Package 'cli'

September 25, 2018

Title Helpers for Developing Command Line Interfaces

Version 1.0.1

Description A suite of tools designed to build attractive command line interfaces ('CLIs'). Includes tools for drawing rules, boxes, trees, and 'Unicode' symbols with 'ASCII' alternatives.

License MIT + file LICENSE

LazyData true

URL https://github.com/r-lib/cli#readme

BugReports https://github.com/r-lib/cli/issues

RoxygenNote 6.1.0

Depends R (>= 2.10)

Imports assertthat, crayon (>= 1.3.4), methods, utils

Suggests covr, fansi, mockery, testthat, webshot, withr

Encoding UTF-8

NeedsCompilation no

Author Gábor Csárdi [aut, cre],
Hadley Wickham [ctb],
Kirill Müller [ctb]

Maintainer Gábor Csárdi <csardi.gabor@gmail.com>

Repository CRAN

Date/Publication 2018-09-25 11:40:07 UTC

R topics documented:

cat_line 2
cli_sitrep 3
console_width 3
demo_spinners 4
get_spinner 4
is_utf8_output 5

list_border_styles 5
list_spinners 7
rule 8
symbol 9
tree 10

Index **12**

`cat_line` `cat()` *helpers*

Description

These helpers provide useful wrappers around `cat()`: most importantly they all set `sep = ""`, and `cat_line()` automatically adds a newline.

Usage

```
cat_line(..., col = NULL, background_col = NULL, file = stdout())

cat_bullet(..., col = NULL, background_col = NULL, bullet = "bullet",
  bullet_col = NULL, file = stdout())

cat_boxx(..., file = stdout())

cat_rule(..., file = stdout())

cat_print(x, file = "")
```

Arguments

`...`	For `cat_line()` and `cat_bullet()`, paste'd together with `collapse = "\n"`. For `cat_rule()` and `cat_boxx()` passed on to `rule()` and `boxx()` respectively.
`col, background_col, bullet_col`	Colours for text, background, and bullets respectively.
`file`	Output destination. Defaults to standard output.
`bullet`	Name of bullet character. Indexes into symbols
`x`	An object to print.

Examples

```
cat_line("This is ", "a ", "line of text.", col = "red")
cat_bullet(letters[1:5])
cat_bullet(letters[1:5], bullet = "tick", bullet_col = "green")
cat_rule()
```

cli_sitrep *cli situation report*

Description

Contains currenty:

- `cli_unicode_option`: whether the `cli.unicode` option is set and its value. See `is_utf8_output()`.
- `symbol_charset`: the selected character set for symbol, UTF-8, Windows, or ASCII.
- `console_utf8`: whether the console supports UTF-8. See `base::l10n_info()`.
- `latex_active`: whether we are inside knitr, creating a LaTeX document.
- `num_colors`: number of ANSI colors. See `crayon::num_colors()`.
- `console_with`: detected console width.

Usage

```
cli_sitrep()
```

Value

Named list with entries listed above. It has a `cli_sitrep` class, with a `print()` and `format()` method.

Examples

```
cli_sitrep()
```

console_width *Determine the width of the console*

Description

It uses the `RSTUDIO_CONSOLE_WIDTH` environment variable, if set. Otherwise it uses the `width` option. If this is not set either, then 80 is used.

Usage

```
console_width()
```

Value

Integer scalar, the console with, in number of characters.

demo_spinners *Show a demo of some (by default all) spinners*

Description

Each spinner is shown for about 2-3 seconds.

Usage

```
demo_spinners(which = NULL)
```

Arguments

which	Character vector, which spinners to demo.

See Also

Other spinners: get_spinner, list_spinners

Examples

```
## Not run:
  demo_spinners(sample(list_spinners(), 10))

## End(Not run)
```

get_spinner *Character vector to put a spinner on the screen*

Description

cli contains many different spinners, you choose one according to your taste.

Usage

```
get_spinner(which = NULL)
```

Arguments

which	The name of the chosen spinner. The default depends on whether the platform supports Unicode.

Value

A list with entries: name, interval: the suggested update interval in milliseconds and frames: the character vector of the spinner's frames.

See Also

Other spinners: demo_spinners, list_spinners

Examples

```
get_spinner()
get_spinner("shark")
```

---

is_utf8_output *Whether cli is emitting UTF-8 characters*

Description

UTF-8 cli characters can be turned on by setting the `cli.unicode` option to `TRUE`. They can be turned on by setting if to `FALSE`. If this option is not set, then `base::l10n_info()` is used to detect UTF-8 support.

Usage

```
is_utf8_output()
```

Value

Flag, whether cli uses UTF-8 characters.

---

list_border_styles *Draw a banner-like box in the console*

Description

Draw a banner-like box in the console

Usage

```
list_border_styles()

boxx(label, border_style = "single", padding = 1, margin = 0,
  float = c("left", "center", "right"), col = NULL,
  background_col = NULL, border_col = col, align = c("left",
  "center", "right"), width = console_width())
```

Arguments

label	Label to show, a character vector. Each element will be in a new line. You can color it using the crayon packag, see examples below.
border_style	String that specifies the border style. list_border_styles lists all current styles.
padding	Padding within the box. Either an integer vector of four numbers (bottom, left, top, right), or a single number x, which is interpreted as c(x, 3*x, x, 3*x).
margin	Margin around the box. Either an integer vector of four numbers (bottom, left, top, right), or a single number x, which is interpreted as c(x, 3*x, x, 3*x).
float	Whether to display the box on the "left", "center", or the "right" of the screen.
col	Color of text, and default border color. Either a crayon style function or a color name that is passed to crayon::make_style().
background_col	Background color of the inside of the box. Either a crayon style function, or a color name which will be used in crayon::make_style() to create a *background* style (i.e. bg = TRUE is used).
border_col	Color of the border. Either a crayon style function or a color name that is passed to crayon::make_style().
align	Alignment of the label within the box: "left", "center", or "right".
width	Width of the screen, defaults to getOption("width").

About fonts and terminal settings

The boxes might or might not look great in your terminal, depending on the box style you use and the font the terminal uses. We found that the Menlo font looks nice in most terminals an also in Emacs.

RStudio currently has a line height greater than one for console output, which makes the boxes ugly.

Examples

```
## Simple box
boxx("Hello there!")

## All border styles
list_border_styles()

## Change border style
boxx("Hello there!", border_style = "double")

## Multiple lines
boxx(c("Hello", "there!"), padding = 1)

## Padding
boxx("Hello there!", padding = 1)
boxx("Hello there!", padding = c(1, 5, 1, 5))

## Margin
```

```
boxx("Hello there!", margin = 1)
boxx("Hello there!", margin = c(1, 5, 1, 5))
boxx("Hello there!", padding = 1, margin = c(1, 5, 1, 5))

## Floating
boxx("Hello there!", padding = 1, float = "center")
boxx("Hello there!", padding = 1, float = "right")

## Text color
boxx(crayon::cyan("Hello there!"), padding = 1, float = "center")

## Backgorund color
boxx("Hello there!", padding = 1, background_col = "brown")
boxx("Hello there!", padding = 1, background_col = crayon::bgRed)

## Border color
boxx("Hello there!", padding = 1, border_col = "green")
boxx("Hello there!", padding = 1, border_col = crayon::red)

## Label alignment
boxx(c("Hi", "there", "you!"), padding = 1, align = "left")
boxx(c("Hi", "there", "you!"), padding = 1, align = "center")
boxx(c("Hi", "there", "you!"), padding = 1, align = "right")

## A very customized box
star <- symbol$star
label <- c(paste(star, "Hello", star), "  there!")
boxx(
  crayon::white(label),
  border_style="round",
  padding = 1,
  float = "center",
  border_col = "tomato3",
  background_col="darkolivegreen"
)
```

`list_spinners` *List all available spinners*

Description

List all available spinners

Usage

```
list_spinners()
```

Value

Character vector of all available spinner names.

See Also

Other spinners: demo_spinners, get_spinner

Examples

```
list_spinners()
get_spinner(list_spinners()[1])
```

`rule` *Make a rule with one or two text labels*

Description

The rule can include either a centered text label, or labels on the left and right side.

Usage

```
rule(left = "", center = "", right = "", line = 1, col = NULL,
  line_col = col, background_col = NULL, width = console_width())
```

Arguments

`left`	Label to show on the left. It interferes with the `center` label, only at most one of them can be present.
`center`	Label to show at the center. It interferes with the `left` and `right` labels.
`right`	Label to show on the right. It interferes with the `center` label, only at most one of them can be present.
`line`	The character or string that is used to draw the line. It can also 1 or 2, to request a single line (Unicode, if available), or a double line. Some strings are interpreted specially, see *Line styles* below.
`col`	Color of text, and default line color. Either a `crayon` style function or a color name that is passed to `crayon::make_style()`.
`line_col, background_col`	Either a color name (used in `crayon::make_style()`), or a style function from `crayon`, to color the line and background.
`width`	Width of the rule. Defaults to the `width` option, see `base::options()`.

Details

To color the labels, use the functions from the `crayon` package directly, see examples below. To color the line, either use the `crayon` colors directly, or the `line_col` option.

Value

Character scalar, the rule.

Line styles

Some strings for the `line` argument are interpreted specially:

- `"single"`: (same as `1`), a single line,
- `"double"`: (same as `2`), a double line,
- `"bar1"`, `"bar2"`, `"bar3"`, etc., `"bar8"` uses varying height bars.

Examples

```
## Simple rule
rule()

## Double rule
rule(line = 2)

## Bars
rule(line = "bar2")
rule(line = "bar5")

## Left label
rule(left = "Results")

## Centered label
rule(center = " * RESULTS * ")

## Colored labels
rule(center = crayon::red(" * RESULTS * "))

## Colored line
rule(center = crayon::red(" * RESULTS * "), line_col = "red")

## Custom line
rule(center = "TITLE", line = "~")

## More custom line
rule(center = "TITLE", line = crayon::blue("~-"))

## Even more custom line
rule(center = crayon::bgRed(" ", symbol$star, "TITLE",
  symbol$star, " "),
  line = "\u2582",
  line_col = "orange")
```

`symbol` *Various handy symbols to use in a command line UI*

Description

Various handy symbols to use in a command line UI

Usage

```
symbol
```

Format

A named list, see `names(symbol)` for all sign names.

Details

On Windows they have a fallback to less fancy symbols.

Examples

```
cat(symbol$tick, " SUCCESS\n", symbol$cross, " FAILURE\n", sep = "")

## All symbols
cat(paste(format(names(symbol), width = 20),
  unlist(symbol)), sep = "\n")
```

---

tree *Draw a tree*

Description

Draw a tree using box drawing characters. Unicode characters are used if available. (Set the `cli.unicode` option if auto-detection fails.)

Usage

```
tree(data, root = data[[1]][[1]], style = NULL,
  width = console_width())
```

Arguments

`data`	Data frame that contains the tree structure. The first column is an id, and the second column is a list column, that contains the ids of the child nodes. The optional third column may contain the text to print to annotate the node.
`root`	The name of the root node.
`style`	Optional box style list.
`width`	Maximum width of the output. Defaults to the `width` option, see `base::options()`.

Details

A node might appear multiple times in the tree, or might not appear at all.

Value

Character vector, the lines of the tree drawing.

Examples

```
data <- data.frame(
  stringsAsFactors = FALSE,
  package = c("processx", "backports", "assertthat", "Matrix",
    "magrittr", "rprojroot", "clisymbols", "prettyunits", "withr",
    "desc", "igraph", "R6", "crayon", "debugme", "digest", "irlba",
    "rcmdcheck", "callr", "pkgconfig", "lattice"),
  dependencies = I(list(
    c("assertthat", "crayon", "debugme", "R6"), character(0),
    character(0), "lattice", character(0), "backports", character(0),
    c("magrittr", "assertthat"), character(0),
    c("assertthat", "R6", "crayon", "rprojroot"),
    c("irlba", "magrittr", "Matrix", "pkgconfig"), character(0),
    character(0), "crayon", character(0), "Matrix",
    c("callr", "clisymbols", "crayon", "desc", "digest", "prettyunits",
      "R6", "rprojroot", "withr"),
    c("processx", "R6"), character(0), character(0)
  ))
)
tree(data)
tree(data, root = "rcmdcheck")

## Colored nodes
data$label <- paste(data$package,
  crayon::blurred(paste0("(", c("2.0.0.1", "1.1.1", "0.2.0", "1.2-11",
    "1.5", "1.2", "1.2.0", "1.0.2", "2.0.0", "1.1.1.9000", "1.1.2",
    "2.2.2", "1.3.4", "1.0.2", "0.6.12", "2.2.1", "1.2.1.9002",
    "1.0.0.9000", "2.0.1", "0.20-35"), ")"))
  )
roots <- ! data$package %in% unlist(data$dependencies)
data$label[roots] <- crayon::cyan(crayon::italic(data$label[roots]))
tree(data)
tree(data, root = "rcmdcheck")
```

Index

base::l10n_info(), *3*, *5*
base::options(), *8*, *10*
boxx *(*list_border_styles*)*, 5
boxx(), *2*

cat(), *2*
cat_boxx *(*cat_line*)*, 2
cat_bullet *(*cat_line*)*, 2
cat_line, 2
cat_print *(*cat_line*)*, 2
cat_rule *(*cat_line*)*, 2
cli_sitrep, 3
console_width, 3
crayon::make_style(), *6*, *8*
crayon::num_colors(), *3*

demo_spinners, 4, *5*, *8*

get_spinner, *4*, 4, *8*

is_utf8_output, 5
is_utf8_output(), *3*

list_border_styles, 5
list_spinners, *4*, *5*, 7

rule, 8
rule(), *2*

symbol, *3*, 9
symbols, *2*

tree, 10