS 30n
531 Toggle between insert mode and overwrite mode.
532 .RE

534 .sp
535 .ne 2
536 .na
537 \fBbeginning-of-line\fR
538 .ad
539 .RS 30n
540 Move the cursor to the beginning of the line.
541 .RE

543 .sp
544 .ne 2
545 .na
546 \fBend-of-line\fR
547 .ad
548 .RS 30n
549 Move the cursor to the end of the line.
550 .RE

552 .sp
553 .ne 2
554 .na
555 \fBdelete-line\fR
556 .ad
557 .RS 30n
558 Delete the contents of the current line.
559 .RE

561 .sp
562 .ne 2
563 .na
564 \fBkill-line\fR
565 .ad
566 .RS 30n
567 Delete everything that follows the cursor.
568 .RE

570 .sp
571 .ne 2
572 .na
573 \fBbackward-kill-line\fR
574 .ad
575 .RS 30n
576 Delete all characters between the cursor and the start of the line.
577 .RE

579 .sp
580 .ne 2
581 .na
582 \fBforward-word\fR
583 .ad
584 .RS 30n
585 Move to the end of the word which follows the cursor.
586 .RE

```

```

588 .sp
589 .ne 2
590 .na
591 \fBforward-to-word\fR
592 .ad
593 .RS 30n
594 Move the cursor to the start of the word that follows the cursor.
595 .RE

597 .sp
598 .ne 2
599 .na
600 \fBbackward-word\fR
601 .ad
602 .RS 30n
603 Move to the start of the word which precedes the cursor.
604 .RE

606 .sp
607 .ne 2
608 .na
609 \fBgoto-column\fR
610 .ad
611 .RS 30n
612 Move the cursor to the l-relative column in the line specified by any preceding
613 digit-argument sequences (see Entering Repeat Counts below).
614 .RE

616 .sp
617 .ne 2
618 .na
619 \fBfind-parenthesis\fR
620 .ad
621 .RS 30n
622 If the cursor is currently over a parenthesis character, move it to the
623 matching parenthesis character. If not over a parenthesis character move right
624 to the next close parenthesis.
625 .RE

627 .sp
628 .ne 2
629 .na
630 \fBforward-delete-char\fR
631 .ad
632 .RS 30n
633 Delete the character under the cursor.
634 .RE

636 .sp
637 .ne 2
638 .na
639 \fBbackward-delete-char\fR
640 .ad
641 .RS 30n
642 Delete the character which precedes the cursor.
643 .RE

645 .sp
646 .ne 2
647 .na
648 \fBlist-or-eof\fR
649 .ad
650 .RS 30n
651 This is intended for binding to \fB^D\fR. When invoked when the cursor is
652 within the line it displays all possible completions then redisplay the line
653 unchanged. When invoked on an empty line, it signals end-of-input (EOF) to the

```

```

654 caller of \fBgl_get_line()\fR.
655 .RE

657 .sp
658 .ne 2
659 .na
660 \fBdel-char-or-list-or-eof\fR
661 .ad
662 .RS 30n
663 This is intended for binding to \fB^D\fR. When invoked when the cursor is
664 within the line it invokes forward-delete-char. When invoked at the end of the
665 line it displays all possible completions then redisplay the line unchanged.
666 When invoked on an empty line, it signals end-of-input (EOF) to the caller of
667 \fBgl_get_line()\fR.
668 .RE

670 .sp
671 .ne 2
672 .na
673 \fBforward-delete-word\fR
674 .ad
675 .RS 30n
676 Delete the word which follows the cursor.
677 .RE

679 .sp
680 .ne 2
681 .na
682 \fBbackward-delete-word\fR
683 .ad
684 .RS 30n
685 Delete the word which precedes the cursor.
686 .RE

688 .sp
689 .ne 2
690 .na
691 \fBupcase-word\fR
692 .ad
693 .RS 30n
694 Convert all of the characters of the word which follows the cursor, to upper
695 case.
696 .RE

698 .sp
699 .ne 2
700 .na
701 \fBdowncase-word\fR
702 .ad
703 .RS 30n
704 Convert all of the characters of the word which follows the cursor, to lower
705 case.
706 .RE

708 .sp
709 .ne 2
710 .na
711 \fBcapitalize-word\fR
712 .ad
713 .RS 30n
714 Capitalize the word which follows the cursor.
715 .RE

717 .sp
718 .ne 2
719 .na

```

```

720 \fBchange-case\fR
721 .ad
722 .RS 30n
723 If the next character is upper case, toggle it to lower case and vice versa.
724 .RE

726 .sp
727 .ne 2
728 .na
729 \fBredisplay\fR
730 .ad
731 .RS 30n
732 Redisplay the line.
733 .RE

735 .sp
736 .ne 2
737 .na
738 \fBclear-screen\fR
739 .ad
740 .RS 30n
741 Clear the terminal, then redisplay the current line.
742 .RE

744 .sp
745 .ne 2
746 .na
747 \fBtranspose-chars\fR
748 .ad
749 .RS 30n
750 Swap the character under the cursor with the character just before the cursor.
751 .RE

753 .sp
754 .ne 2
755 .na
756 \fBset-mark\fR
757 .ad
758 .RS 30n
759 Set a mark at the position of the cursor.
760 .RE

762 .sp
763 .ne 2
764 .na
765 \fBexchange-point-and-mark\fR
766 .ad
767 .RS 30n
768 Move the cursor to the last mark that was set, and move the mark to where the
769 cursor used to be.
770 .RE

772 .sp
773 .ne 2
774 .na
775 \fBkill-region\fR
776 .ad
777 .RS 30n
778 Delete the characters that lie between the last mark that was set, and the
779 cursor.
780 .RE

782 .sp
783 .ne 2
784 .na
785 \fBcopy-region-as-kill\fR

```

```

786 .ad
787 .RS 30n
788 Copy the text between the mark and the cursor to the cut buffer, without
789 deleting the original text.
790 .RE

792 .sp
793 .ne 2
794 .na
795 \fByank\fR
796 .ad
797 .RS 30n
798 Insert the text that was last deleted, just before the current position of the
799 cursor.
800 .RE

802 .sp
803 .ne 2
804 .na
805 \fBappend-yank\fR
806 .ad
807 .RS 30n
808 Paste the current contents of the cut buffer, after the cursor.
809 .RE

811 .sp
812 .ne 2
813 .na
814 \fBup-history\fR
815 .ad
816 .RS 30n
817 Recall the next oldest line that was entered. Note that in \fBvi\fR mode you
818 are left in command mode.
819 .RE

821 .sp
822 .ne 2
823 .na
824 \fBdown-history\fR
825 .ad
826 .RS 30n
827 Recall the next most recent line that was entered. If no history recall session
828 is currently active, the next line from a previous recall session is recalled.
829 Note that in vi mode you are left in command mode.
830 .RE

832 .sp
833 .ne 2
834 .na
835 \fBhistory-search-backward\fR
836 .ad
837 .RS 30n
838 Recall the next oldest line who's prefix matches the string which currently
839 precedes the cursor (in \fBvi\fR command-mode the character under the cursor is
840 also included in the search string). Note that in \fBvi\fR mode you are left in
841 command mode.
842 .RE

844 .sp
845 .ne 2
846 .na
847 \fBhistory-search-forward\fR
848 .ad
849 .RS 30n
850 Recall the next newest line who's prefix matches the string which currently
851 precedes the cursor (in \fBvi\fR command-mode the character under the cursor is

```

```

852 also included in the search string). Note that in \fBvi\fR mode you are left in
853 command mode.
854 .RE

856 .sp
857 .ne 2
858 .na
859 \fBhistory-re-search-backward\fR
860 .ad
861 .RS 30n
862 Recall the next oldest line who's prefix matches that established by the last
863 invocation of either history-search-forward or history-search-backward.
864 .RE

866 .sp
867 .ne 2
868 .na
869 \fBhistory-re-search-forward\fR
870 .ad
871 .RS 30n
872 Recall the next newest line who's prefix matches that established by the last
873 invocation of either history-search-forward or history-search-backward.
874 .RE

876 .sp
877 .ne 2
878 .na
879 \fBcomplete-word\fR
880 .ad
881 .RS 30n
882 Attempt to complete the incomplete word which precedes the cursor. Unless the
883 host program has customized word completion, file name completion is attempted.
884 In \fBvi\fR command mode the character under the cursor is also included in
885 the word being completed, and you are left in \fBvi\fR insert mode.
886 .RE

888 .sp
889 .ne 2
890 .na
891 \fBexpand-filename\fR
892 .ad
893 .RS 30n
894 Within the command line, expand wild cards, tilde expressions and dollar
895 expressions in the file name which immediately precedes the cursor. In \fBvi\fR
896 command mode the character under the cursor is also included in the file name
897 being expanded, and you are left in \fBvi\fR insert mode.
898 .RE

900 .sp
901 .ne 2
902 .na
903 \fBlist-glob\fR
904 .ad
905 .RS 30n
906 List any file names which match the wild-card, tilde and dollar expressions in
907 the file name which immediately precedes the cursor, then redraw the input line
908 unchanged.
909 .RE

911 .sp
912 .ne 2
913 .na
914 \fBlist-history\fR
915 .ad
916 .RS 30n
917 Display the contents of the history list for the current history group. If a

```

918 repeat count of \fB> 1\fR is specified, only that many of the most recent lines
 919 are displayed. See the Entering Repeat Counts section.
 920 .RE

922 .sp
 923 .ne 2
 924 .na
 925 \fBread-from-file\fR
 926 .ad
 927 .RS 30n
 928 Temporarily switch to reading input from the file whose name precedes the
 929 cursor.
 930 .RE

932 .sp
 933 .ne 2
 934 .na
 935 \fBread-init-files\fR
 936 .ad
 937 .RS 30n
 938 Re-read \fBteclarc\fR configuration files.
 939 .RE

941 .sp
 942 .ne 2
 943 .na
 944 \fBbeginning-of-history\fR
 945 .ad
 946 .RS 30n
 947 Move to the oldest line in the history list. Note that in \fBvi\fR mode you are
 948 left in command mode.
 949 .RE

951 .sp
 952 .ne 2
 953 .na
 954 \fBend-of-history\fR
 955 .ad
 956 .RS 30n
 957 Move to the newest line in the history list (ie. the current line). Note that
 958 in \fBvi\fR mode this leaves you in command mode.
 959 .RE

961 .sp
 962 .ne 2
 963 .na
 964 \fBdigit-argument\fR
 965 .ad
 966 .RS 30n
 967 Enter a repeat count for the next key binding function. For details, see the
 968 Entering Repeat Counts section.
 969 .RE

971 .sp
 972 .ne 2
 973 .na
 974 \fBnewline\fR
 975 .ad
 976 .RS 30n
 977 Terminate and return the current contents of the line, after appending a
 978 newline character. The newline character is normally '\en', but will be the
 979 first character of the key sequence that invoked the newline action, if this
 980 happens to be a printable character. If the action was invoked by the '\en'
 981 newline character or the '\er' carriage return character, the line is appended
 982 to the history buffer.
 983 .RE

985 .sp
 986 .ne 2
 987 .na
 988 \fBrepeat-history\fR
 989 .ad
 990 .RS 30n
 991 Return the line that is being edited, then arrange for the next most recent
 992 entry in the history buffer to be recalled when tecla is next called.
 993 Repeatedly invoking this action causes successive historical input lines to be
 994 re-executed. Note that this action is equivalent to the 'Operate' action in
 995 ksh.
 996 .RE

998 .sp
 999 .ne 2
 1000 .na
 1001 \fBbring-bell\fR
 1002 .ad
 1003 .RS 30n
 1004 Ring the terminal bell, unless the bell has been silenced via the nobeep
 1005 configuration option (see The Tecla Configuration File section).
 1006 .RE

1008 .sp
 1009 .ne 2
 1010 .na
 1011 \fBforward-copy-char\fR
 1012 .ad
 1013 .RS 30n
 1014 Copy the next character into the cut buffer (NB. use repeat counts to copy more
 1015 than one).
 1016 .RE

1018 .sp
 1019 .ne 2
 1020 .na
 1021 \fBbackward-copy-char\fR
 1022 .ad
 1023 .RS 30n
 1024 Copy the previous character into the cut buffer.
 1025 .RE

1027 .sp
 1028 .ne 2
 1029 .na
 1030 \fBforward-copy-word\fR
 1031 .ad
 1032 .RS 30n
 1033 Copy the next word into the cut buffer.
 1034 .RE

1036 .sp
 1037 .ne 2
 1038 .na
 1039 \fBbackward-copy-word\fR
 1040 .ad
 1041 .RS 30n
 1042 Copy the previous word into the cut buffer.
 1043 .RE

1045 .sp
 1046 .ne 2
 1047 .na
 1048 \fBforward-find-char\fR
 1049 .ad

```

1050 .RS 30n
1051 Move the cursor to the next occurrence of the next character that you type.
1052 .RE

1054 .sp
1055 .ne 2
1056 .na
1057 \fBbackward-find-char\fR
1058 .ad
1059 .RS 30n
1060 Move the cursor to the last occurrence of the next character that you type.
1061 .RE

1063 .sp
1064 .ne 2
1065 .na
1066 \fBforward-to-char\fR
1067 .ad
1068 .RS 30n
1069 Move the cursor to the character just before the next occurrence of the next
1070 character that the user types.
1071 .RE

1073 .sp
1074 .ne 2
1075 .na
1076 \fBbackward-to-char\fR
1077 .ad
1078 .RS 30n
1079 Move the cursor to the character just after the last occurrence before the
1080 cursor of the next character that the user types.
1081 .RE

1083 .sp
1084 .ne 2
1085 .na
1086 \fBrepeat-find-char\fR
1087 .ad
1088 .RS 30n
1089 Repeat the last backward-find-char, forward-find-char, backward-to-char or
1090 forward-to-char.
1091 .RE

1093 .sp
1094 .ne 2
1095 .na
1096 \fBinvert-refind-char\fR
1097 .ad
1098 .RS 30n
1099 Repeat the last backward-find-char, forward-find-char, backward-to-char, or
1100 forward-to-char in the opposite direction.
1101 .RE

1103 .sp
1104 .ne 2
1105 .na
1106 \fBdelete-to-column\fR
1107 .ad
1108 .RS 30n
1109 Delete the characters from the cursor up to the column that is specified by the
1110 repeat count.
1111 .RE

1113 .sp
1114 .ne 2
1115 .na

```

```

1116 \fBdelete-to-parenthesis\fR
1117 .ad
1118 .RS 30n
1119 Delete the characters from the cursor up to and including the matching
1120 parenthesis, or next close parenthesis.
1121 .RE

1123 .sp
1124 .ne 2
1125 .na
1126 \fBforward-delete-find\fR
1127 .ad
1128 .RS 30n
1129 Delete the characters from the cursor up to and including the following
1130 occurrence of the next character typed.
1131 .RE

1133 .sp
1134 .ne 2
1135 .na
1136 \fBbackward-delete-find\fR
1137 .ad
1138 .RS 30n
1139 Delete the characters from the cursor up to and including the preceding
1140 occurrence of the next character typed.
1141 .RE

1143 .sp
1144 .ne 2
1145 .na
1146 \fBforward-delete-to\fR
1147 .ad
1148 .RS 30n
1149 Delete the characters from the cursor up to, but not including, the following
1150 occurrence of the next character typed.
1151 .RE

1153 .sp
1154 .ne 2
1155 .na
1156 \fBbackward-delete-to\fR
1157 .ad
1158 .RS 30n
1159 Delete the characters from the cursor up to, but not including, the preceding
1160 occurrence of the next character typed.
1161 .RE

1163 .sp
1164 .ne 2
1165 .na
1166 \fBdelete-refind\fR
1167 .ad
1168 .RS 30n
1169 Repeat the last *-delete-find or *-delete-to action.
1170 .RE

1172 .sp
1173 .ne 2
1174 .na
1175 \fBdelete-invert-refind\fR
1176 .ad
1177 .RS 30n
1178 Repeat the last *-delete-find or *-delete-to action, in the opposite direction.
1179 .RE

1181 .sp

```



```

1182 .ne 2
1183 .na
1184 \fBcopy-to-column\fR
1185 .ad
1186 .RS 30n
1187 Copy the characters from the cursor up to the column that is specified by the
1188 repeat count, into the cut buffer.
1189 .RE

1191 .sp
1192 .ne 2
1193 .na
1194 \fBcopy-to-parenthesis\fR
1195 .ad
1196 .RS 30n
1197 Copy the characters from the cursor up to and including the matching
1198 parenthesis, or next close parenthesis, into the cut buffer.
1199 .RE

1201 .sp
1202 .ne 2
1203 .na
1204 \fBforward-copy-find\fR
1205 .ad
1206 .RS 30n
1207 Copy the characters from the cursor up to and including the following occurrence
1208 of the next character typed, into the cut buffer.
1209 .RE

1211 .sp
1212 .ne 2
1213 .na
1214 \fBbackward-copy-find\fR
1215 .ad
1216 .RS 30n
1217 Copy the characters from the cursor up to and including the preceding occurrence
1218 of the next character typed, into the cut buffer.
1219 .RE

1221 .sp
1222 .ne 2
1223 .na
1224 \fBforward-copy-to\fR
1225 .ad
1226 .RS 30n
1227 Copy the characters from the cursor up to, but not including, the following
1228 occurrence of the next character typed, into the cut buffer.
1229 .RE

1231 .sp
1232 .ne 2
1233 .na
1234 \fBbackward-copy-to\fR
1235 .ad
1236 .RS 30n
1237 Copy the characters from the cursor up to, but not including, the preceding
1238 occurrence of the next character typed, into the cut buffer.
1239 .RE

1241 .sp
1242 .ne 2
1243 .na
1244 \fBcopy-refind\fR
1245 .ad
1246 .RS 30n
1247 Repeat the last *-copy-find or *-copy-to action.

```

```

1248 .RE

1250 .sp
1251 .ne 2
1252 .na
1253 \fBcopy-invert-refind\fR
1254 .ad
1255 .RS 30n
1256 Repeat the last *-copy-find or *-copy-to action, in the opposite direction.
1257 .RE

1259 .sp
1260 .ne 2
1261 .na
1262 \fBvi-mode\fR
1263 .ad
1264 .RS 30n
1265 Switch to \fBvi\fR mode from emacs mode.
1266 .RE

1268 .sp
1269 .ne 2
1270 .na
1271 \fBemacs-mode\fR
1272 .ad
1273 .RS 30n
1274 Switch to \fBemacs\fR mode from \fBvi\fR mode.
1275 .RE

1277 .sp
1278 .ne 2
1279 .na
1280 \fBvi-insert\fR
1281 .ad
1282 .RS 30n
1283 From \fBvi\fR command mode, switch to insert mode.
1284 .RE

1286 .sp
1287 .ne 2
1288 .na
1289 \fBvi-overwrite\fR
1290 .ad
1291 .RS 30n
1292 From \fBvi\fR command mode, switch to overwrite mode.
1293 .RE

1295 .sp
1296 .ne 2
1297 .na
1298 \fBvi-insert-at-bol\fR
1299 .ad
1300 .RS 30n
1301 From \fBvi\fR command mode, move the cursor to the start of the line and switch
1302 to insert mode.
1303 .RE

1305 .sp
1306 .ne 2
1307 .na
1308 \fBvi-append-at-eol\fR
1309 .ad
1310 .RS 30n
1311 From \fBvi\fR command mode, move the cursor to the end of the line and switch
1312 to append mode.
1313 .RE

```

```

1315 .sp
1316 .ne 2
1317 .na
1318 \fBvi-append\fR
1319 .ad
1320 .RS 30n
1321 From \fBvi\fR command mode, move the cursor one position right, and switch to
1322 insert mode.
1323 .RE

1325 .sp
1326 .ne 2
1327 .na
1328 \fBvi-replace-char\fR
1329 .ad
1330 .RS 30n
1331 From \fBvi\fR command mode, replace the character under the cursor with the
1332 next character entered.
1333 .RE

1335 .sp
1336 .ne 2
1337 .na
1338 \fBvi-forward-change-char\fR
1339 .ad
1340 .RS 30n
1341 From \fBvi\fR command mode, delete the next character then enter insert mode.
1342 .RE

1344 .sp
1345 .ne 2
1346 .na
1347 \fBvi-backward-change-char\fR
1348 .ad
1349 .RS 30n
1350 From vi command mode, delete the preceding character then enter insert mode.
1351 .RE

1353 .sp
1354 .ne 2
1355 .na
1356 \fBvi-forward-change-word\fR
1357 .ad
1358 .RS 30n
1359 From \fBvi\fR command mode, delete the next word then enter insert mode.
1360 .RE

1362 .sp
1363 .ne 2
1364 .na
1365 \fBvi-backward-change-word\fR
1366 .ad
1367 .RS 30n
1368 From vi command mode, delete the preceding word then enter insert mode.
1369 .RE

1371 .sp
1372 .ne 2
1373 .na
1374 \fBvi-change-rest-of-line\fR
1375 .ad
1376 .RS 30n
1377 From \fBvi\fR command mode, delete from the cursor to the end of the line, then
1378 enter insert mode.
1379 .RE

```

```

1381 .sp
1382 .ne 2
1383 .na
1384 \fBvi-change-line\fR
1385 .ad
1386 .RS 30n
1387 From \fBvi\fR command mode, delete the current line, then enter insert mode.
1388 .RE

1390 .sp
1391 .ne 2
1392 .na
1393 \fBvi-change-to-bol\fR
1394 .ad
1395 .RS 30n
1396 From \fBvi\fR command mode, delete all characters between the cursor and the
1397 beginning of the line, then enter insert mode.
1398 .RE

1400 .sp
1401 .ne 2
1402 .na
1403 \fBvi-change-to-column\fR
1404 .ad
1405 .RS 30n
1406 From \fBvi\fR command mode, delete the characters from the cursor up to the
1407 column that is specified by the repeat count, then enter insert mode.
1408 .RE

1410 .sp
1411 .ne 2
1412 .na
1413 \fBvi-change-to-parenthesis\fR
1414 .ad
1415 .RS 30n
1416 Delete the characters from the cursor up to and including the matching
1417 parenthesis, or next close parenthesis, then enter \fBvi\fR insert mode.
1418 .RE

1420 .sp
1421 .ne 2
1422 .na
1423 \fBvi-forward-change-find\fR
1424 .ad
1425 .RS 30n
1426 From \fBvi\fR command mode, delete the characters from the cursor up to and
1427 including the following occurrence of the next character typed, then enter
1428 insert mode.
1429 .RE

1431 .sp
1432 .ne 2
1433 .na
1434 \fBvi-backward-change-find\fR
1435 .ad
1436 .RS 30n
1437 From vi command mode, delete the characters from the cursor up to and including
1438 the preceding occurrence of the next character typed, then enter insert mode.
1439 .RE

1441 .sp
1442 .ne 2
1443 .na
1444 \fBvi-forward-change-to\fR
1445 .ad

```

```

1446 .RS 30n
1447 From \fBvi\fR command mode, delete the characters from the cursor up to, but
1448 not including, the following occurrence of the next character typed, then enter
1449 insert mode.
1450 .RE

1452 .sp
1453 .ne 2
1454 .na
1455 \fBvi-backward-change-to\fR
1456 .ad
1457 .RS 30n
1458 From \fBvi\fR command mode, delete the characters from the cursor up to, but
1459 not including, the preceding occurrence of the next character typed, then enter
1460 insert mode.
1461 .RE

1463 .sp
1464 .ne 2
1465 .na
1466 \fBvi-change-refind\fR
1467 .ad
1468 .RS 30n
1469 Repeat the last vi-*-change-find or vi-*-change-to action.
1470 .RE

1472 .sp
1473 .ne 2
1474 .na
1475 \fBvi-change-invert-refind\fR
1476 .ad
1477 .RS 30n
1478 Repeat the last vi-*-change-find or vi-*-change-to action, in the opposite
1479 direction.
1480 .RE

1482 .sp
1483 .ne 2
1484 .na
1485 \fBvi-undo\fR
1486 .ad
1487 .RS 30n
1488 In \fBvi\fR mode, undo the last editing operation.
1489 .RE

1491 .sp
1492 .ne 2
1493 .na
1494 \fBvi-repeat-change\fR
1495 .ad
1496 .RS 30n
1497 In \fBvi\fR command mode, repeat the last command that modified the line.
1498 .RE

1500 .SS "Default Key Bindings In \fBemacs\fR Mode"
1501 .LP
1502 The following default key bindings, which can be overridden by the tecla
1503 The following default key bindings, which can be overridden by the tecla
1504 configuration file, are designed to mimic most of the bindings of the unix
1504 \fBtcsh\fR shell, when it is in \fBemacs\fR editing mode.
1504 \fBtcsh shell\fR shell, when it is in \fBemacs\fR editing mode.
1505 .sp
1506 .LP
1507 This is the default editing mode of the tecla library.
1508 .sp
1509 .LP

```

```

1510 Under UNIX the terminal driver sets a number of special keys for certain
1511 functions. The tecla library attempts to use the same key bindings to maintain
1512 consistency. The key sequences shown for the following 6 bindings are thus just
1513 examples of what they will probably be set to. If you have used the stty
1514 command to change these keys, then the default bindings should match.
1515 .sp
1516 .ne 2
1517 .na
1518 \fB\fB^C\fR\fR
1519 .ad
1520 .RS 6n
1521 user-interrupt
1522 .RE

1524 .sp
1525 .ne 2
1526 .na
1527 \fB^e\fR
1528 .ad
1529 .RS 6n
1530 abort
1531 .RE

1533 .sp
1534 .ne 2
1535 .na
1536 \fB\fB^Z\fR\fR
1537 .ad
1538 .RS 6n
1539 suspend
1540 .RE

1542 .sp
1543 .ne 2
1544 .na
1545 \fB\fB^Q\fR\fR
1546 .ad
1547 .RS 6n
1548 start-output
1549 .RE

1551 .sp
1552 .ne 2
1553 .na
1554 \fB\fB^S\fR\fR
1555 .ad
1556 .RS 6n
1557 stop-output
1558 .RE

1560 .sp
1561 .ne 2
1562 .na
1563 \fB\fB^V\fR\fR
1564 .ad
1565 .RS 6n
1566 literal-next
1567 .RE

1569 .sp
1570 .LP
1571 The cursor keys are referred to by name, as follows. This is necessary because
1572 different types of terminals generate different key sequences when their cursor
1573 keys are pressed.
1574 .sp
1575 .ne 2

```

```

1576 .na
1577 \fBright\fR
1578 .ad
1579 .RS 9n
1580 cursor-right
1581 .RE

1583 .sp
1584 .ne 2
1585 .na
1586 \fBleft\fR
1587 .ad
1588 .RS 9n
1589 cursor-left
1590 .RE

1592 .sp
1593 .ne 2
1594 .na
1595 \fBup\fR
1596 .ad
1597 .RS 9n
1598 up-history
1599 .RE

1601 .sp
1602 .ne 2
1603 .na
1604 \fBdown\fR
1605 .ad
1606 .RS 9n
1607 down-history
1608 .RE

1610 .sp
1611 .LP
1612 The remaining bindings don't depend on the terminal settings.
1613 .sp
1614 .ne 2
1615 .na
1616 \fB\F\fR
1617 .ad
1618 .RS 21n
1619 cursor-right
1620 .RE

1622 .sp
1623 .ne 2
1624 .na
1625 \fB^B\fR
1626 .ad
1627 .RS 21n
1628 cursor-left
1629 .RE

1631 .sp
1632 .ne 2
1633 .na
1634 \fB^I\fR
1635 .ad
1636 .RS 21n
1637 insert-mode
1638 .RE

1640 .sp
1641 .ne 2

```

```

1642 .na
1643 \fB^A\fR
1644 .ad
1645 .RS 21n
1646 beginning-of-line
1647 .RE

1649 .sp
1650 .ne 2
1651 .na
1652 \fB^E\fR
1653 .ad
1654 .RS 21n
1655 end-of-line
1656 .RE

1658 .sp
1659 .ne 2
1660 .na
1661 \fB^U\fR
1662 .ad
1663 .RS 21n
1664 delete-line
1665 .RE

1667 .sp
1668 .ne 2
1669 .na
1670 \fB^K\fR
1671 .ad
1672 .RS 21n
1673 kill-line
1674 .RE

1676 .sp
1677 .ne 2
1678 .na
1679 \fB^f\fR
1680 .ad
1681 .RS 21n
1682 forward-word
1683 .RE

1685 .sp
1686 .ne 2
1687 .na
1688 \fB^b\fR
1689 .ad
1690 .RS 21n
1691 backward-word
1692 .RE

1694 .sp
1695 .ne 2
1696 .na
1697 \fB^D\fR
1698 .ad
1699 .RS 21n
1700 del-char-or-list-or-eof
1701 .RE

1703 .sp
1704 .ne 2
1705 .na
1706 \fB^H\fR
1707 .ad

```

```
1708 .RS 21n
1709 backward-delete-char
1710 .RE
```

```
1712 .sp
1713 .ne 2
1714 .na
1715 \fB\fB^?\fR\fR
1716 .ad
1717 .RS 21n
1718 backward-delete-char
1719 .RE
```

```
1721 .sp
1722 .ne 2
1723 .na
1724 \fB\fBM-d\fR\fR
1725 .ad
1726 .RS 21n
1727 forward-delete-word
1728 .RE
```

```
1730 .sp
1731 .ne 2
1732 .na
1733 \fB\fBM-^H\fR\fR
1734 .ad
1735 .RS 21n
1736 backward-delete-word
1737 .RE
```

```
1739 .sp
1740 .ne 2
1741 .na
1742 \fB\fBM-^?\fR\fR
1743 .ad
1744 .RS 21n
1745 backward-delete-word
1746 .RE
```

```
1748 .sp
1749 .ne 2
1750 .na
1751 \fB\fBM-u\fR\fR
1752 .ad
1753 .RS 21n
1754 upcase-word
1755 .RE
```

```
1757 .sp
1758 .ne 2
1759 .na
1760 \fB\fBM-l\fR\fR
1761 .ad
1762 .RS 21n
1763 downcase-word
1764 .RE
```

```
1766 .sp
1767 .ne 2
1768 .na
1769 \fB\fBM-c\fR\fR
1770 .ad
1771 .RS 21n
1772 capitalize-word
1773 .RE
```

```
1775 .sp
1776 .ne 2
1777 .na
1778 \fB\fB^R\fR\fR
1779 .ad
1780 .RS 21n
1781 redisplay
1782 .RE
```

```
1784 .sp
1785 .ne 2
1786 .na
1787 \fB\fB^L\fR\fR
1788 .ad
1789 .RS 21n
1790 clear-screen
1791 .RE
```

```
1793 .sp
1794 .ne 2
1795 .na
1796 \fB\fB^T\fR\fR
1797 .ad
1798 .RS 21n
1799 transpose-chars
1800 .RE
```

```
1802 .sp
1803 .ne 2
1804 .na
1805 \fB\fB^@\fR\fR
1806 .ad
1807 .RS 21n
1808 set-mark
1809 .RE
```

```
1811 .sp
1812 .ne 2
1813 .na
1814 \fB\fB^X^X\fR\fR
1815 .ad
1816 .RS 21n
1817 exchange-point-and-mark
1818 .RE
```

```
1820 .sp
1821 .ne 2
1822 .na
1823 \fB\fB^W\fR\fR
1824 .ad
1825 .RS 21n
1826 kill-region
1827 .RE
```

```
1829 .sp
1830 .ne 2
1831 .na
1832 \fB\fBM-w\fR\fR
1833 .ad
1834 .RS 21n
1835 copy-region-as-kill
1836 .RE
```

```
1838 .sp
1839 .ne 2
```

```

1840 .na
1841 \fB\fB^Y\fR\fR
1842 .ad
1843 .RS 2ln
1844 yank
1845 .RE

1847 .sp
1848 .ne 2
1849 .na
1850 \fB\fB^P\fR\fR
1851 .ad
1852 .RS 2ln
1853 up-history
1854 .RE

1856 .sp
1857 .ne 2
1858 .na
1859 \fB\fB^N\fR\fR
1860 .ad
1861 .RS 2ln
1862 down-history
1863 .RE

1865 .sp
1866 .ne 2
1867 .na
1868 \fB\fB^M-p\fR\fR
1869 .ad
1870 .RS 2ln
1871 history-search-backward
1872 .RE

1874 .sp
1875 .ne 2
1876 .na
1877 \fB\fB^M-n\fR\fR
1878 .ad
1879 .RS 2ln
1880 history-search-forward
1881 .RE

1883 .sp
1884 .ne 2
1885 .na
1886 \fB\fB^I\fR\fR
1887 .ad
1888 .RS 2ln
1889 complete-word
1890 .RE

1892 .sp
1893 .ne 2
1894 .na
1895 \fB\fB^X*\fR\fR
1896 .ad
1897 .RS 2ln
1898 expand-filename
1899 .RE

1901 .sp
1902 .ne 2
1903 .na
1904 \fB\fB^X^F\fR\fR
1905 .ad

```

```

1906 .RS 2ln
1907 read-from-file
1908 .RE

1910 .sp
1911 .ne 2
1912 .na
1913 \fB\fB^X^R\fR\fR
1914 .ad
1915 .RS 2ln
1916 read-init-files
1917 .RE

1919 .sp
1920 .ne 2
1921 .na
1922 \fB\fB^Xg\fR\fR
1923 .ad
1924 .RS 2ln
1925 list-glob
1926 .RE

1928 .sp
1929 .ne 2
1930 .na
1931 \fB\fB^Xh\fR\fR
1932 .ad
1933 .RS 2ln
1934 list-history
1935 .RE

1937 .sp
1938 .ne 2
1939 .na
1940 \fB\fB^M-<\fR\fR
1941 .ad
1942 .RS 2ln
1943 beginning-of-history
1944 .RE

1946 .sp
1947 .ne 2
1948 .na
1949 \fB\fB^M->\fR\fR
1950 .ad
1951 .RS 2ln
1952 end-of-history
1953 .RE

1955 .sp
1956 .ne 2
1957 .na
1958 \fB\fB^M-\fR\fR
1959 .ad
1960 .RS 2ln
1961 newline
1962 .RE

1964 .sp
1965 .ne 2
1966 .na
1967 \fB\fB^M-\fR\fR
1968 .ad
1969 .RS 2ln
1970 newline
1971 .RE

```

```

1973 .sp
1974 .ne 2
1975 .na
1976 \fB\FBM-o\fR\fR
1977 .ad
1978 .RS 2ln
1979 repeat-history
1980 .RE

1982 .sp
1983 .ne 2
1984 .na
1985 \fB\FBM-^v\fR\fR
1986 .ad
1987 .RS 2ln
1988 \fBvi\fR-mode
1989 .RE

1991 .sp
1992 .ne 2
1993 .na
1994 \fB\FBM-0, M-1, ... M-9\fR\fR
1995 .ad
1996 .RS 2ln
1997 digit-argument (see below)
1998 .RE

2000 .sp
2001 .LP
2002 Note that \fB^I\fR is what the TAB key generates, and that \fB^@\fR can be
2003 generated not only by pressing the CONTROL key and the @ key simultaneously,
2004 but also by pressing the CONTROL key and the space bar at the same time.
2005 .SS "Default Key Bindings in \fBvi\fR Mode"
2006 .LP
2007 The following default key bindings are designed to mimic the \fBvi\fR style of
2008 editing as closely as possible. This means that very few editing functions are
2009 provided in the initial character input mode, editing functions instead being
2010 provided by the \fBvi\fR command mode. The \fBvi\fR command mode is entered
2011 whenever the ESCAPE character is pressed, or whenever a key sequence that
2012 starts with a meta character is entered. In addition to mimicing \fBvi\fR,
2013 \fBlibtecla\fR provides bindings for tab completion, wild-card expansion of
2014 file names, and historical line recall.
2015 .sp
2016 .LP
2017 To learn how to tell the tecla library to use \fBvi\fR mode instead of the
2018 default \fBemacsvfR editing mode, see the earlier section entitled The Tecla
2019 Configuration File.
2020 .sp
2021 .LP
2022 Under UNIX the terminal driver sets a number of special keys for certain
2023 functions. The tecla library attempts to use the same key bindings to maintain
2024 consistency, binding them both in input mode and in command mode. The key
2025 sequences shown for the following 6 bindings are thus just examples of what
2026 they will probably be set to. If you have used the \fBbstty\fR command to change
2027 these keys, then the default bindings should match.
2028 .sp
2029 .ne 2
2030 .na
2031 \fB\FB^C\fR\fR
2032 .ad
2033 .RS 8n
2034 user-interrupt
2035 .RE

2037 .sp

```

```

2038 .ne 2
2039 .na
2040 \fB^e\fR
2041 .ad
2042 .RS 8n
2043 abort
2044 .RE

2046 .sp
2047 .ne 2
2048 .na
2049 \fB\FB^Z\fR\fR
2050 .ad
2051 .RS 8n
2052 suspend
2053 .RE

2055 .sp
2056 .ne 2
2057 .na
2058 \fB\FB^Q\fR\fR
2059 .ad
2060 .RS 8n
2061 start-output
2062 .RE

2064 .sp
2065 .ne 2
2066 .na
2067 \fB\FB^S\fR\fR
2068 .ad
2069 .RS 8n
2070 stop-output
2071 .RE

2073 .sp
2074 .ne 2
2075 .na
2076 \fB\FB^V\fR\fR
2077 .ad
2078 .RS 8n
2079 literal-next
2080 .RE

2082 .sp
2083 .ne 2
2084 .na
2085 \fB\FBM-^C\fR\fR
2086 .ad
2087 .RS 8n
2088 user-interrupt
2089 .RE

2091 .sp
2092 .ne 2
2093 .na
2094 \fB\FB-^e\fR
2095 .ad
2096 .RS 8n
2097 abort
2098 .RE

2100 .sp
2101 .ne 2
2102 .na
2103 \fB\FBM-^Z\fR\fR

```

```

2104 .ad
2105 .RS 8n
2106 suspend
2107 .RE

2109 .sp
2110 .ne 2
2111 .na
2112 \fB\fBM-^Q\fR\fR
2113 .ad
2114 .RS 8n
2115 start-output
2116 .RE

2118 .sp
2119 .ne 2
2120 .na
2121 \fB\fBM-^S\fR\fR
2122 .ad
2123 .RS 8n
2124 stop-output
2125 .RE

2127 .sp
2128 .LP
2129 Note that above, most of the bindings are defined twice, once as a raw control
2130 code like \fB^C\fR and then a second time as a META character like \fB^C\fR.
2131 The former is the binding for \fBvi\fR input mode, whereas the latter is the
2132 binding for \fBvi\fR command mode. Once in command mode all key sequences that
2133 the user types that they don't explicitly start with an ESCAPE or a META key,
2134 have their first key secretly converted to a META character before the key
2135 sequence is looked up in the key binding table. Thus, once in command mode,
2136 when you type the letter i, for example, the tecla library actually looks up
2137 the binding for \fBm-i\fR.
2138 .sp
2139 .LP
2140 The cursor keys are referred to by name, as follows. This is necessary because
2141 different types of terminals generate different key sequences when their cursor
2142 keys are pressed.
2143 .sp
2144 .ne 2
2145 .na
2146 \fB\fBright\fR\fR
2147 .ad
2148 .RS 9n
2149 cursor-right
2150 .RE

2152 .sp
2153 .ne 2
2154 .na
2155 \fB\fBleft\fR\fR
2156 .ad
2157 .RS 9n
2158 cursor-left
2159 .RE

2161 .sp
2162 .ne 2
2163 .na
2164 \fB\fBup\fR\fR
2165 .ad
2166 .RS 9n
2167 up-history
2168 .RE

```

```

2170 .sp
2171 .ne 2
2172 .na
2173 \fB\fBdown\fR\fR
2174 .ad
2175 .RS 9n
2176 down-history
2177 .RE

2179 .sp
2180 .LP
2181 The cursor keys normally generate a key sequence that start with an ESCAPE
2182 character, so beware that using the arrow keys will put you into command mode
2183 (if you aren't already in command mode).
2184 .sp
2185 .LP
2186 The following are the terminal-independent key bindings for \fBvi\fR input
2187 mode.
2188 .sp
2189 .ne 2
2190 .na
2191 \fB\fB^D\fR\fR
2192 .ad
2193 .RS 8n
2194 list-or-eof
2195 .RE

2197 .sp
2198 .ne 2
2199 .na
2200 \fB\fB^G\fR\fR
2201 .ad
2202 .RS 8n
2203 list-glob
2204 .RE

2206 .sp
2207 .ne 2
2208 .na
2209 \fB\fB^H\fR\fR
2210 .ad
2211 .RS 8n
2212 backward-delete-char
2213 .RE

2215 .sp
2216 .ne 2
2217 .na
2218 \fB\fB^I\fR\fR
2219 .ad
2220 .RS 8n
2221 complete-word
2222 .RE

2224 .sp
2225 .ne 2
2226 .na
2227 \fB\fB^er\fR\fR
2228 .ad
2229 .RS 8n
2230 newline
2231 .RE

2233 .sp
2234 .ne 2
2235 .na

```



```

2236 \fB\fB\en\fR\fR
2237 .ad
2238 .RS 8n
2239 newline
2240 .RE

2242 .sp
2243 .ne 2
2244 .na
2245 \fB\fB^L\fR\fR
2246 .ad
2247 .RS 8n
2248 clear-screen
2249 .RE

2251 .sp
2252 .ne 2
2253 .na
2254 \fB\fB^N\fR\fR
2255 .ad
2256 .RS 8n
2257 down-history
2258 .RE

2260 .sp
2261 .ne 2
2262 .na
2263 \fB\fB^P\fR\fR
2264 .ad
2265 .RS 8n
2266 up-history
2267 .RE

2269 .sp
2270 .ne 2
2271 .na
2272 \fB\fB^R\fR\fR
2273 .ad
2274 .RS 8n
2275 redisplay
2276 .RE

2278 .sp
2279 .ne 2
2280 .na
2281 \fB\fB^U\fR\fR
2282 .ad
2283 .RS 8n
2284 backward-kill-line
2285 .RE

2287 .sp
2288 .ne 2
2289 .na
2290 \fB\fB^W\fR\fR
2291 .ad
2292 .RS 8n
2293 backward-delete-word
2294 .RE

2296 .sp
2297 .ne 2
2298 .na
2299 \fB\fB^X*\fR\fR
2300 .ad
2301 .RS 8n

```

```

2302 expand-filename
2303 .RE

2305 .sp
2306 .ne 2
2307 .na
2308 \fB\fB^X^F\fR\fR
2309 .ad
2310 .RS 8n
2311 read-from-file
2312 .RE

2314 .sp
2315 .ne 2
2316 .na
2317 \fB\fB^X^R\fR\fR
2318 .ad
2319 .RS 8n
2320 read-init-files
2321 .RE

2323 .sp
2324 .ne 2
2325 .na
2326 \fB\fB^?\fR\fR
2327 .ad
2328 .RS 8n
2329 backward-delete-char
2330 .RE

2332 .sp
2333 .LP
2334 The following are the key bindings that are defined in \fBvi\fR command mode,
2335 this being specified by them all starting with a META character. As mentioned
2336 above, once in command mode the initial meta character is optional. For
2337 example, you might enter command mode by typing ESCAPE, and then press 'H'
2338 twice to move the cursor two positions to the left. Both 'H' characters get
2339 quietly converted to \fB^h\fR before being compared to the key binding table,
2340 the first one because ESCAPE followed by a character is always converted to the
2341 equivalent META character, and the second because command mode was already
2342 active.
2343 .sp
2344 .ne 2
2345 .na
2346 \fB^<space>\fR
2347 .ad
2348 .RS 21n
2349 cursor-right (META-space)
2350 .RE

2352 .sp
2353 .ne 2
2354 .na
2355 \fB^fBM-$\fR\fR
2356 .ad
2357 .RS 21n
2358 end-of-line
2359 .RE

2361 .sp
2362 .ne 2
2363 .na
2364 \fB^fBM-*\fR\fR
2365 .ad
2366 .RS 21n
2367 expand-filename

```

```

2368 .RE

2370 .sp
2371 .ne 2
2372 .na
2373 \fB\fBM+\fR\fR
2374 .ad
2375 .RS 21n
2376 down-history
2377 .RE

2379 .sp
2380 .ne 2
2381 .na
2382 \fB\fBM--\fR\fR
2383 .ad
2384 .RS 21n
2385 up-history
2386 .RE

2388 .sp
2389 .ne 2
2390 .na
2391 \fB\fBM-<\fR\fR
2392 .ad
2393 .RS 21n
2394 beginning-of-history
2395 .RE

2397 .sp
2398 .ne 2
2399 .na
2400 \fB\fBM->\fR\fR
2401 .ad
2402 .RS 21n
2403 end-of-history
2404 .RE

2406 .sp
2407 .ne 2
2408 .na
2409 \fB\fBM-^\fR\fR
2410 .ad
2411 .RS 21n
2412 beginning-of-line
2413 .RE

2415 .sp
2416 .ne 2
2417 .na
2418 \fB\fBM-\fR\fR
2419 .ad
2420 .RS 21n
2421 repeat-find-char
2422 .RE

2424 .sp
2425 .ne 2
2426 .na
2427 \fB\fBM-, \fR\fR
2428 .ad
2429 .RS 21n
2430 invert-refind-char
2431 .RE

2433 .sp

```

```

2434 .ne 2
2435 .na
2436 \fB\fBM-|\fR\fR
2437 .ad
2438 .RS 21n
2439 goto-column
2440 .RE

2442 .sp
2443 .ne 2
2444 .na
2445 \fB\fBM-~\fR\fR
2446 .ad
2447 .RS 21n
2448 change-case
2449 .RE

2451 .sp
2452 .ne 2
2453 .na
2454 \fB\fBM-.\fR\fR
2455 .ad
2456 .RS 21n
2457 vi-repeat-change
2458 .RE

2460 .sp
2461 .ne 2
2462 .na
2463 \fB\fBM-%\fR\fR
2464 .ad
2465 .RS 21n
2466 find-parenthesis
2467 .RE

2469 .sp
2470 .ne 2
2471 .na
2472 \fB\fBM-a\fR\fR
2473 .ad
2474 .RS 21n
2475 vi-append
2476 .RE

2478 .sp
2479 .ne 2
2480 .na
2481 \fB\fBM-A\fR\fR
2482 .ad
2483 .RS 21n
2484 vi-append-at-eol
2485 .RE

2487 .sp
2488 .ne 2
2489 .na
2490 \fB\fBM-b\fR\fR
2491 .ad
2492 .RS 21n
2493 backward-word
2494 .RE

2496 .sp
2497 .ne 2
2498 .na
2499 \fB\fBM-B\fR\fR

```

```

2500 .ad
2501 .RS 21n
2502 backward-word
2503 .RE

2505 .sp
2506 .ne 2
2507 .na
2508 \fB\fBM-C\fR\fR
2509 .ad
2510 .RS 21n
2511 vi-change-rest-of-line
2512 .RE

2514 .sp
2515 .ne 2
2516 .na
2517 \fB\fBM-cb\fR\fR
2518 .ad
2519 .RS 21n
2520 vi-backward-change-word
2521 .RE

2523 .sp
2524 .ne 2
2525 .na
2526 \fB\fBM-cB\fR\fR
2527 .ad
2528 .RS 21n
2529 vi-backward-change-word
2530 .RE

2532 .sp
2533 .ne 2
2534 .na
2535 \fB\fBM-cc\fR\fR
2536 .ad
2537 .RS 21n
2538 vi-change-line
2539 .RE

2541 .sp
2542 .ne 2
2543 .na
2544 \fB\fBM-ce\fR\fR
2545 .ad
2546 .RS 21n
2547 vi-forward-change-word
2548 .RE

2550 .sp
2551 .ne 2
2552 .na
2553 \fB\fBM-cE\fR\fR
2554 .ad
2555 .RS 21n
2556 vi-forward-change-word
2557 .RE

2559 .sp
2560 .ne 2
2561 .na
2562 \fB\fBM-cw\fR\fR
2563 .ad
2564 .RS 21n
2565 vi-forward-change-word

```

```

2566 .RE

2568 .sp
2569 .ne 2
2570 .na
2571 \fB\fBM-cW\fR\fR
2572 .ad
2573 .RS 21n
2574 vi-forward-change-word
2575 .RE

2577 .sp
2578 .ne 2
2579 .na
2580 \fB\fBM-cF\fR\fR
2581 .ad
2582 .RS 21n
2583 vi-backward-change-find
2584 .RE

2586 .sp
2587 .ne 2
2588 .na
2589 \fB\fBM-cf\fR\fR
2590 .ad
2591 .RS 21n
2592 vi-forward-change-find
2593 .RE

2595 .sp
2596 .ne 2
2597 .na
2598 \fB\fBM-cT\fR\fR
2599 .ad
2600 .RS 21n
2601 vi-backward-change-to
2602 .RE

2604 .sp
2605 .ne 2
2606 .na
2607 \fB\fBM-ct\fR\fR
2608 .ad
2609 .RS 21n
2610 vi-forward-change-to
2611 .RE

2613 .sp
2614 .ne 2
2615 .na
2616 \fB\fBM-ci\fR\fR
2617 .ad
2618 .RS 21n
2619 vi-change-refind
2620 .RE

2622 .sp
2623 .ne 2
2624 .na
2625 \fB\fBM-c,\fR\fR
2626 .ad
2627 .RS 21n
2628 vi-change-invert-refind
2629 .RE

2631 .sp

```

```

2632 .ne 2
2633 .na
2634 \fB\fBM-ch\fR\fR
2635 .ad
2636 .RS 21n
2637 vi-backward-change-char
2638 .RE

2640 .sp
2641 .ne 2
2642 .na
2643 \fB\fBM-c^H\fR\fR
2644 .ad
2645 .RS 21n
2646 vi-backward-change-char
2647 .RE

2649 .sp
2650 .ne 2
2651 .na
2652 \fB\fBM-c^?\fR\fR
2653 .ad
2654 .RS 21n
2655 vi-backward-change-char
2656 .RE

2658 .sp
2659 .ne 2
2660 .na
2661 \fB\fBM-cl\fR\fR
2662 .ad
2663 .RS 21n
2664 vi-forward-change-char
2665 .RE

2667 .sp
2668 .ne 2
2669 .na
2670 \fB\fBM-c<space>\fR
2671 .ad
2672 .RS 21n
2673 vi-forward-change-char (META-c-space)
2674 .RE

2676 .sp
2677 .ne 2
2678 .na
2679 \fB\fBM-c^\fR\fR
2680 .ad
2681 .RS 21n
2682 vi-change-to-bol
2683 .RE

2685 .sp
2686 .ne 2
2687 .na
2688 \fB\fBM-c0\fR\fR
2689 .ad
2690 .RS 21n
2691 vi-change-to-bol
2692 .RE

2694 .sp
2695 .ne 2
2696 .na
2697 \fB\fBM-c$\fR\fR

```

```

2698 .ad
2699 .RS 21n
2700 vi-change-rest-of-line
2701 .RE

2703 .sp
2704 .ne 2
2705 .na
2706 \fB\fBM-c|\fR\fR
2707 .ad
2708 .RS 21n
2709 vi-change-to-column
2710 .RE

2712 .sp
2713 .ne 2
2714 .na
2715 \fB\fBM-c%\fR\fR
2716 .ad
2717 .RS 21n
2718 vi-change-to-parenthesis
2719 .RE

2721 .sp
2722 .ne 2
2723 .na
2724 \fB\fBM-dh\fR\fR
2725 .ad
2726 .RS 21n
2727 backward-delete-char
2728 .RE

2730 .sp
2731 .ne 2
2732 .na
2733 \fB\fBM-d^H\fR\fR
2734 .ad
2735 .RS 21n
2736 backward-delete-char
2737 .RE

2739 .sp
2740 .ne 2
2741 .na
2742 \fB\fBM-d^?\fR\fR
2743 .ad
2744 .RS 21n
2745 backward-delete-char
2746 .RE

2748 .sp
2749 .ne 2
2750 .na
2751 \fB\fBM-dl\fR\fR
2752 .ad
2753 .RS 21n
2754 forward-delete-char
2755 .RE

2757 .sp
2758 .ne 2
2759 .na
2760 \fB\fBM-d<space>\fR
2761 .ad
2762 .RS 21n
2763 forward-delete-char (META-d-space)

```

```

2764 .RE
2766 .sp
2767 .ne 2
2768 .na
2769 \fB\fBM-dd\fR\fR
2770 .ad
2771 .RS 2ln
2772 delete-line
2773 .RE
2775 .sp
2776 .ne 2
2777 .na
2778 \fB\fBM-db\fR\fR
2779 .ad
2780 .RS 2ln
2781 backward-delete-word
2782 .RE
2784 .sp
2785 .ne 2
2786 .na
2787 \fB\fBM-dB\fR\fR
2788 .ad
2789 .RS 2ln
2790 backward-delete-word
2791 .RE
2793 .sp
2794 .ne 2
2795 .na
2796 \fB\fBM-de\fR\fR
2797 .ad
2798 .RS 2ln
2799 forward-delete-word
2800 .RE
2802 .sp
2803 .ne 2
2804 .na
2805 \fB\fBM-dE\fR\fR
2806 .ad
2807 .RS 2ln
2808 forward-delete-word
2809 .RE
2811 .sp
2812 .ne 2
2813 .na
2814 \fB\fBM-dw\fR\fR
2815 .ad
2816 .RS 2ln
2817 forward-delete-word
2818 .RE
2820 .sp
2821 .ne 2
2822 .na
2823 \fB\fBM-dW\fR\fR
2824 .ad
2825 .RS 2ln
2826 forward-delete-word
2827 .RE
2829 .sp

```

```

2830 .ne 2
2831 .na
2832 \fB\fBM-dF\fR\fR
2833 .ad
2834 .RS 2ln
2835 backward-delete-find
2836 .RE
2838 .sp
2839 .ne 2
2840 .na
2841 \fB\fBM-df\fR\fR
2842 .ad
2843 .RS 2ln
2844 forward-delete-find
2845 .RE
2847 .sp
2848 .ne 2
2849 .na
2850 \fB\fBM-dT\fR\fR
2851 .ad
2852 .RS 2ln
2853 backward-delete-to
2854 .RE
2856 .sp
2857 .ne 2
2858 .na
2859 \fB\fBM-dt\fR\fR
2860 .ad
2861 .RS 2ln
2862 forward-delete-to
2863 .RE
2865 .sp
2866 .ne 2
2867 .na
2868 \fB\fBM-di\fR\fR
2869 .ad
2870 .RS 2ln
2871 delete-refind
2872 .RE
2874 .sp
2875 .ne 2
2876 .na
2877 \fB\fBM-d,\fR\fR
2878 .ad
2879 .RS 2ln
2880 delete-invert-refind
2881 .RE
2883 .sp
2884 .ne 2
2885 .na
2886 \fB\fBM-d^\fR\fR
2887 .ad
2888 .RS 2ln
2889 backward-kill-line
2890 .RE
2892 .sp
2893 .ne 2
2894 .na
2895 \fB\fBM-d0\fR\fR

```

```

2896 .ad
2897 .RS 21n
2898 backward-kill-line
2899 .RE

2901 .sp
2902 .ne 2
2903 .na
2904 \fB\fBM-d$\fR\fR
2905 .ad
2906 .RS 21n
2907 kill-line
2908 .RE

2910 .sp
2911 .ne 2
2912 .na
2913 \fB\fBM-D\fR\fR
2914 .ad
2915 .RS 21n
2916 kill-line
2917 .RE

2919 .sp
2920 .ne 2
2921 .na
2922 \fB\fBM-d|\fR\fR
2923 .ad
2924 .RS 21n
2925 delete-to-column
2926 .RE

2928 .sp
2929 .ne 2
2930 .na
2931 \fB\fBM-d%\fR\fR
2932 .ad
2933 .RS 21n
2934 delete-to-parenthesis
2935 .RE

2937 .sp
2938 .ne 2
2939 .na
2940 \fB\fBM-e\fR\fR
2941 .ad
2942 .RS 21n
2943 forward-word
2944 .RE

2946 .sp
2947 .ne 2
2948 .na
2949 \fB\fBM-E\fR\fR
2950 .ad
2951 .RS 21n
2952 forward-word
2953 .RE

2955 .sp
2956 .ne 2
2957 .na
2958 \fB\fBM-f\fR\fR
2959 .ad
2960 .RS 21n
2961 forward-find-char

```

```

2962 .RE

2964 .sp
2965 .ne 2
2966 .na
2967 \fB\fBM-F\fR\fR
2968 .ad
2969 .RS 21n
2970 backward-find-char
2971 .RE

2973 .sp
2974 .ne 2
2975 .na
2976 \fB\fBM--\fR\fR
2977 .ad
2978 .RS 21n
2979 up-history
2980 .RE

2982 .sp
2983 .ne 2
2984 .na
2985 \fB\fBM-h\fR\fR
2986 .ad
2987 .RS 21n
2988 cursor-left
2989 .RE

2991 .sp
2992 .ne 2
2993 .na
2994 \fB\fBM-H\fR\fR
2995 .ad
2996 .RS 21n
2997 beginning-of-history
2998 .RE

3000 .sp
3001 .ne 2
3002 .na
3003 \fB\fBM-i\fR\fR
3004 .ad
3005 .RS 21n
3006 vi-insert
3007 .RE

3009 .sp
3010 .ne 2
3011 .na
3012 \fB\fBM-I\fR\fR
3013 .ad
3014 .RS 21n
3015 vi-insert-at-bol
3016 .RE

3018 .sp
3019 .ne 2
3020 .na
3021 \fB\fBM-j\fR\fR
3022 .ad
3023 .RS 21n
3024 down-history
3025 .RE

3027 .sp

```

```

3028 .ne 2
3029 .na
3030 \fB\fBM-J\fR\fR
3031 .ad
3032 .RS 21n
3033 history-search-forward
3034 .RE

3036 .sp
3037 .ne 2
3038 .na
3039 \fB\fBM-k\fR\fR
3040 .ad
3041 .RS 21n
3042 up-history
3043 .RE

3045 .sp
3046 .ne 2
3047 .na
3048 \fB\fBM-K\fR\fR
3049 .ad
3050 .RS 21n
3051 history-search-backward
3052 .RE

3054 .sp
3055 .ne 2
3056 .na
3057 \fB\fBM-l\fR\fR
3058 .ad
3059 .RS 21n
3060 cursor-right
3061 .RE

3063 .sp
3064 .ne 2
3065 .na
3066 \fB\fBM-L\fR\fR
3067 .ad
3068 .RS 21n
3069 end-of-history
3070 .RE

3072 .sp
3073 .ne 2
3074 .na
3075 \fB\fBM-n\fR\fR
3076 .ad
3077 .RS 21n
3078 history-re-search-forward
3079 .RE

3081 .sp
3082 .ne 2
3083 .na
3084 \fB\fBM-N\fR\fR
3085 .ad
3086 .RS 21n
3087 history-re-search-backward
3088 .RE

3090 .sp
3091 .ne 2
3092 .na
3093 \fB\fBM-p\fR\fR

```

```

3094 .ad
3095 .RS 21n
3096 append-yank
3097 .RE

3099 .sp
3100 .ne 2
3101 .na
3102 \fB\fBM-P\fR\fR
3103 .ad
3104 .RS 21n
3105 yank
3106 .RE

3108 .sp
3109 .ne 2
3110 .na
3111 \fB\fBM-r\fR\fR
3112 .ad
3113 .RS 21n
3114 vi-replace-char
3115 .RE

3117 .sp
3118 .ne 2
3119 .na
3120 \fB\fBM-R\fR\fR
3121 .ad
3122 .RS 21n
3123 vi-overwrite
3124 .RE

3126 .sp
3127 .ne 2
3128 .na
3129 \fB\fBM-s\fR\fR
3130 .ad
3131 .RS 21n
3132 vi-forward-change-char
3133 .RE

3135 .sp
3136 .ne 2
3137 .na
3138 \fB\fBM-S\fR\fR
3139 .ad
3140 .RS 21n
3141 vi-change-line
3142 .RE

3144 .sp
3145 .ne 2
3146 .na
3147 \fB\fBM-t\fR\fR
3148 .ad
3149 .RS 21n
3150 forward-to-char
3151 .RE

3153 .sp
3154 .ne 2
3155 .na
3156 \fB\fBM-T\fR\fR
3157 .ad
3158 .RS 21n
3159 backward-to-char

```

```

3160 .RE
3162 .sp
3163 .ne 2
3164 .na
3165 \fB\fBM-u\fR\fR
3166 .ad
3167 .RS 21n
3168 vi-undo
3169 .RE
3171 .sp
3172 .ne 2
3173 .na
3174 \fB\fBM-w\fR\fR
3175 .ad
3176 .RS 21n
3177 forward-to-word
3178 .RE
3180 .sp
3181 .ne 2
3182 .na
3183 \fB\fBM-W\fR\fR
3184 .ad
3185 .RS 21n
3186 forward-to-word
3187 .RE
3189 .sp
3190 .ne 2
3191 .na
3192 \fB\fBM-x\fR\fR
3193 .ad
3194 .RS 21n
3195 forward-delete-char
3196 .RE
3198 .sp
3199 .ne 2
3200 .na
3201 \fB\fBM-X\fR\fR
3202 .ad
3203 .RS 21n
3204 backward-delete-char
3205 .RE
3207 .sp
3208 .ne 2
3209 .na
3210 \fB\fBM-yh\fR\fR
3211 .ad
3212 .RS 21n
3213 backward-copy-char
3214 .RE
3216 .sp
3217 .ne 2
3218 .na
3219 \fB\fBM-y^H\fR\fR
3220 .ad
3221 .RS 21n
3222 backward-copy-char
3223 .RE
3225 .sp

```

```

3226 .ne 2
3227 .na
3228 \fB\fBM-y^?\fR\fR
3229 .ad
3230 .RS 21n
3231 backward-copy-char
3232 .RE
3234 .sp
3235 .ne 2
3236 .na
3237 \fB\fBM-yl\fR\fR
3238 .ad
3239 .RS 21n
3240 forward-copy-char
3241 .RE
3243 .sp
3244 .ne 2
3245 .na
3246 \fB\fBM-y<space>\fR
3247 .ad
3248 .RS 21n
3249 forward-copy-char (META-y-space)
3250 .RE
3252 .sp
3253 .ne 2
3254 .na
3255 \fB\fBM-ye\fR\fR
3256 .ad
3257 .RS 21n
3258 forward-copy-word
3259 .RE
3261 .sp
3262 .ne 2
3263 .na
3264 \fB\fBM-yE\fR\fR
3265 .ad
3266 .RS 21n
3267 forward-copy-word
3268 .RE
3270 .sp
3271 .ne 2
3272 .na
3273 \fB\fBM-yw\fR\fR
3274 .ad
3275 .RS 21n
3276 forward-copy-word
3277 .RE
3279 .sp
3280 .ne 2
3281 .na
3282 \fB\fBM-yW\fR\fR
3283 .ad
3284 .RS 21n
3285 forward-copy-word
3286 .RE
3288 .sp
3289 .ne 2
3290 .na
3291 \fB\fBM-yb\fR\fR

```


3292 .ad
 3293 .RS 21n
 3294 backward-copy-word
 3295 .RE

3297 .sp
 3298 .ne 2
 3299 .na
 3300 \fB\fBM-yB\fR\fR
 3301 .ad
 3302 .RS 21n
 3303 backward-copy-word
 3304 .RE

3306 .sp
 3307 .ne 2
 3308 .na
 3309 \fB\fBM-yf\fR\fR
 3310 .ad
 3311 .RS 21n
 3312 forward-copy-find
 3313 .RE

3315 .sp
 3316 .ne 2
 3317 .na
 3318 \fB\fBM-yF\fR\fR
 3319 .ad
 3320 .RS 21n
 3321 backward-copy-find
 3322 .RE

3324 .sp
 3325 .ne 2
 3326 .na
 3327 \fB\fBM-yt\fR\fR
 3328 .ad
 3329 .RS 21n
 3330 forward-copy-to
 3331 .RE

3333 .sp
 3334 .ne 2
 3335 .na
 3336 \fB\fBM-yT\fR\fR
 3337 .ad
 3338 .RS 21n
 3339 backward-copy-to
 3340 .RE

3342 .sp
 3343 .ne 2
 3344 .na
 3345 \fB\fBM-y;\fR\fR
 3346 .ad
 3347 .RS 21n
 3348 copy-refind
 3349 .RE

3351 .sp
 3352 .ne 2
 3353 .na
 3354 \fB\fBM-y,\fR\fR
 3355 .ad
 3356 .RS 21n
 3357 copy-invert-refind

3358 .RE

3360 .sp
 3361 .ne 2
 3362 .na
 3363 \fB\fBM-y^\fR\fR
 3364 .ad
 3365 .RS 21n
 3366 copy-to-bol
 3367 .RE

3369 .sp
 3370 .ne 2
 3371 .na
 3372 \fB\fBM-y0\fR\fR
 3373 .ad
 3374 .RS 21n
 3375 copy-to-bol
 3376 .RE

3378 .sp
 3379 .ne 2
 3380 .na
 3381 \fB\fBM-y\$\fR\fR
 3382 .ad
 3383 .RS 21n
 3384 copy-rest-of-line
 3385 .RE

3387 .sp
 3388 .ne 2
 3389 .na
 3390 \fB\fBM-yy\fR\fR
 3391 .ad
 3392 .RS 21n
 3393 copy-line
 3394 .RE

3396 .sp
 3397 .ne 2
 3398 .na
 3399 \fB\fBM-Y\fR\fR
 3400 .ad
 3401 .RS 21n
 3402 copy-line
 3403 .RE

3405 .sp
 3406 .ne 2
 3407 .na
 3408 \fB\fBM-y|\fR\fR
 3409 .ad
 3410 .RS 21n
 3411 copy-to-column
 3412 .RE

3414 .sp
 3415 .ne 2
 3416 .na
 3417 \fB\fBM-y%\fR\fR
 3418 .ad
 3419 .RS 21n
 3420 copy-to-parenthesis
 3421 .RE

3423 .sp

```

3424 .ne 2
3425 .na
3426 \fB\fBM-^E\fR\fR
3427 .ad
3428 .RS 21n
3429 emacs-mode
3430 .RE

3432 .sp
3433 .ne 2
3434 .na
3435 \fB\fBM-^H\fR\fR
3436 .ad
3437 .RS 21n
3438 cursor-left
3439 .RE

3441 .sp
3442 .ne 2
3443 .na
3444 \fB\fBM-^?\fR\fR
3445 .ad
3446 .RS 21n
3447 cursor-left
3448 .RE

3450 .sp
3451 .ne 2
3452 .na
3453 \fB\fBM-^L\fR\fR
3454 .ad
3455 .RS 21n
3456 clear-screen
3457 .RE

3459 .sp
3460 .ne 2
3461 .na
3462 \fB\fBM-^N\fR\fR
3463 .ad
3464 .RS 21n
3465 down-history
3466 .RE

3468 .sp
3469 .ne 2
3470 .na
3471 \fB\fBM-^P\fR\fR
3472 .ad
3473 .RS 21n
3474 up-history
3475 .RE

3477 .sp
3478 .ne 2
3479 .na
3480 \fB\fBM-^R\fR\fR
3481 .ad
3482 .RS 21n
3483 redisplay
3484 .RE

3486 .sp
3487 .ne 2
3488 .na
3489 \fB\fBM-^D\fR\fR

```

```

3490 .ad
3491 .RS 21n
3492 list-or-eof
3493 .RE

3495 .sp
3496 .ne 2
3497 .na
3498 \fB\fBM-^I\fR\fR
3499 .ad
3500 .RS 21n
3501 complete-word
3502 .RE

3504 .sp
3505 .ne 2
3506 .na
3507 \fB\fBM-^er\fR
3508 .ad
3509 .RS 21n
3510 newline
3511 .RE

3513 .sp
3514 .ne 2
3515 .na
3516 \fB\fBM-^en\fR\fR
3517 .ad
3518 .RS 21n
3519 newline
3520 .RE

3522 .sp
3523 .ne 2
3524 .na
3525 \fB\fBM-^X^R\fR\fR
3526 .ad
3527 .RS 21n
3528 read-init-files
3529 .RE

3531 .sp
3532 .ne 2
3533 .na
3534 \fB\fBM-^Xh\fR\fR
3535 .ad
3536 .RS 21n
3537 list-history
3538 .RE

3540 .sp
3541 .ne 2
3542 .na
3543 \fB\fBM-0, M-1, ... M-9\fR\fR
3544 .ad
3545 .RS 21n
3546 digit-argument (see below)
3547 .RE

3549 .sp
3550 .LP
3551 Note that \fB^I\fR is what the TAB key generates.
3552 .SS "Entering Repeat Counts"
3553 .LP
3554 Many of the key binding functions described previously, take an optional count,
3555 typed in before the target key sequence. This is interpreted as a repeat count

```

3556 by most bindings. A notable exception is the goto-column binding, which
 3557 interprets the count as a column number.

3558 .sp
 3559 .LP

3560 By default you can specify this count argument by pressing the META key while
 3561 typing in the numeric count. This relies on the digit-argument action being
 3562 bound to 'META-0', 'META-1' etc. Once any one of these bindings has been
 3563 activated, you can optionally take your finger off the META key to type in the
 3564 rest of the number, since every numeric digit thereafter is treated as part of
 3565 the number, unless it is preceded by the literal-next binding. As soon as a
 3566 non-digit, or literal digit key is pressed the repeat count is terminated and
 3567 either causes the just typed character to be added to the line that many times,
 3568 or causes the next key binding function to be given that argument.

3569 .sp
 3570 .LP

3571 For example, in \fBemacs\fR mode, typing:
 3572 .sp
 3573 .in +2
 3574 .nf
 3575 M-12a
 3576 .fi
 3577 .in -2

3579 .sp
 3580 .LP

3581 causes the letter 'a' to be added to the line 12 times, whereas

3582 .sp
 3583 .in +2
 3584 .nf
 3585 M-4M-c
 3586 .fi
 3587 .in -2

3589 .sp
 3590 .LP

3591 Capitalizes the next 4 words.

3592 .sp
 3593 .LP

3594 In \fBvi\fR command mode the meta modifier is automatically added to all
 3595 characters typed in, so to enter a count in \fBvi\fR command-mode, just
 3596 involves typing in the number, just as it does in the \fBvi\fR editor itself.
 3597 So for example, in vi command mode, typing:

3598 .sp
 3599 .in +2
 3600 .nf
 3601 4w2x
 3602 .fi
 3603 .in -2

3605 .sp
 3606 .LP

3607 moves the cursor four words to the right, then deletes two characters.

3608 .sp
 3609 .LP

3610 You can also bind digit-argument to other key sequences. If these end in a
 3611 numeric digit, that digit gets appended to the current repeat count. If it
 3612 doesn't end in a numeric digit, a new repeat count is started with a value of
 3613 zero, and can be completed by typing in the number, after letting go of the key
 3614 which triggered the digit-argument action.

3615 .SH FILES
 3616 .ne 2
 3617 .na
 3618 \fB\fB/usr/lib/libtecla.so\fR
 3619 .ad
 3620 .RS 27n
 3621 The tecla library

3622 .RE

3624 .sp
 3625 .ne 2
 3626 .na
 3627 \fB\fB/usr/include/libtecla.h\fR
 3628 .ad
 3629 .RS 27n
 3630 The tecla header file
 3631 .RE

3633 .sp
 3634 .ne 2
 3635 .na
 3636 \fB\fB~/teclarc\fR
 3637 .ad
 3638 .RS 27n
 3639 The personal tecla customization file
 3640 .RE

3642 .SH ATTRIBUTES
 3643 .LP
 3644 See \fBAttributes\fR(5) for descriptions of the following attributes:
 3645 .sp

3647 .sp
 3648 .TS
 3649 box;
 3650 c | c
 3651 l | l .
 3652 ATTRIBUTE TYPE ATTRIBUTE VALUE
 3653 -
 3654 Interface Stability Evolving
 3655 .TE

3657 .SH SEE ALSO
 3658 .LP
 3659 \fBvi\fR(1), \fBcpl_complete_word\fR(3TECLA), \fBef_expand_file\fR(3TECLA),
 3660 \fBgl_get_line\fR(3TECLA), \fBgl_io_mode\fR(3TECLA), \fBlibtecla\fR(3LIB),
 3661 \fBpca_lookup_file\fR(3TECLA), \fBAttributes\fR(5)

```

*****
7843 Tue Dec 11 09:48:10 2018
new/usr/src/tools/onbld/Checks/SpellCheck.py
10057 Man page misspellings ouput particular overriden
Reviewed by: Gerg Mihly Doma <domag02@gmail.com>
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 #
23 # Copyright 2016 Joyent, Inc.
24 #
25 #
26 import re, sys

28 spellMsg = '%s: Line %d contains "%s", a common misspelling of "%s"\n'
29 altMsg = '%s: Line %d contains "%s"; please use "%s" instead for consistency wit

31 misspellings = {
32     'absense': 'absence',
33     'accessable': 'accessible',
34     'accomodate': 'accommodate',
35     'accomodation': 'accommodation',
36     'accross': 'across',
37     'acheive': 'achieve',
38     'additional': 'additional',
39     'adres': 'address',
40     'administrative': 'administrative',
41     'adminstered': 'administered',
42     'adminstrate': 'administrate',
43     'adminstration': 'administration',
44     'adminstrative': 'administrative',
45     'adminstrator': 'administrator',
46     'admissability': 'admissibility',
47     'adress': 'address',
48     'adressable': 'addressable',
49     'adressed': 'addressed',
50     'addressing': 'addressing, dressing',
51     'aginst': 'against',
52     'agression': 'aggression',
53     'agressive': 'aggressive',
54     'alot': 'a lot, allot',
55     'and and': 'and',
56     'apparrantly': 'apparently',
57     'appearance': 'appearance',
58     'arguement': 'argument',
59     'assasination': 'assassination',
60     'auxilliary': 'auxiliary',

```

```

61     'basicly': 'basically',
62     'begining': 'beginning',
63     'belive': 'believe',
64     'beteen': 'between',
65     'betwen': 'between',
66     'beween': 'between',
67     'bewteen': 'between',
68     'bizzare': 'bizarre',
69     'buisness': 'business',
70     'calender': 'calendar',
71     'cemetary': 'cemetery',
72     'chauffer': 'chauffeur',
73     'colleague': 'colleague',
74     'comming': 'coming',
75     'committed': 'committed',
76     'commitee': 'committee',
77     'commiting': 'committing',
78     'comparision': 'comparison',
79     'comparisions': 'comparisons',
80     'compatability': 'compatibility',
81     'compatable': 'compatible',
82     'compatablity': 'compatibility',
83     'compatiable': 'compatible',
84     'compatibility': 'compatibility',
85     'completly': 'completely',
86     'conciuous': 'conscious',
87     'conditdion': 'condition',
88     'conected': 'connected',
89     'conjunction': 'conjunction',
90     'continous': 'continuous',
91     'curiosity': 'curiosity',
92     'daemon': 'daemon',
93     'definatelly': 'definitely',
94     'desireable': 'desirable',
95     'diffrent': 'different',
96     'dilemma': 'dilemma',
97     'dissapear': 'disappear',
98     'dissappoint': 'disappoint',
99     'ecstasy': 'ecstasy',
100     'embarass': 'embarrass',
101     'enviroment': 'environment',
102     'exept': 'except',
103     'existance': 'existence',
104     'familiar': 'familiar',
105     'finaly': 'finally',
106     'folowing': 'following',
107     'foriegn': 'foreign',
108     'forseeable': 'foreseeable',
109     'fourty': 'forty',
110     'foward': 'forward',
111     'freind': 'friend',
112     'futher': 'further',
113     'gaurd': 'guard',
114     'glamorous': 'glamorous',
115     'government': 'government',
116     'happend': 'happened',
117     'harrassment': 'harassment',
118     'hierachical': 'hierarchical',
119     'hierachies': 'hierarchies',
120     'hierachy': 'hierarchy',
121     'hierarcical': 'hierarchical',
122     'hierarcy': 'hierarchy',
123     'honourary': 'honorary',
124     'humourous': 'humorous',
125     'idiosyncracy': 'idiosyncrasy',
126     'immediatly': 'immediately',

```

```

127     'inaccessible': 'inaccessible',
128     'inbetween': 'between',
129     'incidently': 'incidentally',
130     'independant': 'independent',
131     'infomation': 'information',
132     'interupt': 'interrupt',
133     'intial': 'initial',
134     'intially': 'initially',
135     'irresistable': 'irresistible',
136     'jist': 'gist',
137     'knowlege': 'knowledge',
138     'lenght': 'length',
139     'liase': 'liaise',
140     'liason': 'liaison',
141     'library': 'library',
142     'maching': 'machine, marching, matching',
143     'millenia': 'millennia',
144     'millenium': 'millennium',
145     'neccessary': 'necessary',
146     'negotation': 'negotiation',
147     'nontheless': 'nonetheless',
148     'noticable': 'noticeable',
149     'occassion': 'occasion',
150     'occasional': 'occasional',
151     'occasionally': 'occasionally',
152     'occurance': 'occurrence',
153     'occured': 'occurred',
154     'occurence': 'occurrence',
155     'occurring': 'occurring',
156     'ommission': 'omission',
157     'orginal': 'original',
158     'originally': 'originally',
159     'ouput': 'output',
160     'overriden': 'overridden',
161     'particular': 'particular',
162     'pavillion': 'pavilion',
163     'peice': 'piece',
164     'persistant': 'persistent',
165     'politican': 'politician',
166     'posession': 'possession',
167     'possibility': 'possibility',
168     'preceed': 'precede',
169     'preceeded': 'preceded',
170     'preceeding': 'preceding',
171     'preceeds': 'precedes',
172     'prefered': 'preferred',
173     'prefering': 'preferring',
174     'presense': 'presence',
175     'proces': 'process',
176     'propoganda': 'propaganda',
177     'psuedo': 'pseudo',
178     'publically': 'publicly',
179     'realy': 'really',
180     'reciept': 'receipt',
181     'recieve': 'receive',
182     'recieved': 'received',
183     'reciever': 'receiver',
184     'recievers': 'receivers',
185     'recieves': 'receives',
186     'recieving': 'receiving',
187     'recomend': 'recommend',
188     'recomended': 'recommended',
189     'recommending': 'recommending',
190     'recomends': 'recommends',
191     'recurse': 'recur',
192     'recurses': 'recurs',

```

```

193     'recurring': 'recurring',
194     'refered': 'referred',
195     'refering': 'referring',
196     'religious': 'religious',
197     'rember': 'remember',
198     'remeber': 'remember',
199     'repetition': 'repetition',
200     'reponsible': 'responsible',
201     'resistence': 'resistance',
202     'retreive': 'retrieve',
203     'seige': 'siege',
204     'sence': 'since',
205     'seperate': 'separate',
206     'seperated': 'separated',
207     'seperately': 'separately',
208     'seperates': 'separates',
209     'similiar': 'similar',
210     'somwhere': 'somewhere',
211     'sould': 'could, should, sold, soul',
212     'sturcture': 'structure',
213     'succesful': 'successful',
214     'succesfully': 'successfully',
215     'successfull': 'successful',
216     'sucessful': 'successful',
217     'supercede': 'supersede',
218     'supress': 'suppress',
219     'supressed': 'suppressed',
220     'suprise': 'surprise',
221     'suprisingly': 'surprisingly',
222     'sytem': 'system',
223     'tendancy': 'tendency',
224     'the the': 'the',
225     'the these': 'these',
226     'therefor': 'therefore',
227     'threshold': 'threshold',
228     'tolerence': 'tolerance',
229     'tomorrow': 'tomorrow',
230     'tomorrow': 'tomorrow',
231     'tounge': 'tongue',
232     'tranformed': 'transformed',
233     'transfered': 'transferred',
234     'truely': 'truly',
235     'trustworthyness': 'trustworthiness',
236     'uncommitted': 'uncommitted',
237     'unforseen': 'unforeseen',
238     'unfortunatly': 'unfortunately',
239     'unsuccessfull': 'unsuccessful',
240     'untill': 'until',
241     'upto': 'up to',
242     'whereever': 'wherever',
243     'wich': 'which',
244     'wierd': 'weird',
245     'wtih': 'with',
246 }

```

unchanged portion omitted