Package 'rgrass'

July 11, 2025

Version 0.5-3

Date 2025-07-09

Title Interface Between 'GRASS' Geographical Information System and 'R'

Description An interface between the 'GRASS' geographical information system ('GIS') and 'R', based on starting 'R' from within the 'GRASS' 'GIS' environment, or running a free-standing 'R' session in a temporary 'GRASS' location; the package provides facilities for using all 'GRASS' commands from the 'R' command line. The original interface package for 'GRASS 5' (2000-2010) is described in Bivand (2000) <doi:10.1016/S0098-3004(00)00057-1> and Bivand (2001) <https://www.r-project.org/conferences/DSC-2001/Proceedings/Bivand.pdf>. This was succeeded by 'spgrass6' for 'GRASS 6' (2006-2016) and 'rgrass7' for 'GRASS 7' (2015-present). The 'rgrass' package modernizes the interface for 'GRASS 8' while still permitting the use of 'GRASS 7'.

Depends R (>= 3.5.0)

Imports stats, utils, methods, xml2

Suggests terra (>= 1.6-16), sp (>= 0.9), knitr, rmarkdown, sf, stars, raster (>= 3.6-3), codetools, testthat (>= 3.0.0), withr, fs

VignetteBuilder knitr

SystemRequirements GRASS (>= 7)

License GPL (>= 2)

URL https://osgeo.github.io/rgrass/, https://grass.osgeo.org/, https://github.com/osgeo/rgrass, https://lists.osgeo.org/mailman/listinfo/grass-stats

BugReports https://github.com/osgeo/rgrass/issues/

RoxygenNote 7.3.2

Encoding UTF-8

Config/testthat/edition 3

NeedsCompilation no

Author Roger Bivand [aut] (ORCID: <https://orcid.org/0000-0003-2392-6140>),
Sebastian Jeworutzki [ctb] (ORCID:
<https://orcid.org/0000-0002-2671-5253>),
Rainer Krug [ctb] (ORCID: <https://orcid.org/0000-0002-7490-0066>),
Robin Lovelace [ctb] (ORCID: <https://orcid.org/0000-0001-5679-6536>),
Markus Neteler [ctb] (ORCID: <https://orcid.org/0000-0003-1916-1966>),
Steven Pawley [cre, aut] (ORCID:
<https://orcid.org/0000-0002-5524-3320>),
Floris Vanderhaeghe [ctb] (ORCID:
<https://orcid.org/0000-0002-6378-6229>)

Maintainer Steven Pawley <dr.stevenpawley@gmail.com>

Repository CRAN

Date/Publication 2025-07-11 05:00:02 UTC

Contents

`execGRASS` *Run GRASS commands*

Description

The functions provide an interface to GRASS commands run through `system`, based on the values returned by the `--interface description` flag using XML parsing. If required parameters are omitted, and have declared defaults, the defaults will be used.

Usage

```
execGRASS(
  cmd,
  flags = NULL,
  ...,
  parameters = NULL,
  intern = NULL,
  ignore.stderr = NULL,
  Sys_ignore.stdout = FALSE,
  Sys_wait = TRUE,
  Sys_input = NULL,
  Sys_show.output.on.console = TRUE,
```

```
  Sys_minimized = FALSE,
  Sys_invisible = TRUE,
  echoCmd = NULL,
  redirect = FALSE,
  legacyExec = NULL
)

stringexecGRASS(
  string,
  intern = NULL,
  ignore.stderr = NULL,
  Sys_ignore.stdout = FALSE,
  Sys_wait = TRUE,
  Sys_input = NULL,
  Sys_show.output.on.console = TRUE,
  Sys_minimized = FALSE,
  Sys_invisible = TRUE,
  echoCmd = NULL,
  redirect = FALSE,
  legacyExec = NULL
)

doGRASS(
  cmd,
  flags = NULL,
  ...,
  parameters = NULL,
  echoCmd = NULL,
  legacyExec = NULL
)

parseGRASS(cmd, legacyExec = NULL)

## S3 method for class 'GRASS_interface_desc'
print(x, ...)

getXMLencoding()

setXMLencoding(enc)
```

Arguments

cmd	GRASS command name.
flags	character vector of GRASS command flags.
...	for execGRASS and doGRASS, GRASS module parameters given as R named arguments directly. For the print method, other arguments to print method. The storage modes of values passed must match those required in GRASS, so a single GRASS string must be a character vector of length 1, a single GRASS

integer must be an integer vector of length 1 (may be an integer constant such as 10L), and a single GRASS float must be a numeric vector of length 1. For multiple values, use vectors of suitable length.

`parameters` list of GRASS command parameters, used if GRASS parameters are not given as R arguments directly; the two methods for passing GRASS parameters may not be mixed. The storage modes of values passed must match those required in GRASS, so a single GRASS string must be a character vector of length 1, a single GRASS integer must be an integer vector of length 1 (may be an integer constant such as 10L), and a single GRASS float must be a numeric vector of length 1. For multiple values, use vectors of suitable length.

`intern` default NULL, in which case set internally from `get.useInternOption`; a logical (not 'NA') which indicates whether to make the output of the command an R object. Not available unless 'popen' is supported on the platform.

`ignore.stderr` default NULL, taking the value set by `set.ignore.stderrOption`, a logical indicating whether error messages written to 'stderr' should be ignored.

`Sys_ignore.stdout`, `Sys_wait`, `Sys_input`
pass extra arguments to `system`.

`Sys_show.output.on.console`, `Sys_minimized`, `Sys_invisible`
pass extra arguments to `system` on Windows systems only.

`echoCmd` default NULL, taking the logical value set by `set.echoCmdOption`, print GRASS command to be executed to console.

`redirect` default `FALSE`, if `TRUE`, add "2>&1" to the command string and set `intern` to `TRUE`; only used in legacy mode.

`legacyExec` default NULL, taking the logical value set by `set.legacyExecOption` which is initialised to `FALSE` on "unix" platforms and `TRUE` otherwise. If `TRUE`, use `system`, if `FALSE` use `system2` and divert stderr to temporary file to record error messages and warnings from GRASS modules.

`string` a string representing *one* full GRASS statement, using shell syntax: command name, optionally followed by flags and parameters, all separated by whitespaces. Parameters follow the key=value format; if 'value' contains spaces, then 'value' must be quoted

`x` object to be printed

`enc` character string to replace UTF-8 in header of XML data generated by GRASS module –interface-description output when the internationalised messages are not in UTF-8 (known to apply to French, which is in latin1)

Details

`parseGRASS` checks to see whether the GRASS command has been parsed already and cached in this session; if not, it reads the interface description, parses it and caches it for future use. `doGRASS` assembles a proposed GRASS command with flags and parameters as a string, wrapping `parseGRASS`, and `execGRASS` is a wrapper for `doGRASS`, running the command through `system` (from 0.7-4, the `...` argument is not used for passing extra arguments for `system`). The command string is termed proposed, because not all of the particular needs of commands are provided by the interface description, and no check is made for the existence of input objects. Support for multiple parameter values

added with help from Patrick Caldon. Support for defaults and for direct use of GRASS parameters instead of a parameter list suggested by Rainer Krug.

stringexecGRASS is a wrapper around execGRASS, and accepts a single shell statement as a string (following GRASS's command syntax).

Value

parseGRASS returns a GRASS_interface_desc object, doGRASS returns a character string with a proposed GRASS command - the expanded command name is returned as an attribute, and execGRASS and stringexecGRASS return what system or system2 return, particularly depending on the intern argument when the character strings output by GRASS modules are returned.

If intern is FALSE, system returns the module exit code, while system2 returns the module exit code with "resOut" and "resErr" attributes.

Note

If any package command fails with a UTF-8 error from the XML package, try using setXMLencoding to work around the problem that GRASS modules declare –interface-description output as UTF-8 without ensuring that it is (French is of 6.4.0 RC5 latin1).

Author(s)

Roger S. Bivand, e-mail: Roger.Bivand@nhh.no

See Also

base::system()

Examples

```
# Run examples if in an active GRASS session in the nc_basic_spm_grass7
Sys.setenv("_SP_EVOLUTION_STATUS_" = "2")
run <- FALSE
GISRC <- Sys.getenv("GISRC")
if (nchar(GISRC) > 0) {
  location_name <- read.dcf(GISRC)[1, "LOCATION_NAME"]
  if (location_name == "nc_basic_spm_grass7") {
    run <- TRUE
  }
}

# Save and set echo command option
echoCmdOption <- get.echoCmdOption()
set.echoCmdOption(TRUE)

if (run) {
  # Read and print GRASS interface description for 'r.slope.aspect'
  print(parseGRASS("r.slope.aspect"))
}
if (run) {
  # Assemble the 'r.slope.aspect' command with specified parameters as a string
```

```
  doGRASS(
    "r.slope.aspect",
    flags = c("overwrite"),
    elevation = "elevation.dem",
    slope = "slope",
    aspect = "aspect"
  )
}
if (run) {
  # Alternatively, specify parameters as a list
  params <- list(elevation = "elevation",
                 slope = "slope",
                 aspect = "aspect")
  doGRASS("r.slope.aspect",
          flags = c("overwrite"),
          parameters = params)
}
if (run) {
  # Read and print GRASS interface description for 'r.buffer'
  print(parseGRASS("r.buffer"))
}
if (run) {
  # Assemble the 'r.buffer' with specified parameters as as string
  doGRASS(
    "r.buffer",
    flags = c("overwrite"),
    input = "schools",
    output = "bmap",
    distances = seq(1000, 15000, 1000)
  )
}
if (run) {
  # Alternatively, specify parameters as a list
  params <- list(
    input = "schools",
    output = "bmap",
    distances = seq(1000, 15000, 1000)
  )
  doGRASS("r.buffer", flags = c("overwrite"), parameters = params)
}
if (run) {
  # Restore original echo command option
  set.echoCmdOption(echoCmdOption)

  # Try executing 'r.stats' command which will fail because "fire_blocksgg"
  # does not exist in the mapset
  try(res <- execGRASS("r.stats", input = "fire_blocksgg", flags = c("C", "n")),
      silent = FALSE)
}
if (run) {
  # Execute 'r.stats' with legacyExec and print the result
  res <- execGRASS(
    "r.stats",
```

```
    input = "fire_blocksgg",
    flags = c("C", "n"),
    legacyExec = TRUE
  )
  print(res)
}
if (run) {
  # If the command failed, retrieve error message
  if (res != 0) {
    resERR <- execGRASS(
      "r.stats",
      input = "fire_blocksgg",
      flags = c("C", "n"),
      redirect = TRUE,
      legacyExec = TRUE
    )
    print(resERR)
  }
}
if (run) {

  # Use 'stringexecGRASS' to run a command and print the result
  res <- stringexecGRASS("r.stats -p -l input=geology", intern = TRUE)
  print(res)

  stringexecGRASS(
    "r.random.cells --overwrite --quiet output=samples distance=1000 ncells=100 seed=1"
  )
}
if (run) {
  # Alternatively, run the same command using 'execGRASS'
  execGRASS(
    "r.random.cells",
    flags  = c("overwrite", "quiet"),
    output = "samples",
    distance = 1000,
    ncells = 100L,
    seed = 1L
  )
}
```

---

gmeta *Reads GRASS metadata from the current LOCATION*

Description

GRASS LOCATION metadata are read into a list in R; helper function getLocationProj returns a WKT2 string of projection information. The helper function `gmeta2grd` creates a GridTopology object from the current GRASS mapset region definitions.

Usage

```
gmeta(ignore.stderr = FALSE, g.proj_WKT = NULL)

getLocationProj(ignore.stderr = FALSE, g.proj_WKT = NULL)

gmeta2grd(ignore.stderr = FALSE)

## S3 method for class 'gmeta'
print(x, ...)

get.ignore.stderrOption()

get.stop_on_no_flags_parasOption()

get.echoCmdOption()

get.useInternOption()

get.legacyExecOption()

get.defaultFlagsOption()

get.suppressEchoCmdInFuncOption()

set.ignore.stderrOption(value)

set.stop_on_no_flags_parasOption(value)

set.echoCmdOption(value)

set.useInternOption(value)

set.legacyExecOption(value)

set.defaultFlagsOption(value)

set.suppressEchoCmdInFuncOption(value)
```

Arguments

ignore.stderr	default FALSE, can be set to TRUE to silence system() output to standard error; does not apply on Windows platforms.
g.proj_WKT	default NULL: return WKT2 representation in GRASS >= 7.6 and Proj4 in GRASS < 7.6; may be set to FALSE to return Proj4 for GRASS >= 7.6.
x	S3 object returned by gmeta
...	arguments passed through print method

value — logical value for setting options on ignore.stderr set by default on package load to FALSE, stop_on_no_flags_params set by default on package load to TRUE, echoCmd set by default on package load to FALSE. useIntern sets the intern argument globally; legacyExec sets the legacyExec option globally, but is initialized to FALSE on unix systems (all but Windows) and TRUE on Windows; defaultFlags is initialized to NULL, but may be a character vector with values from c("quiet", "verbose") suppressEchoCmdInFunc default TRUE suppresses the effect of echoCmd within package functions, maybe set FALSE for debugging.

Value

Returns list of g.gisenv, g.region -g3, and g.proj values.

Author(s)

Roger S. Bivand, e-mail: Roger.Bivand@nhh.no

Examples

```
run <- FALSE
if (nchar(Sys.getenv("GISRC")) > 0 &&
    read.dcf(Sys.getenv("GISRC"))[1, "LOCATION_NAME"] == "nc_basic_spm_grass7") {
  run <- TRUE
}

if (run) {
  G <- gmeta()
  print(G)
}

if (run) {
  cat(getLocationProj(), "\n")
  cat(getLocationProj(g.proj_WKT = FALSE), "\n")
}

if (run) {
  grd <- gmeta2grd()
  print(grd)
}

if (run) {
  ncells <- prod(slot(grd, "cells.dim"))
  df <- data.frame(k = rep(1, ncells))
  mask_SG <- sp::SpatialGridDataFrame(grd, data = df)
  print(summary(mask_SG))
}
```

initGRASS *Initiate GRASS session*

Description

Run GRASS interface in an R session not started within GRASS. In general, most users will use initGRASS in throwaway locations, to use GRASS modules on R objects without the need to define and populate a location. The function initializes environment variables used by GRASS, the .gisrc used by GRASS for further environment variables, and a temporary location.

On Windows, if OSGeo4W GRASS is being used, the R session must be started in the OSGeo4W shell. If not, the non-standard placing of files and of environment variables confuses the function. If toupper(gisBase) contains "OSGEO4W64/APPS/GRASS" or "OSGEO4W/APPS/GRASS" (and after converting "\" to "/"), but the environment variable OSGEO4W_ROOT is not defined, initGRASS() will exit with an error before confusion leads to further errors. For further details, see https://github.com/osgeo/rgrass/issues/16 and https://github.com/osgeo/rgrass/issues/16.

The same restriction applies to use of GRASS with QGIS Windows standalone installations, which may be used with initGRASS only if the R session is started from the OSGeo4W shell shipped as part of the standalone installer (see https://github.com/osgeo/rgrass/issues/87). The function will exit with an error if R was not started from the QGIS OSGeo4W shell before confusion leads to further errors.

The locking functions are used internally, but are exposed for experienced R/GRASS scripters needing to use the GRASS module "g.mapset" through initGRASS in an existing GRASS location. In particular, "g.mapset" may leave a .gislock file in the current MAPSET, so it may be important to call unlink_.gislock to clean up before quitting the R session. remove_GISRC may be used to try to remove the file given in the "GISRC" environment variable if created by initGRASS with argument remove_GISRC= TRUE.

Usage

```
initGRASS(
  gisBase = NULL,
  home,
  SG,
  gisDbase,
  addon_base,
  location,
  mapset,
  override = FALSE,
  use_g.dirseps.exe = TRUE,
  pid,
  remove_GISRC = FALSE,
  ignore.stderr = get.ignore.stderrOption()
)

get.GIS_LOCK()
```

```
set.GIS_LOCK(pid)

unset.GIS_LOCK()

unlink_.gislock()

remove_GISRC()
```

Arguments

gisBase	The directory path to GRASS binaries and libraries, containing bin and lib subdirectories among others; if NULL, set from environment variable GRASS_INSTALLATION if found, if not found, `system("grass --config path")` is tried.
home	The directory in which to create the .gisrc file; defaults to `$HOME` on Unix systems and to `USERPROFILE` on Windows systems; can usually be set to `tempdir()`.
SG	An optional `SpatRaster` or `SpatialGrid` object to define the `DEFAULT_WIND` of the temporary location.
gisDbase	if missing, `tempdir()` will be used; GRASS GISDBASE directory for the working session.
addon_base	if missing, assumed to be "$HOME/.grass7/addons" on Unix-like platforms, on MS Windows "\ and checked for existence.
location	if missing, `basename(tempfile())` will be used; GRASS location directory for the working session.
mapset	if missing, `basename(tempfile())` will be used; GRASS mapset directory for the working session.
override	default FALSE, set to TRUE if accidental trashing of GRASS .gisrc files and locations is not a problem.
use_g.dirseps.exe	default TRUE; when TRUE appears to work for WinGRASS Native binaries, when FALSE for QGIS GRASS binaries; ignored on other platforms.
pid	default `as.integer(round(runif(1, 1, 1000)))`, integer used to identify GIS_LOCK; the value here is arbitrary, but probably should be set correctly.
remove_GISRC	default FALSE; if TRUE, attempt to unlink the temporary file named in the "GISRC" environment variable when the R session terminates or when this package is unloaded.
ignore.stderr	default taking the value set by `set.ignore.stderrOption`; can be set to TRUE to silence `system()` output to standard error; does not apply on Windows platforms.

Details

The function establishes an out-of-GRASS working environment providing GRASS commands with the environment variable support required, and may also provide a temporary location for use until the end of the running R session if the `home` argument is set to `tempdir()`, and the `gisDbase` argument is not given. Running `gmeta` shows where the location is, should it be desired to archive it before leaving R.

Value

The function runs `gmeta6` before returning the current values of the running GRASS session that it provides.

Note

If any package command fails with a UTF-8 error from the XML package, try using `setXMLencoding` to work around the problem that GRASS modules declare –interface-description output as UTF-8 without ensuring that it is.

Author(s)

Roger S. Bivand, e-mail: Roger.Bivand@nhh.no

See Also

`gmeta()`

Examples

```
# Run only if GRASS installation is found and 'terra' package is installed
GRASS_INSTALLATION <- Sys.getenv("GRASS_INSTALLATION")
run <- nzchar(GRASS_INSTALLATION) &&
       file.exists(GRASS_INSTALLATION) &&
       file.info(GRASS_INSTALLATION)$isdir &&
       require(terra, quietly = TRUE)

if (run) {
  # Get the terra example dataset
  f <- system.file("ex/elev.tif", package="terra")
  r <- rast(f)
}

# Check for existing GRASS session running
if (run) {
  loc_existing <- try(gmeta(), silent = TRUE)
}

if (run) {
  # Initialize a temporary GRASS project using the example data
  loc <- initGRASS(
    gisBase = GRASS_INSTALLATION,
    home = tempdir(),
    SG = r,
    override = TRUE
  )
}

if (run) {
  # Write the example data to the GRASS database
  write_RAST(r, "elev", flags = "overwrite")
```

```
  execGRASS("r.info", map = "elev")
}

if (run) {
  # Calculate slope and aspect raster
  execGRASS(
    "r.slope.aspect",
    flags    = "overwrite",
    elevation = "elev",
    slope    = "slope",
    aspect   = "aspect"
  )
}

if (run) {
  # Read the results back into R and plot
  u1 <- read_RAST(c("elev", "slope", "aspect"), return_format = "terra")
  plot(u1[["elev"]], col = terrain.colors(50))
}

# Restore the original GRASS session
if (run) {
  if (!inherits(loc_existing, "try-error")) {
    loc <- initGRASS(
      home = tempdir(),
      gisBase = GRASS_INSTALLATION,
      gisDbase = loc_existing$GISDBASE,
      location = loc_existing$LOCATION_NAME,
      mapset = loc_existing$MAPSET,
      override = TRUE
    )
  }
}
```

`read_RAST` *Read and write GRASS raster files*

Description

Read GRASS raster files from GRASS into R `terra::SpatRaster` or `sp::SpatialGridDataFrame` objects, and write single columns of `terra::SpatRaster` or `sp::SpatialGridDataFrame` objects to GRASS. When `return_format="terra"`, temporary binary files and r.out.bin and r.in.bin are used for speed reasons. `read_RAST()` and `write_RAST()` by default use "RRASTER" files written and read by GDAL.

Usage

```
read_RAST(
  vname,
  cat = NULL,
```

```
  NODATA = NULL,
  return_format = "terra",
  close_OK = return_format == "SGDF",
  flags = NULL,
  Sys_ignore.stdout = FALSE,
  ignore.stderr = get.ignore.stderrOption()
)

write_RAST(
  x,
  vname,
  zcol = 1,
  NODATA = NULL,
  flags = NULL,
  ignore.stderr = get.ignore.stderrOption(),
  overwrite = FALSE,
  verbose = TRUE
)
```

Arguments

vname	A vector of GRASS raster file names in mapsets in the current search path, as set by "g.mapsets"; the file names may be given as fully-qualified map names using "name@mapset", in which case only the mapset given in the full path will be searched for the existence of the raster; if more than one raster with the same name is found in mapsets in the current search path, an error will occur, in which case the user should give the fully-qualified map name. If the fully-qualified name is used, `@` will be replaced by underscore in the output object.
cat	default NULL; if not NULL, must be a logical vector matching vname, stating which (CELL) rasters to return as factor.
NODATA	by default NULL, in which case it is set to one less than `floor()` of the data values for FCELL rasters or the range maximum for CELL Byte, UInt16 and Uint32 rasters (with no negative values), and an attempt is made to set NODATA to the upper Int16 and Int32 range if the lower range is occupied; otherwise an integer NODATA value (required to be integer by GRASS r.out.bin).
return_format	default `terra`, optionally `SGDF`.
close_OK	default TRUE - clean up possible open connections used for reading metadata; may be set to FALSE to avoid the side-effect of other user-opened connections being broken.
flags	default NULL, character vector, for example `overwrite`.
Sys_ignore.stdout	Passed to `system`.
ignore.stderr	default taking the value set by `set.ignore.stderrOption`; can be set to TRUE to silence `system()` output to standard error; does not apply on Windows platforms.
x	A terra `terra::SpatRaster` or sp `sp::SpatialGridDataFrame` object,

zcol	Attribute column number or name,
overwrite	default FALSE, if TRUE inserts "overwrite" into the value of the flags argument if not already there to allow existing GRASS rasters to be overwritten,
verbose	default TRUE, report how writing to GRASS is specified,

Value

by default returns a SpatRaster object, but may return a legacy SpatialGridDataFrame object if return_format="SGDF". write_RAST silently returns the object being written to GRASS.

Author(s)

Roger S. Bivand, e-mail: Roger.Bivand@nhh.no

Examples

```
# Run example only if GRASS settings file indicates that the
# nc_basic_spm_grass7 location is active
run <- FALSE
GISRC <- Sys.getenv("GISRC")

if (nchar(GISRC) > 0) {
  location_name <- read.dcf(GISRC)[1, "LOCATION_NAME"]
  if (location_name == "nc_basic_spm_grass7") {
    run <- TRUE
  }
}

# store original environment variables before modifying
GV <- Sys.getenv("GRASS_VERBOSE")
Sys.setenv("GRASS_VERBOSE" = 0)
original_ignore_stderr <- get.ignore.stderrOption()
set.ignore.stderrOption(TRUE)

if (run) {
  # Retrieve GRASS metadata and creata a new mapset
  meta <- gmeta()
  location_path <- file.path(meta$GISDBASE, meta$LOCATION_NAME)
  previous_mapset <- meta$MAPSET
  example_mapset <- "RGRASS_EXAMPLES"
  execGRASS("g.mapset", flags = "c", mapset = example_mapset)
}

if (run) {
  # List available mapsets and raster maps
  mapsets <- unlist(
    strsplit(execGRASS("g.mapsets", flags = "p", intern = TRUE), " ")
  )
  print(mapsets)
}

if (run) {
```

```
  execGRASS("g.list", type = "raster", pattern = "soils", flags = "m",
            intern = TRUE)
}

if (run) {
  execGRASS("g.list", type = "raster", pattern = "soils@PERMANENT",
            mapset = ".", flags = "m", intern = TRUE)
}

if (run) {
  execGRASS("g.list", type = "raster", pattern = "soils",
            mapset = "PERMANENT", flags = "m", intern = TRUE)
}
# Read/write the GRASS "landuse" dataset as a SpatRaster
if (require("terra", quietly = TRUE)) {
 if (run) {
  v1 <- read_RAST("landuse", cat = TRUE, return_format = "terra")
  print(v1)
  print(inMemory(v1))
 }

 if (run) {
  write_RAST(v1, "landuse1", flags = c("o", "overwrite"))
  execGRASS("r.stats", flags = "c", input = "landuse1")
  execGRASS("g.remove", flags = "f", name = "landuse1", type = "raster")
 }
}

# read the geology and elevation GRASS datasets as SpatialGridDataFrames
if (require("sp", quietly = TRUE)) {
 if (run) {
  nc_basic <- read_RAST(c("geology", "elevation"), cat = c(TRUE, FALSE),
                        return_format = "SGDF")
  print(table(nc_basic$geology))
 }

 if (run) {
  execGRASS("r.stats", flags = c("c", "l", "quiet"), input = "geology")
  boxplot(nc_basic$elevation ~ nc_basic$geology)
 }
 if (run) {
 # Compute square root of elevation and write back to GRASS
  nc_basic$sqdem <- sqrt(nc_basic$elevation)
  write_RAST(nc_basic, "sqdemSP", zcol = "sqdem",
             flags = c("quiet", "overwrite"))
  execGRASS("r.info", map = "sqdemSP")
 }

 if (run) {
  # Read the new raster data and measure read times
  print(system.time(
    sqdemSP <- read_RAST(c("sqdemSP", "elevation"), return_format = "SGDF")
  ))
```

```
 }

 if (run) {
  print(system.time(
    sqdem <- read_RAST(c("sqdemSP", "elevation"), return_format = "terra"))
  )
 }

 if (run) {
  print(names(sqdem))
 }

 if (run) {
  sqdem1 <- read_RAST(c("sqdemSP@RGRASS_EXAMPLES", "elevation@PERMANENT"))
  print(names(sqdem1))
 }

 if (run) {
  execGRASS("g.remove", flags = "f", name = "sqdemSP", type = "raster")

  # GRASS r.mapcalc example
  execGRASS("r.mapcalc", expression = "basins0 = basins - 1",
            flags = "overwrite")
  execGRASS("r.stats", flags = "c", input = "basins0")
 }

 if (run) {
  basins0 <- read_RAST("basins0", return_format = "SGDF")
  print(table(basins0$basins0))
  execGRASS("g.remove", flags = "f", name = "basins0", type = "raster")
 }

 if (run) {
  # Create and read a test raster
  execGRASS("r.mapcalc", expression = "test_t = 66000",
            flags = "overwrite")
  execGRASS("r.info", flags = "r", map = "test_t", intern = TRUE)
  tt <- read_RAST("test_t")
  execGRASS("g.remove", flags = "f", name = "test_t", type = "raster")
 }
}

if (run) {
  # Restore the previous mapset and clean up
  execGRASS("g.mapset", mapset = previous_mapset)
  if (example_mapset != previous_mapset) {
    unlink(file.path(location_path, example_mapset), recursive = TRUE)
  }
}

# Restore original GRASS settings
Sys.setenv("GRASS_VERBOSE" = GV)
set.ignore.stderrOption(original_ignore_stderr)
```

read_VECT *Read and write GRASS vector object files*

Description

read_VECT moves one GRASS vector object file with attribute data through a temporary GeoPackage file to a terra::SpatVector object; write_VECT moves a terra::SpatVector object through a temporary GeoPackage file to a GRASS vector object file. vect2neigh returns neighbour pairs with shared boundary length as described by Markus Neteler, in https://stat.ethz.ch/pipermail/r-sig-geo/2005-October/000616.html. cygwin_clean_temp can be called to try to clean the GRASS mapset-specific temporary directory under cygwin.

Usage

```
read_VECT(
  vname,
  layer = "",
  proxy = FALSE,
  use_gdal_grass_driver = TRUE,
  type = NULL,
  flags = "overwrite",
  Sys_ignore.stdout = FALSE,
  ignore.stderr = get.ignore.stderrOption()
)

write_VECT(
  x,
  vname,
  flags = "overwrite",
  ignore.stderr = get.ignore.stderrOption()
)

vInfo(vname, layer, ignore.stderr = NULL)

vColumns(vname, layer, ignore.stderr = NULL)

vDataCount(vname, layer, ignore.stderr = NULL)

vect2neigh(
  vname,
  ID = NULL,
  ignore.stderr = NULL,
  remove = TRUE,
  vname2 = NULL,
  units = "k"
)
```

Arguments

vname	A GRASS vector file name.
layer	a layer name (string); if missing the first layer will be used.
proxy	Default is FALSE. Set as TRUE if you need a SpatVectorProxy object.
use_gdal_grass_driver	Default TRUE. The standalone GDAL-GRASS driver for the vector format will be used if it is installed. The advantage is that no intermediate file needs to be written from GRASS GIS and subsequently read into R; instead the vector layer is read directly from the GRASS GIS database. Please read the **Note** further below!.
type	override type detection when multiple types are non-zero, passed to v.out.ogr.
flags	Character vector containing additional optional flags and/or options for v.in.ogr, particularly "o" and "overwrite".
Sys_ignore.stdout	Passed to system.
ignore.stderr	default the value set by set.ignore.stderrOption; NULL, taking the value set by set.ignore.stderrOption, can be set to TRUE to silence system() output to standard error; does not apply on Windows platforms.
x	A SpatVector object moved to GRASS.
ID	A valid DB column name for unique identifiers (optional).
remove	default TRUE, remove copied vectors created in vect2neigh.
vname2	If on a previous run, remove was FALSE, the name of the temporary vector may be given to circumvent its generation.
units	default "k"; see GRASS 'v.to.db' manual page for alternatives.

Value

read_VECT imports a GRASS vector layer into a SpatVector or SpatVectorProxy object. vect2neigh returns a data frame object with left and right neighbours and boundary lengths, also given class GRASSneigh and spatial.neighbour (as used in spdep). The incantation to retrieve the neighbours list is sn2listw(vect2neigh())$neighbours, and to retrieve the boundary lengths: sn2listw(vect2neigh())$weights. The GRASSneigh object has two other useful attributes: external is a vector giving the length of shared boundary between each polygon and the external area, and total giving each polygon's total boundary length.

Note

Be aware that the GDAL-GRASS driver may have some issues for vector data. In our experience, the error and warning messages for vector data can be ignored. Further, the returned metadata about the coordinate reference system may currently be incomplete, e.g. it may miss the EPSG code.

Author(s)

Roger S. Bivand, e-mail: Roger.Bivand@nhh.no

Examples

```
# Run example if in active GRASS nc_basic_spm_grass7 location
run <- FALSE
if (nchar(Sys.getenv("GISRC")) > 0 &&
    read.dcf(Sys.getenv("GISRC"))[1, "LOCATION_NAME"] == "nc_basic_spm_grass7") {
  run <- require(terra, quietly = TRUE)
}

# Store original environment variable settings
GV <- Sys.getenv("GRASS_VERBOSE")
Sys.setenv("GRASS_VERBOSE" = 0)
ois <- get.ignore.stderrOption()
set.ignore.stderrOption(TRUE)

if (run) {
  # Create a new mapset
  meta <- gmeta()
  location_path <- file.path(meta$GISDBASE, meta$LOCATION_NAME)
  previous_mapset <- meta$MAPSET
  example_mapset <- "RGRASS_EXAMPLES"
  execGRASS("g.mapset", "c", mapset = example_mapset)
 }

if (run) {
  # Report basic metadata about the schools dataset
  execGRASS("v.info", map = "schools", layer = "1")
  print(vInfo("schools"))
 }

if (run) {
  # Read/write as a SpatVector
  schs <- read_VECT("schools")
  print(summary(schs))
 }

if (run) {
  try({
    write_VECT(schs, "newsch", flags = c("o", "overwrite"))
  })
  schs <- read_VECT("schools", use_gdal_grass_driver = FALSE)
 }

if (run) {
  write_VECT(schs, "newsch", flags = c("o", "overwrite"))
  execGRASS("v.info", map = "newsch", layer = "1")
 }

if (run) {
  nschs <- read_VECT("newsch")
  print(summary(nschs))
 }
```

```
if (run) {
  print(all.equal(names(nschs), as.character(vColumns("newsch")[, 2])))
 }

if (run) {
  # Show metadata for the roadsmajor dataset and read as spatVector
  print(vInfo("roadsmajor"))
 }

if (run) {
  roads <- read_VECT("roadsmajor")
  print(summary(roads))
}

# not run: vect2neigh() currently writes 3 new data sources in the PERMANENT
# mapset, despite this mapset not being the active one.
if (FALSE) {
  cen_neig <- vect2neigh("census")
  str(cen_neig)
}

# Cleanup the previously created datasets
if (run) {
  execGRASS(
    "g.remove",
    flags = "f",
    name = c("newsch", "newsch1"),
    type = "vector"
  )
  execGRASS("g.mapset", mapset = previous_mapset)
  if (example_mapset != previous_mapset) {
    unlink(file.path(location_path, example_mapset), recursive = TRUE)
  }
}

# Restore environment variable settings
Sys.setenv("GRASS_VERBOSE" = GV)
set.ignore.stderrOption(ois)
```

Index