

Package ‘cgwtools’

February 12, 2019

Type Package

Title Miscellaneous Tools

Version 3.0.1

Date 2019-02-10

Author Carl Witthoft

Maintainer Carl Witthoft <carl@witthoft.com>

Description Functions for performing quick observations or evaluations of data, including a variety of ways to list objects by size, class, etc. In addition, functions which mimic Unix shell commands, including 'head', 'tail', 'pushd', and 'popd'. The functions 'seqle' and 'reverse.seqle' mimic the base 'rle' but can search for linear sequences. The function 'splatnd' allows the user to generate zero-argument commands without the need for 'makeActiveBinding'.

License LGPL-3

Imports methods

NeedsCompilation no

Repository CRAN

Date/Publication 2019-02-12 09:55:49 UTC

R topics documented:

cgwtools-package	2
approxseq	2
askrm	3
getstack	4
inverse.seqle	5
lsclass	6
lsdata	7
lsdim	8
lssize	9
lstype	10
mystat	10
popd	11
pushd	12

resave	14
seql	15
short	16
splatnd	17
thekurt	18
theskew	19
Index	20

cgwtools-package	<i>A collection of tools that the author finds handy</i>
------------------	--

Description

Most of these tools are small functions to do simple tasks or filtered views of the current environment. In addition the function `splatnd` is provided primarily as a piece of example code to show how to write zero-argument operators. It's based on the code in the package `sos` , and avoids the need to use `makeActiveBinding` (as in ,e.g., `pracma::ans`)

Details

Package: cgwtools
Type: Package
Version: 2.0
Date: 2014-11-25
License: GPL-3

Author(s)

Carl Witthoft, with attributions as noted in the individual help pages
Maintainer:Carl Witthoft carl@witthoft.com

approxeq	<i>Do "fuzzy" equality and return a logical vector.</i>
----------	---

Description

This function compares two vectors (or arrays) of values and returns the near-equality status of corresponding elements. As with `all.equal()` , the intent is primarily to get around machine limits of representation of floating-point numbers. For integer comparison, just use the base `==` operator.

Usage

```
approxeq(x, y, tolerance = .Machine$double.eps^0.5, ...)
```

Arguments

<code>x, y</code>	The two input items, typically vectors or arrays of data.
<code>tolerance</code>	Set the precision to which <code>abs(x[j] - y[j])</code> will be compared. The default argument provided is the R-standard value for floats.
<code>...</code>	Not used at this time.

Details

If `x` and `y` are of different lengths, the shorter one is recycled and a warning issued.

Value

A vector of the same length as the longer of `x` or `y`, consisting of TRUE and FALSE elements, depending on whether the corresponding elements of `x` and `y` are within the approximate equality precision desired.

Author(s)

Carl Witthoft, <carl@witthoft.com>

See Also

[all.equal](#), [Comparison](#), [identical](#)

Examples

```
x<-1:10
y<-x+runif(10)*1e-6
approxeq(x,y) #all FALSE
approxeq(x,y,tolerance=1e-5) #mostly TRUE, probably
```

askrm

Interactive application of selected function a list of objects.

Description

This function was originally written to do the same as the unix `rm -i` command. The user supplies a list of items and the name of a function which is optionally applied to each item in turn.

Usage

```
askrm(items = ls(parent.frame()), fn = "rm", ask = TRUE)
```

Arguments

items	A character vector of names of the objects to be acted upon (such as <code>ls()</code> returns). The default is all objects in the parent working environment.
fn	The name of the function to be applied, supplied as a character string. Possible future upgrades may allow function names to be entered without quotes.
ask	If TRUE, the user is prompted for "y/n" before performing the function on each object in the list. Be cautious about setting to FALSE for obvious reasons. Note that the only accepted positive response is exactly "y" so, e.g. "yes" will be treated as "no."

Value

	A list with three elements.
func	Echo back the input function, for archival reference.
selected	All the items from the input list to which the function <code>fn</code> was applied. In the default case, these are the items deleted from the environment.
evout	A list of the value(s) returned by the function, if any, each time it was executed.

Author(s)

Carl Witthoft, <carl@witthoft.com>

See Also

When interactive prompting is not desired, [sapply](#) or its brethren are recommended.

Examples

```
# get rid of junky objects left around from testing
foo<-1
afoo<-c(foo,2)
foob <- c('a','b','d')
askrm(ls(pattern="foo") )

x<- rep(1,10)
y<- runif(10)
askrm(c('x','y'),'sd',ask=FALSE)
```

getstack

Returns the current directory stack that pushd and popd manipulate

Description

`getstack` goes into the separate environment where `pushd` and `popd` operate and returns the current stack of directories.

Usage

```
getstack()
```

Arguments

none

Details

Allowing a function to modify an object in the GlobalEnvironment is frowned upon by CRAN (and most programmers), so to maintain a directory stack a separate environment is established by `pushd`. Since this environment is not visible at the console level, `getstack` allows the user to check on the current status of the stack.

Value

The current directory stack is returned as a vector of strings.

Author(s)

Carl Witthoft <carl@witthoft.com>

See Also

[popd](#), [pushd](#), [setwd](#)

Examples

```
## depends on your local directory structure and permissions
getwd()
getstack() #empty, probably
pushd('..')
getstack()
pushd('.')
getstack()
popd()
getstack()
popd()
getstack()
getwd() #back where we started
```

inverse.seqle

Inverse of [seqle](#)

Description

As with `inverse.rle`, this function reverses the compression performed with `seqle` so long as you know the `incr` value used to generate the compressed data.

Usage

```
inverse.seqle(x, incr = 1L)
```

Arguments

x	An object of class <code>rle</code>
incr	The increment between elements used to generate the compressed data object. Note that this can be either integer or float. For floating-point sequences, the reconstruction of the original series may differ at the level of floating-point precision used to generate the input object.

Value

a vector of values identical (or nearly so, for floats) to the original sequence.

Note

The bulk of the code is taken directly from `base::inverse.rle`. Thanks to "flodel", <http://www.linkedin.com/in/florentdelm>, on StackOverflow for suggesting code to handle floating-point increments.

Author(s)

Carl Witthoft, <carl@witthoft.com>

See Also

[seqle](#), [inverse.rle](#)

Examples

```
x<- c(2,2,2,3:8,8,4,4,4,5,5.5,6)
y<-seqle(x,incr=0)
inverse.seqle(y,0)
y <- seqle(x,incr=1)
inverse.seqle(y)
inverse.seqle(y,2) # not what you wanted
```

lsclass

Q&D function to list all objects with the specified class attribute.

Description

This is one of the author's collection of `ls*` Q&D functions. Since anyone can define a new class at any time, there is no predefined set of legal or illegal class names. Remember that an object can have multiple classes. This function only allows searching for a single class name in a given call.

Usage

```
lsclass(type = "numeric")
```

Arguments

type The name of the class you're looking for.

Value

A vector of character strings containing the names of matching objects (as would be returned by the base function `ls`).

Author(s)

Carl Witthoft carl@witthoft.com

See Also

[typeof](#), [class](#), [lstype](#)

Examples

```
xyzyz<-structure(vector(),class='grue')
lsclass('integer')
lsclass('grue')
```

lsdata

List all objects in an .Rdata file.

Description

This function opens an .Rdata file, lists the contents, and cleans up after itself.

Usage

```
lsdata(fnam = ".Rdata")
```

Arguments

fnam the name of the datafile to be examined.

Value

The output of `ls` applied to the objects loaded from the specified data file.

Author(s)

Carl Witthoft carl@witthoft.com

References

Various people have published similar code on Stack Overflow.

See Also

[load](#), [resave](#)

Examples

```
xblue<-1
yblue<-2
save(xblue,yblue,file='blue.Rdata')
lsdata('blue.Rdata')
```

lsdim	<i>Return dimensions of arguments, or lengths if dim==NULL.</i>
-------	---

Description

Just a toy to return dim() when it's sensible and length() otherwise # items must be collection or list of character strings, e.g. output of ls()

Usage

```
lsdim(items)
```

Arguments

items	A vector of character strings identifying the objects of interest as would be returned by, e.g. ls(pattern="foo") or lstype("double") .
-------	---

Value

A list, each element of which gives the object dimensions if there are any, and the length if not.

Author(s)

Carl Witthoft, <carl@witthoft.com>

See Also

[dim](#), [length](#),

Examples

```
x1<-1:10
x2<-matrix(1,3,4)
lsdim(c('x1','x2'))
```

lssize	<i>List the sizes of all selected objects.</i>
--------	--

Description

Just a toy to list the number of elements or optionally the bytesize as produced with `object.size` of a specified selection of objects. I find it handy when I want to rid an environment of large (or empty) objects. In the default case, `byte=FALSE`, lists and S4 objects are "taken apart" down to the lowest level so all individual elements are counted.

Usage

```
lssize(items, byte = FALSE)
```

Arguments

items	A vector of character strings identifying the objects of interest as would be returned by, e.g. <code>ls(pattern="foo")</code> or <code>lstype("double")</code> .
byte	If TRUE, calculate the number of bytes taken up by an object. If FALSE, calculate the total number of elements of an object.

Value

A vector of the object sizes, with the object names as names for the elements

Author(s)

Carl Witthoft, <carl@witthoft.com>

References

Many thanks to Martin Morgan of bioconductor.org who provided the recursive function for deconstructing an S4 Object. See <http://stackoverflow.com/questions/14803237/is-there-an-s4-equivalent-to-unlist> for the original question and answer.

See Also

[lstype](#), [object.size](#), [length](#)

Examples

```
x1<-runif(100)
x2<-runif(1000)
x3<-runif(2000)
lssize(ls(pattern='x[1-3]'))
lssize(ls(pattern='x[1-3]'),byte=TRUE)
#depending on what you have in your environment:
lssize(lstype('integer'))
```

lstype	<i>List all objects of the specified type.</i>
--------	--

Description

This is a Q&D tool to list all objects in the current environment of a specified type. As discussed in the base R documentation, these types are the vector types "logical", "integer", "double", "complex", "character", "raw" and "list", "NULL", "closure" (function), "special" and "builtin" (basic functions and operators), "environment", "S4" (some S4 objects).

Usage

```
lstype(type = "closure")
```

Arguments

type	Any valid variable type, or "function," which is redirected to "closure."
------	---

Value

A vector of character strings as is returned by the base function `ls`.

Author(s)

Carl Witthoft carl@witthoft.com

See Also

[ls](#), [lssize](#), [lsclass](#)

Examples

```
lstype('integer') #if you have any such in your environment.
```

mystat	<i>Calculate and display basic statistics for an object.</i>
--------	--

Description

This function calculates the min, max, median, mean, standard deviation, skew and kurtosis for the specified object and displays the results in a semi-tabular form. An option is provided to set the number of digits displayed for the returned values. Note: see the help pages in this package for `theskew` and `thekurt` for information on those implementations.

Usage

```
mystat(x, numdig = 3, na.rm = TRUE, printit = TRUE)
```

Arguments

<code>x</code>	A vector or vectorizable object.
<code>numdig</code>	How many digits to the right of the decimal point to display (when <code>printit</code> is TRUE).
<code>na.rm</code>	Does the user desire NA values to be removed. Rare is the need to set this to FALSE.
<code>printit</code>	Set to TRUE to see the results, nicely formatted, in the console.

Value

A data frame with scalar elements matching their names:

<code>min</code>	minimum
<code>max</code>	maximum
<code>mean</code>	mean value
<code>median</code>	median
<code>sdev</code>	standard deviation
<code>skew</code>	skew
<code>kurtosis</code>	kurtosis

Author(s)

Carl Witthoft, <carl@witthoft.com>

See Also

[theskew](#) , [thekurt](#)

Examples

```
x <- runif(100)
mystat(x)
mystat(x,numdig=6)
```

popd

Performs equivalent of bash command with same name

Description

popd is based on the cygwin bash manpages' description of these commands.

Usage

```
popd(dn=FALSE, pull=0)
```

Arguments

dn	Determines whether a stack "pop" is to be performed. This is the equivalent of the first argument in <code>bash:popd</code> . If dn is FALSE and pull is zero, then set the new directory to the value at the top of the stack. If dn is TRUE then do not change directory, and look to pull for modifying the stack. See details for why the conditions are set this way.
pull	Equivalent of the latter n arguments in bash. Removes the stack entry corresponding to the pull's value ; can be positive or negative. Note that there may be some inconsistency in how this is handled in different implementations of bash.

Details

Recommend reading `man bash` for full details of the operations. This implementation will not change the working directory if dn is TRUE. The directory history is stored in a file in the function's environment (not console environment). `dirhist`, typically first created with [pushd](#).

Value

A status value: 0 for success or 1 if there is no stack file (`.dirhist`). Future upgrades may include other codes for other failure mechanisms, but for now error messages will have to suffice.

Author(s)

Carl Witthoft <carl@witthoft.com>

See Also

[pushd](#) , [setwd](#)

Examples

```
## depends on your local directory structure and permissions
getwd()
pushd("~/..")
getwd()
popd()
getwd()
```

pushd

Performs equivalent of bash command with same name

Description

pushd is based on the cygwin bash manpages' description of these commands.

Usage

```
pushd(path, dn=FALSE, rot=0)
```

Arguments

<code>path</code>	The directory to move into.
<code>dn</code>	Equivalent of the <code>dir</code> argument in <code>bash</code> . When <code>TRUE</code> , adds the current directory to the stack.
<code>rot</code>	Equivalent of the <code>n</code> argument in <code>bash</code> . Rotates the existing stack by the value of <code>rot</code> ; can be positive or negative. Note that there may be some inconsistency in how this is handled in different implementations of <code>bash</code> .

Details

Recommend reading `man bash` for full details of the operations. This implementation should do nothing more than change the working directory (and store directory history in a file in the function's environment (not console environment) `.dirhist`).

Value

A status value, which is always `0` for success. A future upgrade may implement a `trycatch` for conditions such as an inaccessible directory, but for now error messages will have to suffice.

Author(s)

Carl Witthoft <carl@witthoft.com>

See Also

[popd](#) , [setwd](#)

Examples

```
## depends on your local directory structure and permissions
getwd()
pushd("~/..")
getwd()
popd()
getwd()
```

resave*Add some objects to an existing .Rdata - type file.*

Description

Take an existing myfile.Rdata data file and add the specified objects to it. This is achieved by opening the data file in a local environment, "dumping" the new objects into that environment, and re-saving everything to the same file name.

Usage

```
resave(..., list = character(), file)
```

Arguments

...	Names of objects to save.
list	A list of names of the objects to save. Can be used with or without any named arguments in ...
file	The name of the file to open and add items to.

Value

Nothing is returned. This function is used solely to put objects into the file.

Note

Code is essentially the same as that provided by "flodel", <http://www.linkedin.com/in/florentdelmotte>, on StackOverflow.

Author(s)

Carl Witthoft <carl@witthoft.com>

See Also

[lsdata](#), [save](#), [load](#)

Examples

```
foo<-1:4
bar<-5:8
save(foo,file='foo.Rdata')
resave(bar,file='foo.Rdata')
#check your work
lsdata('foo.Rdata')
```

`seqle`*Extends rle to find and encode linear sequences.*

Description

The function `rle`, or "run-length encoder," is a simple compression scheme which identifies sequences of repeating values in a vector. `seqle` extends this scheme by allowing the user to specify a sequence of values with a common "slope," or delta value, between adjacent elements. `seqle` with an increment of zero is the same as `rle`.

Usage

```
seqle(x, incr = 1L, prec = .Machine$double.eps^0.5)
```

Arguments

<code>x</code>	The input vector of values.
<code>incr</code>	The desired increment between elements which specifies the sequences to search for. Note that this can be either integer or float. For floating-point sequences, see the <code>prec</code> argument for determining what level of precision is used to determine whether elements continue a sequence or not.
<code>prec</code>	Defines the required precision to which elements are compared when determining whether they are part of a sequence. Note that for integer inputs, this value is more or less meaningless.

Details

Note: the returned value is assigned the class "`rle`". So far as I can tell, this class has only a `print` method, i.e. defining what is returned to the console when the user types the name of the returned object.

Value

<code>lengths</code>	a vector of the lengths (1 or greater) of all sequences found.
<code>values</code>	a vector of the initial value for each sequence. For example, if <code>incr == 1</code> a values of 5 associated with a lengths of 3 represents the sequence 5, 6, 7.

Note

The bulk of the code is taken directly from `base::rle`. Thanks to "flodel", <http://www.linkedin.com/in/florentdelmotte>, on StackOverflow for suggesting code to handle floating-point increments.

Author(s)

Carl Witthoft, <carl@witthoft.com>

See Also

[rle inverse.seqle](#)

Examples

```
x<- c(2,2,2,3:8,8,8,4,4,4,5,5.5,6)
seqle(x,incr=0)
seqle(x,incr=1)
seqle(x,incr=1.5)
```

short

Returns a small sample of the specified data set.

Description

The user specifies both the number of elements to display and the number of elements at the start and end of the vector to ignore ('skip') when selecting elements. The results are displayed in a nice tabular form. There are options to set the value of N as well as the number of values to "skip" before selecting the values. `short` is similar to a combination of the unix `head` and `tail` functions.

Usage

```
short(x = seq(1, 20), numel = 4, skipel = 0, ynam = deparse(substitute(x)), dorows=FALSE)
```

Arguments

<code>x</code>	The data vector to be examined.
<code>numel</code>	How many elements to display. Note that <code>numel</code> elements of the beginning and of the end of the vector are returned.
<code>skipel</code>	If desired, skip the first <code>skipel</code> elements before returning <code>numel</code> elements.
<code>ynam</code>	Not normally changed by the user. <code>ynam</code> retrieves the name of the object in question, to be used in the output table formatting.
<code>dorows</code>	For matrices only, return the "numel" number of rows rather than elements. <code>dorows</code> is ignored with a warning if the input <code>x</code> has higher dimensionality.

Details

If the argument `x` happens to be a list, `short` unlists everything, so the first `numel` values will be taken from the first list element, going on to the second element as needed, etc.

Value

Nothing is returned of interest. The function is called to provide what is printed directly to the console, which is a formatted table of the lead and tail values selected, with column labels identifying their location in the input vector object.

Author(s)

Carl Witthoft <carl@witthoft.com>

See Also

[head](#), [tail](#)

Examples

```
foo<-matrix(runif(100),nrow=20)
short(foo)
short(foo,numel=6,skipel=10)
short(foo,numel=6,skipel=10,dorows=TRUE)
```

splatnd

Execute simple zero-argument functions

Description

Execute simple zero-argument functions without having to type the "()" , and without having to go through the bother of makeActiveBinding. This code is provided primarily to allow the user to build his own set of command "shortcuts" by modifying the set of arguments to the switch function in the function body. The bulk of the code is copied from the excellent package sos . The name splatnd cannot be called directly, and doesn't even exist after being sourced. It serves to define a variety of operators ![your_string_here] . If the string after ! is not in the switch-list, the function defaults to the normal splat operator, i.e. NOT[your_string_here] .

Arguments

none

Details

There's an obvious risk of undesired results should there exist an object in the environment with the same name as one of the items in the switch options. The workaround is to enclose the object name in parentheses. See the example.

Value

The returned value is the result of whatever function or operator was invoked.

Note

The bulk of the code is taken directly from the sos package.

Author(s)

Carl Witthoft, <carl@witthoft.com>

See Also

The R manuals on creating operators, `findFn` in the package `sos` , normally invoked as ???

Examples

```
# based on the default items in splatnd.R
qapla <- 1:5
!qapla
!(qapla)
```

thekurt

Calculates the kurtosis of the input data set.

Description

Kurtosis is the next moment after skew (which is the moment after the variance). This function is provided primarily to support the function `mystat` . It uses the algorithm provided in the R package `e1071`

Usage

```
thekurt(x)
```

Arguments

`x` A vector or vectorizable data set.

Value

A single scalar, the kurtosis of the data provided.

Author(s)

Carl Witthoft, <carl@witthoft.com>

theskew	<i>Calculates the skew of a dataset.</i>
---------	--

Description

This function is included primarily to support `mystat` . Skew is the next moment after the variance. The algorithm used here is taken from the R package `e1071` .

Usage

```
theskew(x)
```

Arguments

x	A vector of data to be evaluated
---	----------------------------------

Value

A single scalar, the skew of the dataset

Author(s)

Carl Witthoft, <carl@witthoft.com>

Index

! (splatnd), [17](#)

all.equal, [3](#)
approxeq, [2](#)
askrm, [3](#)

cgwtools (cgwtools-package), [2](#)
cgwtools-package, [2](#)
class, [7](#)
Comparison, [3](#)

dim, [8](#)

getstack, [4](#)

head, [17](#)

identical, [3](#)
inverse.rle, [6](#)
inverse.seqle, [5](#), [16](#)

length, [8](#), [9](#)
load, [8](#), [14](#)
ls, [10](#)
lsclass, [6](#), [10](#)
lsdata, [7](#), [14](#)
lsdim, [8](#)
lssize, [9](#), [10](#)
lstype, [7](#), [9](#), [10](#)

mystat, [10](#)

object.size, [9](#)

popd, [5](#), [11](#), [13](#)
pushd, [5](#), [12](#), [12](#)

resave, [8](#), [14](#)
rle, [16](#)

sapply, [4](#)
save, [14](#)

seqle, [5](#), [6](#), [15](#)
setwd, [5](#), [12](#), [13](#)
short, [16](#)
splatnd, [17](#)

tail, [17](#)
thekurt, [11](#), [18](#)
theskew, [11](#), [19](#)
typeof, [7](#)