



Linux Ubuntu 26.04 Manual Pages on command 'regerror.3'

\$ man regerror.3

regex(3) Library Functions Manual regex(3)

NAME

regcomp, regexec, regerror, regfree - POSIX regex functions

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <regex.h>
```

```
int regcomp(regex_t *restrict preg, const char *restrict regex,
```

```
    int cflags);
```

```
int regexec(const regex_t *restrict preg,
```

```
    const char *restrict string,
```

```
    size_t n, regmatch_t pmatch[_Nullable restrict n],
```

```
    int eflags);
```

```
size_t regerror(size_t errbuf_size;
```

```
    int errcode, const regex_t *_Nullable restrict preg,
```

```
    char errbuf[_Nullable restrict errbuf_size],
```

```
    size_t errbuf_size);
```

```
void regfree(regex_t *preg);
```

```
typedef struct {
```

```
    size_t re_nsub;
```

```
} regex_t;
```

```
typedef struct {
```

```
    regoff_t rm_so;
```

```

    regoff_t rm_eo;

} regmatch_t;

typedef /* ... */ regoff_t;

```

DESCRIPTION

Compilation

`regcomp()` is used to compile a regular expression into a form that is suitable for subsequent `regexexec()` searches.

On success, the pattern buffer at `*preg` is initialized. `regex` is a null-terminated string.

The locale must be the same when running `regexexec()`.

After `regcomp()` succeeds, `preg->re_nsub` holds the number of subexpressions in `regex`. Thus, a value of `preg->re_nsub + 1` passed as `n` to `regexexec()` is sufficient to capture all matches.

`cflags` is the bitwise OR of zero or more of the following:

REG_EXTENDED

Use POSIX Extended Regular Expression syntax when interpreting `regex`. If not set, POSIX Basic Regular Expression syntax is used.

REG_ICASE

Do not differentiate case. Subsequent `regexexec()` searches using this pattern buffer will be case insensitive.

REG_NOSUB

Report only overall success. `regexexec()` will use only `pmatch` for `REG_STARTEND`, ignoring `n`.

REG_NEWLINE

Match-any-character operators don't match a newline.

A nonmatching list (`[^...]`) not containing a newline does not match a newline.

Match-beginning-of-line operator (`^`) matches the empty string immediately after a newline, regardless of whether `eflags`, the execution flags of `regexexec()`, contains

`REG_NOTBOL`.

Match-end-of-line operator (`$`) matches the empty string immediately before a newline, regardless of whether `eflags` contains `REG_NOTEOL`.

Matching

`regexexec()` is used to match a null-terminated string against the compiled pattern buffer in `*preg`, which must have been initialised with `regcomp()`. `eflags` is the bitwise OR of zero or more of the following flags:

REG_NOTBOL

The match-beginning-of-line operator always fails to match (but see the compilation flag REG_NEWLINE above). This flag may be used when different portions of a string are passed to regexec() and the beginning of the string should not be interpreted as the beginning of the line.

REG_NOTEOL

The match-end-of-line operator always fails to match (but see the compilation flag REG_NEWLINE above).

REG_STARTEND

Match [string + pmatch[0].rm_so, string + pmatch[0].rm_eo) instead of [string, string + strlen(string)). This allows matching embedded NUL bytes and avoids a strlen(3) on known-length strings. If any matches are returned (REG_NOSUB wasn't passed to regcomp(), the match succeeded, and n > 0), they overwrite pmatch as usual, and the match offsets remain relative to string (not string + pmatch[0].rm_so). This flag is a BSD extension, not present in POSIX.

Match offsets

Unless REG_NOSUB was passed to regcomp(), it is possible to obtain the locations of matches within string: regexec() fills n elements of pmatch with results: pmatch[0] corresponds to the entire match, pmatch[1] to the first subexpression, etc. If there were more matches than n, they are discarded; if fewer, unused elements of pmatch are filled with -1s. Each returned valid (non--1) match corresponds to the range [string + rm_so, string + rm_eo).

regoff_t is a signed integer type capable of storing the largest value that can be stored in either an ptrdiff_t type or a ssize_t type.

Error reporting

regerror() is used to turn the error codes that can be returned by both regcomp() and regexec() into error message strings.

If preg isn't a null pointer, errcode must be the latest error returned from an operation on preg.

If errbuf_size isn't 0, up to errbuf_size bytes are copied to errbuf; the error string is always null-terminated, and truncated to fit.

Freeing

regfree() deinitializes the pattern buffer at *preg, freeing any associated memory; *preg

must have been initialized via `regcomp()`.

RETURN VALUE

`regcomp()` returns zero for a successful compilation or an error code for failure.

`regexexec()` returns zero for a successful match or `REG_NOMATCH` for failure.

`regerror()` returns the size of the buffer required to hold the string.

ERRORS

The following errors can be returned by `regcomp()`:

REG_BADBR

Invalid use of back reference operator.

REG_BADPAT

Invalid use of pattern operators such as group or list.

REG_BADRPT

Invalid use of repetition operators such as using `'**'` as the first character.

REG_EBRACE

Un-matched brace interval operators.

REG_EBRACK

Un-matched bracket list operators.

REG_ECOLLATE

Invalid collating element.

REG_ECTYPE

Unknown character class name.

REG_EEND

Nonspecific error. This is not defined by POSIX.

REG_EESCAPE

Trailing backslash.

REG_EPAREN

Un-matched parenthesis group operators.

REG_ERANGE

Invalid use of the range operator; for example, the ending point of the range occurs prior to the starting point.

REG_ESIZE

Compiled regular expression requires a pattern buffer larger than 64 kB. This is not defined by POSIX.

REG_SPACE

The regex routines ran out of memory.

REG_ESUBREG

Invalid back reference to a subexpression.

ATTRIBUTES

For an explanation of the terms used in this section, see `attributes(7)`.

[illegible]

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

[illegible]

? regcomp(), regexec() ? Thread safety ? MT-Safe locale ?

[illegible]

? regerror() ? Thread safety ? MT-Safe env ?

[illegible]

? regfree()	? Thread safety	? MT-Safe	?
-------------	-----------------	-----------	---

[illegible]

STANDARDS

POSIX.1-2008.

HISTORY

POSIX.1-2001.

Prior to POSIX.1-2008, `regoff_t` was required to be capable of storing the largest value that can be stored in either an `off_t` type or a `ssize_t` type.

CAVEATS

re_nsub is only required to be initialized if REG_NOSUB wasn't specified, but all known im?

plementations initialize it regardless.

Both `regex_t` and `regmatch_t` may (and do) have more members, in any order. Always reference them by name.

EXAMPLES

```
#include <stdcountof.h>
```

```
#include <stdint.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <regex.h>
```

```
static const char *const str =
```

```

"1) John Driverhacker;\n2) John Doe;\n3) John Foo;\n";

static const char *const re = "John.*o";

int main(void)
{
    static const char *s = str;

    regex_t    regex;
    regmatch_t pmatch[1];
    regoff_t    off, len;

    if (regcomp(&regex, re, REG_NEWLINE))
        exit(EXIT_FAILURE);

    printf("String = \"%s\"\n", str);
    printf("Matches:\n");

    for (unsigned int i = 0; ; i++) {
        if (regexexec(&regex, s, countof(pmatch), pmatch, 0))
            break;

        off = pmatch[0].rm_so + (s - str);
        len = pmatch[0].rm_eo - pmatch[0].rm_so;

        printf("#%u:\n", i);
        printf("offset = %jd; length = %jd\n", (intmax_t) off,
              (intmax_t) len);
        printf("substring = \"%.*s\"\n", len, s + pmatch[0].rm_so);
        s += pmatch[0].rm_eo;
    }

    exit(EXIT_SUCCESS);
}

```

SEE ALSO

grep(1), regex(7)

The glibc manual section, Regular Expressions

Linux man-pages 6.17

2026-02-10

regex(3)