Package 'NCmisc'

February 25, 2014

Type Package

Version 1.1

Date 2013-07-25

Title Miscellaneous general purpose functions

Author Nicholas Cooper

Maintainer Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Depends R (>= 2.10), grDevices, graphics, stats, utils, tools

Suggests proftools

Description A set of handy functions. Includes: versatile one line progress bar, one line function timer with detailed output, time delay function, text histogram, object preview, CRAN package search, simpler package installer, Linux command install check,a flexible Mode function, top function, simulation of correlated data, and more

License GPL (>= 2)

R topics documented:

`NCmisc-package` *Miscellaneous one line functions*

Description

A set of handy functions. Includes: versatile one line progress bar, one line function timer with detailed output, time delay function, text histogram, object preview, CRAN package search, simpler package installer, Linux command install check, a flexible Mode function, top function, simulation of correlated data, and more

Details

Package:	NCmisc
Type:	Package
Version:	1.0
Date:	2013-07-25
License:	GPL (>= 2)

A package of general purpose functions that might save time or help tidy up code. Some of these functions are similar to existing functions but are simpler to use or have more features (e.g, timeit and loop.tracker reduce an initialisation, 'during' and close three-line call structure, to a single function call. Also, some of these functions are useful for building packages and pipelines, for instance: Header(), to provide strong visual deliniation between procedures in console output, by an

ascii bordered heading; loop.tracker() to track the progress of loops (called with only 1 line of code), with the option to periodically backup a key object during the loop; estimate.memory() to determine whether the object may exceed some threshold before creating it, timeit(), a one line wrapper for proftools which gives a detailed breakdown of time taken, and time within each function called during a procedure; and check.linux.install() to verify installation status of terminal commands before using system(), top() to examine current memory and CPU usage [using the system 'top' command]. prv() is useful for debugging as it allows a detailed preview of objects, and is as easy as placing print statements within loops/functions but gives more information, and gives compact output for large objects. For testing sim.cor() provides a simple way to simulate a correlated data matrix, as often this is more realistic than completely random data. Otherwise summarise.r.datasets gives a list of all available datasets and their structure and dimensionality.

List of key functions:

- *check.linux.install* Check whether a given system command is installed (e.g, bash)
- *cor.with* simulate a variable with a specified correlation to an existing variable
- *Dim* same as dim() function but works for more objects, including vectors
- *estimate.memory* Estimate the memory required for an object
- *exists.not.function* same as exists() function but ignores functions
- *get.distinct.cols* Return up to 22 distinct colours
- *hlist* A good way to preview large lists
- *Header* Print heading text with a border
- *loop.tracker* Creates a progess bar within a loop with only 1 line
- *Mode* Find the mode(s) of a vector
- *must.use.package* Do everything possible to load an R package
- *narm* Return an object with missing values removed
- *packages.loaded* quietly test whether packages are loaded without using require
- *pad.left* Print a vector with appropriate padding so each has equal char length
- *pctile* Find data thresholds corresponding to percentiles
- *preview* same as prv, but enter arguments as strings
- *prv.large* tidy representation for large matrices/data.frames
- *prv* compact preview of objects (more complete than 'print')
- *Rfile.index* Create an index file for an R function file
- *rmv.spc* Remove leading and trailing spaces (or other character)
- *search.cran* Search all CRAN packages for those containing keyword(s)
- *sim.cor* simulate a correlated dataset
- *spc* Print a character a specified number of times
- *standardize* Convert a numeric vector to Z-scores
- *Substitute* multivariable version of substitute (base)
- *summarise.r.datasets* show and summarise all available example datasets
- *textogram* Make an ascii histogram in the console

- *timeit* Times an expression, with breakdown of time spent in functions
- *toheader* Return a string with each first letter of each word in upper case
- *top* report on CPU and memory usage, overall or by process
- *Unlist* Unlist a list, starting only from a set depth
- *wait* Wait for a period of time

Author(s)

Nicholas Cooper

Maintainer: Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

reader ~~

Examples

```
#text histogram
textogram(rnorm(10000),range=c(-3,3))
# wait 0.2 seconds
wait(0.2,silent=FALSE)
# see whether a system command is installed
check.linux.install("sed")
# a nice progress bar
max <- 100; for (cc in 1:max) { loop.tracker(cc,max); wait(0.004,"s") }
# nice header
Header(c("SPACE","The final frontier"))
# memory reqd for proposed or actual object
estimate.memory(matrix(rnorm(100),nrow=10))
# a mode function (there isnt one included as part of base)
Mode(c(1,2,3,3,4,4,4))
# search for packages containing text, eg, misc
search.cran("misc",repos="http://cran.ma.imperial.ac.uk/")
# breakdown of processing time using proftools
# not run:  timeit(wait(2,"s") ,total.time=TRUE)
# simulate a correlated dataset
corDat <- sim.cor(200,5)
cor(corDat) # show correlation matrix
prv(corDat) # show preview of matrix
# Dim() versus dim()
Dim(1:10); dim(1:10)
# check whether package is loaded (when not required or dependency)
repos <- "http://cran.ma.imperial.ac.uk/" # or repos <- getOption("repos")
packages.loaded("bigmemory",repos=repos)
```

check.linux.install *Check whether a given system command is installed (e.g, bash)*

Description

Tests whether a command is installed and callable by system(). Will return a warning if run on windows

Usage

```
check.linux.install(cmd = c("plink", "perl", "sed"))
```

Arguments

cmd	list of commands to test

Value

returns true or false for each command 'cmd'

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
check.linux.install("ls") # should be standard
check.linux.install(c("perl","sed","fake-cmd"))
```

cor.with *Simulate a correlated variable*

Description

Simulate a variable correlated at level 'r' with cector x (of the same length). Can either 'preserve' the mean and standard-deviation, leave standardizeed, or select new mean 'mn' and standard deviation 'st'.

Usage

```
cor.with(x, r = 0.5, preserve = FALSE, mn = NA, st = NA)
```

Arguments

x	existing variable, to which you want to simulate a new correlated variable
r	the 'expected' correlation you want to target (randomness will mean that the actual correlation will vary around this value)
preserve	logical, whether to preserve the same mean and standard deviation(SD) as x, for the new variable
mn	optional, set the mean for the new simulated variable [must also set st if using this]
st	optional, set the SD for the new simulated variable [must also set mn if using this]

Value

return the new variable with an expected correlation of 'r' with x

Author(s)

Nicholas Cooper

References

http://www.uvm.edu/~dhowell/StatPages/More_Stuff/CorrGen.html

See Also

sim.cor

Examples

```
X <- rnorm(10,100,14)
cor.with(X,r=.5) # create a variable correlated .5 with X
cor(X,cor.with(X)) # check the actual correlation
# some variability in the actual correlation, so run 1000 times:
print(mean(replicate(1000,{cor(X,cor.with(X))})))
cor.with(X,preserve=TRUE) # preserve original mean and standard deviation
X[c(4,10)] <- NA # works fine with NAs, but new var will have same missing
cor.with(X,mn=50,st=2) # specify new mean and standard deviation
```

Dim *A more general dimension function*

Description

A more general 'dim' function. For arrays simply calls the dim() function, but for other data types, tries to provide an equivalent, for instance will call length(x) for vectors, and will recursively report dims for lists, and will attempt something sensible for other datatypes.

Usage

```
Dim(x, cat.lists = TRUE)
```

Arguments

x	the object to find the dimension for
cat.lists	logical, for lists, TRUE will concatenate the dimesions to a single string, or FALSE will return the sizes as a list of the same structure as the original.

Value

dimension(s) of the object

See Also

prv, preview

Examples

```
# create variables of different types to show output styles #
Dim(193)
Dim(1:10)
testvar <- matrix(rnorm(100),nrow=25)
Dim(matrix(rnorm(100),nrow=25))
Dim(list(first="test",second=testvar,third=100:110))
Dim(list(first="test",second=testvar,third=100:110),FALSE)
```

---

estimate.memory *Estimate the memory required for an object.*

Description

An existing object or just dim/length of a proposed object

Usage

```
estimate.memory(dat)
```

Arguments

dat	either a matrix/dataframe, or else dims; c(nrow,ncol)

Value

returns minimum memory requirement in GB (numeric, scalar)

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
estimate.memory(matrix(rnorm(100),nrow=10))
estimate.memory(c(10^6,10^4))
estimate.memory(5.4*10^8)
```

`exists.not.function` *Does object exist ignoring functions*

Description

The exists() function can tell you whether an object exists at all, or whether an object exists with a certain type, but it can be useful to know whether an object exists as genuine data (and not a function) which can be important when a variable or object is accidently or intentionally given the same name as a function. This function usually returns a logical value as to the existence of the object (ignoring functions) but can also be set to return the non-function type if the object exists.

Usage

```
exists.not.function(x, ret.type = FALSE)
```

Arguments

`x`	the object name to search for
`ret.type`	logical, if TRUE then will return the objects' type (if it exists) rather than TRUE or FALSE. If the object doesn't exist the empty string will be returned as the type.

Value

logical, whether non-function object exists, or else the type if ret.type=TRUE

Author(s)

Nicholas Cooper

Examples

```
x <- "test"
# the standard exists function, for all modes, correct mode, and other modes:
exists("x")
exists("x",mode="character")
exists("x",mode="numeric")
# standard case for a non-function variable
exists.not.function("x",TRUE)
# compare results for a non-existent variable
exists("aVarNotSeen")
exists.not.function("aVarNotSeen")
# compare results for variable that is a function
```

```
exists("mean")
exists.not.function("mean")
# define a variable with same name as a function
mean <- 1.4
# exists.not.function returns the type of the variable ignoring the function of the same name
exists.not.function("mean",TRUE)
exists("mean",mode="function")
exists("mean",mode="numeric")
```

`get.distinct.cols` *Return up to 22 distinct colours.*

Description

Useful if you want to colour 22 autosomes, etc, because most R colour palettes only provide 12 or fewer colours, or else provide, a gradient which is not distinguishable for discrete categories. Manually curated so the most similar colours aren't side by side.

Usage

```
get.distinct.cols(n = 22)
```

Arguments

`n`	number of unique colours to return

Value

returns vector of n colours

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
get.distinct.cols(10)
plot(1:22,pch=19,col=get.distinct.cols(22))
```

Header *Print heading text with a border.*

Description

Makes highly visible headings, can separately horizontal, vertical and corner characters

Usage

```
Header(txt, h = "=", v = h, corner = h, align = "center")
```

Arguments

txt	The text to display in the centre
h	the ascii character to use on the horizontal sections of the border, and used for v,corner too if not specified separately
v	the character to use on vertical sections of the border
corner	the character to use on corner sections of the border
align	alignment of the writing, when there are multiple lines, e.g, "right", "left", "centre"/"center"

Value

returns nothing, simply prints the heading to the console

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
Header("Section 1")
Header("Section 1",h="-",v="|",corner="*")
Header(c("SPACE","The final frontier"))
Header(c("MY SCRIPT","Part 1"),align="left",h=".")
```

headl *A good way to preview large lists.*

Description

An alternative to head(list) which allows limiting of large list components in the console display

Usage

```
headl(x, n = 6, skip = 20, skip2 = 10, ind = "",
  ind2 = "  ")
```

Arguments

x	a list to preview
n	The number of values to display for the deepest nodes of the list
skip	number of first level elements to display before skipping the remainder
skip2	number of subsequent level elements to display before skipping the remainder
ind	indent character for first level elements
ind2	indent character for subsequent level elements

Value

prints truncated preview of a large list

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
sub1 <- list(list(1:100),list(2:101),list(101:200),list(201:300),list(301:400))
big.list <- list(sub1,sub1,sub1,sub1,sub1,sub1)
headl(sub1)
headl(big.list,skip=2)
```

`loop.tracker` *Creates a progess bar within a loop*

Description

Only requires a single line within a loop to run, in contrast with the built-in tracker which requires a line to initialise, and a line to close. Also has option to backup objects during long loops. Ideal for a loop with a counter such as a for loop. Tracks progress as either percentage of time remaining or by intermittently displaying the estimated number of minutes to go

Usage

```
loop.tracker(cc, max, st.time = NULL, sav.obj = NULL,
  sav.fn = NA, sav.freq = 10, unit = c("m", "s", "h")[1])
```

Arguments

`cc`	integer, current value of the loop counter
`max`	integer, final value of the loop counter
`st.time`	'start time' when using 'time to go' mode, taken from a call to proc.time()
`sav.obj`	optionally an object to backup during the course of a very long loop, to restore in the event of a crash.
`sav.fn`	the file name to save 'save.obj'
`sav.freq`	how often to update 'sav.obj' to file, in terms of percentage of run-time
`unit`	time units h/m/s if using 'time to go' mode

Value

returns nothing, simply prints progress to the console

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
# simple example with a for-loop
max <- 100; for (cc in 1:max) { loop.tracker(cc,max); wait(0.004,"s") }
#example using the time to go with a while loop
cc <- 0; max <- 10; start <- proc.time()
while(cc < max) { cc <- cc + 1; wait(0.05,"s"); loop.tracker(cc,max,start,unit="s") }
# example with saving an object, and restoring after a crash
X <- matrix(rnorm(5000),nrow=50); max <- nrow(X); sums <- numeric(max)
for (cc in 1:max) {
  sums[cc] <- sum(X[cc,])
  wait(.05) # just so this trivial loop doesnt finish so quickly
  loop.tracker(cc,max, sav.obj=sums, sav.fn="temp.rda", sav.freq=5);
```

```
  if(cc==29) { warning("faked a crash at iteration 29!"); rm(sums); break }
}
cat("\nloaded latest backup from iteration 28:",paste(load("temp.rda")),"\n")
print(sav.obj); unlink("temp.rda")
```

Mode *Find the mode of a vector.*

Description

The mode is the most common value in a series. This function can return multiple values if there are equally most frequent values, and can also work with non-numeric types.

Usage

```
Mode(x, multi = FALSE, warn = FALSE)
```

Arguments

x	The data to take the mode from. Dimensions and NA's are removed if possible, strings, factors, numeric all permitted
multi	Logical, whether to return multiple modes if values have equal frequency
warn	Logical, whether to give warnings when multiple values are found (if multi=FALSE)

Value

The most frequent value, or sorted set of most frequent values if multi==TRUE and there are more than one. Numeric if x is numeric, else as strings

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
Mode(c(1,2,3,3,4,4)) # 2 values are most common, as multi=FALSE,
# selects the last value (after sort)
Mode(c(1,2,3,3,4,4),multi=TRUE) # same test with multi=T,
# returns both most frequent
Mode(matrix(1:16,ncol=4),warn=TRUE) # takes mode of the entire
# matrix treating as a vector, but all values occur once
Mode(c("Tom","Dick","Harry"),multi=FALSE,warn=TRUE) # selects last
# sorted value, but warns there are multiple modes
Mode(c("Tom","Dick","Harry"),multi=TRUE,warn=TRUE) # multi==TRUE so
# warning is negated
```

`must.use.package` *Do everything possible to load an R package.*

Description

Like 'require()' except it will attempt to install a package if necessary, and will also deal automatically with bioconductor packages too.

Usage

```
must.use.package(pcknms, bioC = FALSE, ask = FALSE,
  reload = FALSE, avail = FALSE, quietly = FALSE)
```

Arguments

`pcknms`	list of packages to load/install, shouldn't mix bioconductor/CRAN in one call
`bioC`	whether the listed packages are from bioconductor
`reload`	indicates to reload the package even if loaded
`avail`	when bioC=FALSE, see whether pcknms are in the list of available CRAN packages
`ask`	whether to get the user's permission to install a required package, or just go ahead and do it
`quietly`	passed to library/require, display installation text or not

Value

nothing, simply loads the packages specified if possible

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
# not run : run if you are ok to install/already have these packages
# must.use.package(c("MASS","nlme","lme4"),ask=FALSE)
# must.use.package("limma",bioC=TRUE)
# search() # show packages have loaded, then detach them again:
# sapply(paste("package",c("limma","MASS","nlme","lme4"),sep=":"),detach,character.only=TRUE)
```

narm *Return an object with missing values removed.*

Description

Convenience function, removes NAs from most standard objects. Uses function na.exclude for matrices and dataframes. Main difference to na.exlude is that it simply performs the transformation, without adding attributes For unknown types, leaves unchanged with a warning.

Usage

```
narm(X)
```

Arguments

X	The object to remove NAs, any vector, matrix or data.frame

Value

Vector minus NA's, or the matrix/data.frame minus NA rows

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
narm(c(1,2,4,NA,5))
DF <- data.frame(x = c(1, 2, 3), y = c(0, 10, NA))
DF; narm(DF)
# if a list, will only completely remove NA from the lowest levels
# empty places will be left at top levels
print(narm(list(1,2,3,NA,list(1,2,3,NA))))
```

packages.loaded *Check whether a set of packages has been loaded*

Description

Returns TRUE if the whole set of packages entered has been loaded, or FALSE otherwise. This can be useful when developing a package where there is optional functionality depending if another package is in use (but the other package is not part of 'depends' because it is not essential). Because 'require' cannot be used within functions submitted as part of a CRAN package.

Usage

```
packages.loaded(pcks = "", ..., cran.check = TRUE,
  repos = NULL)
```

Arguments

pcks	character, a package name, or vector of names, if left blank will return all loaded
...	further package names as character (same as entering via pcks, but avoids need for c() in pcks)
cran.check	logical, in the case at least one package is not found, whether to search CRAN and see whether the package(s) even exist on CRAN.
repos	repository to use if package is not loaded and cran.check=TRUE, if NULL, will attempt to use the repository in getOptions("repos") or will default to the imperial.ac.uk mirror.

Value

logical TRUE or FALSE whether the whole list of packages are available

Author(s)

Nicholas Cooper

Examples

```
repos <- "http://cran.ma.imperial.ac.uk/"
packages.loaded("NCmisc","reader",repos=repos)
packages.loaded(c("bigmisc","nonsenseFailTxt"),repos=repos)
packages.loaded(c("bigmisc","nonsenseFailTxt"),cran.check=FALSE)
packages.loaded() # no argument means all loaded packages are listed
```

---

pad.left *Print a vector with appropriate padding so each has equal char length.*

Description

Print a vector with appropriate padding so each has equal char length.

Usage

```
pad.left(X, char = " ", numdigits = NA)
```

Arguments

X	vector of data to pad to equal length
char	character to pad with, space is default, but zero might be a desirable choice for padding numbers
numdigits	if using numeric data, the number of digits to keep

Value

returns the vector in character format with equal nchar()

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
pad.left(1:10)
phone.numbers <- c("07429719234","7876345123","7123543765")
pad.left(phone.numbers,"0")
pad.left(rnorm(10),numdigits=3)
```

---

pctile *Find data thresholds corresponding to percentiles*

Description

Finds the top and bottom bounds corresponding to percentile 'pc' of the data 'dat'.

Usage

```
pctile(dat, pc = 0.01)
```

Arguments

dat	numeric vector of data
pc	the percentile to seek, c(pc, 1-pc)

Value

returns the upper and lower threshold

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
pctile(rnorm(100000),.025)
pctile(sample(100),.9)
```

preview *Output variable states within functions during testing/debugging*

Description

A versatile function to compactly display most common R objects. Will return the object name, type, dimension, and a compact representation of object contents, for instance using prv.large() to display matrices, so as to not overload the console for large objects. Useful for debugging, can be placed inside loops and functions to track values, dimensions, and data types. Particularly when debugging complex code, the automatic display of the variable name prevents confusion versus using regular print statements. By listing variables to track as character(), provides 'cat()' output of compact and informative variable state information, e.g, variable name, value, datatype and dimension. Can also specify array or list elements, or custom labels. prv() is the same as preview() except it can take objects without using double quotes and has no 'labels' command (and doesn't need one).

Usage

```
preview(varlist, labels = NULL, counts = NULL)
```

Arguments

varlist	character vector, the list of variable(s) to report, which will trigger automatic labelling of the variable name, otherwise if entered as the variable value (ie. without quotes, then will by default be displayed as 'unknown variable')
labels,	will label 'unknown variables' (see above) if entered as variables without quotes
counts	a list of array index values; so if calling during a counting loop, the value can be reported each iteration, also printing the count index; if the list is named the name will also appear, e.g, variable[count=1]. This list must be the same length as varlist (and labels if not NULL), and each element [[i]] must contain as many values as the original corresponding varlist[i] has dimensions. The dimensions must result in a 1x1 scalar

See Also

Dim

Examples

```
# create variables of different types to show output styles #
testvar1 <- 193
testvar2 <- "Atol"
testvar3 <- c(1:10)
testvar4 <- matrix(rnorm(100),nrow=25)
testvar5 <- list(first="test",second=testvar4,third=100:110)
preview("testvar1")
preview("testvar4")
```

```
preview(paste("testvar",1:5,sep=""))
preview(testvar1,"myvarname")
preview(testvar1)
# examples with loops and multiple dimensions / lists
for (cc in 1:4) {
 for (dd in 1:4) { preview("testvar4",counts=list(cc,dd)) }}

for (dd in 1:3) { preview("testvar5",counts=list(dd=dd)) }
```

---

prv *Output variable states within functions/loops during testing/debugging*

Description

Same as preview but no labels command, and input is without quotes and should be plain variable names of existing variables (no indices, args, etc) A versatile function to compactly display most common R objects. Will return the object name, type, dimension, and a compact representation of object contents, for instance using prv.large() to display matrices, so as to not overload the console for large objects. Useful for debugging, can be placed inside loops and functions to track values, dimensions, and data types. Particularly when debugging complex code, the automatic display of the variable name prevents confusion versus using regular print statements. By listing variables to track as character(), provides 'cat()' output of compact and informative variable state information, e.g, variable name, value, datatype and dimension. Can also specify array or list elements, or custom labels. prv() is the same as preview() except it can take objects without using double quotes and has no 'labels' command (and doesn't need one). If expressions are entered rather than variable names, then prv() will attempt to pass the arguments to preview().

Usage

```
prv(..., counts = NULL)
```

Arguments

`...`	series of variable(s) to report, separated by commas, which will trigger automatic labelling of the variable name
`counts`	a list of array index values; so if calling during a counting loop, the value can be reported each iteration, also printing the count index; if the list is named the name will also appear, e.g, variable[count=1]. This list must be the same length as the variable list ... , and each element [[i]] must contain as many values as the original corresponding variable list[i] has dimensions

See Also

Dim

Examples

```
# create variables of different types to show output styles #
testvar1 <- 193
testvar2 <- "Atol"
testvar3 <- c(1:10)
testvar4 <- matrix(rnorm(100),nrow=25)
testvar5 <- list(first="test",second=testvar4,third=100:110)
preview("testvar1"); prv(testvar1)
prv(testvar1,testvar2,testvar3,testvar4)
prv(matrix(rnorm(100),nrow=25)) # expression sent to preview() with no label
prv(193) # fails as there are no object names involved
```

---

prv.large *Tidy display function for matrix objects*

Description

This function prints the first and last columns and rows of a matrix, and more, if desired. Allows previewing of a matrix without overloading the console. Most useful when data has row and column names.

Usage

```
prv.large(largeMat, rows = 3, cols = 2, digits = 4,
  rL = "Row#", rlab = "rownames", clab = "colnames",
  rownums = T, ret = FALSE, warn = TRUE)
```

Arguments

largeMat	a matrix
rows	number of rows to display
cols	number of columns to display
digits	number of digits to display for numeric data
rL	row label to describe the row names/numbers, e.g, row number, ID, etc
rlab	label to describe the data rows
clab	label to describe the data columns
rownums	logical, whether to display rownumbers or ignore them
ret	logical, whether to return the result as a formatted object, or just print to console
warn	logical, whether to warn if the object type is not supported

Examples

```
mat <- matrix(rnorm(1000),nrow=50)
rownames(mat) <- paste("ID",1:50,sep="")
colnames(mat) <- paste("Var",1:20,sep="")
prv.large(mat)
prv.large(mat,rows=9,cols=4,digits=1,rlab="samples",clab="variables",rownums=FALSE)
```

`Rfile.index` *Create an index file for an R function file*

Description

Create a html index for an R function file by looking for functions, add descriptions using comments directly next to the function() command. Note that if too much code other than well-formatted functions is in the file then the result is likely not to be a nicely formatted index.

Usage

```
Rfile.index(fn, below = TRUE, fn.out = "out.htm",
  skip.indent = TRUE)
```

Arguments

`fn`	an R file containing functions in standard R script
`below`	whether to search for comment text below or above the function() calls
`fn.out`	optional name for the output file, else will be based on the name of the input file
`skip.indent`	whether to skip functions that are indented, the assumption being they are functions within functions

Value

creates an html file with name and description of each function

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
# not run:  rfile <- file.choose() # choose an R script file with functions
# not run:  out <- Rfile.index(rfile,fn.out="temp.htm")
# unlink("temp.htm") # run once youve inspected this file in a browser
```

rmv.spc *Remove leading and trailing spaces (or other character).*

Description

Remove leading and trailing spaces (or other character).

Usage

```
rmv.spc(str, before = TRUE, after = TRUE, char = " ")
```

Arguments

str	character vector, may containing leading or trailing chars
before	logical, whether to remove leading spaces
after	logical, whether to remove trailing spaces
char	an alternative character to be removed instead of spaces

Value

returns vectors without the leading/trailing characters

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

spc

Examples

```
rmv.spc("  mid sentence  ")
rmv.spc("0012300",after=FALSE,char="0")
rmv.spc(" change nothing ",after=FALSE,before=FALSE)
```

search.cran *Search all CRAN packages for those containing keyword(s).*

Description

Can be useful for trying to find new packages for a particular purpose. No need for these packages to be installed or loaded. Further searching can be done using utils::RSiteSearch()

Usage

```
search.cran(txt, repos = "")
```

Arguments

txt	text to search for, a character vector, not case-sensitive
repos	repository (CRAN mirror) to use, "" defaults to getOption("repos")

Value

list of hits for each keyword (txt)

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
repos <- "http://cran.ma.imperial.ac.uk/" # OR: repos <- getOption("repos")
search.cran("useful",repos)
search.cran(c("hmm","markov","hidden"),repos=repos)
```

sim.cor *Simulate a dataset with correlated measures*

Description

Simulate a dataset with correlated measures (normal simulation with e.g, rnorm() usually only gives small randomly distributed correlations between variables). This is a quick and unsophisticated method, but should be able to provide a dataset with slightly more realistic structure than simple rnorm() type functions. Varying the last three parameters gives some control on the way the data is generated. It starts with a seed random variable, then creates 'k' random variables with an expected correlation of r=genr() with that seed variable. Then after this, one of the variables in the set (including the seed) is randomly selected to run through the same process of generating 'k' new variables; this is repeated until columns are full up. 'mix.order' then randomizes the column order destroying the relationship between column number and correlation structure, although in some cases, such relationships might be desired as representative of some real life datasets.

Usage

```
sim.cor(nrow = 100, ncol = 100, genx = rnorm,
  genr = runif, k = 3, mix.order = TRUE)
```

Arguments

nrow	integer, number of rows to simulate
ncol	integer, number of columns to simulate
genx	the generating function for data, e.g rnorm(), runif(), etc
genr	the generating function for desired correlation, e.g, runif()
k	number of steps generating from the same seed before choosing a new seed
mix.order	whether to randomize the column order after simulating

Author(s)

Nicholas Cooper

See Also

cor.with

Examples

```
corDat <- sim.cor(200,5)
prv(corDat) # preview of simulated normal data with r uniformly varying
cor(corDat) # correlation matrix
corDat <- sim.cor(500,4,genx=runif,genr=function(x) { 0.5 },mix.order=FALSE)
prv(corDat) # preview of simulated uniform data with r fixed at 0.5
cor(corDat) # correlation matrix
```

---

spc *Print a character a specified number of times.*

Description

Returns 'char' X_i number of times for each element i of X. Useful for padding for alignment purposes.

Usage

```
spc(X, char = " ")
```

Arguments

X	numeric vector of number of repeats
char	The character to repeat (longer will be shortened)

Value

returns vectors of strings of char, lengths X

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

See Also

rmv.spc

Examples

```
cat(paste(spc(9),"123\n"))
cat(paste(spc(8),"1234\n"))
spc(c(1:5),".")
```

---

standardize *Convert a numeric vector to Z-scores.*

Description

Transform a vector to z scores by subtracting its mean and dividing by its standard deviation

Usage

```
standardize(X)
```

Arguments

X	numeric vector to standardize

Value

vector of the same length in standardised form

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
x1 <- rnorm(10,100,15); x2 <- sample(10)
print(x1) ;  standardize(x1)
print(x2) ;  standardize(x2)
```

`Substitute` *Convert objects as arguments to object names*

Description

Equivalent to the base function substitute() but can do any length of arguments instead of just one. Converts the objects in parentheses into text arguments as if they had been entered with double quote strings. The objects must exist and be accessible in the environment the function is called from for the function to work (same as for substitute()). One application for this is to be able to create functions where object arguments can be entered without quotation marks (simpler), or where you want to use the name of the object as well as the data in the object.

Usage

```
Substitute(x = NULL, ...)
```

Arguments

`x`	compulsory, simply the first object in the list, no difference to any further objects
`...`	any further objects to return string names for.

Value

character list of x,... object names

Author(s)

Nicholas Cooper

See Also

`prv`, `preview`

Examples

```
myvar <- list(test=c(1,2,3)); var2 <- "testme"; var3 <- 10:14
print(myvar)
# single variable case, equivalent to base::substitute()
print(substitute(myvar))
print(Substitute(myvar))
# multi variable case, substitute wont work
Substitute(myvar,var2,var3)
# prv() is a wrapper for preview() allowing arguments without parentheses
# which is achieved internally by passing the arguments to Substitute()
preview(c("myvar","var2","var3"))
prv(myvar,var2,var3)
```

summarise.r.datasets *Summarise the dimensions and type of available R example datasets*

Description

This function will parse the current workspace to see what R datasets are available. Using the toHTML function from the tools package to interpret the data() call, each dataset is examined in turn for type and dimensionality. Can also use a filter for dataset types, to only show, for instance, matrix datasets. Also you can specify whether to only look for base datasets, or to search for datasets in all available packages. Result is a printout to the console of the available datasets and their characteristics.

Usage

```
summarise.r.datasets(filter = FALSE,
  types = c("data.frame", "matrix"), all = FALSE, ...)
```

Arguments

filter	logical, whether to filter datasets by 'types'
types	if filter=TRUE, which data types to include in the result
all	logical, if all=TRUE, look for datasets in all available packages, else just base
...	if all is false, further arguments to the data() function to search datasets

Author(s)

Nicholas Cooper

Examples

```
summarise.r.datasets()
summarise.r.datasets(filter=TRUE,"matrix")
```

---

textogram *Make an ascii histogram in the console.*

Description

Uses a call to base::hist(...) and uses the densities to make a a text histogram in the console Particularly useful when working in the terminal without graphics.

Usage

```
textogram(X, range = NA, ...)
```

Arguments

X	numeric vector of data
range	optional sub-range of X to test; c(low,high)
...	additional arguments passed to base::hist()

Value

outputs an ascii histogram to the console

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
textogram(runif(100000))
textogram(rnorm(10000),range=c(-3,3))
```

---

timeit *Times an expression, with breakdown of time spent in each function*

Description

A wrapper for the proftools package Rprof() function. It is to Rprof() as system.time() is to proc.time() (base) Useful for identifying which functions are taking the most time. This procedure will return an error unless expr takes more than ~0.1 seconds to evaluate. I could not see any simple way to avoid this limitation.

Usage

```
timeit(expr, suppressResult = F, total.time = TRUE)
```

Arguments

expr	an expression, must take at least 1 second (roughly)
suppressResult	logical, if true, will return timing information rather than the result of expr
total.time	to sort by total.time, else by self.time

Value

returns matrix where rows are function names, and columns are self.time and total.time. total.time is total time spent in that function, including function calls made by that function. self.time doesn't count other functions within a function

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
timeit(wait(0.1,"s") ,total.time=TRUE)
timeit(wait(0.1,"s") ,total.time=FALSE)
```

---

toheader *Return a string with each first letter of each word in upper case.*

Description

Return a string with each first letter of each word in upper case.

Usage

```
toheader(txt, strict = FALSE)
```

Arguments

txt	a character string
strict	whether to force non-leading letters to lowercase

Value

Vector minus NA's, or the matrix/data.frame minus NA rows

Author(s)

via R Core

Examples

```
toheader(c("using AIC for model selection"))
toheader(c("using AIC", "for MODEL selection"), strict=TRUE)
```

---

top *Monitor CPU, RAM and Processes*

Description

This function runs the unix 'top' command and returns the overall CPU and RAM usage, and optionally the table of processes and resource use for each. Works only with unix-based systems such as Mac OS X and Linux, where 'top' is installed. Default is to return CPU and RAM overall stats, to get detailed stats instead, set Table=TRUE.

Usage

```
top(CPU = !Table, RAM = !Table, Table = FALSE,
  procs = 20, mem.key = NULL, cpu.key = NULL)
```

Arguments

CPU	logical, whether to return overall CPU usage information
RAM	logical, whether to return overall RAM usage information
Table	logical, whether to return system information for separate processes. This is returned as table with all of the same columns as a command line 'top' command. If 'Table=TRUE' is set, then the default becomes not to return the overall CPU/RAM usage stats. The dataframe returned will have been sorted by descending memory usage.
procs	integer, if Table=TRUE, then the maximum number of processes to return (default 20)
mem.key	character, default for Linux is 'mem' and for Mac OS X, 'physmem', but if the 'top' command on your system displays memory usage using a different label, then enter it here (case insensitive) to override defaults.
cpu.key	character, default for Linux and Mac OS X is 'cpu', but if the top command on your system displays CPU usage using a different label, then enter it here.

Value

a list containing CPU and RAM usage, or with alternate parameters can return stats for each process

Author(s)

Nicholas Cooper

Examples

```
top()
top(Table=TRUE,proc=5)
```

Unlist *Unlist a list, starting only from a set depth.*

Description

Allows unlisting preserving the top levels of a list. Can specify the number of list depth levels to skip before running unlist()

Usage

```
Unlist(obj, depth = 1)
```

Arguments

obj	the list to unlist
depth	skip to what layer of the list before unlisting; eg. the base unlist() function would correspond to depth=0

Value

returns vectors of strings of char, lengths X

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
complex.list <- list(1,1:3,list(2,2:4,list(3,3:4,list(10))),list(4,5:7,list(3)))
Unlist(complex.list,0) # equivalent to unlist()
Unlist(complex.list,1) # unlist within the top level lists
Unlist(complex.list,2) # unlist within the second level lists
Unlist(complex.list,10) # once depth >= list-depth, no difference
unlist(complex.list,recursive=FALSE) # not the same as any of the above
```

---

wait *Wait for a period of time.*

Description

Waits a number of hours minutes or seconds (doing nothing). This will use 100

Usage

```
wait(dur, unit = "s", silent = TRUE)
```

Arguments

dur	waiting time
unit	time units h/m/s, seconds are the default
silent	print text showing that waiting is in progress

Value

no return value

Author(s)

Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

Examples

```
wait(.5,silent=FALSE) # wait 0.5 seconds
wait(0.001, "m")
wait(0.0001, "Hours", silent=FALSE)
```

Index