Package 'rgrass7'

June 7, 2023

Version 0.2-12

Date 2023-06-06

Title Deprecated Interface Between GRASS Geographical Information System and R

Description This package will be archived in October 2023 together with 'rgdal'. Deprecated interpreted interface between 'GRASS' geographical information system and R. Transition to new package 'rgrass' <https://grass.osgeo.org/news/2023_06_05_retirement_of_rgrass7/>.

Depends R (>= 3.3.0), XML

Imports stats, utils, methods

Suggests rgdal (>= 1.0-6), RSQLite, sp (>= 0.9), sf (>= 0.7.6), stars, terra

SystemRequirements GRASS (>= 7)

License GPL (>= 2)

URL https://grass.osgeo.org/, https://github.com/rsbivand/rgrass

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

Collate AAA.R options.R rgrass.R bin_link.R rast_link.R vect_link.R vect_link_ng.R initGRASS.R xml1.R

NeedsCompilation no

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

Maintainer Roger Bivand <Roger.Bivand@nhh.no>

Repository CRAN

Date/Publication 2023-06-07 08:30:02 UTC

R topics documented:

rgrass7-deprecated 2

Index **14**

---

rgrass7-deprecated *Interface between GRASS geographical information system and R*

Description

Deprecated interpreted interface between GRASS geographical information system, versions 7 and 8, and R, use rgrass instead.

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)
use_sf()
use_sp()
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.useGDALOption()
get.pluginOption()
get.echoCmdOption()
get.useInternOption()
get.legacyExecOption()
get.defaultFlagsOption()
get.suppressEchoCmdInFuncOption()
set.ignore.stderrOption(value)
set.stop_on_no_flags_parasOption(value)
set.useGDALOption(value)
set.pluginOption(value)
set.echoCmdOption(value)
```

```
set.useInternOption(value)
set.legacyExecOption(value)
set.defaultFlagsOption(value)
set.suppressEchoCmdInFuncOption(value)
initGRASS(gisBase, 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()
readRAST(vname, cat=NULL, ignore.stderr = get.ignore.stderrOption(),
 NODATA=NULL, plugin=get.pluginOption(), mapset=NULL,
 useGDAL=get.useGDALOption(), close_OK=TRUE, drivername="GTiff",
 driverFileExt=NULL, return_SGDF=TRUE)
read_RAST(vname, cat=NULL, NODATA=NULL, ignore.stderr=get.ignore.stderrOption(),
 return_format="SGDF", close_OK=return_format=="SGDF", flags=NULL)
writeRAST(x, vname, zcol = 1, NODATA=NULL,
 ignore.stderr = get.ignore.stderrOption(), useGDAL=get.useGDALOption(),
 overwrite=FALSE, flags=NULL, drivername="GTiff")
write_RAST(x, vname, zcol = 1, NODATA=NULL, flags=NULL,
 ignore.stderr = get.ignore.stderrOption(), overwrite=FALSE, verbose=TRUE)
readVECT(vname, layer, type=NULL, plugin=NULL,
 remove.duplicates = TRUE, ignore.stderr=NULL,
 with_prj=TRUE, with_c=FALSE, mapset=NULL,
 pointDropZ=FALSE, driver=NULL)
read_VECT(vname, layer, type=NULL, flags="overwrite",
    ignore.stderr = NULL)
writeVECT(SDF, vname,  v.in.ogr_flags=NULL,
 ignore.stderr = NULL, driver=NULL,
 min_area=0.0001, snap=-1)
write_VECT(x, vname, flags="overwrite", ignore.stderr = NULL)
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

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 thos 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 thos 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

`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

`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

`x` object

`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)

`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

`value` logical value for setting options on `ignore.stderr` set by default on package load to FALSE, `stop_on_no_flags_paras` set by default on package load to TRUE, `useGDAL` set by default on package load to TRUE, `plugin` set by default on package load to NULL, `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, may be set FALSE for debugging.

gisBase	The directory path to GRASS binaries and libraries, containing bin and lib subdirectories among others
home	The directory in which to create the .gisrc file; defaults to $HOME on Unix systems and to USERPROFILE on Windows systems; can usually be set to tempdir()
SG	An optional SpatialGrid object to define the DEFAULT_WIND of the temporary location; if use_sp() has not been called, it will be called internally if SG is given as an object inheriting from "Spatial"; if use_sf() has been called, it will be overridden internally as only objects inheriting from "Spatial" may be given.
gisDbase	if missing, tempdir() will be used; GRASS GISDBASE directory for the working session
addon_base	if missing, assumed to be "$HOME/.grass7/addons" on Unix-like platforms, on MS Windows "%APPDATA%\GRASS7\addons", 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
vname	A vector of GRASS file names
cat	default NULL; if not NULL, must be a logical vector matching vname, stating which (CELL) rasters to return as factor
return_format	For read_RAST(), either "SGDF" or "terra"
plugin	default taking the value set by set.pluginOption; NULL does auto-detection, changes to FALSE if vname is longer than 1, and a sanity check will be run on raster and current region, and the function will revert to FALSE if mismatch is found; if TRUE, the plugin is available and the raster should be read in its original region and resolution; if the plugin is used, no further arguments other than mapset are respected
useGDAL	(effectively defunct, only applies to use of plugin) default taking the value set by set.useGDALOption; use plugin and readGDAL if autodetected or plugin=TRUE; or for writing writeGDAL, GTiff, and r.in.gdal, if FALSE using r.out.bin or r.in.bin
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

drivername	default "GTiff"; a valid GDAL writable driver name to define the file format for intermediate files
driverFileExt	default NULL; otherwise string value of required driver file name extension
return_SGDF	default TRUE returning a SpatialGridDataFrame object, if FALSE, return a list with a GridTopology object, a list of bands, and a proj4string; see example below
zcol	Attribute column number or name
NODATA	by default NULL, in which case it is set to one less than floor()of the data values, otherwise an integer NODATA value (required to be integer by GRASS r.out.bin)
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
layer	a layer name (string); if missing set to default of "1"
type	override type detection when multiple types are non-zero, passed to v.out.ogr
remove.duplicates	In line and area vector objects, multiple geometrical features may be associated with a single cat number, leading to duplication of data rows; this argument attempts to combine the geometrical features so that they match a single data row
with_prj	default TRUE, write ESRI-style PRJ file for transfered data
with_c	default FALSE in GRASS; if FALSE, export features with category (labeled) only; if not default, all features are exported, including GRASS "islands" which are probably spurious exterior rings filling holes.
pointDropZ	default FALSE, if TRUE, discard third coordinates for point geometries; third coordinates are alway discarded for line and polygon geometries
driver	default NULL, which will lead to the choice of the first driver found in a ordered preferred vector, currently c("SQLite", "ESRI Shapefile"); a valid OGR writable driver name to define the file format for intermediate files, one of c("GML", "SQLite"), c("ESRI_Shapefile", "MapInfo_File") is preferred as these construct the names of the intermediate files adequately
min_area	default 0.0001); Minimum size of area to be imported (square meters) Smaller areas and islands are ignored. Should be greater than snap^2
snap	default -1); Snapping threshold for boundaries (map units). '-1' for no snap
SDF	A Spatial*DataFrame to be moved to GRASS as a vector object, for SpatialPointsDataFrame, SpatialLinesDataFrame, and SpatialPolygonsDataFrame objects, or an equivalent "sf" object
v.in.ogr_flags	Character vector containing additional optional flags and/or options for v.in.ogr, particularly "o" and "overwrite"
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

Note

Note that the examples use the smaller subset North Carolina location: https://grass.osgeo.org/sampledata/north_carolina/nc_basic_spm_grass7.tar.gz

Author(s)

Roger Bivand

Examples

```
run <- FALSE
if (nchar(Sys.getenv("GISRC")) > 0 &&
  read.dcf(Sys.getenv("GISRC"))[1,"LOCATION_NAME"] == "nc_basic_spm_grass7") run <- TRUE
  oechoCmd <- get.echoCmdOption()
  set.echoCmdOption(TRUE)
if (run) {
  print(parseGRASS("r.slope.aspect"))
}
if (run) {
  doGRASS("r.slope.aspect", flags=c("overwrite"),
    elevation="elevation.dem", slope="slope", aspect="aspect")
}
if (run) {
  pars <- list(elevation="elevation", slope="slope", aspect="aspect")
  doGRASS("r.slope.aspect", flags=c("overwrite"), parameters=pars)
}
if (run) {
  print(parseGRASS("r.buffer"))
}
if (run) {
  doGRASS("r.buffer", flags=c("overwrite"), input="schools", output="bmap",
    distances=seq(1000,15000,1000))
}
if (run) {
  pars <- list(input="schools", output="bmap", distances=seq(1000,15000,1000))
  doGRASS("r.buffer", flags=c("overwrite"), parameters=pars)
}
if (run) {
  set.echoCmdOption(oechoCmd)
  try(res <- execGRASS("r.stats", input = "fire_blocksgg", # no such file
    flags = c("C", "n")), silent=FALSE)
}
if (run) {
  res <- execGRASS("r.stats", input = "fire_blocksgg", flags = c("C", "n"),
    legacyExec=TRUE)
  print(res)
}
if (run) {
  if (res != 0) {
    resERR <- execGRASS("r.stats", input = "fire_blocksgg",
      flags = c("C", "n"), redirect=TRUE, legacyExec=TRUE)
    print(resERR)
```

```
  }
}
if (run) {
  res <- stringexecGRASS("r.stats -p -l input=geology", intern=TRUE)
  print(res)
}
if (run) {
  stringexecGRASS(paste("r.random.cells --overwrite --quiet output=samples",
    "distance=1000 ncells=100 seed=1"))
}
if (run) {
 execGRASS("r.random.cells", flags=c("overwrite", "quiet"), output="samples", distance=1000,
    ncells=100L, seed=1L)
}
run <- FALSE
if (nchar(Sys.getenv("GISRC")) > 0 &&
  read.dcf(Sys.getenv("GISRC"))[1,"LOCATION_NAME"] == "nc_basic_spm_grass7") run <- TRUE
run <- (run && !inherits(try(use_sp(), silent=TRUE), "try-error"))
if (run) {
  G <- gmeta()
  print(G)
}
if (run) {
  cat(getLocationProj(), "\n")
}
if (run) {
  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))
}
GRASS_INSTALLATION <- Sys.getenv("GRASS_INSTALLATION")
run <- FALSE
if (nzchar(GRASS_INSTALLATION)) run <- file.info(GRASS_INSTALLATION)$isdir
run <- run && require(terra, quietly=TRUE)
if (run) {
f <- system.file("ex/elev.tif", package="terra")
r <- rast(f)
plot(r, col=grDevices::terrain.colors(50))
}
if (run) {
(loc <- initGRASS(GRASS_INSTALLATION, home=tempdir(), SG=r, override=TRUE))
}
if (run) {
write_RAST(r, "elev", flags="overwrite")
execGRASS("r.info", map="elev")
```

```
}
if (run) {
s <- rast(r)
values(s) <- values(r)
write_RAST(s, "elev1", flags="overwrite")
execGRASS("r.info", map="elev1")
}
if (run) {
execGRASS("r.slope.aspect", flags="overwrite", elevation="elev", slope="slope", aspect="aspect")
}
if (run) {
u1 <- read_RAST(c("elev", "slope", "aspect"), return_format="terra")
plot(u1[["elev"]], col=grDevices::terrain.colors(50))
}
run <- FALSE
if (nchar(Sys.getenv("GISRC")) > 0 &&
  read.dcf(Sys.getenv("GISRC"))[1,"LOCATION_NAME"] == "nc_basic_spm_grass7") run <- TRUE
run <- (run && !inherits(try(use_sp(), silent=TRUE), "try-error"))
  GV <- Sys.getenv("GRASS_VERBOSE")
  Sys.setenv("GRASS_VERBOSE"=0)
#  require(rgdal)
  ois <- get.ignore.stderrOption()
  set.ignore.stderrOption(TRUE)
  get.useGDALOption()
if (run) {
  nc_basic <- readRAST(c("geology", "elevation"), cat=c(TRUE, FALSE),
    useGDAL=FALSE)
  nc_basic <- readRAST(c("geology", "elevation"), cat=c(TRUE, FALSE),
    useGDAL=TRUE)
  print(table(nc_basic$geology))
}
if (run) {
  execGRASS("r.stats", flags=c("c", "l", "quiet"), input="geology")
}
if (run) {
  boxplot(nc_basic$elevation ~ nc_basic$geology)
}
if (run) {
  nc_basic$sqdem <- sqrt(nc_basic$elevation)
}
if (run) {
  if ("GRASS" %in% rgdal::gdalDrivers()$name) {
    execGRASS("g.region", raster="elevation")
    dem1 <- readRAST("elevation", plugin=TRUE, mapset="PERMANENT")
    print(summary(dem1))
    execGRASS("g.region", raster="elevation")
  }
}
if (run) {
  writeRAST(nc_basic, "sqdemSP", zcol="sqdem", flags=c("quiet", "overwrite"))
  execGRASS("r.info", map="sqdemSP")
}
if (run) {
```

```
  execGRASS("g.remove", flags="f", name="sqdemSP", type="raster")
}
if (run) {
 writeRAST(nc_basic, "sqdemSP", zcol="sqdem", useGDAL=TRUE, flags=c("quiet", "overwrite"))
  execGRASS("r.info", map="sqdemSP")
}
if (run) {
  print(system.time(sqdemSP <- readRAST(c("sqdemSP", "elevation"),
    useGDAL=TRUE, return_SGDF=FALSE)))
}
if (run) {
  print(system.time(sqdemSP <- readRAST(c("sqdemSP", "elevation"),
    useGDAL=TRUE, return_SGDF=TRUE)))
}
if (run) {
  print(system.time(sqdemSP <- readRAST(c("sqdemSP", "elevation"),
    useGDAL=FALSE, return_SGDF=TRUE)))
}
if (run) {
  print(system.time(sqdemSP <- readRAST(c("sqdemSP", "elevation"),
    useGDAL=FALSE, return_SGDF=FALSE)))
}
if (run) {
  str(sqdemSP)
  mat <- do.call("cbind", sqdemSP$dataList)
  str(mat)
}
if (run) {
  print(system.time(SGDF <- sp::SpatialGridDataFrame(grid=sqdemSP$grid,
    proj4string=sqdemSP$proj4string, data=as.data.frame(sqdemSP$dataList))))
}
if (run) {
  summary(SGDF)
}
if (run) {
  execGRASS("g.remove", flags="f", name="sqdemSP", type="raster")
  execGRASS("r.mapcalc", expression="basins0 = basins - 1")
  execGRASS("r.stats", flags="c", input="basins0")
}
if (run) {
  basins0 <- readRAST("basins0")
  print(table(basins0$basins0))
}
if (run) {
  basins0 <- readRAST("basins0", plugin=FALSE)
  print(table(basins0$basins0))
}
if (run) {
  execGRASS("g.remove", flags="f", name="basins0", type="raster")
}
run <- run && require("terra", quietly=TRUE)
if (run) {
  v1 <- read_RAST("landuse", cat=TRUE, return_format="terra")
```

```
  v1
  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")
}
  Sys.setenv("GRASS_VERBOSE"=GV)
  set.ignore.stderrOption(ois)
run <- FALSE
if (nchar(Sys.getenv("GISRC")) > 0 &&
  read.dcf(Sys.getenv("GISRC"))[1,"LOCATION_NAME"] == "nc_basic_spm_grass7") run <- TRUE
run <- (run && !inherits(try(use_sp(), silent=TRUE), "try-error"))
GV <- Sys.getenv("GRASS_VERBOSE")
Sys.setenv("GRASS_VERBOSE"=0)
#  require(rgdal)
ois <- get.ignore.stderrOption()
set.ignore.stderrOption(TRUE)
if (run) {
  execGRASS("v.info", map="schools", layer="1")
}
if (run) {
  print(vInfo("schools"))
  schs <- readVECT("schools", plugin=NULL)
  print(summary(schs))
}
if (run) {
  schs1 <- readVECT("schools", plugin=FALSE)
  print(summary(schs1))
}
if (run) {
  writeVECT(schs, "newsch", v.in.ogr_flags=c("o", "overwrite"))
  execGRASS("v.info", map="newsch", layer="1")
}
if (run) {
  nschs <- readVECT("newsch")
  print(summary(nschs))
}
if (run) {
  print(all.equal(names(nschs), as.character(vColumns("newsch")[,2])))
}
if (run) {
  names(nschs) <- paste("ABCDEFGHIJKLMNO", names(nschs), sep="")
  writeVECT(nschs, "newsch1", v.in.ogr_flags=c("o", "overwrite"))
}
if (run) {
  print(all.equal(names(nschs), as.character(vColumns("newsch1")[-1,2])))
}
if (run) {
  nschs1 <- readVECT("newsch1")
  print(all.equal(names(nschs), names(nschs1)[-1]))
}
```

```
if (run) {
  print(summary(nschs1))
}
if (run) {
  schs <- readVECT("schools", driver="ESRI Shapefile")
  names(schs) <- paste("ABCDEFGHIJKLMNO", names(schs), sep="")
  writeVECT(schs, "newsch", v.in.ogr_flags=c("o", "overwrite"),
    driver="ESRI Shapefile")
  print(all.equal(names(schs), as.character(vColumns("newsch")[-1,2])))
}
if (run) {
  nschs <- readVECT("newsch", driver="ESRI Shapefile")
  all.equal(names(schs), names(nschs)[-1])
}
if (run) {
  print(vInfo("roadsmajor"))
}
if (run) {
  roads <- readVECT("roadsmajor")
  print(summary(roads))
}
if (run) {
  cen_neig <- vect2neigh("census")
  str(cen_neig)
}
if (run) {
  execGRASS("g.remove", flags="f", name=c("newsch", "newsch1"), type="vector")
}
use_sf()
if (run) {
  print(vInfo("schools"))
  schs <- readVECT("schools", plugin=NULL)
  print(summary(schs))
}
if (run) {
  schs1 <- readVECT("schools", plugin=FALSE)
  print(summary(schs1))
}
if (run) {
  writeVECT(schs, "newsch", v.in.ogr_flags=c("o", "overwrite"))
  execGRASS("v.info", map="newsch", layer="1")
}
if (run) {
  nschs <- readVECT("newsch")
  print(summary(nschs))
}
if (run) {
  execGRASS("g.remove", flags="f", name="newsch", type="vector")
}
run <- run && require("terra", quietly=TRUE)
if (run) {
  v1 <- read_VECT("census")
  v1
```

```
}
if (run) {
  write_VECT(v1, "census_sV")
  execGRASS("v.info", map="census_sV")
}
if (run) {
  execGRASS("g.remove", flags="f", name="census_sV", type="vector")
}
Sys.setenv("GRASS_VERBOSE"=GV)
set.ignore.stderrOption(ois)
if (run) use_sp()
```

Index

∗ **package**
rgrass7-deprecated, 2
∗ **spatial**
rgrass7-deprecated, 2

doGRASS *(*rgrass7-deprecated*)*, 2

execGRASS *(*rgrass7-deprecated*)*, 2

get.defaultFlagsOption
*(*rgrass7-deprecated*)*, 2
get.echoCmdOption *(*rgrass7-deprecated*)*, 2
get.GIS_LOCK *(*rgrass7-deprecated*)*, 2
get.ignore.stderrOption
*(*rgrass7-deprecated*)*, 2
get.legacyExecOption
*(*rgrass7-deprecated*)*, 2
get.pluginOption *(*rgrass7-deprecated*)*, 2
get.stop_on_no_flags_parasOption
*(*rgrass7-deprecated*)*, 2
get.suppressEchoCmdInFuncOption
*(*rgrass7-deprecated*)*, 2
get.useGDALOption *(*rgrass7-deprecated*)*, 2
get.useInternOption
*(*rgrass7-deprecated*)*, 2
getLocationProj *(*rgrass7-deprecated*)*, 2
getXMLencoding *(*rgrass7-deprecated*)*, 2
gmeta *(*rgrass7-deprecated*)*, 2
gmeta2grd *(*rgrass7-deprecated*)*, 2

initGRASS *(*rgrass7-deprecated*)*, 2

parseGRASS *(*rgrass7-deprecated*)*, 2
print.gmeta *(*rgrass7-deprecated*)*, 2
print.GRASS_interface_desc
*(*rgrass7-deprecated*)*, 2

read_RAST *(*rgrass7-deprecated*)*, 2
read_VECT *(*rgrass7-deprecated*)*, 2
readRAST *(*rgrass7-deprecated*)*, 2
readVECT *(*rgrass7-deprecated*)*, 2
remove_GISRC *(*rgrass7-deprecated*)*, 2
rgrass7-deprecated, 2

set.defaultFlagsOption
*(*rgrass7-deprecated*)*, 2
set.echoCmdOption *(*rgrass7-deprecated*)*, 2
set.GIS_LOCK *(*rgrass7-deprecated*)*, 2
set.ignore.stderrOption
*(*rgrass7-deprecated*)*, 2
set.legacyExecOption
*(*rgrass7-deprecated*)*, 2
set.pluginOption *(*rgrass7-deprecated*)*, 2
set.stop_on_no_flags_parasOption
*(*rgrass7-deprecated*)*, 2
set.suppressEchoCmdInFuncOption
*(*rgrass7-deprecated*)*, 2
set.useGDALOption *(*rgrass7-deprecated*)*, 2
set.useInternOption
*(*rgrass7-deprecated*)*, 2
setXMLencoding *(*rgrass7-deprecated*)*, 2
stringexecGRASS *(*rgrass7-deprecated*)*, 2

unlink_.gislock *(*rgrass7-deprecated*)*, 2
unset.GIS_LOCK *(*rgrass7-deprecated*)*, 2
use_sf *(*rgrass7-deprecated*)*, 2
use_sp *(*rgrass7-deprecated*)*, 2

vColumns *(*rgrass7-deprecated*)*, 2
vDataCount *(*rgrass7-deprecated*)*, 2
vect2neigh *(*rgrass7-deprecated*)*, 2
vInfo *(*rgrass7-deprecated*)*, 2

write_RAST *(*rgrass7-deprecated*)*, 2
write_VECT *(*rgrass7-deprecated*)*, 2
writeRAST *(*rgrass7-deprecated*)*, 2
writeVECT *(*rgrass7-deprecated*)*, 2