NAME

Term::UI - Term::ReadLine UI made easy

SYNOPSIS

```
use Term::UI;
use Term::ReadLine;

my $term = Term::ReadLine->new('brand');

my $reply = $term->get_reply(
                prompt => 'What is your favourite colour?',
                choices => [qw|blue red green|],
                default => 'blue',
);

my $bool = $term->ask_yn(
                    prompt => 'Do you like cookies?',
                    default => 'y',
            );

my $string = q[some_command -option --no-foo --quux='this thing'];

my ($options,$munged_input) = $term->parse_options($string);

### don't have Term::UI issue warnings -- default is '1'
$Term::UI::VERBOSE = 0;

### always pick the default (good for non-interactive terms)
### -- default is '0'
$Term::UI::AUTOREPLY = 1;

### Retrieve the entire session as a printable string:
$hist = Term::UI::History->history_as_string;
$hist = $term->history_as_string;
```

DESCRIPTION

`Term::UI` is a transparent way of eliminating the overhead of having to format a question and then validate the reply, informing the user if the answer was not proper and re-issuing the question.

Simply give it the question you want to ask, optionally with choices the user can pick from and a default and `Term::UI` will DWYM.

For asking a yes or no question, there's even a shortcut.

HOW IT WORKS

`Term::UI` places itself at the back of the `Term::ReadLine` `@ISA` array, so you can call its functions through your term object.

`Term::UI` uses `Term::UI::History` to record all interactions with the commandline. You can retrieve this history, or alter the filehandle the interaction is printed to. See the `Term::UI::History` manpage or the `SYNOPSIS` for details.

METHODS

$reply = $term->get_reply(prompt => 'question?', [choices => \@list, default => $list[0], multi => BOOL, print_me => "extra text to print & record", allow => $ref]);

`get_reply` asks a user a question, and then returns the reply to the caller. If the answer is invalid (more on that below), the question will be reposed, until a satisfactory answer has been entered.

You have the option of providing a list of choices the user can pick from using the `choices` argument. If the answer is not in the list of choices presented, the question will be reposed.

If you provide a `default` answer, this will be returned when either `$AUTOREPLY` is set to true, (see the `GLOBAL VARIABLES` section further below), or when the user just hits `enter`.

You can indicate that the user is allowed to enter multiple answers by toggling the `multi` flag. Note that a list of answers will then be returned to you, rather than a simple string.

By specifying an `allow` hander, you can yourself validate the answer a user gives. This can be any of the types that the Params::Check `allow` function allows, so please refer to that manpage for details.

Finally, you have the option of adding a `print_me` argument, which is simply printed before the prompt. It's printed to the same file handle as the rest of the questions, so you can use this to keep track of a full session of Q&A with the user, and retrieve it later using the `Term::UI->history_as_string` function.

See the `EXAMPLES` section for samples of how to use this function.

$bool = $term->ask_yn(prompt => "your question", [default => (y|1,n|0), print_me => "extra text to print & record"])

Asks a simple `yes` or no question to the user, returning a boolean indicating `true` or `false` to the caller.

The `default` answer will automatically returned, if the user hits `enter` or if `$AUTOREPLY` is set to true. See the `GLOBAL VARIABLES` section further below.

Also, you have the option of adding a `print_me` argument, which is simply printed before the prompt. It's printed to the same file handle as the rest of the questions, so you can use this to keep track of a full session of Q&A with the user, and retrieve it later using the `Term::UI->history_as_string` function.

See the `EXAMPLES` section for samples of how to use this function.

($opts, $munged) = $term->parse_options(STRING);

`parse_options` will convert all options given from an input string to a hash reference. If called in list context it will also return the part of the input string that it found no options in.

Consider this example:

```
    my $str =   q[command --no-foo --baz --bar=0 --quux=bleh ] .
                q[--option="some'thing" -one-dash -single=blah' arg];

    my ($options,$munged) =  $term->parse_options($str);

    ### $options would contain: ###
    $options = {
                'foo'       => 0,
                'bar'       => 0,
                'one-dash'  => 1,
                'baz'       => 1,
                'quux'      => 'bleh',
                'single'    => 'blah\'',
                'option'    => 'some\'thing'
    };

    ### and this is the munged version of the input string,
    ### ie what's left of the input minus the options
    $munged = 'command arg';
```

As you can see, you can either use a single or a double `-` to indicate an option. If you prefix an option with `no-` and do not give it a value, it will be set to 0. If it has no prefix and no value, it will be set to 1. Otherwise, it will be set to its value. Note also that it can deal fine with single/double quoting issues.

$str = $term->history_as_string

Convenience wrapper around `Term::UI::History->history_as_string`.

Consult the `Term::UI::History` man page for details.

GLOBAL VARIABLES

The behaviour of Term::UI can be altered by changing the following global variables:

$Term::UI::VERBOSE

This controls whether Term::UI will issue warnings and explanations as to why certain things may have failed. If you set it to 0, Term::UI will not output any warnings. The default is 1;

$Term::UI::AUTOREPLY

This will make every question be answered by the default, and warn if there was no default provided. This is particularly useful if your program is run in non-interactive mode. The default is 0;

$Term::UI::INVALID

This holds the string that will be printed when the user makes an invalid choice. You can override this string from your program if you, for example, wish to do localization. The default is `Invalid selection, please try again:`

$Term::UI::History::HISTORY_FH

This is the filehandle all the print statements from this module are being sent to. Please consult the `Term::UI::History` manpage for details.

This defaults to `*STDOUT`.

EXAMPLES

Basic get_reply sample

```
    ### ask a user (with an open question) for their favourite colour
    $reply = $term->get_reply( prompt => 'Your favourite colour? );
```

which would look like:

```
    Your favourite colour?
```

and `$reply` would hold the text the user typed.

get_reply with choices

```
    ### now provide a list of choices, so the user has to pick one
    $reply = $term->get_reply(
                prompt  => 'Your favourite colour?',
                choices => [qw|red green blue|] );
```

which would look like:

```
      1> red
      2> green
      3> blue

    Your favourite colour?
```

`$reply` will hold one of the choices presented. `Term::UI` will repose the question if the user attempts to enter an answer that's not in the list of choices. The string presented is held in the `$Term::UI::INVALID` variable (see the `GLOBAL VARIABLES` section for details.

get_reply with choices and default

```
    ### provide a sensible default option -- everyone loves blue!
    $reply = $term->get_reply(
                prompt  => 'Your favourite colour?',
                choices => [qw|red green blue|],
                default => 'blue' );
```

which would look like:

```
      1> red
      2> green
      3> blue

    Your favourite colour? [3]:
```

Note the default answer after the prompt. A user can now just hit `enter` (or set `$Term::UI::AUTOREPLY` -- see the `GLOBAL VARIABLES` section) and the sensible answer 'blue' will be returned.

get_reply using print_me & multi

```
    ### allow the user to pick more than one colour and add an
    ### introduction text
    @reply = $term->get_reply(
                print_me    => 'Tell us what colours you like',
                prompt      => 'Your favourite colours?',
                choices     => [qw|red green blue|],
                multi       => 1 );
```

which would look like:

```
    Tell us what colours you like
      1> red
      2> green
      3> blue

    Your favourite colours?
```

An answer of `3 2 1` would fill `@reply` with `blue green red`

get_reply & allow

```
    ### pose an open question, but do a custom verification on
    ### the answer, which will only exit the question loop, if
```

```
### the answer matches the allow handler.
$reply = $term->get_reply(
            prompt  => "What is the magic number?",
            allow   => 42 );
```

Unless the user now enters 42, the question will be reposed over and over again. You can use more sophisticated `allow` handlers (even subroutines can be used). The `allow` handler is implemented using `Params::Check`'s `allow` function. Check its manpage for details.

an elaborate ask_yn sample

```
### ask a user if he likes cookies. Default to a sensible 'yes'
### and inform him first what cookies are.
$bool = $term->ask_yn( prompt   => 'Do you like cookies?',
                       default  => 'y',
                       print_me => 'Cookies are LOVELY!!!' );
```

would print:

```
Cookies are LOVELY!!!
Do you like cookies? [Y/n]:
```

If a user then simply hits `enter`, agreeing with the default, `$bool` would be set to `true`. (Simply hitting 'y' would also return `true`. Hitting 'n' would return `false`)

We could later retrieve this interaction by printing out the Q&A history as follows:

```
print $term->history_as_string;
```

which would then print:

```
Cookies are LOVELY!!!
Do you like cookies? [Y/n]:  y
```

There's a chance we're doing this non-interactively, because a console is missing, the user indicated he just wanted the defaults, etc.

In this case, simply setting `$Term::UI::AUTOREPLY` to true, will return from every question with the default answer set for the question. Do note that if `AUTOREPLY` is true, and no default is set, `Term::UI` will warn about this and return `undef`.

See Also

`Params::Check`, `Term::ReadLine`, `Term::UI::History`

BUG REPORTS

Please report bugs or other issues to <bug-term-ui@rt.cpan.org<gt>.

AUTHOR

This module by Jos Boumans <kane@cpan.org>.

COPYRIGHT