NAME

readpassphrase - get a passphrase from the user

SYNOPSIS

#include <readpassphrase.h>

*char **
readpassphrase(*const char *prompt, char *buf, size_t bufsiz, int flags*);

DESCRIPTION

The **readpassphrase**() function displays a prompt to, and reads in a passphrase from, */dev/tty*. If this file is inaccessible and the RPP_REQUIRE_TTY flag is not set, **readpassphrase**() displays the prompt on the standard error output and reads from the standard input. In this case it is generally not possible to turn off echo.

Up to *bufsiz* - 1 characters (one is for the NUL) are read into the provided buffer *buf*. Any additional characters and the terminating newline (or return) character are discarded.

The **readpassphrase**() function takes the following optional *flags*:

RPP_ECHO_OFF	turn off echo (default behavior)
RPP_ECHO_ON	leave echo on
RPP_REQUIRE_TTY	fail if there is no tty
RPP_FORCELOWER	force input to lower case
RPP_FORCEUPPER	force input to upper case
RPP_SEVENBIT	strip the high bit from input
RPP_STDIN	force read of passphrase from stdin

The calling process should zero the passphrase as soon as possible to avoid leaving the cleartext passphrase visible in the process’s address space.

RETURN VALUES

Upon successful completion, **readpassphrase**() returns a pointer to the NUL-terminated passphrase. If an error is encountered, the terminal state is restored and a NULL pointer is returned.

FILES

/dev/tty

EXAMPLES

The following code fragment will read a passphrase from */dev/tty* into the buffer *passbuf*.

```
char passbuf[1024];

...

if (readpassphrase("Response: ", passbuf, sizeof(passbuf),
    RPP_REQUIRE_TTY) == NULL)
        errx(1, "unable to read passphrase");

if (compare(transform(passbuf), epass) != 0)
        errx(1, "bad passphrase");

...

memset(passbuf, 0, sizeof(passbuf));
```

ERRORS

[EINTR] The **readpassphrase**() function was interrupted by a signal.

[EINVAL] The *bufsiz* argument was zero.

[EIO] The process is a member of a background process attempting to read from its controlling terminal, the process is ignoring or blocking the SIGTTIN signal, or the process group is orphaned.

[EMFILE] The process has already reached its limit for open file descriptors.

[ENFILE] The system file table is full.

[ENOTTY] There is no controlling terminal and the RPP_REQUIRE_TTY flag was specified.

SIGNALS

The **readpassphrase**() function will catch the following signals:

SIGALRM	SIGHUP	SIGINT
SIGPIPE	SIGQUIT	SIGTERM
SIGTSTP	SIGTTIN	SIGTTOU

When one of the above signals is intercepted, terminal echo will be restored if it had previously been

turned off. If a signal handler was installed for the signal when **readpassphrase**() was called, that handler is then executed. If no handler was previously installed for the signal then the default action is taken as per sigaction(2).

The SIGTSTP, SIGTTIN and SIGTTOU signals (stop signals generated from keyboard or due to terminal I/O from a background process) are treated specially. When the process is resumed after it has been stopped, **readpassphrase**() will reprint the prompt and the user may then enter a passphrase.

SEE ALSO

sigaction(2), getpass(3)

STANDARDS

The **readpassphrase**() function is an extension and should not be used if portability is desired.

HISTORY

The **readpassphrase**() function first appeared in FreeBSD 4.6 and OpenBSD 2.9.