Package 'gtools'

June 13, 2022

Title Various R Programming Tools

Description Functions to assist in R programming, including:
- assist in developing, updating, and maintaining R and R packages ('ask', 'checkRVersion', 'getDependencies', 'keywords', 'scat'),
- calculate the logit and inverse logit transformations ('logit', 'inv.logit'),
- test if a value is missing, empty or contains only NA and NULL values ('invalid'),
- manipulate R's .Last function ('addLast'),
- define macros ('defmacro'),
- detect odd and even integers ('odd', 'even'),
- convert strings containing non-ASCII characters (like single quotes) to plain ASCII ('ASCIIfy'),
- perform a binary search ('binsearch'),
- sort strings containing both numeric and character components ('mixedsort'),
- create a factor variable from the quantiles of a continuous variable ('quantcut'),
- enumerate permutations and combinations ('combinations', 'permutation'),
- calculate and convert between fold-change and log-ratio ('foldchange', 'logratio2foldchange', 'foldchange2logratio'),
- calculate probabilities and generate random numbers from Dirichlet distributions ('rdirichlet', 'ddirichlet'),
- apply a function over adjacent subsets of a vector ('running'),
- modify the TCP_NODELAY ('de-Nagle') flag for socket objects,
- efficient 'rbind' of data frames, even if the column names don't match ('smartbind'),
- generate significance stars from p-values ('stars.pval'),
- convert characters to/from ASCII codes ('asc', 'chr'),
- convert character vector to ASCII representation ('ASCIIfy'),
- apply title capitalization rules to a character vector ('capwords').

Version 3.9.2.2

Date 2022-05-28

Author Gregory R. Warnes, Ben Bolker, Thomas Lumley and CRAN team

Maintainer ORPHANED

License GPL-2

Depends R (>= 2.10), methods, stats, utils

URL https://github.com/r-gregmisc/gtools

BugReports https://github.com/r-gregmisc/gtools/issues

Language en-US

Suggests car, gplots, knitr, rstudioapi, SGP, taxize

RoxygenNote 7.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-06-13 15:09:05

R topics documented:

---

asc *Convert between characters and ASCII codes*

Description

Convert between characters and ASCII codes

Usage

```
asc(char, simplify = TRUE)

chr(ascii)
```

Arguments

char	vector of character strings
simplify	logical indicating whether to attempt to convert the result into a vector or matrix object. See sapply for details.
ascii	vector or list of vectors containing integer ASCII codes

Value

asc returns the integer ASCII values for each character in the elements of char. If simplify=FALSE the result will be a list containing one vector per element of char. If simplify=TRUE, the code will attempt to convert the result into a vector or matrix.

asc returns the characters corresponding to the provided ASCII values.

Functions

- asc: return the characters corresponding to the specified ASCII codes
- chr: return the ASCII codes for the specified characters.

Author(s)

Adapted by Gregory R. Warnes <greg@warnes.net> from code posted on the 'Data Debrief' blog on 2011-03-09 at https://datadebrief.blogspot.com/2011/03/ascii-code-table-in-r.html.

See Also

strtoi, charToRaw, rawToChar, as.raw

Examples

```
## ascii codes for lowercase letters
asc(letters)

## uppercase letters from ascii codes
chr(65:90)

## works on muti-character strings
(tmp <- asc("hello!"))
chr(tmp)

## Use 'simplify=FALSE' to return the result as a list
(tmp <- asc("hello!", simplify = FALSE))
chr(tmp)

## When simplify=FALSE the results can be...
asc(c("a", "e", "i", "o", "u", "y")) # a vector
asc(c("ae", "io", "uy")) # or a matrix

## When simplify=TRUE the results are always a list...
asc(c("a", "e", "i", "o", "u", "y"), simplify = FALSE)
asc(c("ae", "io", "uy"), simplify = FALSE)
```

ASCIIfy *Convert Characters to ASCII*

Description

Convert character vector to ASCII, replacing non-ASCII characters with single-byte ('\x00') or two-byte ('\u0000') codes.

Usage

```
ASCIIfy(x, bytes = 2, fallback = "?")
```

Arguments

x	a character vector, possibly containing non-ASCII characters.
bytes	either 1 or 2, for single-byte ('\x00') or two-byte ('\u0000') codes.
fallback	an output character to use, when input characters cannot be converted.

Value

A character vector like x, except non-ASCII characters have been replaced with '\x00' or '\u0000' codes.

Note

To render single backslashes, use these or similar techniques:

```
 write(ASCIIfy(x), "file.txt")
cat(paste(ASCIIfy(x), collapse="\n"), "\n", sep="")
```

The resulting strings are plain ASCII and can be used in R functions and datasets to improve package portability.

Author(s)

Arni Magnusson <arnima@hafro.is>

See Also

showNonASCII identifies non-ASCII characters in a character vector.

Examples

```
cities <- c("S\u00e3o Paulo", "Reykjav\u00edk")
print(cities)
ASCIIfy(cities, 1)
ASCIIfy(cities, 2)

athens <- "\u0391\u03b8\u03ae\u03bd\u03b1"
print(athens)
ASCIIfy(athens)
```

---

ask *Display a prompt and collect the user's response*

Description

Display a prompt and collect the user's response

Usage

```
ask(msg = "Press <RETURN> to continue: ", con = stdin())
```

Arguments

msg	Character vector providing the message to be displayed
con	Character connection to query, defaults to stdin().

Details

The prompt message will be displayed, and then `readLines` is used to collect a single input value (possibly empty), which is then returned.

In most situations using the default `con=stdin()` should work properly. Under RStudio, it is necessary to specify `con=file("stdin")` for proper operation.

Value

A character scalar containing the input provided by the user.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`readLines`, `scan`

Examples

```
# use default prompt
ask()

silly <- function() {
  age <- ask("How old aroe you? ")
  age <- as.numeric(age)
  cat("In 10 years you will be", age + 10, "years old!\n")
}
```

---

assert *Defunct Functions in package* gtools

Description

The functions or variables listed here are no longer part of package `gtools`.

Usage

```
assert(...)

capture(expression, collapse = "\n")

sprint(x, ...)
```

Arguments

`expression, collapse, x, ...`
: ignored

Details

- `assert` is a defunct synonym for `stopifnot`.
- `addLast` has been replaced by `lastAdd`, which has the same purpose but applied using different syntax.
- `capture` and `capture.output` have been removed in favor of `capture.output` from the `utils` package.

See Also

`Defunct`, `stopifnot`, `lastAdd`, `capture.output`

---

badDend *Dataset That Crashes Base:::Plot.Dendogram with 'Node Stack Overflow'*

Description

Base:::Plot.Dendogram() will generate a 'Node Stack Overflow' when run on a dendrogram appropriately constructed from this data set.

Format

The format is: num [1:2047, 1:12] 1 2 3 4 5 6 7 8 9 10 ... - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:12] "X" "V1" "V2" "V3" ...

Note

See help page for `unByteCode` to see how to construct the 'bad' dendrogram from this data and how to work around the issue.

Examples

```
data(badDend)
```

`baseOf` *Transform an integer to an array of base-n digits*

Description

Transform an integer to an array of base-n digits

Usage

```
baseOf(v, base = 10, len = 1)
```

Arguments

`v`	A single integer value to be transformed.
`base`	The base to which to transform to.
`len`	The minimal length of the returned array.

Details

This function converts the elements of an integer vector as an array of its digits. The base of the numbering scheme may be changed away from 10, which defines our decimal system, to any other integer value. For base=2, the number is returned in the binary system. The least significant digit has the highest index in the array, i.e. it appears on the right. The highest exponent is at position 1, i.e. left.

To write decimal values in another base is very common in computer science. In particular at the basis 2 the then possible values 0 and 1 are often interpreted as logical false or true. And at the very interface to electrical engineering, it is indicated as an absence or presence of voltage. When several bit values are transported synchronously, then it is common to give every lane of such a data bus a unique 2^x value and interpret it as a number in the binary system. To distinguish 256 characters one once needed 8 bit ("byte"). It is the common unit in which larger non-printable data is presented. Because of the many non-printable characters and the difficulty for most humans to memorize an even longer alphabet, it is presented as two half bytes ("nibble") of 4 bit in a hexadecimal presentation. Example code is shown below.

For statisticians, it is more likely to use bit representations for hashing. A bit set to 1 (TRUE) at e.g. position 2, 9 or 17 is interpreted as the presence of a particular feature combination of a sample. With baseOf, you can refer to the bit combination as a number, which is more easily and more efficiently dealt with than with an array of binary values. The example code presents a counter of combinations of features which may be interpreted as a Venn diagram.

Author(s)

Steffen Moeller <moeller@debian.org>

Examples

```
# decimal representation
baseOf(123)

# binary representation
baseOf(123, base = 2)

# octal representation
baseOf(123, base = 8)

# hexadecimal representation
baseOf(123, base = 16)

# hexadecimal with more typical letter-notation
c(0:9, LETTERS)[baseOf(123, 16)]

# hexadecimal again, now showing a single string
paste(c(0:9, LETTERS)[baseOf(123, 16)], collapse = "")

# decimal representation but filling leading zeroes
baseOf(123, len = 5)

# and converting that back
sum(2^(4:0) * baseOf(123, len = 5))

# hashing and a tabular venn diagram derived from it
m <- matrix(sample(c(FALSE, TRUE), replace = TRUE, size = 300), ncol = 4)
colnames(m) <- c("strong", "colorful", "nice", "humorous")
names(dimnames(m)) <- c("samples", "features")
head(m)

m.val <- apply(m, 1, function(X) {
  return(sum(2^((ncol(m) - 1):0) * X))
})
m.val.rle <- rle(sort(m.val))
m.counts <- cbind(
  baseOf(m.val.rle$value, base = 2, len = ncol(m)),
  m.val.rle$lengths
)
colnames(m.counts) <- c(colnames(m), "num")
rownames(m.counts) <- apply(m.counts[, 1:ncol(m)], 1, paste, collapse = "")
m.counts[1 == m.counts[, "nice"] & 1 == m.counts[, "humorous"], , drop = FALSE]
m.counts[, "num", drop = TRUE]
```

`binsearch` *Binary Search*

Description

Search within a specified range to locate an integer parameter which results in the the specified monotonic function obtaining a given value.

Usage

```
binsearch(
  fun,
  range,
  ...,
  target = 0,
  lower = ceiling(min(range)),
  upper = floor(max(range)),
  maxiter = 100,
  showiter = FALSE
)
```

Arguments

`fun`	Monotonic function over which the search will be performed.
`range`	2-element vector giving the range for the search.
`...`	Additional parameters to the function `fun`.
`target`	Target value for `fun`. Defaults to 0.
`lower`	Lower limit of search range. Defaults to `min(range)`.
`upper`	Upper limit of search range. Defaults to `max(range)`.
`maxiter`	Maximum number of search iterations. Defaults to 100.
`showiter`	Boolean flag indicating whether the algorithm state should be printed at each iteration. Defaults to FALSE.

Details

This function implements an extension to the standard binary search algorithm for searching a sorted list. The algorithm has been extended to cope with cases where an exact match is not possible, to detect whether that the function may be monotonic increasing or decreasing and act appropriately, and to detect when the target value is outside the specified range.

The algorithm initializes two variable `lo` and `high` to the extremes values of `range`. It then generates a new value `center` halfway between `lo` and `hi`. If the value of `fun` at `center` exceeds `target`, it becomes the new value for `lo`, otherwise it becomes the new value for `hi`. This process is iterated until `lo` and `hi` are adjacent. If the function at one or the other equals the target, this value is returned, otherwise `lo`, `hi`, and the function value at both are returned.

Note that when the specified target value falls between integers, the *two* closest values are returned. If the specified target falls outside of the specified `range`, the closest endpoint of the range will be returned, and an warning message will be generated. If the maximum number if iterations was reached, the endpoints of the current subset of the range under consideration will be returned.

Value

A list containing:

`call`	How the function was called.
`numiter`	The number of iterations performed
`flag`	One of the strings, "Found", "Between Elements", "Maximum number of iterations reached", "Reached lower boundary", or "Reached upper boundary."
`where`	One or two values indicating where the search terminated.
`value`	Value of the function `fun` at the values of `where`.

Note

This function often returns two values for `where` and `value`. Be sure to check the `flag` parameter to see what these values mean.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`optim`, `optimize`, `uniroot`

Examples

```
### Toy examples

# search for x=10
binsearch(function(x) x - 10, range = c(0, 20))

# search for x=10.1
binsearch(function(x) x - 10.1, range = c(0, 20))

### Classical toy example

# binary search for the index of 'M' among the sorted letters
fun <- function(X) {
  ifelse(LETTERS[X] > "M", 1,
    ifelse(LETTERS[X] < "M", -1, 0)
  )
}

binsearch(fun, range = 1:26)
# returns $where=13
LETTERS[13]

### Substantive example, from genetics
## Not run:
```

```
library(genetics)
# Determine the necessary sample size to detect all alleles with
# frequency 0.07 or greater with probability 0.95.
power.fun <- function(N) 1 - gregorius(N = N, freq = 0.07)$missprob

binsearch(power.fun, range = c(0, 100), target = 0.95)

# equivalent to
gregorius(freq = 0.07, missprob = 0.05)

## End(Not run)
```

`capwords` *Capitalize Words for Titles*

Description

This function capitalizes words for use in titles

Usage

```
capwords(
  s,
  strict = FALSE,
  AP = TRUE,
  onlyfirst = FALSE,
  preserveMixed = FALSE,
  sep = " "
)
```

Arguments

`s`	character string to be processed
`strict`	Logical, remove all additional capitalization.
`AP`	Logical, apply the Associated Press (AP) rules for prepositions and conjunctions that should not be capitalized in titles.
`onlyfirst`	Logical, only capitalize the first word.
`preserveMixed`	Logical, preserve the capitalization mixed-case words containing an upper-case letter after a lower-case letter.
`sep`	Character string, word separator

Details

This function separates the provided character string into separate words using `sep` as the word separator. If `firstonly==TRUE`, it then capitalizes the first letter the first word, otherwise (the default), it capitalizes the first letter of every word. If `AP==TRUE`, it then un-capitalizes words in the Associated Press's (AP) list of prepositions and conjunctions should not be capitalized in titles. Next, it capitalizes the first word. It then re-joins the words using the specified separator.

If `preserveMixed==TRUE`, words with an upper-case letter appearing after a lower-case letter will not be changed (e.g. "iDevice").

Value

A character scalar containing the capitalized words.

Author(s)

Gregory R. Warnes <greg@warnes.net> based on code from the `chartr` manual page, and `taxize_capwords` in the taxize package.

References

Fogarty, Mignon. Capitalizing Titles: "Which words should you capitalize? Grammar Girl's Quick and Dirty Tips for Better Writing. 9 Jun. 2011. Quick and Dirty Tips Website." Accessed 22 April 2016 https://www.quickanddirtytips.com/education/grammar/capitalizing-titles

See Also

`chartr`, `taxize_capwords`, `capwords`

Examples

```
capwords("a function to capitalize words in a title")
capwords("a function to capitalize words in a title", AP = FALSE)

capwords("testing the iProduct for defects")
capwords("testing the iProduct for defects", strict = TRUE)
capwords("testing the iProduct for defects", onlyfirst = TRUE)
capwords("testing the iProduct for defects", preserveMixed = TRUE)

capwords("title_using_underscores_as_separators", sep = "_")
```

checkRVersion *Check if a newer version of R is available*

Description

Check if a newer version of R is available

Usage

```
checkRVersion(quiet = FALSE)
```

Arguments

quiet	Logical indicating whether printed output should be suppressed.

Details

This function accesses the R web site to discover the latest released version of R. It then compares this version to the running version. If the running version is the same as the latest version, it prints the message, "The latest version of R is installed:" followed by the version number, and returns NULL. If the running version is older than the current version, it displays the message, "A newer version of R is now available:" followed by the corresponding version number, and returns the version number.

If `quiet=TRUE`, no printing is performed.

Value

Either the version number of the latest version of R, if the running version is less than the latest version, or NULL.

Note

This function utilizes the internet to access the R project web site. If internet access is unavailable, the function will fail.

Author(s)

Gregory R. Warnes <gregory.warnes@rochester.edu>

See Also

R.Version

Examples

```
checkRVersion()

ver <- checkRVersion()
print(ver)
```

`combinations` *Enumerate the Combinations or Permutations of the Elements of a Vector*

Description

`combinations` enumerates the possible combinations of a specified size from the elements of a vector. `permutations` enumerates the possible permutations.

Usage

```
combinations(n, r, v = 1:n, set = TRUE, repeats.allowed = FALSE)

permutations(n, r, v = 1:n, set = TRUE, repeats.allowed = FALSE)
```

Arguments

`n`	Size of the source vector
`r`	Size of the target vectors
`v`	Source vector. Defaults to `1:n`
`set`	Logical flag indicating whether duplicates should be removed from the source vector `v`. Defaults to `TRUE`.
`repeats.allowed`	Logical flag indicating whether the constructed vectors may include duplicated values. Defaults to `FALSE`.

Details

Caution: The number of combinations and permutations increases rapidly with `n` and `r`!.

To use values of `n` above about 45, you will need to increase R's recursion limit. See the `expression` argument to the `options` command for details on how to do this.

Taken from an email by Brian D Ripley <ripley@stats.ox.ac.uk> to r-help dated Tue, 14 Dec 1999 11:14:04 +0000 (GMT) in response to Alex Ahgarin <datamanagement@email.com>. Original version was named "subsets" and was Written by Bill Venables.

Value

Returns a matrix where each row contains a vector of length `r`.

Author(s)

Original versions by Bill Venables <Bill.Venables@cmis.csiro.au>. Extended to handle repeats.allowed by Gregory R. Warnes <greg@warnes.net>.

References

Venables, Bill. "Programmers Note", R-News, Vol 1/1, Jan. 2001. https://cran.r-project.org/doc/Rnews/

See Also

choose, options

Examples

```
combinations(3,2,letters[1:3])
combinations(3,2,letters[1:3],repeats=TRUE)

permutations(3,2,letters[1:3])
permutations(3,2,letters[1:3],repeats=TRUE)

## Not run:
# To use large 'n', you need to change the default recusion limit
options(expressions=1e5)
cmat <- combinations(300,2)
dim(cmat) # 44850 by 2

## End(Not run)
```

---

defmacro *Define a macro*

Description

defmacro define a macro that uses R expression replacement

Usage

```
defmacro(..., expr)

strmacro(..., expr, strexpr)
```

Arguments

...	macro argument list
expr	R expression defining the macro body
strexpr	character string defining the macro body

Details

`strmacro` define a macro that uses string replacement

`defmacro` and `strmacro` create a macro from the expression given in `expr`, with formal arguments given by the other elements of the argument list.

A macro is similar to a function definition except for handling of formal arguments. In a function, formal arguments are simply variables that contains the result of evaluating the expressions provided to the function call. In contrast, macros actually modify the macro body by *replacing* each formal argument by the expression (`defmacro`) or string (`strmacro`) provided to the macro call.

For `defmacro`, the special argument name `DOTS` will be replaced by `...` in the formal argument list of the macro so that `...` in the body of the expression can be used to obtain any additional arguments passed to the macro. For `strmacro` you can mimic this behavior providing a `DOTS=""` argument. This is illustrated by the last example below.

Macros are often useful for creating new functions during code execution.

Value

A macro function.

Note

Note that because [the defmacro code] works on the parsed expression, not on a text string, defmacro avoids some of the problems of traditional string substitution macros such as `strmacro` and the C preprocessor macros. For example, in

```
mul <- defmacro(a, b, expr={a*b})
```

a C programmer might expect `mul(i, j + k)` to expand (incorrectly) to `i*j + k`. In fact it expands correctly, to the equivalent of `i*(j + k)`.

For a discussion of the differences between functions and macros, please Thomas Lumley's R-News article (reference below).

Author(s)

Thomas Lumley wrote `defmacro`. Gregory R. Warnes <greg@warnes.net> enhanced it and created `strmacro`.

References

The original `defmacro` code was directly taken from:

Lumley T. "Programmer's Niche: Macros in R", R News, 2001, Vol 1, No. 3, pp 11–13, https://cran.r-project.org/doc/Rnews/

See Also

`function` `substitute`, `eval`, `parse`, `source`, `parse`,

Examples

```
####
# macro for replacing a specified missing value indicator with NA
# within a dataframe
###
setNA <- defmacro(df, var, values,
  expr = {
    df$var[df$var %in% values] <- NA
  }
)

# create example data using 999 as a missing value indicator
d <- data.frame(
  Grp = c("Trt", "Ctl", "Ctl", "Trt", "Ctl", "Ctl", "Trt", "Ctl", "Trt", "Ctl"),
  V1 = c(1, 2, 3, 4, 5, 6, 999, 8, 9, 10),
  V2 = c(1, 1, 1, 1, 1, 2, 999, 2, 999, 999),
  stringsAsFactors = TRUE
)
d

# Try it out
setNA(d, V1, 999)
setNA(d, V2, 999)
d


###
# Expression macro
###
plot.d <- defmacro(df, var, DOTS,
  col = "red", title = "", expr =
    plot(df$var ~ df$Grp, type = "b", col = col, main = title, ...)
)

plot.d(d, V1)
plot.d(d, V1, col = "blue")
plot.d(d, V1, lwd = 4) # use optional 'DOTS' argument

###
# String macro (note the quoted text in the calls below)
#
# This style of macro can be useful when you are reading
# function arguments from a text file
###
plot.s <- strmacro(DF, VAR,
  COL = "'red'", TITLE = "''", DOTS = "", expr =
    plot(DF$VAR ~ DF$Grp, type = "b", col = COL, main = TITLE, DOTS)
)

plot.s("d", "V1")
plot.s(DF = "d", VAR = "V1", COL = '"blue"')
```

```
plot.s("d", "V1", DOTS = "lwd=4") # use optional 'DOTS' argument



#######
# Create a macro that defines new functions
######
plot.sf <- defmacro(
  type = "b", col = "black",
  title = deparse(substitute(x)), DOTS, expr =
    function(x, y) plot(x, y, type = type, col = col, main = title, ...)
)

plot.red <- plot.sf(col = "red", title = "Red is more Fun!")
plot.blue <- plot.sf(col = "blue", title = "Blue is Best!", lty = 2)

plot.red(1:100, rnorm(100))
plot.blue(1:100, rnorm(100))
```

`dirichlet` *Functions for the Dirichlet Distribution*

Description

Functions to compute the density of or generate random deviates from the Dirichlet distribution

Usage

```
ddirichlet(x, alpha)

rdirichlet(n, alpha)
```

Arguments

`x`	A vector containing a single random deviate or matrix containing one random deviate per row.
`alpha`	Vector or (for `ddirichlet`) matrix containing shape parameters.
`n`	Number of random vectors to generate.

Details

The Dirichlet distribution is the multidimensional generalization of the beta distribution. It is the canonical Bayesian distribution for the parameter estimates of a multinomial distribution.

Value

`ddirichlet` returns a vector containing the Dirichlet density for the corresponding rows of `x`.

`rdirichlet` returns a matrix with `n` rows, each containing a single Dirichlet random deviate.

Functions

- `ddirichlet`: Dirichlet distribution function.
- `rdirichlet`: Generate dirichlet random deviates.

Author(s)

Code original posted by Ben Bolker to R-News on Fri Dec 15 2000. See https://stat.ethz.ch/pipermail/r-help/2000-December/009561.html. Ben attributed the code to Ian Wilson <i.wilson@maths.abdn.ac.uk>. Subsequent modifications by Gregory R. Warnes <greg@warnes.net>.

See Also

dbeta, rbeta

Examples

```
x <- rdirichlet(20, c(1, 1, 1))

ddirichlet(x, c(1, 1, 1))
```

ELISA *Data from an ELISA assay*

Description

Observed signals and (for some observations) nominal concentrations for samples that were aliquoted to multiple assay plates, which were read multiple times on multiple days.

Format

a data frame with the following columns:

- PlateDay factor. Specifies one of four physically distinct 96 well plates
- Read factor. The signal was read 3 times for each plate.
- Description character. Indicates contents of sample.
- Concentration numeric. Nominal concentration of standards (NA for all other samples).
- Signal numeric. Assay signal. Specifically, optical density (a colorimetric assay).

Source

Anonymized data.

`foldchange` *Compute fold-change or convert between log-ratio and fold-change.*

Description

`foldchange` computes the fold change for two sets of values. `logratio2foldchange` converts values from log-ratios to fold changes. `foldchange2logratio` does the reverse.

Usage

```
foldchange(num, denom)

logratio2foldchange(logratio, base = 2)

foldchange2logratio(foldchange, base = 2)
```

Arguments

`num, denom`	vector/matrix of numeric values
`logratio`	vector/matrix of log-ratio values
`base`	Exponential base for the log-ratio.
`foldchange`	vector/matrix of fold-change values

Details

Fold changes are commonly used in the biological sciences as a mechanism for comparing the relative size of two measurements. They are computed as: $\frac{num}{denom}$ if $num > denom$, and as $\frac{-denom}{num}$ otherwise.

Fold-changes have the advantage of ease of interpretation and symmetry about $num = denom$, but suffer from a discontinuity between -1 and 1, which can cause significant problems when performing data analysis. Consequently statisticians prefer to use log-ratios.

Value

A vector or matrix of the same dimensions as the input containing the converted values.

Functions

- `foldchange`: Compute fold-change.
- `logratio2foldchange`: Compute foldchange from log-ratio values.
- `foldchange2logratio`: Compute log-ratio from fold-change values.

Author(s)

Gregory R. Warnes <greg@warnes.net>

Examples

```
a <- 1:21
b <- 21:1

f <- foldchange(a, b)

cbind(a, b, f)
```

`getDependencies` *Get package dependencies*

Description

Get package dependencies

Usage

```
getDependencies(
  pkgs,
  dependencies = c("Depends", "Imports", "LinkingTo"),
  installed = TRUE,
  available = TRUE,
  base = FALSE,
  recommended = FALSE
)
```

Arguments

`pkgs`	character vector of package names
`dependencies`	character vector of dependency types to include. Choices are "Depends", "Imports", "LinkingTo", "Suggests", and "Enhances". Defaults to `c("Depends", "Imports", "LinkingTo")`.
`installed`	Logical indicating whether to pull dependency information from installed packages. Defaults to TRUE.
`available`	Logical indicating whether to pull dependency information from available packages. Defaults to TRUE.
`base`	Logical indicating whether to include dependencies on base packages that are included in the R installation. Defaults to FALSE.
`recommended`	Logical indicating whether to include dependencies on recommended packages that are included in the R installation. Defaults to FALSE.

Details

This function recursively constructs the list of dependencies for the packages given by `pkgs`. By default, the dependency information is extracted from both installed and available packages. As a consequence, it works both for local and CRAN packages.

Value

A character vector of package names.

Note

If `available=TRUE` R will attempt to access the currently selected CRAN repository, prompting for one if necessary.

Author(s)

Gregory R. Warnes <greg@warnes.net> based on the non exported `utils:::getDependencies` and `utils:::.clean_up_dependencies2`.

See Also

`installed.packages`, `available.packages`

Examples

```
## A locally installed package
getDependencies("MASS", installed = TRUE, available = FALSE)
## Not run:
## A package on CRAN
getDependencies("gregmisc", installed = FALSE, available = TRUE)

## End(Not run)

## Show base and recommended dependencies
getDependencies("MASS", available = FALSE, base = TRUE, recommended = TRUE)
## Not run:
## Download the set of packages necessary to support a local package
deps <- getDependencies("MyLocalPackage", available = FALSE)
download.packages(deps, destdir = "./R_Packages")

## End(Not run)
```

---

gtools *gtools: Various R Programming Tools*

Description

Functions to assist in R programming, including:

developing, updating, and maintaining R and R packages 'ask', 'checkRVersion', 'getDependencies', 'keywords', 'scat'

calculate the logit and inverse logit transformations 'logit', 'inv.logit'

test if a value is missing, empty, contains only NA and NULL values, or is a 'try-error' 'invalid'

manipulate R's .Last function 'addLast'

define macros 'defmacro', 'strmacro'

detect odd and even integers 'odd', 'even'

convert strings containing non-ASCII characters (like single quotes) to plain ASCII 'ASCIIfy'

perform a binary search 'binsearch'

sort strings containing both numeric and character components 'mixedsort', 'mixedorder'

create a factor variable from the quantiles of a continuous variable 'quantcut'

enumerate permutations and combinations 'combinations', 'permutation'

calculate and convert between fold-change and log-ratio 'foldchange', 'logratio2foldchange', 'foldchange2logratio'

calculate probabilities and generate random numbers from Dirichlet distributions 'rdirichlet', 'ddirichlet'

apply a function over adjacent subsets of a vector 'running'

modify the TCP_NODELAY ('de-Nagle') flag for socket objects 'tcpNoDelay'

efficient 'rbind' of data frames, even if the column names don't match 'smartbind'

generate significance stars from p-values 'stars.pval'

convert characters to/from ASCII codes asc, chr

convert character vector to ASCII representation 'ASCIIfy'

apply title capitalization rules to a character vector 'capwords'

`gtools-deprecated` *Deprecated Functions in the gtools package*

Description

These functions are provided for compatibility with older versions of gtools, and may be defunct as soon as the next release.

Details

gtools currently contains no deprecated functions.

`help("oldName-deprecated")` (note the quotes).

See Also

`Deprecated`

`invalid` *Test if a value is missing, empty, contains only NA or NULL values, or is a try-error.*

Description

Test if a value is missing, empty, contains only NA or NULL values, or is a try-error.

Usage

```
invalid(x)
```

Arguments

`x`	value to be tested

Value

Logical value.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`missing`, `is.na`, `is.null`

Examples

```
invalid(NA)
invalid()
invalid(c(NA, NA, NULL, NA))

invalid(list(a = 1, b = NULL))

x <- try(log("A"))
invalid(x)

# example use in a function
myplot <- function(x, y) {
  if (invalid(y)) {
    y <- x
    x <- 1:length(y)
  }
  plot(x, y)
}
```

```
myplot(1:10)
myplot(1:10, NA)
```

---

keywords *List valid keywords for R man pages*

Description

List valid keywords for R man pages

Usage

```
keywords(topic)
```

Arguments

topic	object or man page topic

Details

If `topic` is provided, return a list of the keywords associated with `topic`. Otherwise, display the list of valid R keywords from the R doc/KEYWORDS file.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

help

Examples

```
## Show all valid R keywords
keywords()

## Show keywords associated with the 'merge' function
keywords(merge)
keywords("merge")
```

`lastAdd` *Non-destructively construct a .Last function to be executed when R exits.*

Description

Non-destructively construct a `.Last` function to be executed when R exits.

Usage

```
lastAdd(fun)
```

Arguments

`fun`	Function to be called.

Details

`lastAdd` constructs a new function which can be used to replace the existing definition of `.Last`, which will be executed when R terminates normally.

If a `.Last` function already exists in the global environment, the original definition is stored in a private environment, and the new function is defined to call the function `fun` and then to call the previous (stored) definition of `.Last`.

If no `.Last` function exists in the global environment, `lastAdd` simply returns the function `fun`.

Value

A new function to be used for `.Last`.

Note

This function replaces the (now defunct) `addLast` function.

Author(s)

Gregory R. Warnes `<greg@warnes.net>`

See Also

`.Last`

Examples

```
## Print a couple of cute messages when R exits.
helloWorld <- function() cat("\nHello World!\n")
byeWorld <- function() cat("\nGoodbye World!\n")

.Last <- lastAdd(byeWorld)
.Last <- lastAdd(helloWorld)

## Not run:
q("no")

## Should yield:
##
##   Save workspace image? [y/n/c]: n
##
##   Hello World!
##
##   Goodbye World!
##
##   Process R finished at Tue Nov 22 10:28:55 2005

## End(Not run)
```

`loadedPackages` *Provide Name, Version, and Path of Loaded Package Namespaces*

Description

Provide name, version, and path of loaded package namespaces

Usage

```
loadedPackages(silent = FALSE)
```

Arguments

`silent`	Logical indicating whether the results should be printed

Value

Invisibly returns a data frame containing one row per loaded package namespace, with columns:

`Package`	Package name
`Version`	Version string
`Path`	Path to package files

SearchPath — Either the index of the package namespace in the current search path, or '-' if the package namespace is not in the search path. '1' corresponds to the top of the search path (the first namespace searched for values).

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

loadedNamespaces, packageVersion, search, find.package

Examples

```
loadedPackages()
```

---

logit — *Generalized logit and inverse logit function*

Description

Compute generalized logit and generalized inverse logit functions.

Usage

```
logit(x, min = 0, max = 1)

inv.logit(x, min = 0, max = 1)
```

Arguments

x	value(s) to be transformed
min	Lower end of logit interval
max	Upper end of logit interval

Details

The generalized logit function takes values on [min, max] and transforms them to span [-Inf,Inf] it is defined as:

$$y = log(\frac{p}{(1-p)})$$

where

$$p = \frac{(x - min)}{(max - min)}$$

The generalized inverse logit function provides the inverse transformation:

$$x = p'(max - min) + min$$

where

$$p' = \frac{exp(y)}{(1 + exp(y))}$$

Value

Transformed value(s).

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

logit

Examples

```
x <- seq(0, 10, by = 0.25)
xt <- logit(x, min = 0, max = 10)
cbind(x, xt)

y <- inv.logit(xt, min = 0, max = 10)
cbind(x, xt, y)
```

`mixedsort` *Order or Sort strings with embedded numbers so that the numbers are in the correct order*

Description

These functions sort or order character strings containing embedded numbers so that the numbers are numerically sorted rather than sorted by character value. I.e. "Aspirin 50mg" will come before "Aspirin 100mg". In addition, case of character strings is ignored so that "a", will come before "B" and "C".

Usage

```
mixedsort(
  x,
  decreasing = FALSE,
  na.last = TRUE,
  blank.last = FALSE,
  numeric.type = c("decimal", "roman"),
  roman.case = c("upper", "lower", "both"),
  scientific = TRUE
)

mixedorder(
  x,
  decreasing = FALSE,
  na.last = TRUE,
  blank.last = FALSE,
  numeric.type = c("decimal", "roman"),
  roman.case = c("upper", "lower", "both"),
  scientific = TRUE
)
```

Arguments

x	Vector to be sorted.
decreasing	logical. Should the sort be increasing or decreasing? Note that descending=TRUE reverses the meanings of na.last and blanks.last.
na.last	for controlling the treatment of NA values. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed.
blank.last	for controlling the treatment of blank values. If TRUE, blank values in the data are put last; if FALSE, they are put first; if NA, they are removed.
numeric.type	either "decimal" (default) or "roman". Are numeric values represented as decimal numbers (numeric.type="decimal") or as Roman numerals (numeric.type="roman")?
roman.case	one of "upper", "lower", or "both". Are roman numerals represented using only capital letters ('IX') or lower-case letters ('ix') or both?
scientific	logical. Should exponential notation be allowed for numeric values.

Details

I often have character vectors (e.g. factor labels), such as compound and dose, that contain both text and numeric data. This function is useful for sorting these character vectors into a logical order.

It does so by splitting each character vector into a sequence of character and numeric sections, and then sorting along these sections, with numbers being sorted by numeric value (e.g. "50" comes before "100"), followed by characters strings sorted by character value (e.g. "A" comes before "B") *ignoring case* (e.g. 'A' has the same sort order as 'a').

By default, sort order is ascending, empty strings are sorted to the front, and NA values to the end. Setting descending=TRUE changes the sort order to descending and reverses the meanings of na.last and blank.last.

Parsing looks for decimal numbers unless `numeric.type="roman"`, in which parsing looks for roman numerals, with character case specified by `roman.case`.

Value

`mixedorder` returns a vector giving the sort order of the input elements. `mixedsort` returns the sorted vector.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`sort`, `order`

Examples

```
## compound & dose labels
Treatment <- c(
  "Control", "Asprin 10mg/day", "Asprin 50mg/day",
  "Asprin 100mg/day", "Acetomycin 100mg/day",
  "Acetomycin 1000mg/day"
)

## ordinary sort puts the dosages in the wrong order
sort(Treatment)

## but mixedsort does the 'right' thing
mixedsort(Treatment)

## Here is a more complex example
x <- rev(c(
  "AA 0.50 ml", "AA 1.5 ml", "AA 500 ml", "AA 1500 ml",
  "EXP 1", "AA 1e3 ml", "A A A", "1 2 3 A", "NA", NA, "1e2",
  "", "-", "1A", "1 A", "100", "100A", "Inf"
))

mixedorder(x)

mixedsort(x) # Notice that plain numbers, including 'Inf' show up
# before strings, NAs at the end, and blanks at the
# beginning .


mixedsort(x, na.last = TRUE) # default
mixedsort(x, na.last = FALSE) # push NAs to the front


mixedsort(x, blank.last = FALSE) # default
mixedsort(x, blank.last = TRUE) # push blanks to the end
```

```
mixedsort(x, decreasing = FALSE) # default
mixedsort(x, decreasing = TRUE) # reverse sort order

## Roman numerals
chapters <- c(
  "V. Non Sequiturs", "II. More Nonsense",
  "I. Nonsense", "IV. Nonesensical Citations",
  "III. Utter Nonsense"
)
mixedsort(chapters, numeric.type = "roman")

## Lower-case Roman numerals
vals <- c(
  "xix", "xii", "mcv", "iii", "iv", "dcclxxii", "cdxcii",
  "dcxcviii", "dcvi", "cci"
)
(ordered <- mixedsort(vals, numeric.type = "roman", roman.case = "lower"))
roman2int(ordered)

## Control scientific notation for number matching:
vals <- c("3E1", "2E3", "4e0")

mixedsort(vals) # With scientfic notation
mixedsort(vals, scientific = FALSE) # Without scientfic notation
```

`na.replace` *Replace Missing Values*

Description

Replace missing values

Usage

```
na.replace(x, replace, ...)
```

Arguments

`x`	vector possibly containing missing (`NA`) values
`replace`	either a scalar replacement value, or a function returning a scalar value
`...`	Optional arguments to be passed to `replace`

Details

This is a convenience function that is the same as x[is.na(x)] <- replace

Value

Vector with missing values (`NA`) replaced by the value of `replace`.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

is.na, na.omit

Examples

```
x <- c(1, 2, 3, NA, 6, 7, 8, NA, NA)

# Replace with a specified value
na.replace(x, "999")

# Replace with the calculated median
na.replace(x, median, na.rm = TRUE)
```

---

oddeven *Detect odd/even integers*

Description

detect odd/even integers

Usage

```
odd(x)

even(x)
```

Arguments

x	vector of integers

Value

Vector of TRUE/FALSE values.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

round

Examples

```
odd(4)
even(4)

odd(1:10)
even(1:10)
```

`permute` *Randomly Permute the Elements of a Vector*

Description

Randomly Permute the elements of a vector

Usage

```
permute(x)
```

Arguments

`x`	Vector of items to be permuted

Details

This is simply a wrapper function for `sample`.

Value

Vector with the original items reordered.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`sample`

Examples

```
x <- 1:10
permute(x)
```

quantcut *Create a Factor Variable Using the Quantiles of a Continuous Variable*

Description

Create a factor variable using the quantiles of a continuous variable.

Usage

```
quantcut(x, q = 4, na.rm = TRUE, ...)
```

Arguments

x	Continuous variable.
q	Either a integer number of equally spaced quantile groups to create, or a vector of quantiles used for creating groups. Defaults to q=4 which is equivalent to q=seq(0, 1, by=0.25). See quantile for details.
na.rm	Boolean indicating whether missing values should be removed when computing quantiles. Defaults to TRUE.
...	Optional arguments passed to cut.

Details

This function uses quantile to obtain the specified quantiles of x, then calls cut to create a factor variable using the intervals specified by these quantiles.

It properly handles cases where more than one quantile obtains the same value, as in the second example below. Note that in this case, there will be fewer generated factor levels than the specified number of quantile intervals.

Value

Factor variable with one level for each quantile interval.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

cut, quantile

Examples

```
## create example data
# testonly{
set.seed(1234)
# }
x <- rnorm(1000)

## cut into quartiles
quartiles <- quantcut(x)
table(quartiles)

## cut into deciles
deciles.1 <- quantcut(x, 10)
table(deciles.1)
# or equivalently
deciles.2 <- quantcut(x, seq(0, 1, by = 0.1))

# testonly{
stopifnot(identical(deciles.1, deciles.2))
# }

## show handling of 'tied' quantiles.
x <- round(x) # discretize to create ties
stem(x) # display the ties
deciles <- quantcut(x, 10)

table(deciles) # note that there are only 5 groups (not 10)
# due to duplicates
```

`roman2int` *Convert Roman Numerals to Integers*

Description

Convert roman numerals to integers

Usage

```
roman2int(roman)
```

Arguments

`roman`	character vector containing roman numerals

Details

This function will convert roman numerals to integers without the upper bound imposed by R (3899), ignoring case.

Value

A integer vector with the same length as roman. Character strings which are not valid roman numerals will be converted to NA.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

as.roman

Examples

```
roman2int(c("I", "V", "X", "C", "L", "D", "M"))

# works regardless of case
roman2int("MMXVI")
roman2int("mmxvi")

# works beyond R's limit of 3899
val.3899 <- "MMMDCCCXCIX"
val.3900 <- "MMMCM"
val.4000 <- "MMMM"
as.numeric(as.roman(val.3899))
as.numeric(as.roman(val.3900))
as.numeric(as.roman(val.4000))

roman2int(val.3899)
roman2int(val.3900)
roman2int(val.4000)
```

---

running *Apply a Function Over Adjacent Subsets of a Vector*

Description

Applies a function over subsets of the vector(s) formed by taking a fixed number of previous points.

Usage

```
running(
  X,
  Y = NULL,
  fun = mean,
  width = min(length(X), 20),
  allow.fewer = FALSE,
```

```
  pad = FALSE,
  align = c("right", "center", "left"),
  simplify = TRUE,
  by,
  ...
)
```

Arguments

X	data vector
Y	data vector (optional)
fun	Function to apply. Default is mean
width	Integer giving the number of vector elements to include in the subsets. Defaults to the lesser of the length of the data and 20 elements.
allow.fewer	Boolean indicating whether the function should be computed for subsets with fewer than width points
pad	Boolean indicating whether the returned results should be 'padded' with NAs corresponding to sets with less than width elements. This only applies when when allow.fewer is FALSE.
align	One of "right", "center", or "left". This controls the relative location of 'short' subsets with less then width elements: "right" allows short subsets only at the beginning of the sequence so that all of the complete subsets are at the end of the sequence (i.e. 'right aligned'), "left" allows short subsets only at the end of the data so that the complete subsets are 'left aligned', and "center" allows short subsets at both ends of the data so that complete subsets are 'centered'.
simplify	Boolean. If FALSE the returned object will be a list containing one element per evaluation. If TRUE, the returned object will be coerced into a vector (if the computation returns a scalar) or a matrix (if the computation returns multiple values). Defaults to FALSE.
by	Integer separation between groups. If by=width will give non-overlapping windows. Default is missing, in which case groups will start at each value in the X/Y range.
...	parameters to be passed to fun

Details

running applies the specified function to a sequential windows on X and (optionally) Y. If Y is specified the function must be bivariate.

Value

List (if simplify==TRUE), vector, or matrix containing the results of applying the function fun to the subsets of X (running) or X and Y.

Note that this function will create a vector or matrix even for objects which are not simplified by sapply.

Author(s)

Gregory R. Warnes <greg@warnes.net>, with contributions by Nitin Jain <nitin.jain@pfizer.com>.

See Also

wapply to apply a function over an x-y window centered at each x point, sapply, lapply

Examples

```
# show effect of pad
running(1:20, width = 5)
running(1:20, width = 5, pad = TRUE)

# show effect of align
running(1:20, width = 5, align = "left", pad = TRUE)
running(1:20, width = 5, align = "center", pad = TRUE)
running(1:20, width = 5, align = "right", pad = TRUE)

# show effect of simplify
running(1:20, width = 5, fun = function(x) x) # matrix
running(1:20, width = 5, fun = function(x) x, simplify = FALSE) # list

# show effect of by
running(1:20, width = 5) # normal
running(1:20, width = 5, by = 5) # non-overlapping
running(1:20, width = 5, by = 2) # starting every 2nd


# Use 'pad' to ensure correct length of vector, also show the effect
# of allow.fewer.
par(mfrow = c(2, 1))
plot(1:20, running(1:20, width = 5, allow.fewer = FALSE, pad = TRUE), type = "b")
plot(1:20, running(1:20, width = 5, allow.fewer = TRUE, pad = TRUE), type = "b")
par(mfrow = c(1, 1))

# plot running mean and central 2 standard deviation range
# estimated by *last* 40 observations
dat <- rnorm(500, sd = 1 + (1:500) / 500)
plot(dat)
sdfun <- function(x, sign = 1) mean(x) + sign * sqrt(var(x))
lines(running(dat, width = 51, pad = TRUE, fun = mean), col = "blue")
lines(running(dat, width = 51, pad = TRUE, fun = sdfun, sign = -1), col = "red")
lines(running(dat, width = 51, pad = TRUE, fun = sdfun, sign = 1), col = "red")


# plot running correlation estimated by last 40 observations (red)
# against the true local correlation (blue)
sd.Y <- seq(0, 1, length = 500)

X <- rnorm(500, sd = 1)
```

```
Y <- rnorm(500, sd = sd.Y)

plot(running(X, X + Y, width = 20, fun = cor, pad = TRUE), col = "red", type = "s")

r <- 1 / sqrt(1 + sd.Y^2) # true cor of (X,X+Y)
lines(r, type = "l", col = "blue")
```

---

scat *Display debugging text*

Description

If `getOption('DEBUG')==TRUE`, write text to STDOUT and flush so that the text is immediately displayed. Otherwise, do nothing.

Usage

```
scat(...)
```

Arguments

`...`	Arguments passed to `cat`

Value

NULL (invisibly)

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`cat`

Examples

```
options(DEBUG = NULL) # makee sure DEBUG isn't set
scat("Not displayed")

options(DEBUG = TRUE)
scat("This will be displayed immediately (even in R BATCH output \n")
scat("files), provided options()$DEBUG is TRUE.")
```

`script_file` *Determine the directory or full path to the currently executing script*

Description

Determine the directory or full path to the currently executing script

Usage

```
script_file(fail = c("stop", "warning", "quiet"))

script_path(fail = c("stop", "warning", "quiet"))
```

Arguments

`fail`	character, one of "stop", "warning", "quiet". specifying what should be done when the script path cannot be determined: "stop" causes an error to be generated, "warn" generates a warning message and returns NA, "quiet" silently returns NA. These function should work with `Rscript`, `source()`, `Rmarkdown` `RStudio`'s "Run selection", and the `RStudio` Console.

Value

A character scalar containing the full path to the currently executing script file (`script_file`) or its directory (`script_path`). If unable to determine the script path, it generates a warning and returns `""` (empty string).

Functions

- `script_file`: Determine the full path of the currently executing script
- `script_path`: Determine the directory of the currently executing script

Author(s)

Greg Warnes <greg@warnes.net> based on on a Stack Overflow post by jerry-t (https://stackoverflow.com/users/2292993/jerry-t) at https://stackoverflow.com/a/36777602/2744062.

Examples

```
getwd()
commandArgs(trailingOnly = FALSE)

script_file("warning")
script_path("warning")
```

setTCPNoDelay *Modify the TCP_NODELAY ('de-Nagle') flag for socket objects*

Description

Modify the TCP_NODELAY ('de-Nagle') flag for socket objects

Usage

```
setTCPNoDelay(socket, value = TRUE)
```

Arguments

socket	A socket connection object
value	Logical indicating whether to set (TRUE) or unset (FALSE) the flag

Details

By default, TCP connections wait a small fixed interval before actually sending data, in order to permit small packets to be combined. This algorithm is named after its inventor, John Nagle, and is often referred to as 'Nagling'.

While this reduces network resource utilization in these situations, it imposes a delay on all outgoing message data, which can cause problems in client/server situations.

This function allows this feature to be disabled (de-Nagling, value=TRUE) or enabled (Nagling, value=FALSE) for the specified socket.

Value

The character string "SUCCESS" will be returned invisible if the operation was successful. On failure, an error will be generated.

Author(s)

Gregory R. Warnes <greg@warnes.net>

References

"Nagle's algorithm" https://en.wikipedia.org/wiki/Nagle's_algorithm,

Nagle, John. "Congestion Control in IP/TCP Internetworks", IETF Request for Comments 896, January 1984. https://www.ietf.org/rfc/rfc0896.txt?number=896

See Also

make.socket, socketConnection

Examples

```
## Not run:
host <- "www.r-project.org"
socket <- make.socket(host, 80)
print(socket)
setTCPNoDelay(socket, TRUE)

write.socket(socket, "GET /\n\n")
write.socket(socket, "A")
write.socket(socket, "B\n")
while ((str <- read.socket(socket)) > "") {
  cat(str)
}
close.socket(socket)

## End(Not run)
```

`smartbind` *Efficient rbind of data frames, even if the column names don't match*

Description

Efficient rbind of data frames, even if the column names don't match

Usage

```
smartbind(..., list, fill = NA, sep = ":", verbose = FALSE)
```

Arguments

`...`	Data frames to combine
`list`	List containing data frames to combine
`fill`	Value to use when 'filling' missing columns. Defaults to `NA`.
`sep`	Character string used to separate column names when pasting them together.
`verbose`	Logical flag indicating whether to display processing messages. Defaults to `FALSE`.

Value

The returned data frame will contain:

`columns`	all columns present in any provided data frame
`rows`	a set of rows from each provided data frame, with values in columns not present in the given data frame filled with missing (`NA`) values.

The data type of columns will be preserved, as long as all data frames with a given column name agree on the data type of that column. If the data frames disagree, the column will be converted into a character strings. The user will need to coerce such character columns into an appropriate type.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

rbind, cbind

Examples

```
df1 <- data.frame(A = 1:10, B = LETTERS[1:10], C = rnorm(10))
df2 <- data.frame(A = 11:20, D = rnorm(10), E = letters[1:10])

# rbind would fail
## Not run:
rbind(df1, df2)
# Error in match.names(clabs, names(xi)) : names do not match previous
# names:
#  D, E

## End(Not run)
# but smartbind combines them, appropriately creating NA entries
smartbind(df1, df2)

# specify fill=0 to put 0 into the missing row entries
smartbind(df1, df2, fill = 0)
```

---

split_path *Split a File Path into Components*

Description

This function converts a character scalar containing a *valid* file path into a character vector of path components (e.g. directories).

Usage

```
split_path(x, depth_first = TRUE)
```

Arguments

x	character scalar. Path to be processed.
depth_first	logical. Should path be returned depth first? Defaults to TRUE.

Value

Character vector of path components, depth first.

`stars.pval` *Generate significance stars from p-values*

Description

Generate significance stars (e.g. '***', '**', '*', '+') from p-values using R's standard definitions.

Usage

```
stars.pval(p.value)
```

Arguments

`p.value`	numeric vector of p-values

Details

Mapping from p-value ranges to symbols:

0 - 0.001 '***'

0.001 - 0.01 '**'

0.01 - 0.05 '*'

0.05 - 0.1 '+'

0.1 - 1.0 '' (No symbol)

Value

A character vector containing the same number of elements as `p-value`, with an attribute "legend" providing the conversion pattern.

Author(s)

Gregory R. Warnes <greg@warnes.net>

See Also

`symnum`

Examples

```
p.val <- c(0.0004, 0.0015, 0.013, 0.044, 0.067, 0.24)
stars.pval(p.val)
```

stat_mode *Most frequently occurring value*

Description

Most frequently occurring value

Usage

```
stat_mode(x, na.rm = TRUE, ties = c("all", "first", "last", "missing"), ...)
```

Arguments

x	vector of values
na.rm	logical. Should NA values be removed before processing?
ties	character. Which value(s) should be returned in the case of ties?
...	optional additional parameters.

Value

vector of the same class as x

Examples

```
# Character vector
chr_vec <- c("a", "d", "d", "h", "h", NA, NA) # Multiple modes
stat_mode(x = chr_vec)
stat_mode(x = chr_vec, na.rm = FALSE)
stat_mode(x = chr_vec, na.rm = FALSE, ties = "first")
stat_mode(x = chr_vec, na.rm = FALSE, ties = "last")

# - # Numeric vector
# See that it keeps the original vector type
num_vec <- c(2, 3, 3, 4, 4, NA, NA)
stat_mode(x = num_vec)
stat_mode(x = num_vec, na.rm = FALSE)
stat_mode(x = num_vec, na.rm = FALSE, ties = "first")
stat_mode(x = num_vec, na.rm = FALSE, ties = "last")

# The default option is ties="all" but it is very easy for the user to control
# the ties without changing this parameter.
# Select always just one mode, being that the first mode
stat_mode(x = num_vec)[1]

# Select the first and the second stat_mode
stat_mode(x = num_vec)[c(1, 2)]
```

```
# Logical Vectors
stat_mode(x = c(TRUE, TRUE))
stat_mode(x = c(FALSE, FALSE, TRUE, TRUE))

# - # Single element cases
stat_mode(x = c(NA_real_))
stat_mode(x = 2)
stat_mode(x = NA)
stat_mode(x = c("a"))

# Not allowing multiple stat_mode, returning NA if that happens
stat_mode(x = c(1, 1, 2, 2), multiple_modes = FALSE) # multiple stat_mode
stat_mode(x = c(1, 1), multiple_modes = FALSE) # single mode

# Empty vector cases
# The ties of any empty vector will be itself (an empty vector of the same type)
stat_mode(x = double())
stat_mode(x = complex())
stat_mode(x = vector("numeric"))
stat_mode(x = vector("character"))
```

---

unByteCode *Convert a byte-code function to an interpreted-code function*

Description

The purpose of these functions is to allow a byte coded function to be converted back into a fully interpreted function as a *temporary* work around for issues in byte-code interpretation.

Usage

```
unByteCode(fun)

assignEdgewise(name, env, value)

unByteCodeAssign(fun)
```

Arguments

fun	function to be modified
name	object name
env	namespace
value	new function body

Details

unByteCode returns a copy of the function that is directly interpreted from text rather than from byte-code.

assignEdgewise makes an assignment into a locked environment.

unByteCodeAssign changes the specified function *in its source environment* to be directly interpreted from text rather than from byte-code.

Value

All three functions return a copy of the modified function or assigned value.

Note

These functions are not intended as a permanent solution to issues with byte-code compilation or interpretation. Any such issues should be promptly reported to the R maintainers via the R Bug Tracking System at https://bugs.r-project.org and via the R-devel mailing list https://stat.ethz.ch/mailman/listinfo/r-devel.

Author(s)

Gregory R. Warnes <greg@warnes.net>

References

These functions were inspired as a work-around to R bug https://bugs.r-project.org/bugzilla/show_bug.cgi?id=15215.

See Also

disassemble, assign

Examples

```
data(badDend)
dist2 <- function(x) as.dist(1 - cor(t(x), method = "pearson"))
hclust1 <- function(x) hclust(x, method = "single")

distance <- dist2(badDend)
cluster <- hclust1(distance)

dend <- as.dendrogram(cluster)
## Not run:
## In R 2.3.0 and earlier crashes with a node stack overflow error
plot(dend)
## Error in xy.coords(x, y, recycle = TRUE) : node stack overflow

## End(Not run)

## convert stats:::plotNode from byte-code to interpreted-code
```

```
unByteCodeAssign(stats:::plotNode)

# increase recursion limit
options("expressions" = 5e4)

# now the function does not crash
plot(dend)
```

Index