

Package: textshaping (via r-universe)

September 30, 2026

Title Bindings to the 'HarfBuzz' and 'Fribidi' Libraries for Text Shaping

Version 1.0.5.9000

Description Provides access to the text shaping functionality in the 'HarfBuzz' library and the bidirectional algorithm in the 'Fribidi' library. 'textshaping' is a low-level utility package mainly for graphic devices that expands upon the font tool-set provided by the 'systemfonts' package.

License MIT + file LICENSE

URL <https://github.com/r-lib/textshaping>

BugReports <https://github.com/r-lib/textshaping/issues>

Depends R (>= 3.2.0)

Imports lifecycle, stats, stringi, systemfonts (>= 1.3.2.9000), utils

Suggests covr, grDevices, grid, knitr, rmarkdown, testthat (>= 3.0.0)

LinkingTo cpp11 (>= 0.2.1), systemfonts (>= 1.0.0)

VignetteBuilder knitr

Config/build/compilation-database true

Config/testthat/edition 3

Config/usethis/last-upkeep 2025-04-23

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

SystemRequirements freetype2, harfbuzz, fribidi

Remotes r-lib/systemfonts

Config/pak/sysreqs

libfontconfig1-dev libfreetype6-dev libfribidi-dev libharfbuzz-dev libicu-dev

Repository <https://r-lib.r-universe.dev>

Date/Publication 2026-03-18 10:34:44 UTC

RemoteUrl <https://github.com/r-lib/textshaping>
RemoteRef HEAD
RemoteSha 67c94db6939cd4ca9c2915d7ffb6858a79ce80cd

Contents

get_font_features	2
lorem_text	3
math_grob	4
plot_math	6
plot_shape	7
shape_math	8
shape_text	12
text_width	15

Index	18
--------------	-----------

get_font_features	<i>Get available OpenType features in a font</i>
-------------------	--

Description

This is a simply functions that returns the available OpenType feature tags for one or more fonts. See [font_feature\(\)](#) for more information on how to use the different feature with a font.

Usage

```
get_font_features(  
  family = "",  
  italic = FALSE,  
  bold = FALSE,  
  path = NULL,  
  index = 0  
)
```

Arguments

family	The name of the font families to match
italic	logical indicating the font slant
bold	[Deprecated] Use <code>weight = "bold"</code> instead
path, index	path and index of a font file to circumvent lookup based on family and style

Value

A list with an element for each of the input fonts containing the supported feature tags for that font.

Examples

```
# Select a random font on the system
sys_fonts <- systemfonts::system_fonts()
random_font <- sys_fonts$family[sample(nrow(sys_fonts), 1)]

# Get the features
get_font_features(random_font)
```

lorem_text	<i>Get gibberish text in various scripts</i>
------------	--

Description

Textshaping exists partly to allow all the various scripts that exists in the world to be used in R graphics. This function returns gibberish filler text (lorem ipsum text) in various scripts for testing purpose. Some of these are transliterations of the original lorem ipsum text while others are based an a distribution model.

Usage

```
lorem_text(
  script = c("latin", "chinese", "arabic", "devanagari", "cyrillic", "kana", "hangul",
    "greek", "hebrew", "armenian", "georgian"),
  n = 1
)

lorem_bidi(
  ltr = c("latin", "chinese", "devanagari", "cyrillic", "kana", "hangul", "greek",
    "armenian", "georgian"),
  rtl = c("arabic", "hebrew"),
  ltr_prop = 0.9,
  n = 1
)
```

Arguments

script	A string giving the script to fetch gibberish for
n	The number of paragraphs to fetch. Each paragraph will be its own element in the returned character vector.
ltr, rtl	scripts to use for left-to-right and right-to-left text
ltr_prop	The approximate proportion of left-to-right text in the final string

Value

a character vector of length `n`

References

<https://generator.lorem-ipsum.info>

Examples

```
# Defaults to standard lorem ipsum
lorem_text()

# Get two paragraphs of hangul (Korean)
lorem_text("hangul", 2)

# Get gibberish bi-directional text
lorem_bidi()
```

`math_grob`

Create a grid grob for a math expression

Description

This function creates a grid grob that renders a math expression. The grob can be used with standard grid functions like `grid::grid.draw()`, combined with other grobs using `grid::grobTree()`, or used in ggplot2 annotations. Like the standard grid grobs, `math_grob()` creates a grob object to pass around while `grid.math()` both creates the grob and render it in one pass.

Usage

```
math_grob(
  math,
  x = 0.5,
  y = 0.5,
  hjust = "left",
  vjust = "baseline",
  default.units = "npc",
  name = NULL,
  gp = NULL,
  vp = NULL,
  family = "math",
  size = 12,
  res = 72,
  inline = TRUE,
  path = NULL,
  index = 0,
```

```

    color = "black"
)

grid.math(
    math,
    x = 0.5,
    y = 0.5,
    hjust = "left",
    vjust = "baseline",
    default.units = "npc",
    name = NULL,
    gp = NULL,
    vp = NULL,
    family = "math",
    size = 12,
    res = 72,
    inline = TRUE,
    path = NULL,
    index = 0,
    color = "black",
    draw = TRUE
)

```

Arguments

math	A single math expression string. Use raw strings for cleaner syntax, e.g., <code>r"(\frac{a}{b})"</code> .
x, y	The position of the grob. Can be a unit object or a numeric value (interpreted as npc units).
hjust	Horizontal justification. Either a numeric value (0 = left, 0.5 = center, 1 = right) or a character string ("left", "center", "right").
vjust	Vertical justification. Either a numeric value (0 = bottom, 0.5 = center, 1 = top) or a character string ("bottom", "center", "top", "baseline"). The default "baseline" places the math baseline at the y position, which is most useful for aligning with text.
default.units	A string indicating the default units to use if x or y are only given as numeric values.
name	A character identifier for the grob.
gp	A <code>grid::gpar()</code> object containing graphical parameters, or NULL for defaults. Supported parameters are <code>fontsize</code> , <code>col</code> , and <code>fontfamily</code> .
vp	A <code>grid::viewport()</code> object (or NULL).
family	Font family name. For best results, use a font with an OpenType MATH table such as "STIX Two Math", "Latin Modern Math", "Cambria Math", or "Libertinus Math". Fonts without a MATH table will use fallback positioning constants based on TeX defaults, which produces reasonable but less precise results. Default uses the system's default math font.

size	The size in points to use for the font
res	The resolution to use when doing the shaping. Should optimally match the resolution used when rendering the glyphs.
inline	Logical. If TRUE (default), uses inline style. If FALSE , uses display style (larger operators with limits above/below).
path, index	path an index of a font file to circumvent lookup based on family and style
color	Base color for the equation (default "black"). Individual elements can override this using <code>\textcolor{color}{content}</code> .
draw	Should the created grob be rendered

Value

A grob object of class "math_grob".

Examples

```
# Create a math grob (baseline at y=0.5)
g <- math_grob(r"(\frac{a}{b})", x = 0.5, y = 0.5)

# Draw it (requires a device that supports glyph rendering)
try(grid::grid.draw(g))

# Center the expression in the viewport
g <- math_grob(r"(E = mc^2)", x = 0.5, y = 0.5,
              hjust = "center", vjust = "center")
try(grid::grid.draw(g))

# Combine with other grobs
tree <- grid::grobTree(
  grid::rectGrob(gp = grid::gpar(fill = "lightblue")),
  math_grob(r"(E = mc^2)", x = 0.5, y = 0.5,
            hjust = "center", vjust = "center")
)
try(grid::grid.draw(tree))
```

plot_math

Preview shaped math expression

Description

This function allows you to preview the layout that `shape_math()` calculates. It renders the positioned glyphs and lines (fraction bars, radical overlines, cancel lines, etc.) using grid graphics, and shows the position of the baseline relative to the equation. This is primarily a debugging and visualization tool.

Usage

```
plot_math(shaped, id = 1)
```

Arguments

shaped The output of a call to `shape_math()`

id The index of the expression to show if **shaped** contains multiples

Value

This function is called for its side effects

Examples

```
## Not run:
shaped <- shape_math(r"(\frac{a}{b})", family = "STIX Two Math")
plot_math(shaped)

shaped <- shape_math("x^2 + y^2 = z^2", family = "STIX Two Math")
plot_math(shaped)

## End(Not run)
```

plot_shape	<i>Preview shaped text and the metrics for the text box</i>
------------	---

Description

This function allows you to preview the layout that `shape_text()` calculates. It is purely meant as a sanity check to make sure that the values calculated are sensible and shouldn't be used as a plotting function for rendering text on its own.

Usage

```
plot_shape(shape, id = 1)
```

Arguments

shape The output of a call to `shape_text()`

id The index of the text run to show in case **shape** contains multiples

Value

This function is called for its side effects

Examples

```

arab_text <- lorem_text("arabic", 2)
shape <- shape_text(
  arab_text,
  max_width = 5,
  indent = 0.2
)

try(
  plot_shape(shape)
)

```

shape_math	<i>Shape LaTeX math expression</i>
------------	------------------------------------

Description

Parses a LaTeX math expression and returns positioned glyphs for rendering. The function uses HarfBuzz's OpenType MATH table support to properly position mathematical symbols, scripts, fractions, and other structures according to the font's mathematical typesetting rules.

Usage

```

shape_math(
  math,
  family = "math",
  size = 12,
  res = 72,
  inline = TRUE,
  color = "black",
  path = NULL,
  index = 0
)

```

Arguments

math	A character vector of LaTeX math expressions. Use raw strings for cleaner syntax, e.g., <code>r"(\alpha)"</code> for α , <code>r"(\frac{a}{b})"</code> for fractions. Alternatively escape backslashes: <code>"\\alpha"</code> .
family	Font family name. For best results, use a font with an OpenType MATH table such as "STIX Two Math", "Latin Modern Math", "Cambria Math", or "Libertinus Math". Fonts without a MATH table will use fallback positioning constants based on TeX defaults, which produces reasonable but less precise results. Default uses the system's default math font.
size	The size in points to use for the font

res	The resolution to use when doing the shaping. Should optimally match the resolution used when rendering the glyphs.
inline	Logical. If TRUE (default), uses inline style. If FALSE , uses display style (larger operators with limits above/below).
color	Base color for the equation (default "black"). Individual elements can override this using <code>\textcolor{color}{content}</code> .
path, index	path an index of a font file to circumvent lookup based on family and style

Details

Font Requirements:

Mathematical typesetting works best with fonts that include an OpenType MATH table, which provides precise positioning parameters for scripts, fractions, radicals, and other structures. When using a font without a MATH table (such as Arial or Times New Roman), the function falls back to default positioning constants based on TeX conventions. The results will be reasonable but may not match the typographic quality of a dedicated math font.

Supported LaTeX Commands:

LaTeX math expressions is a huge specification and also includes numerous extensions with additional functionality. Supporting the full set would be unfeasible but textshaping tries hard to cover all but the most esoteric needs. The list of supported functionality is:

Greek letters `\alpha`, `\beta`, `\gamma`, `\delta`, `\epsilon`, `\varepsilon`, `\zeta`, `\eta`, `\theta`, `\vartheta`, `\iota`, `\kappa`, `\lambda`, `\mu`, `\nu`, `\xi`, `\pi`, `\varpi`, `\rho`, `\varrho`, `\sigma`, `\varsigma`, `\tau`, `\upsilon`, `\phi`, `\varphi`, `\chi`, `\psi`, `\omega`, `\Gamma`, `\Delta`, `\Theta`, `\Lambda`, `\Xi`, `\Pi`, `\Sigma`, `\Upsilon`, `\Phi`, `\Psi`, `\Omega`

Binary operators `+`, `-`, `=`, `\pm`, `\mp`, `\times`, `\div`, `\cdot`, `\ast`, `\star`, `\circ`, `\bullet`, `\cap`, `\cup`, `\vee`, `\wedge`, `\setminus`, `\oplus`, `\ominus`, `\otimes`, `\oslash`, `\odot`, `\ltimes`, `\rtimes`, `\boxplus`, `\boxminus`, `\boxtimes`

Relations `\leq`, `\leqslant`, `\geq`, `\geqslant`, `\neq`, `\neqslant`, `\equiv`, `\approx`, `\sim`, `\simeq`, `\cong`, `\propto`, `\ll`, `\gg`, `\subset`, `\supset`, `\subseteq`, `\supseteq`, `\in`, `\ni`, `\notin`, `\perp`, `\parallel`, `\mid`, `\prec`, `\succ`, `\preceq`, `\succeq`, `\leqslant`, `\geqslant`, `\lesssim`, `\gtrsim`, `\vdash`, `\Vdash`, `\triangleq`

Arrows `\rightarrow`, `\longrightarrow`, `\leftarrow`, `\Longrightarrow`, `\Leftarrow`, `\leftrightarrow`, `\Leftrightarrow`, `\mapsto`, `\hookrightarrow`, `\hookleftarrow`, `\uparrow`, `\downarrow`, `\Uparrow`, `\Downarrow`, `\updownarrow`, `\nearrow`, `\searrow`, `\swarrow`, `\nwarrow`, `\nleftarrow`, `\nrightarrow`, `\nLeftarrow`, `\nRightarrow`, `\nleftrightarrow`, `\nLeftrightarrow`, `\leftarrowtail`, `\rightarrowtail`, `\twoheadleftarrow`, `\twoheadrightarrow`, `\curvearrowleft`, `\curvearrowright`

Scripts `x^2`, `x_i`, `x_i^2`

Fractions `\frac{num}{denom}`, `\dfrac{}{} (display style)`, `\tfrac{}{} (text style)`, `\cfrac{}{} (continued fraction)`

Roots `\sqrt{x}`, `\sqrt[n]{x}`

Big operators `\sum`, `\prod`, `\coprod`, `\int`, `\iint`, `\iiint`, `\oint`, `\bigcup`, `\bigcap`, `\bigvee`, `\bigwedge`, `\bigoplus`, `\bigotimes`, `\bigodot`, `\biguplus`, `\bigsqcup`

(with $_{{\text{lower}}}^{{\text{upper}}}$ limits). Use `\limits` to force limits above/below or `\nolimits` to force side scripts

Delimiters `\left(, \right), \left[, \right], \left\{, \right\}, \langle, \rangle, \lfloor, \rfloor, \lceil, \rceil, \vert, \Vert, \ulcorner, \urcorner, \llcorner, \lrcorner`

Manual delimiter sizing `\big, \Big, \bigg, \Bigg` and variants `\bigl, \bigr, \bigrm, \Bigl, \Bigr, \Bigm`, etc. for manual control of delimiter size

Accents `\hat{x}, \check{x}, \tilde{x}, \acute{x}, \grave{x}, \dot{x}, \ddot{x}, \dotted{x}, \breve{x}, \bar{x}, \vec{x}, \mathring{x}`

Wide accents `\widehat{abc}, \widetilde{abc}, \overline{abc}, \underline{abc}, \overbrace{abc}, \underbrace{abc}, \overrightarrow{AB}, \overleftarrow{AB}, \overleftrightarrow{AB}, \underrightarrow{ }, \underleftarrow{ }, \underleftrightarrow{ }.`
Overbrace/underbrace support annotations: `\overbrace{x+y}^n, \underbrace{x+y}_n`

Functions `\sin, \cos, \tan, \cot, \sec, \csc, \arcsin, \arccos, \arctan, \arccot, \sinh, \cosh, \tanh, \coth, \log, \ln, \lg, \exp, \lim, \liminf, \limsup, \max, \min, \sup, \inf, \det, \dim, \ker, \hom, \arg, \deg, \gcd, \Pr, \mod`

Text `\text{...}`

Font styles `\mathbf{x}, \mathit{x}, \mathrm{x}, \mathsf{x}, \mathtt{x}, \mathcal{X}, \mathscr{X}, \mathfrak{X}, \mathbb{R}, \boldsymbol{x}, \bm{x}`

Math styles `\displaystyle, \textstyle, \scriptstyle, \scriptscriptstyle` to control operator size and limit placement

Matrices `\begin{matrix}, \begin{pmatrix}, \begin{bmatrix}, \begin{vmatrix}, \begin{Vmatrix}, \begin{Bmatrix}, \begin{smallmatrix}` (with `&` for column separators and `\\` for row separators)

Alignment environments `\begin{aligned}, \begin{gathered}, \begin{split}, \begin{array}{lcr}` for multi-line equations with alignment points

Cases `\begin{cases}, \begin{rcases}, \begin{dcases}, \begin{drcases}`

Stacked expressions `\binom{n}{k}, \tbinom{n}{k}, \dbinom{n}{k}, \overset{a}{b}, \underset{a}{b}, \stackrel{a}{b}, \atop`

Operator names `\operatorname{name}, \operatorname*{name}` (with limits)

Phantoms `\phantom{x}` (invisible with full dimensions), `\hphantom{x}` (width only), `\vphantom{x}` (height/depth only)

Colors `\textcolor{color}{text}, \color{color}{text}, \colorbox{color}{text}`

Cancellation `\cancel{x}, \bcancel{x}, \xcancel{x}, \cancelto{v}{x}`

Boxing `\boxed{expression}`

Substack `\substack{a \\ b}` for multi-line subscripts

Negation `\not` followed by a symbol (e.g., `\not=`)

Extensible arrows `\xrightarrow{above}, \xleftarrow[below]{above}, \xrightarrow{ }, \xleftarrow{ }, \xleftrightarrow{ }, \xhookrightarrow{ }, \xmapsto{ }, \xlongequal{ }`

Spacing `!` (negative thin), `\,` (thin), `\:` (medium), `\;` (thick), `\` (space), `\quad, \qquad`

Smash `\smash{x}` (zero height and depth), `\smash[t]{x}` (zero height), `\smash[b]{x}` (zero depth) for fine vertical spacing control

Modular arithmetic `\pmod{n}` for "(mod n)", `\bmod` as binary operator, `\pod{n}` for " $\textstyle\bigl(n \bigr)$ "

Rule `\rule{width}{height}` for custom-sized horizontal rules

Miscellaneous symbols `\infty`, `\partial`, `\nabla`, `\forall`, `\exists`, `\nexists`, `\emptyset`, `\varnothing`, `\aleph`, `\hbar`, `\ell`, `\wp`, `\Re`, `\Im`, `\angle`, `\triangle`, `\prime`, `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\neg`, `\lnot`, `\bot`, `\top`, `\dagger`, `\ddagger`, `\diamond`, `\square`, `\checkmark`, `\imath`, `\jmath`, `\complement`, `\eth`, `\hslash`, `\mho`, `\dotsb`, `\dotsc`, `\dotsi`, `\dotsm`, `\dotso`

Value

A list with four element: **shape** contains the position of each glyph, relative to the origin in the enclosing textbox. **metrics** contain metrics about the full math expression. **lines** contain line segments (fraction bars, overlines, cancel lines, etc.). **boxes** contain position and dimensions of boxes.

shape is a data.frame with the following columns:

glyph Row number within the string

index Glyph index in the font

metric_id Index of the metrics row (same as `string_id`)

string_id Index of the input string this glyph belongs to

x_offset X position relative to the origin

y_offset Y position relative to the baseline (positive is up)

font_path Path to the font file

font_index Index of the font face in the font file

font_size Font size used for this glyph (may differ for scripts)

advance Horizontal advance width of the glyph

color Color for this glyph

metrics is a data.frame with the following columns:

width Total width of the expression

height Height above the baseline

depth Depth below the baseline

lines is a data.frame with the following columns:

x1 X position of start point

y1 Y position of start point

x2 X position of end point

y2 Y position of end point

thickness Line thickness

string_id Index of the input string this line belongs to

color Color for this line

boxes is a data.frame with the following columns:

x X position of the left edge
y Y position of the bottom edge
width Width of the box
height Height of the box
string_id Index of the input string this box belongs to
color Fill color for the box

Examples

```
# Simple superscript
shape_math("x^2")

# Greek letters
shape_math(r"(\alpha + \beta)")

# Fraction
shape_math(r"(\frac{a}{b})")

# Display mode sum (inline = FALSE)
shape_math(r"(\sum_{i=1}^n i)", inline = FALSE)

# Quadratic formula
shape_math(r"(x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a})")

# Matrix
shape_math(r"(\begin{pmatrix} a & b \\ c & d \end{pmatrix})")

# Binomial coefficient
shape_math(r"(\binom{n}{k})")

# Cases environment
shape_math(r"(f(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases})")

# Colored text
shape_math(r"(x = \textcolor{red}{-b})")

# Cancellation
shape_math(r"(\frac{\cancel{2}x}{\cancel{2}} = x)")

# Extensible arrow with text
shape_math(r"(A \xrightarrow{f} B)")
```

shape_text

Calculate glyph positions for strings

Description

Performs advanced text shaping of strings including font fallbacks, bidirectional script support, word wrapping and various character and paragraph level formatting settings.

Usage

```

shape_text(
  strings,
  id = NULL,
  family = "",
  italic = FALSE,
  weight = "normal",
  width = "undefined",
  features = font_feature(),
  size = 12,
  res = 72,
  lineheight = 1,
  align = "auto",
  hjust = 0,
  vjust = 0,
  max_width = NA,
  tracking = 0,
  indent = 0,
  hanging = 0,
  space_before = 0,
  space_after = 0,
  direction = "auto",
  path = NULL,
  index = 0,
  bold = deprecated()
)

```

Arguments

strings	A character vector of strings to shape
id	A vector grouping the strings together. If strings share an id the shaping will continue between strings
family	The name of the font families to match
italic	logical indicating the font slant
weight	The weight to query for, either in numbers (0, 100, 200, 300, 400, 500, 600, 700, 800, or 900) or strings ("undefined", "thin", "ultralight", "light", "normal", "medium", "semibold", "bold", "ultrabold", or "heavy"). NA will be interpreted as "undefined"/0
width	The width to query for either in numbers (0, 1, 2, 3, 4, 5, 6, 7, 8, or 9) or strings ("undefined", "ultracondensed", "extracondensed", "condensed", "semicondensed", "normal", "semiexpanded", "expanded", "extraexpanded", or "ultraexpanded"). NA will be interpreted as "undefined"/0
features	A <code>systemfonts::font_feature()</code> object or a list of them, giving the OpenType font features to set
size	The size in points to use for the font

res	The resolution to use when doing the shaping. Should optimally match the resolution used when rendering the glyphs.
lineheight	A multiplier for the lineheight
align	Within text box alignment, either 'auto', 'left', 'center', 'right', 'justified', 'justified-left', 'justified-right', 'justified-center', or 'distributed'. 'auto' and 'justified' will chose the left or right version depending on the direction of the text.
hjust, vjust	The justification of the textbox surrounding the text
max_width	The requested width of the string in inches. Setting this to something other than NA will turn on word wrapping.
tracking	Tracking of the glyphs (space adjustment) measured in 1/1000 em.
indent	The indent of the first line in a paragraph measured in inches.
hanging	The indent of the remaining lines in a paragraph measured in inches.
space_before, space_after	The spacing above and below a paragraph, measured in points
direction	The overall directional flow of the text. The default ("auto") will guess the direction based on the content of the string. Use "ltr" (left-to-right) and "rtl" (right-to-left) to turn detection of and set it manually.
path, index	path an index of a font file to circumvent lookup based on family and style
bold	logical indicating whether the font weight

Value

A list with two element: **shape** contains the position of each glyph, relative to the origin in the enclosing textbox. **metrics** contain metrics about the full strings.

shape is a data.frame with the following columns:

glyph The placement of the the first character contributing to the glyph within the string

index The index of the glyph in the font file

metric_id The index of the string the glyph is part of (referencing a row in the **metrics** data.frame)

string_id The index of the string the glyph came from (referencing an element in the **strings** input)

x_offset The x offset in pixels from the origin of the textbox

y_offset The y offset in pixels from the origin of the textbox

font_path The path to the font file used during shaping of the glyph

font_index The index of the font used to shape the glyph in the font file

font_size The size of the font used during shaping

advance The advancement amount to the next glyph

ascender The ascend of the font used for the glyph. This does not measure the actual glyph

descender The descend of the font used for the glyph. This does not measure the actual glyph

metrics is a data.frame with the following columns:

string The text the string consist of

width The width of the string

height The height of the string

left_bearing The distance from the left edge of the textbox and the leftmost glyph

right_bearing The distance from the right edge of the textbox and the rightmost glyph

top_bearing The distance from the top edge of the textbox and the topmost glyph

bottom_bearing The distance from the bottom edge of the textbox and the bottommost glyph

left_border The position of the leftmost edge of the textbox related to the origin

top_border The position of the topmost edge of the textbox related to the origin

pen_x The horizontal position of the next glyph after the string

pen_y The vertical position of the next glyph after the string

ltr The global direction of the string. If TRUE then it is left-to-right, otherwise it is right-to-left

Examples

```
string <- "This is a long string\nLook; It spans multiple lines\nand all"

# Shape with default settings
shape_text(string)

# Mix styles within the same string
string <- c(
  "This string will have\na ",
  "very large",
  " text style\nin the middle"
)

shape_text(string, id = c(1, 1, 1), size = c(12, 24, 12))
```

text_width

Calculate the width of a string, ignoring new-lines

Description

This is a very simple alternative to `systemfonts::shape_string()` that simply calculates the width of strings without taking any newline into account. As such it is suitable to calculate the width of words or lines that have already been split by `\n`. Input is recycled to the length of `strings`.

Usage

```
text_width(
  strings,
  family = "",
  italic = FALSE,
  weight = "normal",
  width = "undefined",
  features = font_feature(),
  size = 12,
  res = 72,
  include_bearing = TRUE,
  path = NULL,
  index = 0,
  bold = deprecated()
)
```

Arguments

strings	A character vector of strings
family	The name of the font families to match
italic	logical indicating the font slant
weight	The weight to query for, either in numbers (0, 100, 200, 300, 400, 500, 600, 700, 800, or 900) or strings ("undefined", "thin", "ultralight", "light", "normal", "medium", "semibold", "bold", "ultrabold", or "heavy"). NA will be interpreted as "undefined"/0
width	The width to query for either in numbers (0, 1, 2, 3, 4, 5, 6, 7, 8, or 9) or strings ("undefined", "ultracondensed", "extracondensed", "condensed", "semicondensed", "normal", "semiexpanded", "expanded", "extraexpanded", or "ultraexpanded"). NA will be interpreted as "undefined"/0
features	A <code>systemfonts::font_feature()</code> object or a list of them, giving the OpenType font features to set
size	The size in points to use for the font
res	The resolution to use when doing the shaping. Should optimally match the resolution used when rendering the glyphs.
include_bearing	Logical, should left and right bearing be included in the string width?
path, index	path an index of a font file to circumvent lookup based on family and style
bold	logical indicating whether the font weight

Value

A numeric vector giving the width of the strings in pixels. Use the provided **res** value to convert it into absolute values.

Examples

```
strings <- c('A short string', 'A very very looong string')
text_width(strings)
```

Index

`font_feature()`, [2](#)

`get_font_features`, [2](#)

`grid.math` (*math_grob*), [4](#)

`grid::gpar()`, [5](#)

`grid::grid.draw()`, [4](#)

`grid::grobTree()`, [4](#)

`grid::viewport()`, [5](#)

`lorem_bidi` (*lorem_text*), [3](#)

`lorem_text`, [3](#)

`math_grob`, [4](#)

`plot_math`, [6](#)

`plot_shape`, [7](#)

`shape_math`, [8](#)

`shape_math()`, [6](#), [7](#)

`shape_text`, [12](#)

`shape_text()`, [7](#)

`systemfonts::font_feature()`, [13](#), [16](#)

`systemfonts::shape_string()`, [15](#)

`text_width`, [15](#)