



Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'error_at_line.3'

\$ man error_at_line.3

ERROR(3) Linux Programmer's Manual ERROR(3)

NAME

error, error_at_line, error_message_count, error_one_per_line, error_print_progname -
glibc error reporting functions

SYNOPSIS

```
#include <error.h>

void error(int status, int errnum, const char *format, ...);

void error_at_line(int status, int errnum, const char *filename,
                  unsigned int linenum, const char *format, ...);

extern unsigned int error_message_count;

extern int error_one_per_line;

extern void (*error_print_progname) (void);
```

DESCRIPTION

error() is a general error-reporting function. It flushes stdout, and then outputs to stderr the program name, a colon and a space, the message specified by the printf(3)-style format string format, and, if errnum is nonzero, a second colon and a space followed by the string given by strerror(errnum). Any arguments required for format should follow format in the argument list. The output is terminated by a newline character.

The program name printed by error() is the value of the global variable program_invocation_name(3). program_invocation_name initially has the same value as main()'s argv[0]. The value of this variable can be modified to change the output of error().

If status has a nonzero value, then error() calls exit(3) to terminate the program using the given value as the exit status.

The `error_at_line()` function is exactly the same as `error()`, except for the addition of the arguments `filename` and `linenum`. The output produced is as for `error()`, except that after the program name are written: a colon, the value of `filename`, a colon, and the value of `linenum`. The preprocessor values `__LINE__` and `__FILE__` may be useful when calling `error_at_line()`, but other values can also be used. For example, these arguments could refer to a location in an input file.

If the global variable `error_one_per_line` is set nonzero, a sequence of `error_at_line()` calls with the same value of `filename` and `linenum` will result in only one message (the first) being output.

The global variable `error_message_count` counts the number of messages that have been output by `error()` and `error_at_line()`.

If the global variable `error_print_progname` is assigned the address of a function (i.e., is not `NULL`), then that function is called instead of prefixing the message with the program name and colon. The function should print a suitable string to `stderr`.

ATTRIBUTES

For an explanation of the terms used in this section, see [attributes\(7\)](#).

????????????????????????????????????????????????????????????????????????????????????

?Interface	? Attribute	? Value	?
------------	-------------	---------	---

????????????????????????????????????????????????????????????????????????????????????

?error()	? Thread safety	? MT-Safe	locale	?
----------	-----------------	-----------	--------	---

????????????????????????????????????????????????????????????????????????????????????

?error_at_line()	? Thread safety	? MT-Unsafe	race: error_at_line/er?	?
------------------	-----------------	-------------	-------------------------	---

?	?	? error_one_per_line	locale	?
---	---	----------------------	--------	---

????????????????????????????????????????????????????????????????????????????????????

The internal `error_one_per_line` variable is accessed (without any form of synchronization, but since it's an `int` used once, it should be safe enough) and, if `error_one_per_line` is set nonzero, the internal static variables (not exposed to users) used to hold the last printed `filename` and `line number` are accessed and modified without synchronization; the update is not atomic and it occurs before disabling cancellation, so it can be interrupted only after one of the two variables is modified. After that, `error_at_line()` is very much like `error()`.

CONFORMING TO

These functions and variables are GNU extensions, and should not be used in programs in?

tended to be portable.

SEE ALSO

`err(3)`, `errno(3)`, `exit(3)`, `perror(3)`, `program_invocation_name(3)`, `strerror(3)`

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.

GNU

2017-09-15

ERROR(3)