Package ‘reader’

February 27, 2014

Type Package

Title A suite of functions to flexibly read data from files

Version 1.0.1

Date 2014-02-25

Author Nicholas Cooper

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

Depends R (>= 2.10), utils, NCmisc (>= 1.1)

Imports grDevices, graphics, stats

Description A set of functions to simplify reading data from files. The main function, reader(), should read most common R datafile types without needing any parameters except the filename. Other functions provide simple ways of handling file paths and extensions, and automatically detecting file format and structure.

License GPL (>= 2)

Collate 'reader.R'

R topics documented:

`reader-package` *Functions to simplify reading from common file types*

Description

A set of functions to simplify reading data from files. The main function, reader(), should read most common R datafile types without needing any parameters except the filename. Other functions provide simple ways of handling file paths and extensions, and automatically detecting file format and structure.

Details

Package:	reader
Type:	Package
Version:	1.0
Date:	2014-02-25
License:	GPL (>= 2)

The reader() function, for which the package is named, should be able to read most of the common types of datafiles used in R without needing any arguments other than the filename. The structure, header, file-format and delimiter are determined automatically. Usually no extra parameters are needed. Other functions provide similarly flexibility to run contigent on data type and file format, or can look for an input file in multiple directory locations. The function cat.path() provides a simple interface to construct file paths using directories, suffixes, prefixes and file extension. Functions in this package can be nested inside new functions, providing flexible parameter format, without having to use multiple if-statements to cope with contigencies. Supported types included delimited text files, R binary files, big.matrix files, text list files, and unstructured text. Note that the file type that will be attempted to read in is initially determine by the file extension, using the function: 'classify.ext()'.

List of key functions:

- *cat.path* Simple and foolproof way to create full-path file names.
- *classify.ext* Classify file types readable by standard R I/O functions.
- *column.salvage* Change column name in different form to desired form.
- *file.ncol* Find the number of columns (lines) in a file.

- *file.nrow* Find the number of rows (lines) in a file.
- *find.id.col* Find which column in a dataframe contains a specified set of values.
- *shift.rownames* Shift the first column of a dataframe to rownames()
- *force.frame* returns a dataframe if 'unknown.data' can in anyway relate to such
- *force.vec* returns a vector if 'unknown.data' can in anyway relate to such
- *get.delim* Determine the delimiter for a text data file.
- *get.ext* Get the file extension from a file-name.
- *is.file* Test whether a file exists in a target directory.
- *make.fixed.width* Convert a matrix or dataframe to fixed-width.
- *n.readLines* Read 'n' lines (ignoring comments and header) from a file.
- *parse.args* Function to collect arguments when running R from the command line.
- *reader* Flexibly load from a text or binary file, accepts multiple file formats.
- *rmv.ext* Remove the file extension from a file-name.
- *find.file* Construct a path to a file, where multiple directories can be searched to find an existing file.

Author(s)

Nicholas Cooper

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

See Also

NCmisc ~~

Examples

```
mydir <- "/Documents"
cat.path(mydir,"temp.doc","NEW",suf=5)
## example for the reader() function ##
df <- data.frame(ID=paste("ID",101:110,sep=""),
                 scores=sample(70,10,TRUE)+30,age=sample(7,10,TRUE)+11)
test.files <- c("temp.txt","temp2.csv","temp3.rda")
write.table(df,file=test.files[1],col.names=TRUE,row.names=TRUE,sep="\t",quote=FALSE)
# file.nrow and file.ncol examples
file.nrow(test.files[1])
file.ncol(test.files[1])
write.csv(df,file=test.files[2])
save(df,file=test.files[3])
# use the same simple reader() function call to read in each file type
for(cc in 1:length(test.files)) {
    cat(test.files[cc],"\n")
    myobj <- reader(test.files[cc])  # add quiet=F to see some working
    print(myobj); cat("\n\n")
}
# inspect files before deleting if desired:
```

```
#  unlink(test.files)
#
# find id column in data frame
new.frame <- data.frame(day=c("M","T","W"),time=c(9,12,3),staff=c("Mary","Jane","John"))
staff.ids <- c("Mark","Jane","John","Andrew","Sally","Mary")
new.frame; find.id.col(new.frame,staff.ids)
```

---

cat.path *Simple and robust way to create full-path file names.*

Description

Create a path with a file name, plus optional directory, prefix, suffix, and file extension. dir/ext are robust, so that if they already exist, the path produced will still make sense. Prefix is applied after the directory, and suffix before the file extension.

Usage

```
cat.path(dir = "", fn, pref = "", suf = "", ext = "",
  must.exist = FALSE)
```

Arguments

`dir`	directory for the full path, if 'fn' already has a dir, then dir will be overridden. Auto add file separator if not present
`fn`	compulsory vector of file names/paths
`pref`	prefix to add in front of the file name
`suf`	suffix to add after the file name, before the extension
`ext`	file extension, will override an existing extension
`must.exist`	the specified file must already exist, else error

Value

returns vector of file names with the full paths

Author(s)

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

Examples

```
mydir <- "/Documents"
cat.path(mydir,"temp.doc")
# dir not added if one already present
cat.path(mydir,"/Downloads/me/temp.doc")
# using prefix and suffix
cat.path(mydir,"temp.doc","NEW",suf=5)
# changing the extension from .docx to .doc
cat.path(mydir,"temp.docx",ext="doc")
```

classify.ext *Classify file types readable by standard R I/O functions.*

Description

Look for known file extensions and classify as binary, comma-separated, text format, or OTH=other; other files are assumed to be unreadable. To read other files, need to specify more types manually.

Usage

```
classify.ext(ext = NULL, more.txt = NULL,
  more.bin = NULL, more.csv = NULL, print.all = FALSE)
```

Arguments

`ext`	filenames or extensions to classify
`more.txt`	more extensions that should be treated as txt
`more.bin`	more extensions that should be treated as binary
`more.csv`	more extensions that should be treated as csv
`print.all`	setting to T, simply prints the list of supported ext

Value

returns the 4 way classification for each file/extension

Author(s)

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

See Also

`get.delim`

Examples

```
classify.ext(c("test.txt","*.csv","tot","other","rda","test.RDatA"))
```

column.salvage *Change column name in different form to desired form.*

Description

Searches for possible equivalents for a desired column in a dataframe and replaces first name match with desired name. Useful when parsing different annotation files which may have standard columns with slightly different names, e.g, Gender=SEX=sex=M/F, or ID=id=ids=samples=subjectID

Usage

```
column.salvage(frame, desired, testfor,
  ignore.case = TRUE)
```

Arguments

`frame`	a dataframe or matrix with column names
`desired`	the column name wanted
`testfor`	possible alternate forms of the desired column name
`ignore.case`	whether to ignore the upper/lower case of the column names

Value

returns the original dataframe with the target column renamed

Author(s)

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

Examples

```
df <- data.frame(Sex=c("M","F","F"),time=c(9,12,3),ID=c("ID3121","ID3122","ID2124"))
# standard example
new.df <- column.salvage(df,"sex",c("gender","sex","M/F")); df; new.df
# exact column already present so no change
new.df <- column.salvage(df,"ID",c("ID","id","ids","samples","subjectID")); df; new.df
# ignore case==TRUE potentially results in not finding desired column:
new.df <- column.salvage(df,"sex",c("gender","sex","M/F"),ignore.case=FALSE); df; new.df
```

`conv.fixed.width` *Convert a matrix or dataframe to fixed-width for nice file output*

Description

Pads each column to a common size so write.table() produces a fixed width format that looks nice

Usage

```
conv.fixed.width(dat)
```

Arguments

`dat`	data.frame or matrix

Value

returns dat with space padding as character

Author(s)

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

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

Examples

```
df <- data.frame(ID=paste("ID",99:108,sep=""),
  scores=sample(150,10,TRUE)+30,age=sample(16,10,TRUE))
dff <- conv.fixed.width(df)
write.table(df,file="notFW.txt",row.names=FALSE,col.names=FALSE,quote=FALSE)
write.table(dff,file="isFW.txt",row.names=FALSE,col.names=FALSE,quote=FALSE)
cat("Fixed-width:\n",paste(readLines("isFW.txt"),"\n"),sep="")
cat("standard-format:\n",paste(readLines("notFW.txt"),"\n"),sep="")
unlink(c("isFW.txt","notFW.txt"))
```

`file.ncol` *Find the number of columns (lines) in a file.*

Description

Returns the number of columns in a datafile. File equivalent of ncol()

Usage

```
file.ncol(fn, reader = FALSE, del = NULL, comment = "#",
  skip = 0, force = FALSE, excl.rn = FALSE)
```

Arguments

`fn`	name of the file(s) to get the length of
`reader`	try to read the entire file to get a result, else looks at the top few lines (ignoring comments)
`del`	specify a delimiter (else this will be auto-detected)
`comment`	a comment symbol to ignore lines in files
`skip`	number of lines to skip at top of file before processing
`force`	try to read the file regardless of whether it looks like an invalid file type. Only use when you know the files are valid
`excl.rn`	exclude rownames from column count (essentially subtract 1)

Value

returns number of columns in file(s). If no delimiter, then =1

Author(s)

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

See Also

`file.nrow`

Examples

```
write.table(matrix(rnorm(100),nrow=10),"temp.txt",col.names=FALSE,row.names=FALSE)
file.ncol("temp.txt",excl.rn=TRUE)
unlink("temp.txt")
# find ncol for all files in current directory:
# [NB: use with caution, will be slow if dir contains large files]
# not run # lf <- list.files(); if(length(lf)==0) { print("no files in dir") }
# lf <- lf[classify.ext(lf)=="TXT"]
# not run (only works if length(lf)>0) # file.ncol(lf)
```

`file.nrow` *Find the number of rows (lines) in a file.*

Description

Returns the number of lines in a file, which in the case of a datafile will often correspond to the number of rows, or rows+1. Can also do this for all files in the directory. File equivalent of nrow()

Usage

```
file.nrow(fn = "", dir = "", all.in.dir = FALSE)
```

Arguments

`fn`	name of the file(s) to get the length of
`dir`	optional path for fn location, or specify all files in dir
`all.in.dir`	select whether to extract length for all files in dir

Value

returns length of file (or all files)

Author(s)

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

See Also

`file.ncol`

Examples

```
write.table(matrix(rnorm(100),nrow=10),"temp.txt",col.names=FALSE)
file.nrow("temp.txt")
# use with caution, will be slow if dir contains large files
# not run # file.nrow(all.in.dir=TRUE)
unlink("temp.txt")
```

`find.file` *Search for a directory to add to the path so that a file exists.*

Description

Looks for a file named 'fn' in 'dir', and if not found there, broadens the search to the list or vector of directorys, 'dirs'. Returns the full path of the first match that exists.

Usage

```
  find.file(fn, dir = "", dirs = NULL)
```

Arguments

`fn`	name of the file to search for
`dir`	the first directory to look in (expected location)
`dirs`	vector/list, a set of directories to look in should the file not be found in 'dir'.

Value

if the file is found, returns the full path of the file, else returns an empty string ""

Author(s)

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

See Also

is.file

Examples

```
l.fn <- "temp.txt"
writeLines("test",con=l.fn)
find.file(l.fn)
find.file(l.fn,dir=getwd())
unlink(l.fn)
# not run # common.places <- ## <<add local folder here>> ##
# not run # d.fn <- cat.path(common.places[1],l.fn)
# write this example file to the first of the folders #
# not run # if(!file.exists(d.fn)) {  writeLines("test2",con=d.fn) }
# search the local folders for a
# a file named temp.txt
# not run # find.file(l.fn,dir=getwd(),dirs=common.places)
# unlink(d.fn) # run only if test file produced
```

`find.id.col` *Find which column in a dataframe contains a specified set of values.*

Description

Starting with a list of ids, each column is searched. The column with the highest non-zero percentage matching is assumed to correspond to the id list. The search terminates early if a perfect match is found. Useful for assembling annotation from multiple sources.

Usage

```
find.id.col(frame, ids,
  ret = c("col", "maxpc", "index", "result"))
```

Arguments

frame	a data.frame, or similarly 2 dimensional object which might contain ids
ids	a vector of IDs/value that might be found in at least 1 column of frame
ret	specify what should be returned, see values

Value

ret can specify a list returning, 'col': the column number (col=0 for rownames) with the best match; 'maxpc': the percentage of ids found in the best matching column; 'index': the matching vector that maps the frame rows onto ids; 'results': the (sub)set of ids found in frame. NAs given for ids not found

Author(s)

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

Examples

```
new.frame <- data.frame(day=c("M","T","W"),time=c(9,12,3),staff=c("Mary","Jane","John"))
staff.ids <- c("Mark","Jane","John","Andrew","Sally","Mary")
new.frame; staff.ids; find.id.col(new.frame,staff.ids)
```

`force.frame` *returns a dataframe if 'unknown.data' can in anyway relate to such:*

Description

it can be: - dataframe, matrix, big.matrix, sub.big.matrix, big.matrix.descriptor, a bigmatrix description file, an RData file containing one of these objects, the name of a text or RData file, a named vector (names become rownames), or a list containing a matrix or dataframe. Using this within functions allows flexibility in specification of a datasource

Usage

```
force.frame(unknown.data, too.big = 10^7)
```

Arguments

`unknown.data`	something that is or can refer to a 2d dataset
`too.big`	max size in GB, to prevent unintended conversion to matrix of a very large big.matrix object.

Value

returns a data.frame regardless of the original object type

Author(s)

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

See Also

`force.vec`

Examples

```
# create a matrix, binary file, text file, big.matrix.descriptor
test.files <- c("temp.rda","temp.txt")
mymat <- matrix(rnorm(100),nrow=10)
# not run yet # require(bigmemory)
save(mymat,file=test.files[1])
write.table(mymat,file=test.files[2],col.names=FALSE,row.names=FALSE)
test.frames <- list(mymat = mymat,
 myrda = test.files[1], mytxt = test.files[2] )
 # not run yet #: ,mybig = describe(as.big.matrix(mymat)) )
sapply(sapply(test.frames,is),"[",1)
# run the function on each, reporting specs of the object returned
for (cc in 1:length(test.frames)) {
  the.frame <- force.frame(test.frames[[cc]])
  cat(names(test.frames)[cc],": dim() => ",
      paste(dim(the.frame),collapse=","),
      "; is() => ",is(the.frame)[1],"\n",sep="")
}
unlink(test.files)
```

---

force.vec *returns a vector if 'unknown.data' can in anyway relate to such:*

Description

if the name of a file with a vector or vector, then reads the file, if a matrix or dataframe, then preferentially return rownames, otherwise return first column - designed to search for IDs. Using this within functions allows flexibility in the specification of a datasource for vectors

Usage

```
force.vec(unknown.data, most.unique = TRUE, dir = NULL,
  warn = FALSE)
```

Arguments

`unknown.data`	something that is or can refer to a 2d dataset
`most.unique`	if TRUE, select most unique column if a unknown.data is a matrix, else select the first column
`dir`	if unknown.data is a file name, specifies directory(s) to look for the file
`warn`	whether to display a warning if unknown.data is a matrix

Value

returns a vector regardless of the original object type

Author(s)

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

See Also

force.frame

Examples

```
# create a matrix, binary file, and simple vector
my.ids <- paste("ID",1:4,sep="")
my.dat <- sample(2,4,replace=TRUE)
test.files <- c("temp.rda")
mymat <- cbind(my.ids,my.dat)
save(mymat,file=test.files[1])
test.vecs <- list(myvec = my.ids,
 myrda = test.files[1],mymat=mymat)
# show dimensions of each test object
sapply(test.vecs,function(x) {  if(is.null(dim(x))){ length(x)} else {dim(x)}})
# run the function on each, reporting specs of the object returned
for (cc in 1:3) {
  the.vec <- force.vec(test.vecs[[cc]])
  cat(names(test.vecs)[cc],": length() => ",
      length(the.vec),"; is() => ",is(the.vec)[1],"\n",sep="")
}
unlink(test.files)
```

`get.delim` *Determine the delimiter for a text data file.*

Description

Reads the first few lines of data in a text file and attempts to infer what delimiter is in use, based on the 'delims' argument that would result in the most consistent number of columns in the first 'n' lines of data. Searches preferentially for delimiters implying between 2 and 'large' columns, then for >large, and lastly for 1 column if nothing else gives a match.

Usage

```
get.delim(fn, n = 10, comment = "#", skip = 0,
  delims = c("\t", " ", "\t| +", ";", ","), large = 10,
  one.byte = TRUE)
```

Arguments

fn	name of the file to parse
n	the number of lines to read to make the inference
comment	a comment symbol to ignore lines in files

skip	number of lines to skip at top of file before processing
delims	the set of delimiters to test for
one.byte	only check for one-byte delimiters, [e.g, whitespace regular expr is >1 byte]
large	search initially for delimiters that imply more than 1, and less than this 'large' columns; if none in this range, look next at >large.

Value

returns character of the most likely delimiter

Author(s)

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

See Also

reader

Examples

```
df <- data.frame(ID=paste("ID",101:110,sep=""),
  scores=sample(70,10,TRUE)+30,age=sample(7,10,TRUE)+11)
# save data to various file formats
test.files <- c("temp.txt","temp2.txt","temp3.csv")
write.table(df,file=test.files[1],col.names=FALSE,row.names=FALSE,sep="|",quote=TRUE)
write.table(df,file=test.files[2],col.names=TRUE,row.names=TRUE,sep="\t",quote=FALSE)
write.csv(df,file=test.files[3])
# report the delimiters
for (cc in 1:length(test.files)) {
  cat("\n",test.files[cc],": ")
  print(get.delim(test.files[cc])) }
unlink(test.files)
```

get.ext *Get the file extension from a file-name.*

Description

Get the file extension from a file-name.

Usage

```
get.ext(fn)
```

Arguments

fn	filename(s) (with full path is ok too)

Value

returns the (usually) 3 character file extension of a filename

Author(s)

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

See Also

rmv.ext

Examples

```
get.ext("/documents/nick/mydoc.xlsx")
get.ext(c("temp.cnv","temp.txt"))
```

`is.file` *Test whether a file exists in a target directory, or alternative list of directories.*

Description

Looks for a file named 'fn' in 'dir', and if not found there, broadens the search to the list or vector of directorys, 'dirs'. Returns TRUE or FALSE as to whether the file exists.

Usage

```
is.file(fn, dir = "", dirs = NULL, combine = TRUE)
```

Arguments

`fn`	name of the file to search for
`dir`	the first directory to look in (expected location)
`dirs`	vector/list, a set of directories to look in should the file not be found in 'dir'.
`combine`	if a list is given, test whether ALL files valid

Value

logical vector of whether each file was found, or if combine is true, then a single value whether ALL valid or not.

Author(s)

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

See Also

find.file

Examples

```
l.fn <- "temp.txt"
writeLines("test",con=l.fn)
some.local.files <- narm(list.files()[1:10])
print(some.local.files)
is.file(l.fn)
is.file(l.fn,dir=getwd())
is.file(some.local.files)
# add a non-valid file to the list to see what happens
is.file(c(some.local.files,"fakefile.unreal"))
is.file(c(some.local.files,"fakefile.unreal"),combine=FALSE)
unlink(l.fn)
```

---

n.readLines *Read 'n' lines (ignoring comments and header) from a file.*

Description

Useful when you don't know the length/structure of a file and want a useful sample to look at. Can skip ahead in the file too. Copes well when there are less than 'n' lines in the file.

Usage

```
n.readLines(fn, n, comment = "#", skip = 0,
  header = TRUE)
```

Arguments

`fn`	name of the file(s) to get the length of
`n`	number of valid lines to attempt to read looks at the top few lines (ignoring comments)
`comment`	a comment symbol to ignore lines in files
`skip`	number of lines to skip at top of file before processing
`header`	whether to allow for, and skip, a header row

Value

returns the first n lines of the file meeting the criteria, or if 'skip' implies lines beyond the length of the file, the result,will be truncated - although in this case, the last line will always be read.

Author(s)

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

Examples

```
dat <- matrix(sample(100),nrow=10)
write.table(dat,"temp.txt",col.names=FALSE,row.names=FALSE)
n.readLines("temp.txt",n=2,skip=2,header=FALSE)
dat[3:4,]
unlink("temp.txt")
```

---

parse.args — *Function to collect arguments when running R from the command line*

Description

Allows parameter specification by A=..., B=... in the command line e.g, R < myScript.R M=1 NAME=John X=10.5, using commandArgs()

Usage

```
parse.args(arg.list = NULL, coms = c("X"), def = 0,
  list.out = F, verbose = TRUE)
```

Arguments

`arg.list`	the result of a commandArgs() call, or else NULL to initiate this call within the function
`coms`	list of valid commands to look for, not case sensitive
`def`	list of default values for each parameter (in same order)
`verbose`	logical, whether to print to the console which assignments are made and warning messages
`list.out`	logical, whether to return output as a list or data.frame

Value

returns dataframe showing the resulting values [column 1, "value"] for each 'coms' (rownames); or, if list.out=TRUE, then returns a list with names corresponding to 'coms' and values equivalent to 'value' column of the data.frame that would be returned if list.out=FALSE

Author(s)

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

Examples

```
parse.args(c("M=1","NAME=John","X=10.5"),coms=c("M","X","NAME"))
parse.args(c("N=1")) # invalid command entered, ignored with warning
temp.fn <- "tempScript1234.R"
# make a temporary R Script file to call using the command line
# not run # writeLines(c("require(reader)","parse.args(coms=c(M,X,NAME))"),con=temp.fn)
bash.cmd <- "R --no-save < tempScript1234.R M=1 NAME=John X=10.5"
# run above command in the terminal, or using system below:
# not run # arg <- system(bash.cmd)
# not run # unlink(temp.fn) # delete temporary file
```

`reader` *Flexibly load from a text or binary file, accepts multiple file formats.*

Description

Uses file extension to distinguish between binary, csv or other text formats. Then tries to automatically determine other parameters necessary to read the file. Will attempt to detect the delimiter, and detect whether there is a heading/column names, and whether the first column should be rownames, or left as a data column. Internal calls to standard file reading functions use 'stringsAsFactors=FALSE'.

Usage

```
reader(fn, dir = "", want.type = NULL, def = "\t",
  force.read = TRUE, header = NA, h.test.p = 0.05,
  quiet = TRUE, treatas = NULL, override = FALSE,
  more.types = NULL, auto.vec = TRUE, ...)
```

Arguments

`fn`	filename (with or without path if dir is specified)
`dir`	optional directory if separate path/filename is preferred
`want.type`	if loading a binary file with multiple objects, specify here the is() type of object you are trying to load
`def`	the default delimiter to try first
`force.read`	attempt to read the file even if the file type looks unsupported
`header`	presence of a header should be autodetected, but can specify header status if you don't trust the autodetection
`h.test.p`	p value to discriminate between number of characters in a column name versus a column value (sensitivity parameter for automatic header detection)
`quiet`	run without messages and warnings
`treatas`	a standard file extension, e.g, 'txt', to treat file as
`override`	assume first col is rownames, regardless of heuristic

`more.types`	optionally add more file types which are read as text
`auto.vec`	if the file seems to only have a single column, automatically return the result as a vector rather than a dataframe with 1 column
`...`	further arguments to the function used by 'reader' to parse the file, e.g, depending on file.type, can be read.table(), read.delim(), read.csv().

Value

returns the most appropriate object depending on the file type, which is usually a data.frame except for binary files

Author(s)

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

Examples

```
# create some datasets
df <- data.frame(ID=paste("ID",101:110,sep=""),
  scores=sample(70,10,TRUE)+30,age=sample(7,10,TRUE)+11)
DNA <- apply(matrix(c("A","C","G","T")[sample(4,100,TRUE)],nrow=10),
                                                1,paste,collapse="")
fix.wid <- c("    MyVal    Results        Check",
  "    0.234      42344          yes",
  "    0.334        351          yes","    0.224         46           no",
  "    0.214     445391          yes")
# save data to various file formats
test.files <- c("temp.txt","temp2.txt","temp3.csv",
                              "temp4.rda","temp5.fasta","temp6.txt")
write.table(df,file=test.files[1],col.names=FALSE,row.names=FALSE,sep="|",quote=TRUE)
write.table(df,file=test.files[2],col.names=TRUE,row.names=TRUE,sep="\t",quote=FALSE)
write.csv(df,file=test.files[3])
save(df,file=test.files[4])
writeLines(DNA,con=test.files[5])
writeLines(fix.wid,con=test.files[6])
# use the same reader() function call to read in each file
for(cc in 1:length(test.files)) {
  cat(test.files[cc],"\n")
  myobj <- reader(test.files[cc])  # add quiet=FALSE to see some working
  print(myobj); cat("\n\n")
}
# inspect files before deleting if desired
unlink(test.files)
# myobj <- reader(file.choose()); myobj # run this to attempt opening a file
```

rmv.ext *Remove the file extension from a file-name.*

Description

Default is to only remove from a known list of file types, this is to protect files with '.' which may not have an extension This option can be changed, and more types can be specified too.

Usage

```
rmv.ext(fn = NULL, only.known = TRUE, more.known = NULL,
  print.known = FALSE)
```

Arguments

`fn`	filename(s) (with full path is ok too)
`only.known`	logical, only remove extension if in the 'known' list
`more.known`	character vector, add to the list of known extensions
`print.known`	return the list of 'known' file extensions

Value

returns the file name/path without the file extension

Author(s)

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

See Also

`get.ext`

Examples

```
rmv.ext(print.known=TRUE)
rmv.ext("/documents/nick/mydoc.xlsx")
rmv.ext(c("temp.cnv","temp.txt","temp.epi"))
# remove anything that looks like an extension
rmv.ext(c("temp.cnv","temp.txt","temp.epi"),only.known=FALSE)
# add to list of known extensions
rmv.ext(c("temp.cnv","temp.txt","temp.epi"),more.known="epi")
```

`shift.rownames` *Shift the first column of a dataframe to rownames() if appropriate.*

Description

Checks whether the first column looks like IDs, and if so will. remove the column, and move these values to rownames.

Usage

```
shift.rownames(dataf, override = FALSE, warn = FALSE)
```

Arguments

`dataf`	data.frame to run the conversion on
`override`	assume col 1 is rownames, regardless of numeric() test
`warn`	whether to display warnings if assumptions aren't met

Value

returns vectors of strings of char, lengths X

Author(s)

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

See Also

`reader`

Examples

```
df1 <- data.frame(ID=paste("ID",101:110,sep=""),
                  scores=sample(70,10,TRUE)+30,age=sample(7,10,TRUE)+11)
shift.rownames(df1)
df2 <- data.frame(ID=paste(101:110),
                  scores=sample(70,10,TRUE)+30,age=sample(7,10,TRUE)+11)
shift.rownames(df2) # first col are all numbers, so no convert
shift.rownames(df2,override=TRUE) # override forces conversion
```

Index