

## NAME

Term::ReadLine - Perl interface to various readline packages. If no real package is found, substitutes stubs instead of basic functions.

## SYNOPSIS

```
use Term::ReadLine;
my $term = new Term::ReadLine 'Simple Perl calc';
my $prompt = "Enter your arithmetic expression: ";
my $OUT = $term->OUT || \*STDOUT;
while ( defined ( $_ = $term->readline($prompt) ) ) {
    my $res = eval($_);
    warn $@ if $@;
    print $OUT $res, "\n" unless $@;
    $term->addhistory($_) if /\S/;
}
```

## DESCRIPTION

This package is just a front end to some other packages. It's a stub to set up a common interface to the various ReadLine implementations found on CPAN (under the `Term::ReadLine::*` namespace).

### Minimal set of supported functions

All the supported functions should be called as methods, i.e., either as

```
$term = new Term::ReadLine 'name';
```

or as

```
$term->addhistory('row');
```

where `$term` is a return value of `Term::ReadLine->new()`.

`ReadLine`

returns the actual package that executes the commands. Among possible values are `Term::ReadLine::Gnu`, `Term::ReadLine::Perl`, `Term::ReadLine::Stub`.

`new`

returns the handle for subsequent calls to following functions. Argument is the name of the application. Optionally can be followed by two arguments for `IN` and `OUT` filehandles. These arguments should be globs.

`readline`

gets an input line, *possibly* with actual `readline` support. Trailing newline is removed. Returns `undef` on EOF.

`addhistory`

adds the line to the history of input, from where it can be used if the actual `readline` is present.

`IN, OUT`

return the filehandles for input and output or `undef` if `readline` input and output cannot be used for Perl.

`MinLine`

If argument is specified, it is an advice on minimal size of line to be included into history. `undef` means do not include anything into history. Returns the old value.

`findConsole`

returns an array with two strings that give most appropriate names for files for input and output using conventions "`<$in`", "`>out`".

`Attribs`

returns a reference to a hash which describes internal configuration of the package. Names of keys in this hash conform to standard conventions with the leading `rl_` stripped.

`Features`

Returns a reference to a hash with keys being features present in current implementation. Several optional features are used in the minimal interface: `appname` should be present if the first argument to `new` is recognized, and `minline` should be present if `MinLine` method is not dummy. `autohistory` should be present if lines are put into history automatically (maybe subject to `MinLine`), and `addhistory` if `addhistory` method is not dummy.

If `Features` method reports a feature `attribs` as present, the method `Attribs` is not dummy.

## Additional supported functions

Actually `Term::ReadLine` can use some other package, that will support a richer set of commands.

All these commands are callable via method interface and have names which conform to standard conventions with the leading `rl_` stripped.

The stub package included with the perl distribution allows some additional methods:

`tkRunning`

makes Tk event loop run when waiting for user input (i.e., during `readline` method).

`ornaments`

makes the command line stand out by using termcap data. The argument to `ornaments` should be 0, 1, or a string of a form "`aa,bb,cc,dd`". Four components of this string should be names of *terminal capacities*, first two will be issued to make the prompt standout, last two to make the input line standout.

`newTTY`

takes two arguments which are input filehandle and output filehandle. Switches to use these filehandles.

One can check whether the currently loaded `ReadLine` package supports these methods by checking for corresponding `Features`.

## EXPORTS

None

## ENVIRONMENT

The environment variable `PERL_RL` governs which `ReadLine` clone is loaded. If the value is false, a dummy interface is used. If the value is true, it should be tail of the name of the package to use, such as `Perl` or `Gnu`.

As a special case, if the value of this variable is space-separated, the tail might be used to disable the ornaments by setting the tail to be `o=0` or `ornaments=0`. The head should be as described above, say

If the variable is not set, or if the head of space-separated list is empty, the best available package is loaded.

```
export "PERL_RL=Perl o=0" # Use Perl ReadLine without ornaments
export "PERL_RL= o=0"    # Use best available ReadLine without ornaments
```

(Note that processing of `PERL_RL` for ornaments is in the discretion of the particular used `Term::ReadLine::*` package).

## CAVEATS

It seems that using `Term::ReadLine` from Emacs minibuffer doesn't work quite right and one will get an error message like

```
Cannot open /dev/tty for read at ...
```

One possible workaround for this is to explicitly open `/dev/tty` like this

```
open (FH, "/dev/tty" )
    or eval 'sub Term::ReadLine::findConsole { ("&STDIN", "&STDERR") }';
die $@ if $@;
close (FH);
```

or you can try using the 4-argument form of `Term::ReadLine->new()`.