Package 'diffobj'

October 5, 2020

Type Package

Title Diffs for R Objects

Description Generate a colorized diff of two R objects for an intuitive visualization of their differences.

Version 0.3.2

Depends R (>= 3.1.0)

License GPL (>= 2)

LazyData true

URL https://github.com/brodieG/diffobj

BugReports https://github.com/brodieG/diffobj/issues

RoxygenNote 7.1.0

VignetteBuilder knitr

Encoding UTF-8

Suggests knitr, rmarkdown, testthat

Collate 'capt.R' 'options.R' 'pager.R' 'check.R' 'finalizer.R' 'misc.R' 'html.R' 'styles.R' 's4.R' 'core.R' 'diff.R' 'get.R' 'guides.R' 'hunks.R' 'layout.R' 'myerssimple.R' 'rdiff.R' 'rds.R' 'set.R' 'subset.R' 'summmary.R' 'system.R' 'text.R' 'tochar.R' 'trim.R' 'word.R'

Imports crayon (>= 1.3.2), tools, methods, utils, stats

NeedsCompilation yes

Author Brodie Gaslam [aut, cre], Michael B. Allen [ctb, cph] (Original C implementation of Myers Diff Algorithm)

Maintainer Brodie Gaslam <brodie.gaslam@yahoo.com>

Repository CRAN

Date/Publication 2020-10-05 17:40:06 UTC

R topics documented:

diffobj-package *Diffs for R Objects*

Description

Generate a colorized diff of two R objects for an intuitive visualization of their differences. See ‘vignette(package="diffobj", "diffobj")‘ for details.

AlignThreshold-class *Controls How Lines Within a Diff Hunk Are Aligned*

Description

Controls How Lines Within a Diff Hunk Are Aligned

Slots

`threshold` numeric(1L) between 0 and 1, what proportion of words in the lines must match in order to align them. Set to 1 to effectively turn aligning off. Defaults to 0.25.

`min.chars` integer(1L) positive, minimum number of characters that must match across lines in order to align them. This requirement is in addition to `threshold` and helps minimize spurious alignments. Defaults to 3.

`count.alnum.only` logical(1L) modifier for `min.chars`, whether to count alpha numeric characters only. Helps reduce spurious alignment caused by meta character sequences such as “[[1]]” that would otherwise meet the `min.chars` limit

Examples

```
a1 <- AlignThreshold(threshold=0)
a2 <- AlignThreshold(threshold=1)
a3 <- AlignThreshold(threshold=0, min.chars=2)
## Note how "e f g" is aligned
diffChr(c("a b c e", "d e f g"), "D e f g", align=a1, pager="off")
## But now it is not
diffChr(c("a b c e", "d e f g"), "D e f g", align=a2, pager="off")
## "e f" are not enough chars to align
diffChr(c("a b c", "d e f"), "D e f", align=a1, pager="off")
## Override with min.chars, so now they align
diffChr(c("a b c", "d e f"), "D e f", align=a3, pager="off")
```

---

any,Diff-method *Determine if Diff Object Has Differences*

Description

Determine if Diff Object Has Differences

Usage

```
## S4 method for signature 'Diff'
any(x, ..., na.rm = FALSE)
```

Arguments

x	a Diff object
...	unused, for compatibility with generic
na.rm	unused, for compatibility with generic

Value

TRUE if there are differences, FALSE if not, FALSE with warning if there are no differences but objects are not all.equal

Examples

```
any(diffChr(letters, letters))
any(diffChr(letters, letters[-c(1, 5, 8)]))
```

---

as.character,DiffSummary-method
Generate Character Representation of DiffSummary Object

Description

Generate Character Representation of DiffSummary Object

Usage

```
## S4 method for signature 'DiffSummary'
as.character(x, ...)
```

Arguments

x	a DiffSummary object
...	not used, for compatibility with generic

Value

the summary as a character vector intended to be `cat`ed to terminal

Examples

```
as.character(
  summary(diffChr(letters, letters[-c(5, 15)], format="raw", pager="off"))
)
```

`auto_context` *Configure Automatic Context Calculation*

Description

Helper functions to help define parameters for selecting an appropriate `context` value.

Usage

```
auto_context(
  min = getOption("diffobj.context.auto.min"),
  max = getOption("diffobj.context.auto.max")
)
```

Arguments

`min`	integer(1L), positive, set to zero to allow any context
`max`	integer(1L), set to negative to allow any context

Value

S4 object containing configuration parameters, for use as the `context` or parameter value in `diff*` methods

Examples

```
## `pager="off"` for CRAN compliance; you may omit in normal use
diffChr(letters, letters[-13], context=auto_context(0, 3), pager="off")
diffChr(letters, letters[-13], context=auto_context(0, 10), pager="off")
diffChr(
  letters, letters[-13], context=auto_context(0, 10), line.limit=3L,
  pager="off"
)
```

console_lines *Attempt to Compute Console Height in Text Lines*

Description

Returns the value of the LINES system variable if it is reasonable, 48 otherwise.

Usage

```
console_lines()
```

Value

integer(1L)

Examples

```
console_lines()
```

Diff-class *Diff Result Object*

Description

Return value for the diff* methods. Has show, as.character, summmary, [, head, tail, and any methods.

diffChr *Diff Character Vectors Element By Element*

Description

Will perform the diff on the actual string values of the character vectors instead of capturing the printed screen output. Each vector element is treated as a line of text. NA elements are treated as the string "NA". Non character inputs are coerced to character and attributes are dropped with c.

Usage

```
diffChr(target, current, ...)

## S4 method for signature 'ANY'
diffChr(
  target,
  current,
  mode = gdo("mode"),
  context = gdo("context"),
  format = gdo("format"),
  brightness = gdo("brightness"),
  color.mode = gdo("color.mode"),
  word.diff = gdo("word.diff"),
  pager = gdo("pager"),
  guides = gdo("guides"),
  trim = gdo("trim"),
  rds = gdo("rds"),
  unwrap.atomic = gdo("unwrap.atomic"),
  max.diffs = gdo("max.diffs"),
  disp.width = gdo("disp.width"),
  ignore.white.space = gdo("ignore.white.space"),
  convert.hz.white.space = gdo("convert.hz.white.space"),
  tab.stops = gdo("tab.stops"),
  line.limit = gdo("line.limit"),
  hunk.limit = gdo("hunk.limit"),
  align = gdo("align"),
  style = gdo("style"),
  palette.of.styles = gdo("palette"),
  frame = par_frame(),
  interactive = gdo("interactive"),
  term.colors = gdo("term.colors"),
  tar.banner = NULL,
  cur.banner = NULL,
  strip.sgr = gdo("strip.sgr"),
  sgr.supported = gdo("sgr.supported"),
  extra = list()
)
```

Arguments

`target`	the reference object
`current`	the object being compared to `target`
`...`	unused, for compatibility of methods with generics
`mode`	character(1L), one of: • "unified": diff mode used by `git diff` • "sidebyside": line up the differences side by side

- "context": show the target and current hunks in their entirety; this mode takes up a lot of screen space but makes it easier to see what the objects actually look like
- "auto": default mode; pick one of the above, will favor "sidebyside" unless `getOption("width")` is less than 80, or in `diffPrint` and objects are dimensioned and do not fit side by side, or in `diffChr`, `diffDeparse`, `diffFile` and output does not fit in side by side without wrapping

`context` integer(1L) how many lines of context are shown on either side of differences (defaults to 2). Set to `-1L` to allow as many as there are. Set to "auto" to display as many as 10 lines or as few as 1 depending on whether total screen lines fit within the number of lines specified in `line.limit`. Alternatively pass the return value of `auto_context` to fine tune the parameters of the auto context calculation.

`format` character(1L), controls the diff output format, one of:

- "auto": to select output format based on terminal capabilities; will attempt to use one of the ANSI formats if they appear to be supported, and if not or if you are in the Rstudio console it will attempt to use HTML and browser output if in interactive mode.
- "raw": plain text
- "ansi8": color and format diffs using basic ANSI escape sequences
- "ansi256": like "ansi8", except using the full range of ANSI formatting options
- "html": color and format using HTML markup; the resulting string is processed with `enc2utf8` when output as a full web page (see docs for `html.output` under `Style`).

Defaults to "auto". See `palette.of.styles` for details on customization, `style` for full control of output format. See 'pager' parameter for more discussion of Rstudio behavior.

`brightness` character, one of "light", "dark", "neutral", useful for adjusting color scheme to light or dark terminals. "neutral" by default. See `PaletteOfStyles` for details and limitations. Advanced: you may specify brightness as a function of `format`. For example, if you typically wish to use a "dark" color scheme, except for when in "html" format when you prefer the "light" scheme, you may use `c("dark",html="light")` as the value for this parameter. This is particularly useful if `format` is set to "auto" or if you want to specify a default value for this parameter via options. Any names you use should correspond to a `format`. You must have one unnamed value which will be used as the default for all `formats` that are not explicitly specified.

`color.mode` character, one of "rgb" or "yb". Defaults to "yb". "yb" stands for "Yellow-Blue" for color schemes that rely primarily on those colors to style diffs. Those colors can be easily distinguished by individuals with limited red-green color sensitivity. See `PaletteOfStyles` for details and limitations. Also offers the same advanced usage as the `brightness` parameter.

`word.diff` TRUE (default) or FALSE, whether to run a secondary word diff on the in-hunk differences. For atomic vectors setting this to FALSE could make the diff *slower* (see the `unwrap.atomic` parameter). For other uses, particularly with `diffChr` setting this to FALSE can substantially improve performance.

pager — one of "auto" (default), "on", "off", a Pager object, or a list; controls whether and how a pager is used to display the diff output. If you require a particular pager behavior you must use a Pager object, or "off" to turn off the pager. All other settings will interact with other parameters such as format, style, as well as with your system capabilities in order to select the pager expected to be most useful.

"auto" and "on" are the same, except that in non-interactive mode "auto" is equivalent to "off". "off" will always send output to the console. If "on", whether the output actually gets routed to the pager depends on the pager threshold setting (see Pager). The default behavior is to use the pager associated with the Style object. The Style object is itself is determined by the format or style parameters.

Depending on your system configuration different styles and corresponding pagers will get selected, unless you specify a Pager object directly. On a system with a system pager that supports ANSI CSI SGR colors, the pager will only trigger if the output is taller than one window. If the system pager is not known to support ANSI colors then the output will be sent as HTML to the IDE viewer if available or to the web browser if not. Even though Rstudio now supports ANSI CSI SGR at the console output is still formatted as HTML and sent to the IDE viewer. Partly this is for continuity of behavior, but also because the default Rstudio pager does not support ANSI CSI SGR, at least as of this writing.

If pager is a list, then the same as with "on", except that the Pager object associated with the selected Style object is re-instantiated with the union of the list elements and the existing settings of that Pager. The list should contain named elements that correspond to the Pager instantiation parameters. The names must be specified in full as partial parameter matching will not be carried out because the pager is re-instantiated with new.

See Pager, Style, and PaletteOfStyles for more details and for instructions on how to modify the default behavior.

guides — TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Guides are additional context lines that are not strictly part of a hunk, but provide important contextual data (e.g. column headers). If TRUE, the context lines are shown in addition to the normal diff output, typically in a different color to indicate they are not part of the hunk. If a function, the function should accept as the first argument the object being diffed, and the second the character representation of the object. The function should return the indices of the elements of the character representation that should be treated as guides. See guides for more details.

trim — TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Function should compute for each line in captured output what portion of those lines should be diffed. By default, this is used to remove row meta data differences (e.g. [1,]) so they alone do not show up as differences in the diff. See trim for more details.

rds — TRUE (default) or FALSE, if TRUE will check whether target and/or current point to a file that can be read with readRDS and if so, loads the R object contained in the file and carries out the diff on the object instead of the original argument. Currently there is no mechanism for specifying additional arguments to readRDS

`unwrap.atomic` TRUE (default) or FALSE. Relevant primarily for `diffPrint`, if TRUE, and `word.diff` is also TRUE, and both `target` and `current` are *unnamed* one-dimension atomics , the vectors are unwrapped and diffed element by element, and then re-wrapped. Since `diffPrint` is fundamentally a line diff, the re-wrapped lines are lined up in a manner that is as consistent as possible with the unwrapped diff. Lines that contain the location of the word differences will be paired up. Since the vectors may well be wrapped with different periodicities this will result in lines that are paired up that look like they should not be paired up, though the locations of the differences should be. If is entirely possible that setting this parameter to FALSE will result in a slower diff. This happens if two vectors are actually fairly similar, but their line representations are not. For example, in comparing `1:100` to `c(100,1:99)`, there is really only one difference at the "word" level, but every screen line is different. `diffChr` will also do the unwrapping if it is given a character vector that contains output that looks like the atomic vectors described above. This is a bug, but as the functionality could be useful when diffing e.g. `capture.output` data, we now declare it a feature.

`max.diffs` integer(1L), number of *differences* after which we abandon the `O(n^2)` diff algorithm in favor of a naive element by element comparison. Set to `-1L` to always stick to the original algorithm (defaults to 50000L).

`disp.width` integer(1L) number of display columns to take up; note that in "sidebyside" `mode` the effective display width is half this number (set to 0L to use default widths which are `getOption("width")` for normal styles and `80L` for HTML styles. Future versions of `diffobj` may change this to larger values for two dimensional objects for better diffs (see details).

`ignore.white.space`
TRUE or FALSE, whether to consider differences in horizontal whitespace (i.e. spaces and tabs) as differences (defaults to TRUE).

`convert.hz.white.space`
TRUE or FALSE, whether modify input strings that contain tabs and carriage returns in such a way that they display as they would **with** those characters, but without using those characters (defaults to TRUE). The conversion assumes that tab stops are spaced evenly eight characters apart on the terminal. If this is not the case you may specify the tab stops explicitly with `tab.stops`.

`tab.stops` integer, what tab stops to use when converting hard tabs to spaces. If not integer will be coerced to integer (defaults to 8L). You may specify more than one tab stop. If display width exceeds that addressable by your tab stops the last tab stop will be repeated.

`line.limit` integer(2L) or integer(1L), if length 1 how many lines of output to show, where `-1` means no limit. If length 2, the first value indicates the threshold of screen lines to begin truncating output, and the second the number of lines to truncate to, which should be fewer than the threshold. Note that this parameter is implemented on a best-efforts basis and should not be relied on to produce the exact number of lines requested. In particular do not expect it to work well for for values small enough that the banner portion of the diff would have to be trimmed. If you want a specific number of lines use `[` or `head` / `tail`. One advantage of `line.limit` over these other options is that you can combine it with `context="auto"` and auto `max.level` selection (the latter for `diffStr`), which

allows the diff to dynamically adjust to make best use of the available display lines. [, head, and tail just subset the text of the output.

hunk.limit — integer(2L) or integer (1L), how many diff hunks to show. Behaves similarly to line.limit. How many hunks are in a particular diff is a function of how many differences, and also how much context is used since context can cause two hunks to bleed into each other and become one.

align — numeric(1L) between 0 and 1, proportion of words in a line of target that must be matched in a line of current in the same hunk for those lines to be paired up when displayed (defaults to 0.25), or an AlignThreshold object. Set to 1 to turn off alignment which will cause all lines in a hunk from target to show up first, followed by all lines from current. Note that in order to be aligned lines must meet the threshold and have at least 3 matching alphanumeric characters (see AlignThreshold for details).

style — "auto", a Style object, or a list. "auto" by default. If a Style object, will override the the format, brightness, and color.mode parameters. The Style object provides full control of diff output styling. If a list, then the same as "auto", except that if the auto-selected Style requires instantiation (see PaletteOfStyles), then the list contents will be used as arguments when instantiating the style object. See Style for more details, in particular the examples.

palette.of.styles — PaletteOfStyles object; advanced usage, contains all the Style objects or "classRepresentation" objects extending Style that are selected by specifying the format, brightness, and color.mode parameters. See PaletteOfStyles for more details.

frame — an environment to use as the evaluation frame for the print/show/str, calls and for diffObj, the evaluation frame for the diffPrint / diffStr calls. Defaults to the return value of par_frame.

interactive — TRUE or FALSE whether the function is being run in interactive mode, defaults to the return value of interactive. If in interactive mode, pager will be used if pager is "auto", and if ANSI styles are not supported and style is "auto", output will be send to viewer/browser as HTML.

term.colors — integer(1L) how many ANSI colors are supported by the terminal. This variable is provided for when crayon::num_colors does not properly detect how many ANSI colors are supported by your terminal. Defaults to return value of crayon::num_colors and should be 8 or 256 to allow ANSI colors, or any other number to disallow them. This only impacts output format selection when style and format are both set to "auto".

tar.banner — character(1L), language, or NULL, used to generate the text to display ahead of the diff section representing the target output. If NULL will use the deparsed target expression, if language, will use the language as it would the target expression, if character(1L), will use the string with no modifications. The language mode is provided because diffStr modifies the expression prior to display (e.g. by wrapping it in a call to str). Note that it is possible in some cases that the substituted value of target actually is character(1L), but if you provide a character(1L) value here it will be assumed you intend to use that value literally.

cur.banner	character(1L) like tar.banner, but for current
strip.sgr	TRUE, FALSE, or NULL (default), whether to strip ANSI CSI SGR sequences prior to comparison and for display of diff. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and FALSE otherwise. The default behavior is to avoid confusing diffs where the original SGR and the SGR added by the diff are mixed together.
sgr.supported	TRUE, FALSE, or NULL (default), whether to assume the standard output device supports ANSI CSI SGR sequences. If TRUE, strings will be manipulated accounting for the SGR sequences. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and to 'crayon::has_color()' otherwise. This only controls how the strings are manipulated, not whether SGR is added to format the diff, which is controlled by the 'style' parameter. This parameter is exposed for the rare cases where you might wish to control string manipulation behavior directly.
extra	list additional arguments to pass on to the functions used to create text representation of the objects to diff (e.g. print, str, etc.)

Value

a Diff object; see diffPrint.

See Also

diffPrint for details on the diff* functions, diffObj, diffStr, diffDeparse to compare deparsed objects, ses for a minimal and fast diff

Examples

```
## `pager="off"` for CRAN compliance; you may omit in normal use
diffChr(LETTERS[1:5], LETTERS[2:6], pager="off")
```

---

diffCsv *Diff CSV Files*

Description

Reads CSV files with read.csv and passes the resulting data frames onto diffPrint. extra values are passed as arguments are passed to both read.csv and print. To the extent you wish to use different extra arguments for each of those functions you will need to read.csv the files and pass them to diffPrint yourself.

Usage

```
diffCsv(target, current, ...)

## S4 method for signature 'ANY'
diffCsv(
  target,
  current,
  mode = gdo("mode"),
  context = gdo("context"),
  format = gdo("format"),
  brightness = gdo("brightness"),
  color.mode = gdo("color.mode"),
  word.diff = gdo("word.diff"),
  pager = gdo("pager"),
  guides = gdo("guides"),
  trim = gdo("trim"),
  rds = gdo("rds"),
  unwrap.atomic = gdo("unwrap.atomic"),
  max.diffs = gdo("max.diffs"),
  disp.width = gdo("disp.width"),
  ignore.white.space = gdo("ignore.white.space"),
  convert.hz.white.space = gdo("convert.hz.white.space"),
  tab.stops = gdo("tab.stops"),
  line.limit = gdo("line.limit"),
  hunk.limit = gdo("hunk.limit"),
  align = gdo("align"),
  style = gdo("style"),
  palette.of.styles = gdo("palette"),
  frame = par_frame(),
  interactive = gdo("interactive"),
  term.colors = gdo("term.colors"),
  tar.banner = NULL,
  cur.banner = NULL,
  strip.sgr = gdo("strip.sgr"),
  sgr.supported = gdo("sgr.supported"),
  extra = list()
)
```

Arguments

`target`	character(1L) or file connection with read capability; if character should point to a CSV file
`current`	like `target`
`...`	unused, for compatibility of methods with generics
`mode`	character(1L), one of: • "unified": diff mode used by `git diff` • "sidebyside": line up the differences side by side

- "context": show the target and current hunks in their entirety; this mode takes up a lot of screen space but makes it easier to see what the objects actually look like
- "auto": default mode; pick one of the above, will favor "sidebyside" unless `getOption("width")` is less than 80, or in `diffPrint` and objects are dimensioned and do not fit side by side, or in `diffChr`, `diffDeparse`, `diffFile` and output does not fit in side by side without wrapping

`context` integer(1L) how many lines of context are shown on either side of differences (defaults to 2). Set to `-1L` to allow as many as there are. Set to "auto" to display as many as 10 lines or as few as 1 depending on whether total screen lines fit within the number of lines specified in `line.limit`. Alternatively pass the return value of `auto_context` to fine tune the parameters of the auto context calculation.

`format` character(1L), controls the diff output format, one of:

- "auto": to select output format based on terminal capabilities; will attempt to use one of the ANSI formats if they appear to be supported, and if not or if you are in the Rstudio console it will attempt to use HTML and browser output if in interactive mode.
- "raw": plain text
- "ansi8": color and format diffs using basic ANSI escape sequences
- "ansi256": like "ansi8", except using the full range of ANSI formatting options
- "html": color and format using HTML markup; the resulting string is processed with `enc2utf8` when output as a full web page (see docs for `html.output` under `Style`).

Defaults to "auto". See `palette.of.styles` for details on customization, `style` for full control of output format. See 'pager' parameter for more discussion of Rstudio behavior.

`brightness` character, one of "light", "dark", "neutral", useful for adjusting color scheme to light or dark terminals. "neutral" by default. See `PaletteOfStyles` for details and limitations. Advanced: you may specify brightness as a function of `format`. For example, if you typically wish to use a "dark" color scheme, except for when in "html" format when you prefer the "light" scheme, you may use `c("dark",html="light")` as the value for this parameter. This is particularly useful if `format` is set to "auto" or if you want to specify a default value for this parameter via options. Any names you use should correspond to a `format`. You must have one unnamed value which will be used as the default for all `formats` that are not explicitly specified.

`color.mode` character, one of "rgb" or "yb". Defaults to "yb". "yb" stands for "Yellow-Blue" for color schemes that rely primarily on those colors to style diffs. Those colors can be easily distinguished by individuals with limited red-green color sensitivity. See `PaletteOfStyles` for details and limitations. Also offers the same advanced usage as the `brightness` parameter.

`word.diff` TRUE (default) or FALSE, whether to run a secondary word diff on the in-hunk differences. For atomic vectors setting this to FALSE could make the diff *slower* (see the `unwrap.atomic` parameter). For other uses, particularly with `diffChr` setting this to FALSE can substantially improve performance.

pager — one of "auto" (default), "on", "off", a Pager object, or a list; controls whether and how a pager is used to display the diff output. If you require a particular pager behavior you must use a Pager object, or "off" to turn off the pager. All other settings will interact with other parameters such as format, style, as well as with your system capabilities in order to select the pager expected to be most useful.

"auto" and "on" are the same, except that in non-interactive mode "auto" is equivalent to "off". "off" will always send output to the console. If "on", whether the output actually gets routed to the pager depends on the pager threshold setting (see Pager). The default behavior is to use the pager associated with the Style object. The Style object is itself is determined by the format or style parameters.

Depending on your system configuration different styles and corresponding pagers will get selected, unless you specify a Pager object directly. On a system with a system pager that supports ANSI CSI SGR colors, the pager will only trigger if the output is taller than one window. If the system pager is not known to support ANSI colors then the output will be sent as HTML to the IDE viewer if available or to the web browser if not. Even though Rstudio now supports ANSI CSI SGR at the console output is still formatted as HTML and sent to the IDE viewer. Partly this is for continuity of behavior, but also because the default Rstudio pager does not support ANSI CSI SGR, at least as of this writing.

If pager is a list, then the same as with "on", except that the Pager object associated with the selected Style object is re-instantiated with the union of the list elements and the existing settings of that Pager. The list should contain named elements that correspond to the Pager instantiation parameters. The names must be specified in full as partial parameter matching will not be carried out because the pager is re-instantiated with new.

See Pager, Style, and PaletteOfStyles for more details and for instructions on how to modify the default behavior.

guides — TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Guides are additional context lines that are not strictly part of a hunk, but provide important contextual data (e.g. column headers). If TRUE, the context lines are shown in addition to the normal diff output, typically in a different color to indicate they are not part of the hunk. If a function, the function should accept as the first argument the object being diffed, and the second the character representation of the object. The function should return the indices of the elements of the character representation that should be treated as guides. See guides for more details.

trim — TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Function should compute for each line in captured output what portion of those lines should be diffed. By default, this is used to remove row meta data differences (e.g. [1,]) so they alone do not show up as differences in the diff. See trim for more details.

rds — TRUE (default) or FALSE, if TRUE will check whether target and/or current point to a file that can be read with readRDS and if so, loads the R object contained in the file and carries out the diff on the object instead of the original argument. Currently there is no mechanism for specifying additional arguments to readRDS

`unwrap.atomic` TRUE (default) or FALSE. Relevant primarily for `diffPrint`, if TRUE, and `word.diff` is also TRUE, and both `target` and `current` are *unnamed* one-dimension atomics , the vectors are unwrapped and diffed element by element, and then re-wrapped. Since `diffPrint` is fundamentally a line diff, the re-wrapped lines are lined up in a manner that is as consistent as possible with the unwrapped diff. Lines that contain the location of the word differences will be paired up. Since the vectors may well be wrapped with different periodicities this will result in lines that are paired up that look like they should not be paired up, though the locations of the differences should be. If is entirely possible that setting this parameter to FALSE will result in a slower diff. This happens if two vectors are actually fairly similar, but their line representations are not. For example, in comparing `1:100` to `c(100,1:99)`, there is really only one difference at the "word" level, but every screen line is different. `diffChr` will also do the unwrapping if it is given a character vector that contains output that looks like the atomic vectors described above. This is a bug, but as the functionality could be useful when diffing e.g. `capture.output` data, we now declare it a feature.

`max.diffs` integer(1L), number of *differences* after which we abandon the `O(n^2)` diff algorithm in favor of a naive element by element comparison. Set to `-1L` to always stick to the original algorithm (defaults to 50000L).

`disp.width` integer(1L) number of display columns to take up; note that in "sidebyside" `mode` the effective display width is half this number (set to 0L to use default widths which are `getOption("width")` for normal styles and `80L` for HTML styles. Future versions of `diffobj` may change this to larger values for two dimensional objects for better diffs (see details).

`ignore.white.space`
TRUE or FALSE, whether to consider differences in horizontal whitespace (i.e. spaces and tabs) as differences (defaults to TRUE).

`convert.hz.white.space`
TRUE or FALSE, whether modify input strings that contain tabs and carriage returns in such a way that they display as they would **with** those characters, but without using those characters (defaults to TRUE). The conversion assumes that tab stops are spaced evenly eight characters apart on the terminal. If this is not the case you may specify the tab stops explicitly with `tab.stops`.

`tab.stops` integer, what tab stops to use when converting hard tabs to spaces. If not integer will be coerced to integer (defaults to 8L). You may specify more than one tab stop. If display width exceeds that addressable by your tab stops the last tab stop will be repeated.

`line.limit` integer(2L) or integer(1L), if length 1 how many lines of output to show, where `-1` means no limit. If length 2, the first value indicates the threshold of screen lines to begin truncating output, and the second the number of lines to truncate to, which should be fewer than the threshold. Note that this parameter is implemented on a best-efforts basis and should not be relied on to produce the exact number of lines requested. In particular do not expect it to work well for for values small enough that the banner portion of the diff would have to be trimmed. If you want a specific number of lines use `[` or `head` / `tail`. One advantage of `line.limit` over these other options is that you can combine it with `context="auto"` and auto `max.level` selection (the latter for `diffStr`), which

allows the diff to dynamically adjust to make best use of the available display lines. [, head, and tail just subset the text of the output.

hunk.limit — integer(2L) or integer (1L), how many diff hunks to show. Behaves similarly to line.limit. How many hunks are in a particular diff is a function of how many differences, and also how much context is used since context can cause two hunks to bleed into each other and become one.

align — numeric(1L) between 0 and 1, proportion of words in a line of target that must be matched in a line of current in the same hunk for those lines to be paired up when displayed (defaults to 0.25), or an AlignThreshold object. Set to 1 to turn off alignment which will cause all lines in a hunk from target to show up first, followed by all lines from current. Note that in order to be aligned lines must meet the threshold and have at least 3 matching alphanumeric characters (see AlignThreshold for details).

style — "auto", a Style object, or a list. "auto" by default. If a Style object, will override the the format, brightness, and color.mode parameters. The Style object provides full control of diff output styling. If a list, then the same as "auto", except that if the auto-selected Style requires instantiation (see PaletteOfStyles), then the list contents will be used as arguments when instantiating the style object. See Style for more details, in particular the examples.

palette.of.styles — PaletteOfStyles object; advanced usage, contains all the Style objects or "classRepresentation" objects extending Style that are selected by specifying the format, brightness, and color.mode parameters. See PaletteOfStyles for more details.

frame — an environment to use as the evaluation frame for the print/show/str, calls and for diffObj, the evaluation frame for the diffPrint / diffStr calls. Defaults to the return value of par_frame.

interactive — TRUE or FALSE whether the function is being run in interactive mode, defaults to the return value of interactive. If in interactive mode, pager will be used if pager is "auto", and if ANSI styles are not supported and style is "auto", output will be send to viewer/browser as HTML.

term.colors — integer(1L) how many ANSI colors are supported by the terminal. This variable is provided for when crayon::num_colors does not properly detect how many ANSI colors are supported by your terminal. Defaults to return value of crayon::num_colors and should be 8 or 256 to allow ANSI colors, or any other number to disallow them. This only impacts output format selection when style and format are both set to "auto".

tar.banner — character(1L), language, or NULL, used to generate the text to display ahead of the diff section representing the target output. If NULL will use the deparsed target expression, if language, will use the language as it would the target expression, if character(1L), will use the string with no modifications. The language mode is provided because diffStr modifies the expression prior to display (e.g. by wrapping it in a call to str). Note that it is possible in some cases that the substituted value of target actually is character(1L), but if you provide a character(1L) value here it will be assumed you intend to use that value literally.

cur.banner	character(1L) like tar.banner, but for current
strip.sgr	TRUE, FALSE, or NULL (default), whether to strip ANSI CSI SGR sequences prior to comparison and for display of diff. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and FALSE otherwise. The default behavior is to avoid confusing diffs where the original SGR and the SGR added by the diff are mixed together.
sgr.supported	TRUE, FALSE, or NULL (default), whether to assume the standard output device supports ANSI CSI SGR sequences. If TRUE, strings will be manipulated accounting for the SGR sequences. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and to 'crayon::has_color()' otherwise. This only controls how the strings are manipulated, not whether SGR is added to format the diff, which is controlled by the 'style' parameter. This parameter is exposed for the rare cases where you might wish to control string manipulation behavior directly.
extra	list additional arguments to pass on to the functions used to create text representation of the objects to diff (e.g. print, str, etc.)

Value

a Diff object; see diffPrint.

See Also

diffPrint for details on the diff* functions, diffObj, diffStr, diffChr to compare character vectors directly, ses for a minimal and fast diff

Examples

```
iris.2 <- iris
iris.2$Sepal.Length[5] <- 99
f1 <- tempfile()
f2 <- tempfile()
write.csv(iris, f1, row.names=FALSE)
write.csv(iris.2, f2, row.names=FALSE)
## `pager="off"` for CRAN compliance; you may omit in normal use
diffCsv(f1, f2, pager="off")
unlink(c(f1, f2))
```

---

diffDeparse *Diff Deparsed Objects*

Description

Perform diff on the character vectors produced by deparseing the objects. Each element counts as a line. If an element contains newlines it will be split into elements new lines by the newlines.

Usage

```
diffDeparse(target, current, ...)

## S4 method for signature 'ANY'
diffDeparse(
  target,
  current,
  mode = gdo("mode"),
  context = gdo("context"),
  format = gdo("format"),
  brightness = gdo("brightness"),
  color.mode = gdo("color.mode"),
  word.diff = gdo("word.diff"),
  pager = gdo("pager"),
  guides = gdo("guides"),
  trim = gdo("trim"),
  rds = gdo("rds"),
  unwrap.atomic = gdo("unwrap.atomic"),
  max.diffs = gdo("max.diffs"),
  disp.width = gdo("disp.width"),
  ignore.white.space = gdo("ignore.white.space"),
  convert.hz.white.space = gdo("convert.hz.white.space"),
  tab.stops = gdo("tab.stops"),
  line.limit = gdo("line.limit"),
  hunk.limit = gdo("hunk.limit"),
  align = gdo("align"),
  style = gdo("style"),
  palette.of.styles = gdo("palette"),
  frame = par_frame(),
  interactive = gdo("interactive"),
  term.colors = gdo("term.colors"),
  tar.banner = NULL,
  cur.banner = NULL,
  strip.sgr = gdo("strip.sgr"),
  sgr.supported = gdo("sgr.supported"),
  extra = list()
)
```

Arguments

`target`	the reference object
`current`	the object being compared to `target`
`...`	unused, for compatibility of methods with generics
`mode`	character(1L), one of: • "unified": diff mode used by `git diff` • "sidebyside": line up the differences side by side

- "context": show the target and current hunks in their entirety; this mode takes up a lot of screen space but makes it easier to see what the objects actually look like
- "auto": default mode; pick one of the above, will favor "sidebyside" unless `getOption("width")` is less than 80, or in `diffPrint` and objects are dimensioned and do not fit side by side, or in `diffChr`, `diffDeparse`, `diffFile` and output does not fit in side by side without wrapping

`context` integer(1L) how many lines of context are shown on either side of differences (defaults to 2). Set to `-1L` to allow as many as there are. Set to "auto" to display as many as 10 lines or as few as 1 depending on whether total screen lines fit within the number of lines specified in `line.limit`. Alternatively pass the return value of `auto_context` to fine tune the parameters of the auto context calculation.

`format` character(1L), controls the diff output format, one of:

- "auto": to select output format based on terminal capabilities; will attempt to use one of the ANSI formats if they appear to be supported, and if not or if you are in the Rstudio console it will attempt to use HTML and browser output if in interactive mode.
- "raw": plain text
- "ansi8": color and format diffs using basic ANSI escape sequences
- "ansi256": like "ansi8", except using the full range of ANSI formatting options
- "html": color and format using HTML markup; the resulting string is processed with `enc2utf8` when output as a full web page (see docs for `html.output` under `Style`).

Defaults to "auto". See `palette.of.styles` for details on customization, `style` for full control of output format. See 'pager' parameter for more discussion of Rstudio behavior.

`brightness` character, one of "light", "dark", "neutral", useful for adjusting color scheme to light or dark terminals. "neutral" by default. See `PaletteOfStyles` for details and limitations. Advanced: you may specify brightness as a function of `format`. For example, if you typically wish to use a "dark" color scheme, except for when in "html" format when you prefer the "light" scheme, you may use `c("dark",html="light")` as the value for this parameter. This is particularly useful if `format` is set to "auto" or if you want to specify a default value for this parameter via options. Any names you use should correspond to a `format`. You must have one unnamed value which will be used as the default for all `formats` that are not explicitly specified.

`color.mode` character, one of "rgb" or "yb". Defaults to "yb". "yb" stands for "Yellow-Blue" for color schemes that rely primarily on those colors to style diffs. Those colors can be easily distinguished by individuals with limited red-green color sensitivity. See `PaletteOfStyles` for details and limitations. Also offers the same advanced usage as the `brightness` parameter.

`word.diff` TRUE (default) or FALSE, whether to run a secondary word diff on the in-hunk differences. For atomic vectors setting this to FALSE could make the diff *slower* (see the `unwrap.atomic` parameter). For other uses, particularly with `diffChr` setting this to FALSE can substantially improve performance.

pager — one of "auto" (default), "on", "off", a Pager object, or a list; controls whether and how a pager is used to display the diff output. If you require a particular pager behavior you must use a Pager object, or "off" to turn off the pager. All other settings will interact with other parameters such as format, style, as well as with your system capabilities in order to select the pager expected to be most useful.

"auto" and "on" are the same, except that in non-interactive mode "auto" is equivalent to "off". "off" will always send output to the console. If "on", whether the output actually gets routed to the pager depends on the pager threshold setting (see Pager). The default behavior is to use the pager associated with the Style object. The Style object is itself is determined by the format or style parameters.

Depending on your system configuration different styles and corresponding pagers will get selected, unless you specify a Pager object directly. On a system with a system pager that supports ANSI CSI SGR colors, the pager will only trigger if the output is taller than one window. If the system pager is not known to support ANSI colors then the output will be sent as HTML to the IDE viewer if available or to the web browser if not. Even though Rstudio now supports ANSI CSI SGR at the console output is still formatted as HTML and sent to the IDE viewer. Partly this is for continuity of behavior, but also because the default Rstudio pager does not support ANSI CSI SGR, at least as of this writing.

If pager is a list, then the same as with "on", except that the Pager object associated with the selected Style object is re-instantiated with the union of the list elements and the existing settings of that Pager. The list should contain named elements that correspond to the Pager instantiation parameters. The names must be specified in full as partial parameter matching will not be carried out because the pager is re-instantiated with new.

See Pager, Style, and PaletteOfStyles for more details and for instructions on how to modify the default behavior.

guides — TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Guides are additional context lines that are not strictly part of a hunk, but provide important contextual data (e.g. column headers). If TRUE, the context lines are shown in addition to the normal diff output, typically in a different color to indicate they are not part of the hunk. If a function, the function should accept as the first argument the object being diffed, and the second the character representation of the object. The function should return the indices of the elements of the character representation that should be treated as guides. See guides for more details.

trim — TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Function should compute for each line in captured output what portion of those lines should be diffed. By default, this is used to remove row meta data differences (e.g. [1,]) so they alone do not show up as differences in the diff. See trim for more details.

rds — TRUE (default) or FALSE, if TRUE will check whether target and/or current point to a file that can be read with readRDS and if so, loads the R object contained in the file and carries out the diff on the object instead of the original argument. Currently there is no mechanism for specifying additional arguments to readRDS

`unwrap.atomic` TRUE (default) or FALSE. Relevant primarily for `diffPrint`, if TRUE, and `word.diff` is also TRUE, and both `target` and `current` are *unnamed* one-dimension atomics , the vectors are unwrapped and diffed element by element, and then re-wrapped. Since `diffPrint` is fundamentally a line diff, the re-wrapped lines are lined up in a manner that is as consistent as possible with the unwrapped diff. Lines that contain the location of the word differences will be paired up. Since the vectors may well be wrapped with different periodicities this will result in lines that are paired up that look like they should not be paired up, though the locations of the differences should be. If is entirely possible that setting this parameter to FALSE will result in a slower diff. This happens if two vectors are actually fairly similar, but their line representations are not. For example, in comparing `1:100` to `c(100,1:99)`, there is really only one difference at the "word" level, but every screen line is different. `diffChr` will also do the unwrapping if it is given a character vector that contains output that looks like the atomic vectors described above. This is a bug, but as the functionality could be useful when diffing e.g. `capture.output` data, we now declare it a feature.

`max.diffs` integer(1L), number of *differences* after which we abandon the `O(n^2)` diff algorithm in favor of a naive element by element comparison. Set to `-1L` to always stick to the original algorithm (defaults to 50000L).

`disp.width` integer(1L) number of display columns to take up; note that in "sidebyside" `mode` the effective display width is half this number (set to 0L to use default widths which are `getOption("width")` for normal styles and `80L` for HTML styles. Future versions of `diffobj` may change this to larger values for two dimensional objects for better diffs (see details).

`ignore.white.space`
TRUE or FALSE, whether to consider differences in horizontal whitespace (i.e. spaces and tabs) as differences (defaults to TRUE).

`convert.hz.white.space`
TRUE or FALSE, whether modify input strings that contain tabs and carriage returns in such a way that they display as they would **with** those characters, but without using those characters (defaults to TRUE). The conversion assumes that tab stops are spaced evenly eight characters apart on the terminal. If this is not the case you may specify the tab stops explicitly with `tab.stops`.

`tab.stops` integer, what tab stops to use when converting hard tabs to spaces. If not integer will be coerced to integer (defaults to 8L). You may specify more than one tab stop. If display width exceeds that addressable by your tab stops the last tab stop will be repeated.

`line.limit` integer(2L) or integer(1L), if length 1 how many lines of output to show, where `-1` means no limit. If length 2, the first value indicates the threshold of screen lines to begin truncating output, and the second the number of lines to truncate to, which should be fewer than the threshold. Note that this parameter is implemented on a best-efforts basis and should not be relied on to produce the exact number of lines requested. In particular do not expect it to work well for for values small enough that the banner portion of the diff would have to be trimmed. If you want a specific number of lines use `[` or `head` / `tail`. One advantage of `line.limit` over these other options is that you can combine it with `context="auto"` and auto `max.level` selection (the latter for `diffStr`), which

allows the diff to dynamically adjust to make best use of the available display lines. [, head, and tail just subset the text of the output.

hunk.limit — integer(2L) or integer (1L), how many diff hunks to show. Behaves similarly to line.limit. How many hunks are in a particular diff is a function of how many differences, and also how much context is used since context can cause two hunks to bleed into each other and become one.

align — numeric(1L) between 0 and 1, proportion of words in a line of target that must be matched in a line of current in the same hunk for those lines to be paired up when displayed (defaults to 0.25), or an AlignThreshold object. Set to 1 to turn off alignment which will cause all lines in a hunk from target to show up first, followed by all lines from current. Note that in order to be aligned lines must meet the threshold and have at least 3 matching alphanumeric characters (see AlignThreshold for details).

style — "auto", a Style object, or a list. "auto" by default. If a Style object, will override the the format, brightness, and color.mode parameters. The Style object provides full control of diff output styling. If a list, then the same as "auto", except that if the auto-selected Style requires instantiation (see PaletteOfStyles), then the list contents will be used as arguments when instantiating the style object. See Style for more details, in particular the examples.

palette.of.styles — PaletteOfStyles object; advanced usage, contains all the Style objects or "classRepresentation" objects extending Style that are selected by specifying the format, brightness, and color.mode parameters. See PaletteOfStyles for more details.

frame — an environment to use as the evaluation frame for the print/show/str, calls and for diffObj, the evaluation frame for the diffPrint / diffStr calls. Defaults to the return value of par_frame.

interactive — TRUE or FALSE whether the function is being run in interactive mode, defaults to the return value of interactive. If in interactive mode, pager will be used if pager is "auto", and if ANSI styles are not supported and style is "auto", output will be send to viewer/browser as HTML.

term.colors — integer(1L) how many ANSI colors are supported by the terminal. This variable is provided for when crayon::num_colors does not properly detect how many ANSI colors are supported by your terminal. Defaults to return value of crayon::num_colors and should be 8 or 256 to allow ANSI colors, or any other number to disallow them. This only impacts output format selection when style and format are both set to "auto".

tar.banner — character(1L), language, or NULL, used to generate the text to display ahead of the diff section representing the target output. If NULL will use the deparsed target expression, if language, will use the language as it would the target expression, if character(1L), will use the string with no modifications. The language mode is provided because diffStr modifies the expression prior to display (e.g. by wrapping it in a call to str). Note that it is possible in some cases that the substituted value of target actually is character(1L), but if you provide a character(1L) value here it will be assumed you intend to use that value literally.

`cur.banner`	character(1L) like `tar.banner`, but for `current`
`strip.sgr`	TRUE, FALSE, or NULL (default), whether to strip ANSI CSI SGR sequences prior to comparison and for display of diff. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and FALSE otherwise. The default behavior is to avoid confusing diffs where the original SGR and the SGR added by the diff are mixed together.
`sgr.supported`	TRUE, FALSE, or NULL (default), whether to assume the standard output device supports ANSI CSI SGR sequences. If TRUE, strings will be manipulated accounting for the SGR sequences. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and to 'crayon::has_color()' otherwise. This only controls how the strings are manipulated, not whether SGR is added to format the diff, which is controlled by the 'style' parameter. This parameter is exposed for the rare cases where you might wish to control string manipulation behavior directly.
`extra`	list additional arguments to pass on to the functions used to create text representation of the objects to diff (e.g. `print`, `str`, etc.)

Value

a `Diff` object; see `diffPrint`.

See Also

`diffPrint` for details on the `diff*` functions, `diffObj`, `diffStr`, `diffChr` to compare character vectors directly, `ses` for a minimal and fast diff

Examples

```
## `pager="off"` for CRAN compliance; you may omit in normal use
diffDeparse(matrix(1:9, 3), 1:9, pager="off")
```

`diffFile` *Diff Files*

Description

Reads text files with `readLines` and performs a diff on the resulting character vectors.

Usage

```
diffFile(target, current, ...)

## S4 method for signature 'ANY'
diffFile(
  target,
  current,
  mode = gdo("mode"),
```

```
  context = gdo("context"),
  format = gdo("format"),
  brightness = gdo("brightness"),
  color.mode = gdo("color.mode"),
  word.diff = gdo("word.diff"),
  pager = gdo("pager"),
  guides = gdo("guides"),
  trim = gdo("trim"),
  rds = gdo("rds"),
  unwrap.atomic = gdo("unwrap.atomic"),
  max.diffs = gdo("max.diffs"),
  disp.width = gdo("disp.width"),
  ignore.white.space = gdo("ignore.white.space"),
  convert.hz.white.space = gdo("convert.hz.white.space"),
  tab.stops = gdo("tab.stops"),
  line.limit = gdo("line.limit"),
  hunk.limit = gdo("hunk.limit"),
  align = gdo("align"),
  style = gdo("style"),
  palette.of.styles = gdo("palette"),
  frame = par_frame(),
  interactive = gdo("interactive"),
  term.colors = gdo("term.colors"),
  tar.banner = NULL,
  cur.banner = NULL,
  strip.sgr = gdo("strip.sgr"),
  sgr.supported = gdo("sgr.supported"),
  extra = list()
)
```

Arguments

`target`	character(1L) or file connection with read capability; if character should point to a text file
`current`	like `target`
`...`	unused, for compatibility of methods with generics
`mode`	character(1L), one of: • "unified": diff mode used by `git diff` • "sidebyside": line up the differences side by side • "context": show the target and current hunks in their entirety; this mode takes up a lot of screen space but makes it easier to see what the objects actually look like • "auto": default mode; pick one of the above, will favor "sidebyside" unless `getOption("width")` is less than 80, or in `diffPrint` and objects are dimensioned and do not fit side by side, or in `diffChr`, `diffDeparse`, `diffFile` and output does not fit in side by side without wrapping

`context` integer(1L) how many lines of context are shown on either side of differences (defaults to 2). Set to `-1L` to allow as many as there are. Set to "auto" to display as many as 10 lines or as few as 1 depending on whether total screen lines fit within the number of lines specified in `line.limit`. Alternatively pass the return value of `auto_context` to fine tune the parameters of the auto context calculation.

`format` character(1L), controls the diff output format, one of:

- "auto": to select output format based on terminal capabilities; will attempt to use one of the ANSI formats if they appear to be supported, and if not or if you are in the Rstudio console it will attempt to use HTML and browser output if in interactive mode.
- "raw": plain text
- "ansi8": color and format diffs using basic ANSI escape sequences
- "ansi256": like "ansi8", except using the full range of ANSI formatting options
- "html": color and format using HTML markup; the resulting string is processed with `enc2utf8` when output as a full web page (see docs for `html.output` under `Style`).

Defaults to "auto". See `palette.of.styles` for details on customization, `style` for full control of output format. See 'pager' parameter for more discussion of Rstudio behavior.

`brightness` character, one of "light", "dark", "neutral", useful for adjusting color scheme to light or dark terminals. "neutral" by default. See `PaletteOfStyles` for details and limitations. Advanced: you may specify brightness as a function of `format`. For example, if you typically wish to use a "dark" color scheme, except for when in "html" format when you prefer the "light" scheme, you may use `c("dark",html="light")` as the value for this parameter. This is particularly useful if `format` is set to "auto" or if you want to specify a default value for this parameter via options. Any names you use should correspond to a `format`. You must have one unnamed value which will be used as the default for all `formats` that are not explicitly specified.

`color.mode` character, one of "rgb" or "yb". Defaults to "yb". "yb" stands for "Yellow-Blue" for color schemes that rely primarily on those colors to style diffs. Those colors can be easily distinguished by individuals with limited red-green color sensitivity. See `PaletteOfStyles` for details and limitations. Also offers the same advanced usage as the `brightness` parameter.

`word.diff` TRUE (default) or FALSE, whether to run a secondary word diff on the in-hunk differences. For atomic vectors setting this to FALSE could make the diff *slower* (see the `unwrap.atomic` parameter). For other uses, particularly with `diffChr` setting this to FALSE can substantially improve performance.

`pager` one of "auto" (default), "on", "off", a `Pager` object, or a list; controls whether and how a pager is used to display the diff output. If you require a particular pager behavior you must use a `Pager` object, or "off" to turn off the pager. All other settings will interact with other parameters such as `format`, `style`, as well as with your system capabilities in order to select the pager expected to be most useful.

"auto" and "on" are the same, except that in non-interactive mode "auto" is equivalent to "off". "off" will always send output to the console. If "on", whether the output actually gets routed to the pager depends on the pager `threshold` setting (see `Pager`). The default behavior is to use the pager associated with the `Style` object. The `Style` object is itself is determined by the `format` or `style` parameters.

Depending on your system configuration different styles and corresponding pagers will get selected, unless you specify a `Pager` object directly. On a system with a system pager that supports ANSI CSI SGR colors, the pager will only trigger if the output is taller than one window. If the system pager is not known to support ANSI colors then the output will be sent as HTML to the IDE viewer if available or to the web browser if not. Even though Rstudio now supports ANSI CSI SGR at the console output is still formatted as HTML and sent to the IDE viewer. Partly this is for continuity of behavior, but also because the default Rstudio pager does not support ANSI CSI SGR, at least as of this writing.

If `pager` is a list, then the same as with "on", except that the `Pager` object associated with the selected `Style` object is re-instantiated with the union of the list elements and the existing settings of that `Pager`. The list should contain named elements that correspond to the `Pager` instantiation parameters. The names must be specified in full as partial parameter matching will not be carried out because the pager is re-instantiated with `new`.

See `Pager`, `Style`, and `PaletteOfStyles` for more details and for instructions on how to modify the default behavior.

`guides` TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Guides are additional context lines that are not strictly part of a hunk, but provide important contextual data (e.g. column headers). If TRUE, the context lines are shown in addition to the normal diff output, typically in a different color to indicate they are not part of the hunk. If a function, the function should accept as the first argument the object being diffed, and the second the character representation of the object. The function should return the indices of the elements of the character representation that should be treated as guides. See `guides` for more details.

`trim` TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Function should compute for each line in captured output what portion of those lines should be diffed. By default, this is used to remove row meta data differences (e.g. `[1,]`) so they alone do not show up as differences in the diff. See `trim` for more details.

`rds` TRUE (default) or FALSE, if TRUE will check whether `target` and/or `current` point to a file that can be read with `readRDS` and if so, loads the R object contained in the file and carries out the diff on the object instead of the original argument. Currently there is no mechanism for specifying additional arguments to `readRDS`

`unwrap.atomic` TRUE (default) or FALSE. Relevant primarily for `diffPrint`, if TRUE, and `word.diff` is also TRUE, and both `target` and `current` are *unnamed* one-dimension atomics , the vectors are unwrapped and diffed element by element, and then re-wrapped. Since `diffPrint` is fundamentally a line diff, the re-wrapped lines are lined up in a manner that is as consistent as possible with the

unwrapped diff. Lines that contain the location of the word differences will be paired up. Since the vectors may well be wrapped with different periodicities this will result in lines that are paired up that look like they should not be paired up, though the locations of the differences should be. If is entirely possible that setting this parameter to FALSE will result in a slower diff. This happens if two vectors are actually fairly similar, but their line representations are not. For example, in comparing `1:100` to `c(100,1:99)`, there is really only one difference at the "word" level, but every screen line is different. `diffChr` will also do the unwrapping if it is given a character vector that contains output that looks like the atomic vectors described above. This is a bug, but as the functionality could be useful when diffing e.g. `capture.output` data, we now declare it a feature.

`max.diffs` integer(1L), number of *differences* after which we abandon the `O(n^2)` diff algorithm in favor of a naive element by element comparison. Set to `-1L` to always stick to the original algorithm (defaults to 50000L).

`disp.width` integer(1L) number of display columns to take up; note that in "sidebyside" `mode` the effective display width is half this number (set to 0L to use default widths which are `getOption("width")` for normal styles and `80L` for HTML styles. Future versions of `diffobj` may change this to larger values for two dimensional objects for better diffs (see details).

`ignore.white.space`
TRUE or FALSE, whether to consider differences in horizontal whitespace (i.e. spaces and tabs) as differences (defaults to TRUE).

`convert.hz.white.space`
TRUE or FALSE, whether modify input strings that contain tabs and carriage returns in such a way that they display as they would **with** those characters, but without using those characters (defaults to TRUE). The conversion assumes that tab stops are spaced evenly eight characters apart on the terminal. If this is not the case you may specify the tab stops explicitly with `tab.stops`.

`tab.stops` integer, what tab stops to use when converting hard tabs to spaces. If not integer will be coerced to integer (defaults to 8L). You may specify more than one tab stop. If display width exceeds that addressable by your tab stops the last tab stop will be repeated.

`line.limit` integer(2L) or integer(1L), if length 1 how many lines of output to show, where `-1` means no limit. If length 2, the first value indicates the threshold of screen lines to begin truncating output, and the second the number of lines to truncate to, which should be fewer than the threshold. Note that this parameter is implemented on a best-efforts basis and should not be relied on to produce the exact number of lines requested. In particular do not expect it to work well for for values small enough that the banner portion of the diff would have to be trimmed. If you want a specific number of lines use `[` or `head` / `tail`. One advantage of `line.limit` over these other options is that you can combine it with `context="auto"` and auto `max.level` selection (the latter for `diffStr`), which allows the diff to dynamically adjust to make best use of the available display lines. `[`, `head`, and `tail` just subset the text of the output.

`hunk.limit` integer(2L) or integer (1L), how many diff hunks to show. Behaves similarly to `line.limit`. How many hunks are in a particular diff is a function of how

many differences, and also how much context is used since context can cause two hunks to bleed into each other and become one.

align — numeric(1L) between 0 and 1, proportion of words in a line of target that must be matched in a line of current in the same hunk for those lines to be paired up when displayed (defaults to 0.25), or an AlignThreshold object. Set to 1 to turn off alignment which will cause all lines in a hunk from target to show up first, followed by all lines from current. Note that in order to be aligned lines must meet the threshold and have at least 3 matching alphanumeric characters (see AlignThreshold for details).

style — "auto", a Style object, or a list. "auto" by default. If a Style object, will override the the format, brightness, and color.mode parameters. The Style object provides full control of diff output styling. If a list, then the same as "auto", except that if the auto-selected Style requires instantiation (see PaletteOfStyles), then the list contents will be used as arguments when instantiating the style object. See Style for more details, in particular the examples.

palette.of.styles — PaletteOfStyles object; advanced usage, contains all the Style objects or "classRepresentation" objects extending Style that are selected by specifying the format, brightness, and color.mode parameters. See PaletteOfStyles for more details.

frame — an environment to use as the evaluation frame for the print/show/str, calls and for diffObj, the evaluation frame for the diffPrint / diffStr calls. Defaults to the return value of par_frame.

interactive — TRUE or FALSE whether the function is being run in interactive mode, defaults to the return value of interactive. If in interactive mode, pager will be used if pager is "auto", and if ANSI styles are not supported and style is "auto", output will be send to viewer/browser as HTML.

term.colors — integer(1L) how many ANSI colors are supported by the terminal. This variable is provided for when crayon::num_colors does not properly detect how many ANSI colors are supported by your terminal. Defaults to return value of crayon::num_colors and should be 8 or 256 to allow ANSI colors, or any other number to disallow them. This only impacts output format selection when style and format are both set to "auto".

tar.banner — character(1L), language, or NULL, used to generate the text to display ahead of the diff section representing the target output. If NULL will use the deparsed target expression, if language, will use the language as it would the target expression, if character(1L), will use the string with no modifications. The language mode is provided because diffStr modifies the expression prior to display (e.g. by wrapping it in a call to str). Note that it is possible in some cases that the substituted value of target actually is character(1L), but if you provide a character(1L) value here it will be assumed you intend to use that value literally.

cur.banner — character(1L) like tar.banner, but for current

strip.sgr — TRUE, FALSE, or NULL (default), whether to strip ANSI CSI SGR sequences prior to comparison and for display of diff. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and FALSE otherwise. The default behavior

is to avoid confusing diffs where the original SGR and the SGR added by the diff are mixed together.

`sgr.supported` TRUE, FALSE, or NULL (default), whether to assume the standard output device supports ANSI CSI SGR sequences. If TRUE, strings will be manipulated accounting for the SGR sequences. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and to 'crayon::has_color()' otherwise. This only controls how the strings are manipulated, not whether SGR is added to format the diff, which is controlled by the 'style' parameter. This parameter is exposed for the rare cases where you might wish to control string manipulation behavior directly.

`extra` list additional arguments to pass on to the functions used to create text representation of the objects to diff (e.g. `print`, `str`, etc.)

Value

a `Diff` object; see `diffPrint`.

See Also

`diffPrint` for details on the `diff*` functions, `diffObj`, `diffStr`, `diffChr` to compare character vectors directly, `ses` for a minimal and fast diff

Examples

```
## Not run:
url.base <- "https://raw.githubusercontent.com/wch/r-source"
f1 <- file.path(url.base, "29f013d1570e1df5dc047fb7ee304ff57c99ea68/README")
f2 <- file.path(url.base, "daf0b5f6c728bd3dbcd0a3c976a7be9beee731d9/README")
diffFile(f1, f2)

## End(Not run)
```

`diffObj` *Diff Objects*

Description

Compare either the `printed` or `str` screen representation of R objects depending on which is estimated to produce the most useful diff. The selection process tries to minimize screen lines while maximizing differences shown subject to display constraints. The decision algorithm is likely to evolve over time, so do not rely on this function making a particular selection under specific circumstances. Instead, use `diffPrint` or `diffStr` if you require one or the other output.

Usage

```
diffObj(target, current, ...)
```

Arguments

target	the reference object
current	the object being compared to target
...	unused, for compatibility of methods with generics

Value

a Diff object; see diffPrint.

See Also

diffPrint for details on the diff* methods, diffStr, diffChr to compare character vectors directly diffDeparse to compare deparsed objects, ses for a minimal and fast diff

Examples

```
## `pager="off"` for CRAN compliance; you may omit in normal use
diffObj(letters, c(letters[1:10], LETTERS[11:26]), pager="off")
with(mtcars, diffObj(lm(mpg ~ hp)$qr, lm(mpg ~ disp)$qr, pager="off"))
```

---

diffobj_set_def_opts *Set All diffobj Options to Defaults*

Description

Used primarily for testing to ensure all options are set to default values.

Usage

```
diffobj_set_def_opts()
```

Value

list for use with options that contains values of diffob options before they were forced to defaults

Examples

```
## Not run:
  diffobj_set_def_opts()

## End(Not run)
```

diffPrint *Diff* print*ed Objects*

Description

Runs the diff between the print or show output produced by target and current. Given the extensive parameter list, this documentation page is intended as a reference for all the diff* methods. For a high level introduction see vignette("diffobj").

Usage

```
diffPrint(target, current, ...)

## S4 method for signature 'ANY'
diffPrint(
  target,
  current,
  mode = gdo("mode"),
  context = gdo("context"),
  format = gdo("format"),
  brightness = gdo("brightness"),
  color.mode = gdo("color.mode"),
  word.diff = gdo("word.diff"),
  pager = gdo("pager"),
  guides = gdo("guides"),
  trim = gdo("trim"),
  rds = gdo("rds"),
  unwrap.atomic = gdo("unwrap.atomic"),
  max.diffs = gdo("max.diffs"),
  disp.width = gdo("disp.width"),
  ignore.white.space = gdo("ignore.white.space"),
  convert.hz.white.space = gdo("convert.hz.white.space"),
  tab.stops = gdo("tab.stops"),
  line.limit = gdo("line.limit"),
  hunk.limit = gdo("hunk.limit"),
  align = gdo("align"),
  style = gdo("style"),
  palette.of.styles = gdo("palette"),
  frame = par_frame(),
  interactive = gdo("interactive"),
  term.colors = gdo("term.colors"),
  tar.banner = NULL,
  cur.banner = NULL,
  strip.sgr = gdo("strip.sgr"),
  sgr.supported = gdo("sgr.supported"),
  extra = list()
)
```

Arguments

target	the reference object
current	the object being compared to target
...	unused, for compatibility of methods with generics
mode	character(1L), one of: • "unified": diff mode used by git diff • "sidebyside": line up the differences side by side • "context": show the target and current hunks in their entirety; this mode takes up a lot of screen space but makes it easier to see what the objects actually look like • "auto": default mode; pick one of the above, will favor "sidebyside" unless getOption("width") is less than 80, or in diffPrint and objects are dimensioned and do not fit side by side, or in diffChr, diffDeparse, diffFile and output does not fit in side by side without wrapping
context	integer(1L) how many lines of context are shown on either side of differences (defaults to 2). Set to -1L to allow as many as there are. Set to "auto" to display as many as 10 lines or as few as 1 depending on whether total screen lines fit within the number of lines specified in line.limit. Alternatively pass the return value of auto_context to fine tune the parameters of the auto context calculation.
format	character(1L), controls the diff output format, one of: • "auto": to select output format based on terminal capabilities; will attempt to use one of the ANSI formats if they appear to be supported, and if not or if you are in the Rstudio console it will attempt to use HTML and browser output if in interactive mode. • "raw": plain text • "ansi8": color and format diffs using basic ANSI escape sequences • "ansi256": like "ansi8", except using the full range of ANSI formatting options • "html": color and format using HTML markup; the resulting string is processed with enc2utf8 when output as a full web page (see docs for html.output under Style). Defaults to "auto". See palette.of.styles for details on customization, style for full control of output format. See 'pager' parameter for more discussion of Rstudio behavior.
brightness	character, one of "light", "dark", "neutral", useful for adjusting color scheme to light or dark terminals. "neutral" by default. See PaletteOfStyles for details and limitations. Advanced: you may specify brightness as a function of format. For example, if you typically wish to use a "dark" color scheme, except for when in "html" format when you prefer the "light" scheme, you may use c("dark",html="light") as the value for this parameter. This is particularly useful if format is set to "auto" or if you want to specify a default value for this parameter via options. Any names you use should correspond to a format. You must have one unnamed value which will be used as the default for all formats that are not explicitly specified.

`color.mode` character, one of "rgb" or "yb". Defaults to "yb". "yb" stands for "Yellow-Blue" for color schemes that rely primarily on those colors to style diffs. Those colors can be easily distinguished by individuals with limited red-green color sensitivity. See `PaletteOfStyles` for details and limitations. Also offers the same advanced usage as the `brightness` parameter.

`word.diff` TRUE (default) or FALSE, whether to run a secondary word diff on the in-hunk differences. For atomic vectors setting this to FALSE could make the diff *slower* (see the `unwrap.atomic` parameter). For other uses, particularly with `diffChr` setting this to FALSE can substantially improve performance.

`pager` one of "auto" (default), "on", "off", a `Pager` object, or a list; controls whether and how a pager is used to display the diff output. If you require a particular pager behavior you must use a `Pager` object, or "off" to turn off the pager. All other settings will interact with other parameters such as `format`, `style`, as well as with your system capabilities in order to select the pager expected to be most useful.

"auto" and "on" are the same, except that in non-interactive mode "auto" is equivalent to "off". "off" will always send output to the console. If "on", whether the output actually gets routed to the pager depends on the pager `threshold` setting (see `Pager`). The default behavior is to use the pager associated with the `Style` object. The `Style` object is itself is determined by the `format` or `style` parameters.

Depending on your system configuration different styles and corresponding pagers will get selected, unless you specify a `Pager` object directly. On a system with a system pager that supports ANSI CSI SGR colors, the pager will only trigger if the output is taller than one window. If the system pager is not known to support ANSI colors then the output will be sent as HTML to the IDE viewer if available or to the web browser if not. Even though Rstudio now supports ANSI CSI SGR at the console output is still formatted as HTML and sent to the IDE viewer. Partly this is for continuity of behavior, but also because the default Rstudio pager does not support ANSI CSI SGR, at least as of this writing.

If `pager` is a list, then the same as with "on", except that the `Pager` object associated with the selected `Style` object is re-instantiated with the union of the list elements and the existing settings of that `Pager`. The list should contain named elements that correspond to the `Pager` instantiation parameters. The names must be specified in full as partial parameter matching will not be carried out because the pager is re-instantiated with `new`.

See `Pager`, `Style`, and `PaletteOfStyles` for more details and for instructions on how to modify the default behavior.

`guides` TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Guides are additional context lines that are not strictly part of a hunk, but provide important contextual data (e.g. column headers). If TRUE, the context lines are shown in addition to the normal diff output, typically in a different color to indicate they are not part of the hunk. If a function, the function should accept as the first argument the object being diffed, and the second the character representation of the object. The function should return the indices of the elements of the character representation that should be treated as guides. See `guides` for more details.

`trim` TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Function should compute for each line in captured output what portion of those lines should be diffed. By default, this is used to remove row meta data differences (e.g. `[1,]`) so they alone do not show up as differences in the diff. See `trim` for more details.

`rds` TRUE (default) or FALSE, if TRUE will check whether `target` and/or `current` point to a file that can be read with `readRDS` and if so, loads the R object contained in the file and carries out the diff on the object instead of the original argument. Currently there is no mechanism for specifying additional arguments to `readRDS`

`unwrap.atomic` TRUE (default) or FALSE. Relevant primarily for `diffPrint`, if TRUE, and `word.diff` is also TRUE, and both `target` and `current` are *unnamed* one-dimension atomics , the vectors are unwrapped and diffed element by element, and then re-wrapped. Since `diffPrint` is fundamentally a line diff, the re-wrapped lines are lined up in a manner that is as consistent as possible with the unwrapped diff. Lines that contain the location of the word differences will be paired up. Since the vectors may well be wrapped with different periodicities this will result in lines that are paired up that look like they should not be paired up, though the locations of the differences should be. If is entirely possible that setting this parameter to FALSE will result in a slower diff. This happens if two vectors are actually fairly similar, but their line representations are not. For example, in comparing `1:100` to `c(100,1:99)`, there is really only one difference at the "word" level, but every screen line is different. `diffChr` will also do the unwrapping if it is given a character vector that contains output that looks like the atomic vectors described above. This is a bug, but as the functionality could be useful when diffing e.g. `capture.output` data, we now declare it a feature.

`max.diffs` integer(1L), number of *differences* after which we abandon the `O(n^2)` diff algorithm in favor of a naive element by element comparison. Set to `-1L` to always stick to the original algorithm (defaults to 50000L).

`disp.width` integer(1L) number of display columns to take up; note that in "sidebyside" `mode` the effective display width is half this number (set to 0L to use default widths which are `getOption("width")` for normal styles and `80L` for HTML styles. Future versions of `diffobj` may change this to larger values for two dimensional objects for better diffs (see details).

`ignore.white.space` TRUE or FALSE, whether to consider differences in horizontal whitespace (i.e. spaces and tabs) as differences (defaults to TRUE).

`convert.hz.white.space` TRUE or FALSE, whether modify input strings that contain tabs and carriage returns in such a way that they display as they would **with** those characters, but without using those characters (defaults to TRUE). The conversion assumes that tab stops are spaced evenly eight characters apart on the terminal. If this is not the case you may specify the tab stops explicitly with `tab.stops`.

`tab.stops` integer, what tab stops to use when converting hard tabs to spaces. If not integer will be coerced to integer (defaults to 8L). You may specify more than one tab stop. If display width exceeds that addressable by your tab stops the last tab stop will be repeated.

`line.limit` integer(2L) or integer(1L), if length 1 how many lines of output to show, where `-1` means no limit. If length 2, the first value indicates the threshold of screen lines to begin truncating output, and the second the number of lines to truncate to, which should be fewer than the threshold. Note that this parameter is implemented on a best-efforts basis and should not be relied on to produce the exact number of lines requested. In particular do not expect it to work well for for values small enough that the banner portion of the diff would have to be trimmed. If you want a specific number of lines use `[` or `head` / `tail`. One advantage of `line.limit` over these other options is that you can combine it with `context="auto"` and auto `max.level` selection (the latter for `diffStr`), which allows the diff to dynamically adjust to make best use of the available display lines. `[`, `head`, and `tail` just subset the text of the output.

`hunk.limit` integer(2L) or integer (1L), how many diff hunks to show. Behaves similarly to `line.limit`. How many hunks are in a particular diff is a function of how many differences, and also how much `context` is used since context can cause two hunks to bleed into each other and become one.

`align` numeric(1L) between 0 and 1, proportion of words in a line of `target` that must be matched in a line of `current` in the same hunk for those lines to be paired up when displayed (defaults to 0.25), or an `AlignThreshold` object. Set to 1 to turn off alignment which will cause all lines in a hunk from `target` to show up first, followed by all lines from `current`. Note that in order to be aligned lines must meet the threshold and have at least 3 matching alphanumeric characters (see `AlignThreshold` for details).

`style` "auto", a `Style` object, or a list. "auto" by default. If a `Style` object, will override the the `format`, `brightness`, and `color.mode` parameters. The `Style` object provides full control of diff output styling. If a list, then the same as "auto", except that if the auto-selected `Style` requires instantiation (see `PaletteOfStyles`), then the list contents will be used as arguments when instantiating the style object. See `Style` for more details, in particular the examples.

`palette.of.styles` `PaletteOfStyles` object; advanced usage, contains all the `Style` objects or "classRepresentation" objects extending `Style` that are selected by specifying the `format`, `brightness`, and `color.mode` parameters. See `PaletteOfStyles` for more details.

`frame` an environment to use as the evaluation frame for the `print/show/str`, calls and for `diffObj`, the evaluation frame for the `diffPrint` / `diffStr` calls. Defaults to the return value of `par_frame`.

`interactive` TRUE or FALSE whether the function is being run in interactive mode, defaults to the return value of `interactive`. If in interactive mode, pager will be used if `pager` is "auto", and if ANSI styles are not supported and `style` is "auto", output will be send to viewer/browser as HTML.

`term.colors` integer(1L) how many ANSI colors are supported by the terminal. This variable is provided for when `crayon::num_colors` does not properly detect how many ANSI colors are supported by your terminal. Defaults to return value of `crayon::num_colors` and should be 8 or 256 to allow ANSI colors, or any other number to disallow them. This only impacts output format selection when `style` and `format` are both set to "auto".

tar.banner	character(1L), language, or NULL, used to generate the text to display ahead of the diff section representing the target output. If NULL will use the deparsed target expression, if language, will use the language as it would the target expression, if character(1L), will use the string with no modifications. The language mode is provided because diffStr modifies the expression prior to display (e.g. by wrapping it in a call to str). Note that it is possible in some cases that the substituted value of target actually is character(1L), but if you provide a character(1L) value here it will be assumed you intend to use that value literally.
cur.banner	character(1L) like tar.banner, but for current
strip.sgr	TRUE, FALSE, or NULL (default), whether to strip ANSI CSI SGR sequences prior to comparison and for display of diff. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and FALSE otherwise. The default behavior is to avoid confusing diffs where the original SGR and the SGR added by the diff are mixed together.
sgr.supported	TRUE, FALSE, or NULL (default), whether to assume the standard output device supports ANSI CSI SGR sequences. If TRUE, strings will be manipulated accounting for the SGR sequences. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and to 'crayon::has_color()' otherwise. This only controls how the strings are manipulated, not whether SGR is added to format the diff, which is controlled by the 'style' parameter. This parameter is exposed for the rare cases where you might wish to control string manipulation behavior directly.
extra	list additional arguments to pass on to the functions used to create text representation of the objects to diff (e.g. print, str, etc.)

Details

Almost all aspects of how the diffs are computed and displayed are controllable through the diff* methods parameters. This results in a lengthy parameter list, but in practice you should rarely need to adjust anything past the color.mode parameter. Default values are specified as options so that users may configure diffs in a persistent manner. gdo is a shorthand function to access diffobj options.

Parameter order after color.mode is not guaranteed. Future versions of diffobj may add parameters and re-order existing parameters past color.mode.

This and other diff* functions are S4 generics that dispatch on the target and current parameters. Methods with signature c("ANY","ANY") are defined and act as the default methods. You can use this to set up methods to pre-process or set specific parameters for selected classes that can then callNextMethod for the actual diff. Note that while the generics include ... as an argument, none of the methods do.

Strings are re-encoded to UTF-8 with enc2utf8 prior to comparison to avoid spurious encoding-only differences.

Value

a Diff object; this object has a show method that will display the diff to screen or pager, as well as summary, any, and as.character methods. If you store the return value instead of displaying it to

screen, and display it later, it is possible for the display to be thrown off if there are environment changes (e.g. display width changes) in between the time you compute the diff and the time you display it.

Matrices and Data Frames

While `diffPrint` attempts to handle the default R behavior that wraps wide tables, the results are often sub-optimal. A better approach is to set the `disp.width` parameter to a large enough value such that wrapping is not necessary, and a browser-based `pager`. In the future we will add the capability to specify different capture widths and wrap widths so that this is an option for terminal output (see issue 109).

One thing to keep in mind is that `diffPrint` is not designed to work with very large data frames.

See Also

`diffObj`, `diffStr`, `diffChr` to compare character vectors directly, `diffDeparse` to compare deparsed objects, `ses` for a minimal and fast diff @param target the reference object

Examples

```
## `pager="off"` for CRAN compliance; you may omit in normal use
diffPrint(letters, letters[-5], pager="off")
```

`diffStr` *Diff Object Structures*

Description

Compares the `str` output of `target` and `current`. If the `max.level` parameter to `str` is left unspecified, will attempt to find the largest `max.level` that fits within `line.limit` and shows at least one difference.

Usage

```
diffStr(target, current, ...)

## S4 method for signature 'ANY'
diffStr(
  target,
  current,
  mode = gdo("mode"),
  context = gdo("context"),
  format = gdo("format"),
  brightness = gdo("brightness"),
  color.mode = gdo("color.mode"),
  word.diff = gdo("word.diff"),
  pager = gdo("pager"),
```

```
  guides = gdo("guides"),
  trim = gdo("trim"),
  rds = gdo("rds"),
  unwrap.atomic = gdo("unwrap.atomic"),
  max.diffs = gdo("max.diffs"),
  disp.width = gdo("disp.width"),
  ignore.white.space = gdo("ignore.white.space"),
  convert.hz.white.space = gdo("convert.hz.white.space"),
  tab.stops = gdo("tab.stops"),
  line.limit = gdo("line.limit"),
  hunk.limit = gdo("hunk.limit"),
  align = gdo("align"),
  style = gdo("style"),
  palette.of.styles = gdo("palette"),
  frame = par_frame(),
  interactive = gdo("interactive"),
  term.colors = gdo("term.colors"),
  tar.banner = NULL,
  cur.banner = NULL,
  strip.sgr = gdo("strip.sgr"),
  sgr.supported = gdo("sgr.supported"),
  extra = list()
)
```

Arguments

target	the reference object
current	the object being compared to target
...	unused, for compatibility of methods with generics
mode	character(1L), one of: • "unified": diff mode used by git diff • "sidebyside": line up the differences side by side • "context": show the target and current hunks in their entirety; this mode takes up a lot of screen space but makes it easier to see what the objects actually look like • "auto": default mode; pick one of the above, will favor "sidebyside" unless getOption("width") is less than 80, or in diffPrint and objects are dimensioned and do not fit side by side, or in diffChr, diffDeparse, diffFile and output does not fit in side by side without wrapping
context	integer(1L) how many lines of context are shown on either side of differences (defaults to 2). Set to -1L to allow as many as there are. Set to "auto" to display as many as 10 lines or as few as 1 depending on whether total screen lines fit within the number of lines specified in line.limit. Alternatively pass the return value of auto_context to fine tune the parameters of the auto context calculation.
format	character(1L), controls the diff output format, one of:

- "auto": to select output format based on terminal capabilities; will attempt to use one of the ANSI formats if they appear to be supported, and if not or if you are in the Rstudio console it will attempt to use HTML and browser output if in interactive mode.
- "raw": plain text
- "ansi8": color and format diffs using basic ANSI escape sequences
- "ansi256": like "ansi8", except using the full range of ANSI formatting options
- "html": color and format using HTML markup; the resulting string is processed with `enc2utf8` when output as a full web page (see docs for `html.output` under `Style`).

Defaults to "auto". See `palette.of.styles` for details on customization, `style` for full control of output format. See 'pager' parameter for more discussion of Rstudio behavior.

`brightness` character, one of "light", "dark", "neutral", useful for adjusting color scheme to light or dark terminals. "neutral" by default. See `PaletteOfStyles` for details and limitations. Advanced: you may specify brightness as a function of `format`. For example, if you typically wish to use a "dark" color scheme, except for when in "html" format when you prefer the "light" scheme, you may use `c("dark",html="light")` as the value for this parameter. This is particularly useful if `format` is set to "auto" or if you want to specify a default value for this parameter via options. Any names you use should correspond to a `format`. You must have one unnamed value which will be used as the default for all `formats` that are not explicitly specified.

`color.mode` character, one of "rgb" or "yb". Defaults to "yb". "yb" stands for "Yellow-Blue" for color schemes that rely primarily on those colors to style diffs. Those colors can be easily distinguished by individuals with limited red-green color sensitivity. See `PaletteOfStyles` for details and limitations. Also offers the same advanced usage as the `brightness` parameter.

`word.diff` TRUE (default) or FALSE, whether to run a secondary word diff on the in-hunk differences. For atomic vectors setting this to FALSE could make the diff *slower* (see the `unwrap.atomic` parameter). For other uses, particularly with `diffChr` setting this to FALSE can substantially improve performance.

`pager` one of "auto" (default), "on", "off", a `Pager` object, or a list; controls whether and how a pager is used to display the diff output. If you require a particular pager behavior you must use a `Pager` object, or "off" to turn off the pager. All other settings will interact with other parameters such as `format`, `style`, as well as with your system capabilities in order to select the pager expected to be most useful.

"auto" and "on" are the same, except that in non-interactive mode "auto" is equivalent to "off". "off" will always send output to the console. If "on", whether the output actually gets routed to the pager depends on the pager `threshold` setting (see `Pager`). The default behavior is to use the pager associated with the `Style` object. The `Style` object is itself is determined by the `format` or `style` parameters.

Depending on your system configuration different styles and corresponding pagers will get selected, unless you specify a `Pager` object directly. On a system with

a system pager that supports ANSI CSI SGR colors, the pager will only trigger if the output is taller than one window. If the system pager is not known to support ANSI colors then the output will be sent as HTML to the IDE viewer if available or to the web browser if not. Even though Rstudio now supports ANSI CSI SGR at the console output is still formatted as HTML and sent to the IDE viewer. Partly this is for continuity of behavior, but also because the default Rstudio pager does not support ANSI CSI SGR, at least as of this writing.

If `pager` is a list, then the same as with "on", except that the `Pager` object associated with the selected `Style` object is re-instantiated with the union of the list elements and the existing settings of that `Pager`. The list should contain named elements that correspond to the `Pager` instantiation parameters. The names must be specified in full as partial parameter matching will not be carried out because the pager is re-instantiated with `new`.

See `Pager`, `Style`, and `PaletteOfStyles` for more details and for instructions on how to modify the default behavior.

`guides` TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Guides are additional context lines that are not strictly part of a hunk, but provide important contextual data (e.g. column headers). If TRUE, the context lines are shown in addition to the normal diff output, typically in a different color to indicate they are not part of the hunk. If a function, the function should accept as the first argument the object being diffed, and the second the character representation of the object. The function should return the indices of the elements of the character representation that should be treated as guides. See `guides` for more details.

`trim` TRUE (default), FALSE, or a function that accepts at least two arguments and requires no more than two arguments. Function should compute for each line in captured output what portion of those lines should be diffed. By default, this is used to remove row meta data differences (e.g. `[1,]`) so they alone do not show up as differences in the diff. See `trim` for more details.

`rds` TRUE (default) or FALSE, if TRUE will check whether `target` and/or `current` point to a file that can be read with `readRDS` and if so, loads the R object contained in the file and carries out the diff on the object instead of the original argument. Currently there is no mechanism for specifying additional arguments to `readRDS`

`unwrap.atomic` TRUE (default) or FALSE. Relevant primarily for `diffPrint`, if TRUE, and `word.diff` is also TRUE, and both `target` and `current` are *unnamed* one-dimension atomics , the vectors are unwrapped and diffed element by element, and then re-wrapped. Since `diffPrint` is fundamentally a line diff, the re-wrapped lines are lined up in a manner that is as consistent as possible with the unwrapped diff. Lines that contain the location of the word differences will be paired up. Since the vectors may well be wrapped with different periodicities this will result in lines that are paired up that look like they should not be paired up, though the locations of the differences should be. If is entirely possible that setting this parameter to FALSE will result in a slower diff. This happens if two vectors are actually fairly similar, but their line representations are not. For example, in comparing `1:100` to `c(100,1:99)`, there is really only one difference at the "word" level, but every screen line is different. `diffChr` will also do the

unwrapping if it is given a character vector that contains output that looks like the atomic vectors described above. This is a bug, but as the functionality could be useful when diffing e.g. `capture.output` data, we now declare it a feature.

`max.diffs` integer(1L), number of *differences* after which we abandon the `O(n^2)` diff algorithm in favor of a naive element by element comparison. Set to `-1L` to always stick to the original algorithm (defaults to 50000L).

`disp.width` integer(1L) number of display columns to take up; note that in "sidebyside" `mode` the effective display width is half this number (set to 0L to use default widths which are `getOption("width")` for normal styles and `80L` for HTML styles. Future versions of `diffobj` may change this to larger values for two dimensional objects for better diffs (see details).

`ignore.white.space`
TRUE or FALSE, whether to consider differences in horizontal whitespace (i.e. spaces and tabs) as differences (defaults to TRUE).

`convert.hz.white.space`
TRUE or FALSE, whether modify input strings that contain tabs and carriage returns in such a way that they display as they would **with** those characters, but without using those characters (defaults to TRUE). The conversion assumes that tab stops are spaced evenly eight characters apart on the terminal. If this is not the case you may specify the tab stops explicitly with `tab.stops`.

`tab.stops` integer, what tab stops to use when converting hard tabs to spaces. If not integer will be coerced to integer (defaults to 8L). You may specify more than one tab stop. If display width exceeds that addressable by your tab stops the last tab stop will be repeated.

`line.limit` integer(2L) or integer(1L), if length 1 how many lines of output to show, where `-1` means no limit. If length 2, the first value indicates the threshold of screen lines to begin truncating output, and the second the number of lines to truncate to, which should be fewer than the threshold. Note that this parameter is implemented on a best-efforts basis and should not be relied on to produce the exact number of lines requested. In particular do not expect it to work well for for values small enough that the banner portion of the diff would have to be trimmed. If you want a specific number of lines use `[` or `head` / `tail`. One advantage of `line.limit` over these other options is that you can combine it with `context="auto"` and auto `max.level` selection (the latter for `diffStr`), which allows the diff to dynamically adjust to make best use of the available display lines. `[`, `head`, and `tail` just subset the text of the output.

`hunk.limit` integer(2L) or integer (1L), how many diff hunks to show. Behaves similarly to `line.limit`. How many hunks are in a particular diff is a function of how many differences, and also how much `context` is used since context can cause two hunks to bleed into each other and become one.

`align` numeric(1L) between 0 and 1, proportion of words in a line of `target` that must be matched in a line of `current` in the same hunk for those lines to be paired up when displayed (defaults to 0.25), or an `AlignThreshold` object. Set to `1` to turn off alignment which will cause all lines in a hunk from `target` to show up first, followed by all lines from `current`. Note that in order to be aligned lines must meet the threshold and have at least 3 matching alphanumeric characters (see `AlignThreshold` for details).

`style` "auto", a `Style` object, or a list. "auto" by default. If a `Style` object, will override the the `format`, `brightness`, and `color.mode` parameters. The `Style` object provides full control of diff output styling. If a list, then the same as "auto", except that if the auto-selected `Style` requires instantiation (see `PaletteOfStyles`), then the list contents will be used as arguments when instantiating the style object. See `Style` for more details, in particular the examples.

`palette.of.styles` `PaletteOfStyles` object; advanced usage, contains all the `Style` objects or "classRepresentation" objects extending `Style` that are selected by specifying the `format`, `brightness`, and `color.mode` parameters. See `PaletteOfStyles` for more details.

`frame` an environment to use as the evaluation frame for the `print/show/str`, calls and for `diffObj`, the evaluation frame for the `diffPrint` / `diffStr` calls. Defaults to the return value of `par_frame`.

`interactive` TRUE or FALSE whether the function is being run in interactive mode, defaults to the return value of `interactive`. If in interactive mode, pager will be used if `pager` is "auto", and if ANSI styles are not supported and `style` is "auto", output will be send to viewer/browser as HTML.

`term.colors` integer(1L) how many ANSI colors are supported by the terminal. This variable is provided for when `crayon::num_colors` does not properly detect how many ANSI colors are supported by your terminal. Defaults to return value of `crayon::num_colors` and should be 8 or 256 to allow ANSI colors, or any other number to disallow them. This only impacts output format selection when `style` and `format` are both set to "auto".

`tar.banner` character(1L), language, or NULL, used to generate the text to display ahead of the diff section representing the target output. If NULL will use the deparsed `target` expression, if language, will use the language as it would the `target` expression, if character(1L), will use the string with no modifications. The language mode is provided because `diffStr` modifies the expression prior to display (e.g. by wrapping it in a call to `str`). Note that it is possible in some cases that the substituted value of `target` actually is character(1L), but if you provide a character(1L) value here it will be assumed you intend to use that value literally.

`cur.banner` character(1L) like `tar.banner`, but for `current`

`strip.sgr` TRUE, FALSE, or NULL (default), whether to strip ANSI CSI SGR sequences prior to comparison and for display of diff. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and FALSE otherwise. The default behavior is to avoid confusing diffs where the original SGR and the SGR added by the diff are mixed together.

`sgr.supported` TRUE, FALSE, or NULL (default), whether to assume the standard output device supports ANSI CSI SGR sequences. If TRUE, strings will be manipulated accounting for the SGR sequences. If NULL, resolves to TRUE if 'style' resolves to an ANSI formatted diff, and to 'crayon::has_color()' otherwise. This only controls how the strings are manipulated, not whether SGR is added to format the diff, which is controlled by the 'style' parameter. This parameter is exposed for the rare cases where you might wish to control string manipulation behavior directly.

extra	list additional arguments to pass on to the functions used to create text representation of the objects to diff (e.g. print, str, etc.)

Details

Due to the seemingly inconsistent nature of max.level when used with objects with nested attributes, and also due to the relative slowness of str, this function simulates the effect of max.level by hiding nested lines instead of repeatedly calling str with varying values of max.level.

Value

a Diff object; see diffPrint.

See Also

diffPrint for details on the diff* functions, diffObj, diffStr, diffChr to compare character vectors directly, diffDeparse to compare deparsed objects, ses for a minimal and fast diff

Examples

```
## `pager="off"` for CRAN compliance; you may omit in normal use
with(mtcars, diffStr(lm(mpg ~ hp)$qr, lm(mpg ~ disp)$qr, pager="off"))
```

---

dimnames,PaletteOfStyles-method

Retrieve Dimnames for PaletteOfStyles Objects

Description

Retrieve Dimnames for PaletteOfStyles Objects

Usage

```
## S4 method for signature 'PaletteOfStyles'
dimnames(x)
```

Arguments

x	a PaletteOfStyles object

Value

list the dimension names dimnames(PaletteOfStyles())

finalizeHtml *Finalizing Methods for HTML Output*

Description

Used as the `finalizer` slot to `StyleHtml` objects to wrap character output prior to output to device. Used primarily by styles that output to HTML to properly configure HTML page structure, including injecting JS, CSS, etc..

Usage

```
finalizeHtml(x, ...)

## S4 method for signature 'ANY'
finalizeHtml(x, x.chr, js, ...)

## S4 method for signature 'Diff'
finalizeHtml(x, x.chr, ...)

## S4 method for signature 'DiffSummary'
finalizeHtml(x, x.chr, ...)
```

Arguments

`x`	object to finalize
`...`	arguments to pass on to methods
`x.chr`	character text representation of `x`, typically generated with the `as.character` method for `x`
`js`	character javascript code to append to HTML representation

gdo *Shorthand Function for Accessing diffobj Options*

Description

`gdo(x)` is equivalent to `getOption(sprintf("diffobj.%s",x))`.

Usage

```
gdo(x)
```

Arguments

`x`	character(1L) name off `diffobj` option to retrieve, without the "diffobj." prefix

Examples

```
gdo("format")
```

---

guides *Generic Methods to Implement Flexible Guide Line Computations*

Description

Guides are context lines that would normally be omitted from the diff because they are too far from any differences, but provide particularly useful contextual information. Column headers are a common example. Modifying guide finding is an advanced feature intended for package developers that want special treatment for the display output of their objects.

Usage

```
guidesPrint(obj, obj.as.chr)

## S4 method for signature 'ANY,character'
guidesPrint(obj, obj.as.chr)

guidesStr(obj, obj.as.chr)

## S4 method for signature 'ANY,character'
guidesStr(obj, obj.as.chr)

guidesChr(obj, obj.as.chr)

## S4 method for signature 'ANY,character'
guidesChr(obj, obj.as.chr)

guidesDeparse(obj, obj.as.chr)

## S4 method for signature 'ANY,character'
guidesDeparse(obj, obj.as.chr)

guidesFile(obj, obj.as.chr)

## S4 method for signature 'ANY,character'
guidesFile(obj, obj.as.chr)
```

Arguments

obj	an R object
obj.as.chr	the character representation of obj that is used for computing the diffs

Details

Diff detects these important context lines by looking for patterns in the text of the diff, and then displays these lines in addition to the normal diff output. Guides are marked by a tilde in the gutter, and are typically styled differently than normal context lines, by default in grey. Guides may be far from the diff hunk they are juxtaposed to. We eschew the device of putting the guides in the hunk header as git diff does because often the column alignment of the guide line is meaningful.

Guides are detected by the guides* methods documented here. Each of the diff* methods (e.g. diffPrint) has a corresponding guides* method (e.g. guidesPrint), with the exception of diffCsv since that method uses diffPrint internally. The guides* methods expect an R object as the first parameter and the captured display representation of the object in a character vector as the second. The function should then identify which elements in the character representation should be treated as guides, and should return the numeric indices for them.

The original object is passed as the first argument so that the generic can dispatch on it, and so the methods may adjust their guide finding behavior to data that is easily retrievable from the object, but less so from the character representation thereof.

The default method for guidesPrint has special handling for 2D objects (e.g. data frames, matrices), arrays, time series, tables, lists, and S4 objects that use the default show method. Guide finding is on a best efforts basis and may fail if your objects contain "pathological" display representations. Since the diff will still work with failed guides finding we consider this an acceptable compromise. Guide finding is more likely to fail with nested recursive structures. A known issue is that list-like S3 objects without print methods [reset the tag buffers](https://bugs.r-project.org/bugzilla/show_bug.cgi?id=17610) so the guides become less useful for them.

guidesStr highlights top level objects. The default methods for the other guide* generics do not do anything and exist only as a mechanism for providing custom guide line methods.

If you dislike the default handling you can also define your own methods for matrices, arrays, etc., or alternatively you can pass a guide finding function directly via the guides parameter to the diff* methods.

If you have classed objects with special patterns you can define your own methods for them (see examples), though if your objects are S3 you will need to use setOldClass as the guides* generics are S4.

Value

integer containing values in seq_along(obj.as.chr)

Note

The mechanism for identifying guides will almost certainly change in the future to allow for better handling of nested guides, so if you do implement custom guideline methods do so with the understanding that they will likely be deprecated in one of the future releases.

Examples

```
## Roundabout way of suppressing guides for matrices
setMethod("guidesPrint", c("matrix", "character"),
  function(obj, obj.as.chr) integer(0L)
)
```

```
## Special guides for "zulu" S3 objects that match lines
## starting in "zulu###" where ### is a nuber
setOldClass("zulu")
setMethod("guidesPrint", c("zulu", "character"),
  function(obj, obj.as.chr) {
    if(length(obj) > 20) grep("^zulu[0-9]*", obj.as.chr)
    else integer(0L)
} )
```

---

has_Rdiff *Attempt to Detect Whether diff Utility is Available*

Description

Checks whether `tools::Rdiff` issues a warning when running with `useDiff=TRUE` and if it does assumes this is because the diff utility is not available. Intended primarily for testing purposes.

Usage

```
has_Rdiff(test.with = tools::Rdiff)
```

Arguments

`test.with`	function to test for diff presence with, typically Rdiff

Value

TRUE or FALSE

Examples

```
has_Rdiff()
```

---

make_blocking *Create a Blocking Version of a Function*

Description

Wraps `fun` in a function that runs `fun` and then issues a `readline` prompt to prevent further R code evaluation until user presses a key.

Usage

```
make_blocking(fun, msg = "Press ENTER to continue...", invisible.res = TRUE)
```

Arguments

fun	a function
msg	character(1L) a message to use as the readline prompt
invisible.res	whether to return the result of fun invisibly

Value

fun, wrapped in a function that does the blocking.

Examples

```
make_blocking(sum, invisible.res=FALSE)(1:10)
```

---

nchar_html *Count Text Characters in HTML*

Description

Very simple implementation that will fail if there are any ">" in the HTML that are not closing tags, and assumes that HTML entities are all one character wide. Also, spaces are counted as one width each because the HTML output is intended to be displayed inside <PRE> tags.

Usage

```
nchar_html(x, ...)
```

Arguments

x	character
...	unused for compatibility with internal use

Value

integer(length(x)) with number of characters of each element

Examples

```
nchar_html("<a href='http:www.domain.com'>hello</a>")
```

Pager *Objects for Specifying Pager Settings*

Description

Initializers for pager configuration objects that modify pager behavior. These objects can be used as the `pager` argument to the `diff*` methods, or as the `pager` slot for `Style` objects. In this documentation we use the "pager" term loosely and intend it to refer to any device other than the terminal that can be used to render output.

Usage

```
Pager(
  pager = function(x) writeLines(readLines(x)),
  file.ext = "",
  threshold = 0L,
  ansi = FALSE,
  file.path = NA_character_,
  make.blocking = FALSE
)

PagerOff(...)

PagerSystem(pager = file.show, threshold = -1L, file.ext = "", ...)

PagerSystemLess(
  pager = file.show,
  threshold = -1L,
  flags = "R",
  file.ext = "",
  ansi = TRUE,
  ...
)

PagerBrowser(
  pager = view_or_browse,
  threshold = 0L,
  file.ext = "html",
  make.blocking = NA,
  ...
)
```

Arguments

`pager`	a function that accepts at least one parameter and does not require a parameter other than the first parameter. This function will be called with a file path passed as the first argument. The referenced file will contain the text of the diff.

By default this is a temporary file that will be deleted as soon as the pager function completes evaluation. PagerSystem and PagerSystemLess use file.show by default, and PagerBrowser uses view_or_browse for HTML output. For asynchronous pagers such as view_or_browse it is important to make the pager function blocking by setting the make.blocking parameter to TRUE, or to specify a pager file path explicitly with file.path.

file.ext — character(1L) an extension to append to file path passed to pager, *without* the period. For example, PagerBrowser uses "html" to cause browseURL to launch the web browser. This parameter will be overridden if file.path is used.

threshold — integer(1L) number of lines of output that triggers the use of the pager; negative values lead to using console_lines + 1, and zero leads to always using the pager irrespective of how many lines the output has.

ansi — TRUE or FALSE, whether the pager supports ANSI CSI SGR sequences.

file.path — character(1L), if not NA the diff will be written to this location, ignoring the value of file.ext. If NA_character_ (default), a temporary file is used and removed after the pager function completes evaluation. If not NA, the file is preserved. Beware that the file will be overwritten if it already exists.

make.blocking — TRUE, FALSE, or NA. Whether to wrap pager with make_blocking prior to calling it. This suspends R code execution until there is user input so that temporary diff files are not deleted before the pager has a chance to read them. This typically defaults to FALSE, except for PagerBrowser where it defaults to NA, which resolves to is.na(file.path) (i.e. it is TRUE if the diff is being written to a temporary file, and FALSE otherwise).

... — additional arguments to pass on to new that are passed on to parent classes.

flags — character(1L), only for PagerSystemLess, what flags to set with the LESS system environment variable. By default the "R" flag is set to ensure ANSI escape sequences are interpreted if it appears your terminal supports ANSI escape sequences. If you want to leave the output on the screen after you exit the pager you can use "RX". You should only provide the flag letters (e.g. ""RX"", not "-RX"). The system variable is only modified for the duration of the evaluation and is reset / unset afterwards. *Note:* you must specify this slot via the constructor as in the example. If you set the slot directly it will not have any effect.

Default Output Behavior

diff* methods use "pagers" to help manage large outputs and also to provide an alternative colored diff when the terminal does not support them directly.

For OS X and *nix systems where less is the pager and the terminal supports ANSI escape sequences, output is colored with ANSI escape sequences. If the output exceeds one screen height in size (as estimated by console_lines) it is sent to the pager.

If the terminal does not support ANSI escape sequences, or if the system pager is not less as detected by pager_is_less, then the output is rendered in HTML and sent to the IDE viewer (getOption("viewer")) if defined, or to the browser with browseURL if not. This behavior may seem sub-optimal for systems that have ANSI aware terminals and ANSI aware pagers other than less, but these should be rare and it is possible to configure diffobj to produce the correct output for them (see examples).

Pagers and Styles

There is a close relationship between pagers and Style. The Style objects control whether the output is raw text, formatted with ANSI escape sequences, or marked up with HTML. In order for these different types of outputs to render properly, they need to be sent to the right device. For this reason Style objects come with a Pager configuration object pre-assigned so the output can render correctly. The exact Pager configuration object depends on the Style as well as the system configuration.

In any call to the diff* methods you can always specify both the Style and Pager configuration object directly for full control of output formatting and rendering. We have tried to set-up sensible defaults for most likely use cases, but given the complex interactions involved it is possible you may need to configure things explicitly. Should you need to define explicit configurations you can save them as option values with options(diffobj.pager=...,diffobj.style=...) so that you do not need to specify them each time you use diffobj.

Pager Configuration Objects

The Pager configuration objects allow you to specify what device to use as the pager and under what circumstances the pager should be used. Several pre-defined pager configuration objects are available via constructor functions:

- Pager: Generic pager just outputs directly to terminal; not useful unless the default parameters are modified.
- PagerOff: Turn off pager
- PagerSystem: Use the system pager as invoked by file.show
- PagerSystemLess: Like PagerSystem, but provides additional configuration options if the system pager is less. Note this object does not change the system pager; it only allows you to configure it via the $LESS environment variable which will have no effect unless the system pager is set to be less.
- PagerBrowser: Use getOption("viewer") if defined, or browseURL if not

The default configuration for PagerSystem and PagerSystemLess leads to output being sent to the pager if it exceeds the estimated window size, whereas PagerBrowser always sends output to the pager. This behavior can be configured via the threshold parameter.

PagerSystemLess's primary role is to correctly configure the $LESS system variable so that less renders the ANSI escape sequences as intended. On OS X more is a faux-alias to less, except it does not appear to read the $LESS system variable. Should you configure your system pager to be the more version of less, pager_is_less will be tricked into thinking you are using a "normal" version of less and you will likely end up seeing gibberish in the pager. If this is your use case you will need to set-up a custom pager configuration object that sets the correct system variables.

Custom Pager Configurations

In most cases the simplest way to generate new pager configurations is to use a list specification in the diff* call. Alternatively you can start with an existing Pager object and change the defaults. Both these cases are covered in the examples.

You can change what system pager is used by PagerSystem by changing it with options(pager=...) or by changing the $PAGER environment variable. You can also explicitly set a function to act as the pager when you instantiate the Pager configuration object (see examples).

If you wish to define your own pager object you should do so by extending the any of the Pager classes. If the function you use to handle the actual paging is non-blocking (i.e. allows R code evaluation to continue after it is spawned, you should set the make.blocking parameter to TRUE to pause execution prior to deleting the temporary file that contains the diff.

See Also

Style, pager_is_less

Examples

```
## We `dontrun` these examples as they involve pagers that should only be run
## in interactive mode
## Not run:
## Specify Pager parameters via list; this lets the `diff*` functions pick
## their preferred pager based on format and other output parameters, but
## allows you to modify the pager behavior.

f <- tempfile()
diffChr(1:200, 180:300, format='html', pager=list(file.path=f))
head(readLines(f))  # html output
unlink(f)

## Assuming system pager is `less` and terminal supports ANSI ESC sequences
## Equivalent to running `less -RFX`

diffChr(1:200, 180:300, pager=PagerSystemLess(flags="RFX"))

## If the auto-selected pager would be the system pager, we could
## equivalently use:

diffChr(1:200, 180:300, pager=list(flags="RFX"))

## System pager is not less, but it supports ANSI escape sequences

diffChr(1:200, 180:300, pager=PagerSystem(ansi=TRUE))

## Use a custom pager, in this case we make up a trivial one and configure it
## always page (`threshold=0L`)

page.fun <- function(x) cat(paste0("| ", readLines(x)), sep="\n")
page.conf <- PagerSystem(pager=page.fun, threshold=0L)
diffChr(1:200, 180:300, pager=page.conf, disp.width=getOption("width") - 2)

## Set-up the custom pager as the default pager

options(diffobj.pager=page.conf)
diffChr(1:200, 180:300)

## A blocking pager (this is effectively very similar to what `PagerBrowser`
## does); need to block b/c otherwise temp file with diff could be deleted
## before the device has a chance to read it since `browseURL` is not
```

```
## blocking itself.  On OS X we need to specify the extension so the correct
## program opens it (in this case `TextEdit`):

page.conf <- Pager(pager=browseURL, file.ext="txt", make.blocking=TRUE)
diffChr(1:200, 180:300, pager=page.conf, format='raw')

## An alternative to a blocking pager is to disable the
## auto-file deletion; here we also specify a file location
## explicitly so we can recover the diff text.

f <- paste0(tempfile(), ".html")  # must specify .html
diffChr(1:5, 2:6, format='html', pager=list(file.path=f))
tail(readLines(f))
unlink(f)

## End(Not run)
```

`pager_is_less` *Check Whether System Has less as Pager*

Description

If `getOption(pager)` is set to the default value, checks whether `Sys.getenv("PAGER")` appears to be `less` by trying to run the pager with the "version" and parsing the output. If `getOption(pager)` is not the default value, then checks whether it points to the `less` program by the same mechanism.

Usage

```
pager_is_less()
```

Details

Some systems may have `less` pagers installed that do not respond to the `$LESS` environment variable. For example, `more` on at least some versions of OS X is `less`, but does not actually respond to `$LESS`. If such as pager is the system pager you will likely end up seeing gibberish in the pager. If this is your use case you will need to set-up a custom pager configuration object that sets the correct system variables (see `Pager`).

Value

TRUE or FALSE

See Also

`Pager`

Examples

```
pager_is_less()
```

PaletteOfStyles-class *Class for Tracking Default Styles by Style Type*

Description

Provides a mechanism for specifying a style based on the style properties along dimensions of format, brightness, and color. This allows a user to request a style that meets a certain description (e.g. a "light" scheme in "ansi256" format), without having to provide a specific `Style` object.

An Array of Styles

A `PaletteOfStyles` object is an "array" containing either "classRepresentation" objects that extend `StyleHtml` or are instances of objects that inherit from `StyleHtml`. The `diff*` methods then pick an object/class from this array based on the values of the `format`, `brightness`, and `color.mode` parameters.

For the most part the distinction between actual `Style` objects vs "classRepresentation" ones is academic, except that with the latter you can control the instantiation by providing a parameter list as the `style` argument to the `diff*` methods. This is not an option with already instantiated objects. See examples.

Dimensions

There are three general orthogonal dimensions of styles that can be used when rendering diffs: the type of format, the "brightness" of the output, and whether the colors used are distinguishable if you assume reds and greens are not distinguishable. Defaults for the intersections each of these dimensions are encoded as a three dimensional list. This list is just an atomic vector of type "list" with a length 3 `dim` attribute.

The array/list dimensions are:

- `format`: the format type, one of "raw", "ansi8", "ansi256", or "html"
- `brightness`: whether the colors are bright or not, which allows user to chose a scheme that is compatible with their console, one of: "light", "dark", "normal"
- `color.mode`: "rgb" for full color or "yb" for dichromats (yb stands for Yellow Blue).

Each of these dimensions can be specified directly via the corresponding parameters to the `diff*` methods.

Methods

`PaletteOfStyles` objects have The following methods implemented:

- `[`, `[<-`, `[[`
- show
- summary
- dimnames

Structural Details

The array/list is stored in the `data` slot of `PaletteOfStyles` objects. Subsetting methods are provided so you may operate directly on the S4 object as you would on a regular array.

The array/list must be fully populated with objects that are or inherit `Style`, or are "classRepresentation" objects (i.e. those of the type returned by `getClassDef`) that extend `Style`. By default the array is populated only with "classRepresentation" objects as that allows the list form of the `style` parameter to the `diff*` methods. If there is a particular combination of coordinates that does not have a corresponding defined style a reasonable substitution must be provided. For example, this package only defines "light" HTML styles, so it simply uses that style for all the possible `brightness` values.

There is no explicit check that the objects in the list comply with the descriptions implied by their coordinates, although the default object provided by the package does comply for the most part. One check that is carried out is that any element that has a "html" value in the `format` dimension extends `StyleHtml`.

While the list may only have the three dimensions described, you can add values to the dimensions provided the values described above are the first ones in each of their corresponding dimensions. For example, if you wanted to allow for styles that would render in `grid` graphics, you could generate a default list with a ""grid"" value appended to the values of the `format` dimension.

Examples

```
## Not run:
## Look at all "ansi256" styles (assumes compatible terminal)
PaletteOfStyles()["ansi256",,]

## End(Not run)
## Generate the default style object palette, and replace
## the ansi256 / light / rgb style with our modified one
## which for illustrative purposes is the raw style
my.pal <- PaletteOfStyles()
my.style <- StyleRaw()   # See `?Style` for custom styles
my.style@funs@word.delete <- function(x) sprintf("--%s--", x)
my.pal["ansi256", "light", "rgb"] <- list(my.style) # note `list()`
## Output has no format now for format/color.mode/brightness
## we modified ...
## `pager="off"` for CRAN compliance; you may omit in normal use
diffPrint(
   1:3, 2:5, format="ansi256", color.mode="rgb", brightness="light",
   palette.of.styles=my.pal, pager="off", disp.width=80
)
## If so desired, set our new style palette as the default
## one; could also pass directly as argument to `diff*` funs
## Not run:
options(diffobj.palette=defs)

## End(Not run)
```

par_frame *Get Parent Frame of S4 Call Stack*

Description

Implementation of the `function(x=parent.frame()) ...` pattern for the `diff*` methods since the normal pattern does not work with S4 methods. Works by looking through the call stack and identifying what call likely initiated the S4 dispatch.

Usage

```
par_frame()
```

Details

The function is not exported and intended only for use as the default value for the `frame` argument for the `diff*` methods.

Matching is done purely by looking for the last repeated call followed by `.local(target,current,...)` that is not a call to `eval`. This pattern seems to match the correct call most of the time. Since methods can be renamed by the user we make no attempt to verify method names. This method could potentially be tricked if you implement custom `diff*` methods that somehow issue two identical sequential calls before calling `callNextMethod`. Failure in this case means the wrong `frame` will be returned.

Value

an environment

Rdiff_chr *Run Rdiff Directly on R Objects*

Description

These functions are here for reference and testing purposes. They are wrappers to `tools::Rdiff` and rely on an existing system diff utility. You should be using `ses` or `diffChr` instead of `Rdiff_chr` and `diffPrint` instead of `Rdiff_obj`. See limitations in note.

Usage

```
Rdiff_chr(from, to, silent = FALSE, minimal = FALSE, nullPointers = TRUE)

Rdiff_obj(from, to, silent = FALSE, minimal = FALSE, nullPointers = TRUE)
```

Arguments

from	character or object coercible to character for Rdiff_chr, any R object with Rdiff_obj, or a file pointing to an RDS object
to	character same as from
silent	TRUE or FALSE, whether to display output to screen
minimal	TRUE or FALSE, whether to exclude the lines that show the actual differences or only the actual edit script commands
nullPointers	passed to tools::Rdiff

Details

Rdiff_chr runs diffs on character vectors or objects coerced to character vectors, where each value in the vectors is treated as a line in a file. Rdiff_chr always runs with the useDiff and Log parameters set to TRUE.

Rdiff_obj runs diffs on the printed representation of the provided objects. For each of from, to, will check if they are 1 length character vectors referencing an RDS file, and will use the contents of that RDS file as the object to compare.

Value

the Rdiff output, invisibly if silent is FALSE Rdiff_chr(letters[1:5], LETTERS[1:5]) Rdiff_obj(letters[1:5], LETTERS[1:5])

Note

These functions will try to use the system diff utility. This will fail in systems that do not have that utility available (e.g. windows installation without Rtools).

See Also

ses, diff*

---

ses *Shortest Edit Script*

Description

Computes shortest edit script to convert a into b by removing elements from a and adding elements from b. Intended primarily for debugging or for other applications that understand that particular format. See GNU diff docs for how to interpret the symbols.

Usage

```
ses(a, b, max.diffs = gdo("max.diffs"), warn = gdo("warn"))

ses_dat(a, b, extra = TRUE, max.diffs = gdo("max.diffs"), warn = gdo("warn"))
```

Arguments

a	character
b	character
max.diffs	integer(1L), number of *differences* after which we abandon the O(n^2) diff algorithm in favor of a naive element by element comparison. Set to -1L to always stick to the original algorithm (defaults to 50000L).
warn	TRUE (default) or FALSE whether to warn if we hit max.diffs.
extra	TRUE (default) or FALSE, whether to also return the indices in a and b the diff values are taken from. Set to FALSE for a small performance gain.

Details

ses will be much faster than any of the diff* methods, particularly for large inputs with limited numbers of differences.

NAs are treated as the string "NA". Non-character inputs are coerced to character.

ses_dat provides a semi-processed "machine-readable" version of precursor data to ses that may be useful for those desiring to use the raw diff data and not the printed output of diffobj, but do not wish to manually parse the ses output. Whether it is faster than ses or not depends on the ratio of matching to non-matching values as ses_dat includes matching values whereas ses does not. See examples.

Value

character shortest edit script, or a machine readable version of it as a data.frame with columns op (factor, values "Match", "Insert", or "Delete"), val character corresponding to the value taken from either a or b, and if extra is TRUE, integer columns id.a and id.b corresponding to the indices in a or b that val was taken from. See Details.

Examples

```
a <- letters[1:6]
b <- c('b', 'CC', 'DD', 'd', 'f')
ses(a, b)
(dat <- ses_dat(a, b))

## use `ses_dat` output to construct a minimal diff
## color with ANSI CSI SGR
diff <- dat[['val']]
del <- dat[['op']] == 'Delete'
ins <- dat[['op']] == 'Insert'
if(any(del))
  diff[del] <- paste0("\033[33m- ", diff[del], "\033[m")
if(any(ins))
  diff[ins] <- paste0("\033[34m+ ", diff[ins], "\033[m")
if(any(!ins & !del))
  diff[!ins & !del] <- paste0("  ", diff[!ins & !del])
writeLines(diff)
```

```
## We can recover `a` and `b` from the data
identical(subset(dat, op != 'Insert', val)[[1]], a)
identical(subset(dat, op != 'Delete', val)[[1]], b)
```

`show,DiffSummary-method`
Display DiffSummary Objects

Description

Display DiffSummary Objects

Usage

```
## S4 method for signature 'DiffSummary'
show(object)
```

Arguments

`object`	a `DiffSummary` object

Value

NULL, invisbly show(summary(diffChr(letters, letters[-c(5, 15)], format="raw", pager="off")))

`show,PaletteOfStyles-method`
Display a PaletteOfStyles

Description

Display a PaletteOfStyles

Usage

```
## S4 method for signature 'PaletteOfStyles'
show(object)
```

Arguments

`object`	a `PaletteOfStyles` object

Value

NULL, invisibly

show,Style-method *Show Method for Style Objects*

Description

Display a small sample diff with the Style object styles applied. For ANSI light and dark styles, will also temporarily set the background and foreground colors to ensure they are compatible with the style, even though this is not done in normal output (i.e. if you intend on using a "light" style, you should set your terminal background color to be light or expect sub-optimal rendering).

Usage

```
## S4 method for signature 'Style'
show(object)

## S4 method for signature 'StyleHtml'
show(object)
```

Arguments

object	a Style S4 object

Value

NULL, invisibly

Examples

```
show(StyleAnsi256LightYb())  # assumes ANSI colors supported
```

Style-class *Customize Appearance of Diff*

Description

S4 objects that expose the formatting controls for Diff objects. Many predefined formats are defined as classes that extend the base Style class. You may fine tune styles by either extending the pre-defined classes, or modifying an instance thereof.

Arguments

funs	a StyleFuns object that contains all the functions represented above
text	a StyleText object that contains the non-content text used by the diff (e.g. gutter.insert.txt)
summary	a StyleSummary object that contains formatting functions and other meta data for rendering summaries
pad	TRUE or FALSE, whether text should be right padded
pager	what type of Pager to use
nchar.fun	function to use to count characters; intended mostly for internal use (used only for gutters as of version 0.2.0).
wrap	TRUE or FALSE, whether text should be hard wrapped at disp.width
na.sub	what character value to substitute for NA elements; NA elements are generated when lining up side by side diffs by adding padding rows; by default the text styles replace these with a blank character string, and the HTML styles leave them as NA for the HTML formatting functions to deal with
blank	sub what character value to replace blanks with; needed in particular for HTML rendering (uses " ") to prevent lines from collapsing
disp.width	how many columns the text representation of the objects to diff is allowed to take up before it is hard wrapped (assuming wrap is TRUE). See param disp.width for diffPrint.
finalizer	function that accepts at least two parameters and requires no more than two parameters, will receive as the first parameter the the object to render (either a Diff or a DiffSummary object), and the text representation of that object as the second argument. This allows final modifications to the character output so that it is displayed correctly by the pager. For example, StyleHtml objects use it to generate HTML headers if the Diff is destined to be displayed in a browser. The object themselves are passed along to provide information about the paging device and other contextual data to the function.
html.output	(StyleHtml objects only) one of: • "page": Include all HTML/CSS/JS required to create a stand-alone web page with the diff; in this mode the diff string will be re-encoded with enc2utf8 and the HTML page encoding will be declared as UTF-8. • "diff.w.style": The CSS and HTML, but without any of the outer tags that would make it a proper HTML page (i.e. no <html>/<head> tags or the like) and without the JS; note that technically this is illegal HTML since we have <style> tags floating outside of <head> tags, but it seems to work in most browsers. • "diff.only": Like "diff.w.style", but without the CSS • "auto": Pick one of the above based on Pager, will chose "page" if the pager is of type PagerBrowser (as in that case the output is destined to be displayed in a browser like device), or "diff.only" if it is not.
escape.html.entities	(StyleHtml objects only) TRUE (default) or FALSE, whether to escape HTML entities in the input

scale	(StyleHtml objects only) TRUE (default) or FALSE, whether to scale HTML output to fit to the viewport
css	(StyleHtml objects only) path to file containing CSS styles to style HTML output with
js	(StyleHtml objects only) path to file containing Javascript used for scaling output to viewports.

Value

Style S4 object

Pre-defined Classes

Pre-defined classes are used to populate the PaletteOfStyles object, which in turn allows the diff* methods to pick the appropriate Style for each combination of the format, color.mode, and brightness parameters when the style parameter is set to "auto". The following classes are pre-defined:

- StyleRaw: No styles applied
- StyleAnsi8NeutralRgb
- StyleAnsi8NeutralYb
- StyleAnsi256LightRgb
- StyleAnsi256LightYb
- StyleAnsi256DarkRgb
- StyleAnsi256DarkYb
- StyleHtmlLightRgb
- StyleHtmlLightYb

Each of these classes has an associated constructor function with the same name (see examples). Objects instantiated from these classes may also be used directly as the value for the style parameter to the diff* methods. This will override the automatic selection process that uses PaletteOfStyles. If you wish to tweak an auto-selected style rather than explicitly specify one, pass a parameter list instead of a Style objects as the style parameter to the diff* methods (see examples).

There are predefined classes for most combinations of format/color.mode/brightness, but not all. For example, there are only "light" brightness defined for the "html" format, and those classes are re-used for all possible brightness values, and the 8 color ANSI neutral classes are used for the 256 color neutral selections as well.

To get a preview of what a style looks like just instantiate an object; the show method will output a trivial diff to screen with styles applied. Note that for ANSI styles of the dark and light variety the show method colors the terminal background and foregrounds in compatible colors. In normal usage the terminal background and foreground colors are left untouched so you should not expect light styles to look good on dark background and vice versa even if they render correctly when showing the style object.

Style Structure

Most of the customization is done by specifying functions that operate on character vectors and return a modified character vector of the same length. The intended use case is to pass `crayon` functions such as `crayon::red`, although you may pass any function of your liking that behaves as described. Formatting functions are expected to return their inputs formatted in such a way that their *display* width is unchanged. If your formatting functions change display width output may not render properly, particularly when using `mode="sidebyside"`.

The visual representation of the diff has many nested components. The functions you specify here will be applied starting with the innermost ones. A schematic of the various component that represent an inserted line follows (note "insert" abbreviated to "ins", and "gutter" abbreviated to "gtr"):

```
+- line ---------------------------------------------------+
|+- line.ins -----------------------------------------------+|
||+- gtr -------------------------++- text ----------------+||
|||+- gtr.ins ---++- gtr.pad ---+||+- text.ins ----------+|||
||||             ||             ||||      +- word.ins -+||||
|||| gtr.ins.txt || gtr.pad.txt |||| DIFF | TEXT HERE  |||||
||||             ||             ||||      +------------+||||
|||+-------------++-------------+||+---------------------+|||
||+-------------------------------++----------------------+||
|+-------------------------------------------------------+|
+---------------------------------------------------------+
```

A similar model applies to deleted and matching lines. The boxes represent functions. `gutter.insert.txt` represents the text to use in the gutter and is not a function. `DIFF TEXT HERE` is text from the objects being diffed, with the portion that has different words inside the `word.insert`. `gutter.pad` and `gutter.pad.txt` are used to separate the gutter from the text and usually end up resolving to a space.

Most of the functions defined here default to `identity`, but you are given the flexibility to fully format the diff. See `StyleFuns` and `StyleText` for a full listing of the adjustable elements.

In side-by-side mode there are two "lines" per screen line, each with the structure described here.

The structure described here may change in the future.

HTML Styles

If you use a `Style` that inherits from `StyleHtml` the diff will be wrapped in HTML tags, styled with CSS, and output to `getOption("viewer")` if your IDE supports it (e.g. Rstudio), or directly to the browser otherwise, assuming that the default `Pager` or a correctly configured pager that inherits from `PagerBrowser` is in effect. Otherwise, the raw HTML will be output to your terminal.

By default HTML output sent to the viewer/browser is a full stand-alone webpage with CSS styles to format and color the diff, and JS code to handle scaling. The CSS and JS is read from the default files and injected into the HTML to simplify packaging of the output. You can customize the CSS and JS by using the `css` and `js` arguments respectively, but read the rest of this documentation section if you plan on doing so.

Should you want to capture the HTML output for use elsewhere, you can do so by using `as.character` on the return value of the `diff*` methods. If you want the raw HTML without any of the headers,

CSS, and JS use `html.ouput="diff.only"` when you instantiate the `StyleHtml` object (see examples), or disable the `Pager`. Another option is `html.output="diff.w.style"` which will add `<style>` tags with the CSS, but without wrapping those in `<head>` tags. This last option results in illegal HTML with a `<style>` block outside of the `<head>` block, but appears to work and is useful if you want to embed HTML someplace but do not have access to the headers.

If you wish to modify the CSS styles you should do so cautiously. The HTML and CSS work well together out of the box, but may not take to kindly to modifications. The safest changes you can make are to the colors of the scheme. You also probably should not modify the functions in the `@funs` slot of the `StyleHtml` object. If you want to provide your own custom styles make a copy of the file at the location returned by `diffobj_css()`, modify it to your liking, and pass the location of your modified sheet back via the `css` argument (see examples).

The javascript controls the scaling of the output such that its width fits in the viewport. If you wish to turn of this behavior you can do so via the `scale` argument. You may need to modify the javascript if you modify the `@funs` functions, but otherwise you are probably best off leaving the javascript untouched. You can provide the location of a modified javascript file via the `js` argument.

Both the CSS and JS files can be specified via options, "diffobj.html.css", and "diffobj.html.js" respectively.

If you define your own custom `StyleHtml` object you may want to modify the slot `@funs@container`. This slot contains a function that is applied to the entire diff output. For example, `StyleHtmlLightRgb` uses `@funs@container <-cont_f("light","rgb")`. `cont_f` returns a function that accepts a character vector as an argument and returns that value wrapped in a `DIV` block with class ""diffobj-container light rgb"". This allows the CSS style sheet to target the `Diff` elements with the correct styles.

Modifying Style Parameters Directly

Often you will want to specify some of the style parameters (e.g. `scale` for html styles) while still relying on the default style selection to pick the specific style. You can do so by passing a list to the `style` parameter of the `diff*` methods. See examples.

New Classes

You can in theory create entirely new classes that extent `Style`. For example you could generate a class that renders the diff in `grid` graphics. Note however that we have not tested such extensions and it is possible there is some embedded code that will misbehave with such a new class.

Examples

```
## Not run:
## Create a new style based on existing style by changing
## gutter symbols and guide color; see `?StyleFuns` and
## `?StyleText` for a full list of adjustable elements
my.style <- StyleAnsi8NeutralYb()
my.style   ## `show` method gives you a preview of the style
my.style@text@gutter.insert <- "+++"
my.style@text@gutter.delete <- "---"
my.style@funs@text.guide <- crayon::green
my.style   ## Notice gutters and guide color
```

```
## Provide a custom style sheet; here we assume there is a style sheet at
## `HOME/web/mycss.css`
my.css <- file.path(path.expand("~"), "web", "mycss.css")
diffPrint(1:5, 2:6, style=StyleHtmlLightYb(css=my.css))

## Turn of scaling; notice how we pass a list to `style`
## and we do not need to specify a specific style
diffPrint(letters, letters[-5], format="html", style=list(scale=FALSE))

## Alternatively we can do the same by specifying a style, but we must
## give an exact html style instead of relying on preferences to pick
## one for us
my.style <- StyleHtmlLightYb(scale=FALSE)
diffPrint(letters, letters[-5], style=my.style)

## End(Not run)
## Return only the raw HTML without any of the headers
as.character(
  diffPrint(1:5, 2:6, format="html", style=list(html.output="diff.only"))
)
```

`StyleFuns-class` *Functions Used for Styling Diff Components*

Description

Except for `container` every function specified here should be vectorized and apply formatting to each element in a character vectors. The functions must accept at least one argument and require no more than one argument. The text to be formatted will be passed as a character vector as the first argument to each function.

Arguments

`container`	function used primarily by HTML styles to generate an outermost `DIV` that allows for CSS targeting of its contents (see `cont_f` for a function generator appropriate for use here)
`line`	function
`line.insert`	function
`line.delete`	function
`line.match`	function
`line.guide`	function formats guide lines (see `guides`)
`text`	function
`text.insert`	function
`text.delete`	function
`text.match`	function

`text.guide`	function formats guide lines (see `guides`)
`gutter`	function
`gutter.insert`	function
`gutter.delete`	function
`gutter.match`	function
`gutter.guide`	function
`gutter.pad`	function
`header`	function to format each hunk header with
`banner`	function to format entire banner
`banner.insert`	function to format insertion banner
`banner.delete`	function to format deletion banner
`meta`	function format meta information lines
`context.sep`	function to format the separator used to visually distinguish the A and B hunks in "context" `mode`

Details

These functions are applied in post processing steps. The `diff*` methods do not do any of the formatting. Instead, the formatting is done only if the user requests to `show` the object. Internally, `show` first converts the object to a character vector using `as.character`, which applies every formatting function defined here except for `container`. Then `show` applies `container` before forwarding the result to the screen or pager.

Value

a StyleFuns S4 object

Note

the slots are set to class "ANY" to allow classed functions such as those defined in the `crayon` package. Despite this seemingly permissive slot definition, only functions are allowed in the slots by the validation functions.

See Also

`Style`

StyleSummary-class *Styling Information for Summaries*

Description

Styling Information for Summaries

Slots

`container` function applied to entire summary

`body` function applied to everything except the actual map portion of the summary

`detail` function applied to section showing how many deletions / insertions, etc. occurred

`map` function applied to the map portion of the summary

StyleText-class *Character Tokens Used in Diffs*

Description

Various character tokens are used throughout diffs to provide visual cues. For example, gutters will contain characters that denote deletions and insertions (< and > by default).

Arguments

`gutter.insert`	character(1L) text to use as visual cue to indicate whether a diff line is an insertion, defaults to "> "
`gutter.insert.ctd`	character(1L) if a diff line is wrapped, the visual cue shifts to this character to indicate wrapping occured
`gutter.delete`	character(1L) see `gutter.insert` above
`gutter.delete.ctd`	character(1L) see `gutter.insert.ctd` above
`gutter.match`	character(1L) see `gutter.insert` above
`gutter.match.ctd`	character(1L) see `gutter.insert.ctd` above
`gutter.guide`	character(1L) see `gutter.insert` above
`gutter.guide.ctd`	character(1L) see `gutter.insert.ctd` above
`gutter.fill`	character(1L) see `gutter.insert` above
`gutter.fill.ctd`	character(1L) see `gutter.insert.ctd` above
`gutter.pad`	character(1L) separator between gutter characters and the rest of a line in a diff
`pad.col`	character(1L) separator between columns in side by side mode

Value

a StyleText S4 object

See Also

Style

---

summary,Diff-method *Summary Method for Diff Objects*

Description

Provides high level count of insertions, deletions, and matches, as well as a "map" of where the differences are.

Usage

```
## S4 method for signature 'Diff'
summary(
  object,
  scale.threshold = 0.1,
  max.lines = 50L,
  width = getOption("width"),
  ...
)
```

Arguments

object	at Diff object
scale.threshold	numeric(1L) between 0 and 1, how much distortion to allow when creating the summary map, where 0 is none and 1 is as much as needed to fit under max.lines, defaults to 0.1
max.lines	integer(1L) how many lines to allow for the summary map, defaults to 50
width	integer(1L) how many columns wide the output should be, defaults to getOption("width")
...	unused, for compatibility with generic

Details

Sequences of single operations (e.g. "DDDDD") are compressed provided that compressing them does not distort the relative size of the sequence relative to the longest such sequence in the map by more than scale.threshold. Since length 1 sequences cannot be further compressed scale.threshold does not apply to them.

Value

a DiffSummary object ## 'pager="off"' for CRAN compliance; you may omit in normal use summary(diffChr(letters, letters[-c(5, 15)], format="raw", pager="off"))

summary,PaletteOfStyles-method *Display a Summarized Version of a PaletteOfStyles*

Description

Display a Summarized Version of a PaletteOfStyles

Usage

```
## S4 method for signature 'PaletteOfStyles'
summary(object, ...)
```

Arguments

object	a PaletteOfStyles object
...	unused, for compatibility with generic

Value

character representation showing classes and/or objects in PaletteOfStyles summary(PaletteOfStyles())

tag_f *Make Functions That Wrap Text in HTML Tags*

Description

Helper functions to generate functions to use as slots for the StyleHtml@funs classes. These are functions that return *functions*.

Usage

```
tag_f(tag, class = character(), style = character())

div_f(class = character(), style = character())

span_f(class = character(), style = character())

cont_f(class = character())
```

Arguments

tag	character(1L) a name of an HTML tag
class	character the CSS class(es)
style	named character inline styles, where the name is the CSS property and the value the value.

Details

tag_f and related functions (div_f, span_f) produce functions that are vectorized and will apply opening and closing tags to each element of a character vector. container_f on the other hand produces a function will collapse a character vector into length 1, and only then applies the tags. Additionally, container_f already comes with the "diffobj-container" class specified.

Value

a function that accepts a character parameter. If applied, each element in the character vector will be wrapped in the div tags

Note

inputs are assumed to be valid class names or CSS styles.

Examples

```
## Assuming class 'ex1' has CSS styles defined elsewhere
tag_f("div", "ex1")(LETTERS[1:5])
## Use convenience function, and add some inline styles
div_f("ex2", c(color="green", `font-family`="arial"))(LETTERS[1:5])
## Notice how this is a div with pre-specifed class,
## and only one div is created around the entire data
cont_f()(LETTERS[1:5])
```

---

trim *Methods to Remove Unsemantic Text Prior to Diff*

Description

diff* methods, in particular diffPrint, modify the text representation of an object prior to running the diff to reduce the incidence of spurious mismatches caused by unsemantic differences. For example, we look to remove matrix row indices and atomic vector indices (i.e. the '[1,]' or '[1]' strings at the beginning of each display line).

Usage

```
trimPrint(obj, obj.as.chr)

## S4 method for signature 'ANY,character'
trimPrint(obj, obj.as.chr)

trimStr(obj, obj.as.chr)

## S4 method for signature 'ANY,character'
trimStr(obj, obj.as.chr)

trimChr(obj, obj.as.chr)
```

```
## S4 method for signature 'ANY,character'
trimChr(obj, obj.as.chr)

trimDeparse(obj, obj.as.chr)

## S4 method for signature 'ANY,character'
trimDeparse(obj, obj.as.chr)

trimFile(obj, obj.as.chr)

## S4 method for signature 'ANY,character'
trimFile(obj, obj.as.chr)
```

Arguments

obj	the object
obj.as.chr	character the printed representation of the object

Details

Consider:

```
> matrix(10:12)
     [,1]
[1,]   10
[2,]   11
[3,]   12
> matrix(11:12)
     [,1]
[1,]   11
[2,]   12
```

In this case, the line by line diff would find all rows of the matrix to be mismatched because where the data matches (rows containing 11 and 12) the indices do not. By trimming out the row indices before the diff, the diff can recognize that row 2 and 3 from the first matrix should be matched to row 1 and 2 of the second.

These methods follow a similar interface as the guide* methods, with one available for each diff* method except for diffCsv since that one uses diffPrint internally. The unsemantic differences are added back after the diff for display purposes, and are colored in grey to indicate they are ignored in the diff.

Currently only trimPrint and trimStr do anything meaningful. trimPrint removes row index headers provided that they are of the default un-named variety. If you add row names, or if numeric row indices are not ascending from 1, they will not be stripped as those have meaning. trimStr removes the '..$', '..-', and '..@' tokens to minimize spurious matches.

You can modify how text is trimmed by providing your own functions to the trim argument of the diff* methods, or by defining trim* methods for your objects. Note that the return value for these functions is the start and end columns of the text that should be *kept* and used in the diff.

As with guides, trimming is on a best efforts basis and may fail with "pathological" display representations. Since the diff still works even with failed trimming this is considered an acceptable compromise. Trimming is more likely to fail with nested recursive structures.

Value

a `length(obj.as.chr)` row and 2 column integer matrix with the start (first column) and end (second column) character positions of the sub string to run diffs on.

Note

`obj.as.chr` will be as processed by `strip_hz_control` and as such will not be identical to the captured output if it contains tabs, newlines, or carriage returns.

`view_or_browse` *Invoke IDE Viewer If Available, browseURL If Not*

Description

Use `getOption("viewer")` to view HTML output if it is available as per RStudio. Fallback to `browseURL` if not available.

Usage

```
view_or_browse(url)
```

Arguments

`url`	character(1L) a location containing a file to display

Value

the return vaue of `getOption("viewer")` if it is a function, or of `browseURL` if the viewer is not available

---

`webfiles` *Return Location of Default HTML Support Files*

Description

File location for default CSS and JS files. Note that these files are read and injected into the output HTML rather than referenced to simplify serving.

Usage

```
diffobj_css()

diffobj_js()
```

Value

path to the default CSS or JS file

Examples

```
diffobj_css()
diffobj_js()
```

[,Diff,numeric,missing,missing-method

Subsetting Methods for Diff Objects

Description

Methods to subset the character representation of the diff output. The subsetting bears no link to the line numbers in the diffs, only to the actual displayed diff.

Usage

```
## S4 method for signature 'Diff,numeric,missing,missing'
x[i]

## S4 method for signature 'Diff'
head(x, n, ...)

## S4 method for signature 'Diff'
tail(x, n, ...)
```

Arguments

x	Diff object
i	subsetting index, must be numeric
n	integer(1L), the size for the resulting object
...	unused, for compatibility with generics

Details

[only supports numeric indices, and returns without error if you specify out of bound indices. If you apply multiple subsetting methods they will be applied in the following order irrespective of what order you actually specify them in: [, then head, then tail. If you use the same subsetting method multiple times on the same object, the last call will define the outcome.

These methods are implemented by storing the chosen indices in the Diff object and using them to subset the as.character output. This mechanism explains the seemingly odd behavior documented above.

Value

Diff object with subsetting indices recorded for use by show ## ‘pager="off"‘ for CRAN compliance; you may omit in normal use diff <- diffChr(letters, LETTERS, format="raw", pager="off") diff[5:15] head(diff, 5) tail(diff, 5) head(head(diff, 5), 8) ## note not ’typical’ behavior

---

[<-,PaletteOfStyles-method

Extract/Replace a Style Class or Object from PaletteOfStyles

Description

Extract/Replace a Style Class or Object from PaletteOfStyles

Usage

```
## S4 replacement method for signature 'PaletteOfStyles'
x[i, j, ...] <- value

## S4 method for signature 'PaletteOfStyles,ANY,ANY,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'PaletteOfStyles'
x[[i, j, ..., exact = TRUE]]
```

Arguments

x	a PaletteOfStyles object
i	numeric, or character corresponding to a valid style format
j	numeric, or character corresponding to a valid style brightness
...	pass a numeric or character corresponding to a valid color.mode
value	a *list* of Style class or Style objects
drop	TRUE or FALSE, whether to drop dimensions, defaults to FALSE, which is different than generic
exact	passed on to generic

Value

a Style ClassRepresentation object or Style object for [[, and a list of the same for [

See Also

diffPrint for explanations of format, brightness, and color.mode

Examples

```
pal <- PaletteOfStyles()
pal[["ansi256", "light", "rgb"]]
pal["ansi256", "light", ]
pal["ansi256", "light", "rgb"] <- list(StyleAnsi8NeutralRgb())
```

Index